

# SKRIPSI

**PERANCANGAN DAN PEMBUATAN SISTEM PENGATURAN  
LAMPU LALU LINTAS DILENGKAPI PERALATAN *CELLULAR*  
SEBAGAI PENGENDALI**



*Disusun Oleh:*  
**BAMBANG HUSONO**  
NIM: 02.17.086

**KONSENTRASI TEKNIK ELEKTRONIKA  
JURUSAN TEKNIK ELEKTRO S-1  
FAKULTAS TEKNOLOGI INDUSTRI  
INSTITUT TEKNOLOGI NASIONAL MALANG  
2009**

---

**LEMBAR PERSETUJUAN**

**PERANCANGAN DAN PEMBUATAN SISTEM PENGATURAN  
LAMPU LALU LINTAS DILENGKAPI PERALATAN  
CELLULAR SEBAGAI PENGENDALI**

**SKRIPSI**

*Disusun dan Diajukan Untuk Melengkapi dan Memenuhi Syarat  
Guna Mencapai Gelar Sarjana Teknik*

Disusun oleh:

**BAMBANG HUSONO**

**NIM : 02.17.086**

**Diperiksa dan Disetujui  
Dosen Pembimbing**

**Mengetahui  
Ketua Jurusan T. Elektro S-1**

**( Ir. F. Yudi Limpraptono, MT )  
NIP.Y.1039500274**

**( Ir. F. Yudi Limpraptono, MT )  
NIP.Y.1039500274**

**KONSENTRASI TEKNIK ELEKTRONIKA  
JURUSAN TEKNIK ELEKTRO S-1  
FAKULTAS TEKNOLOGI INDUSTRI  
INSTITUT TEKNOLOGI NASIONAL MALANG  
2009**



INSTITUT TEKNOLOGI NASIONAL MALANG  
FAKULTAS TEKNOLOGI INDUSTRI  
JURUSAN TEKNIK ELEKTRO S-1  
KONSENTRASI TEKNIK ELEKTRONIKA

**BERITA ACARA UJIAN SKRIPSI  
FAKULTAS TEKNOLOGI INDUSTRI**

Nama Mahasiswa : BAMBANG HUSONO  
NIM : 02.17.086  
Jurusan : Teknik Elektro S-1  
Konsentrasi : Teknik Elektronika  
Judul Skripsi : "PERANCANGAN DAN PEMBUATAN  
SISTEM PENGATURAN LAMPU LALU  
LINTAS DILENGKAPI PERALATAN  
CELLULAR SEBAGAI PENGENDALI"

Dipertahankan Dihadapan Team Penguji Skripsi Jenjang Strata Satu ( S-1), pada:

Hari : Senin  
Tanggal : 23 Maret 2009  
Dengan nilai : 73,7 (B+) *af*

**Panitia Ujian Skripsi**



**Ketua Majelis Penguji**

( Ir.Sidik Noertjahjono, MT )  
NIP.Y.1028700163

**Sekretaris Majelis Penguji**

( Ir.F.Yudi Limpraptono, MT )  
NIP.Y.1039500274

**Anggota Penguji**

**Penguji I**

( Ir. Eko Nurcahyo )  
NIP.Y.1028700172

**Penguji II**

( M.Ibrahim Ashari, ST, MT )  
NIP.P.1030100358

## ABSTRAKSI

### PERANCANGAN DAN PEMBUATAN SISTEM PENGATURAN LAMPU LALU LINTAS DILENGKAPI PERALATAN *CELLULAR* SEBAGAI PENGENDALI

(Bambang Husono, 02.17.086, Teknik Elektro/Elektronika S-1)

(Dosen Pembimbing Ir.F.Yudi Limpraptono,MT)

Kata kunci : *Mikrokontroller, Seven Segment, Lampu Lalu lintas*

Kemajuan Teknologi dibidang elektronika pada saat ini telah berkembang sangat pesat. Hal ini berdampak dengan terciptanya alat-alat elektronika yang dapat membantu manusia dalam kehidupan sehari-hari. Salah satu permasalahan yang dapat diselesaikan adalah masalah pengaturan lampu lalu lintas. karena apabila kontrol dari lampu lalu lintas akan dirubah, maka harus datang ke tempat lampu lalu lintas tersebut berada. Sehingga perlu adanya solusi untuk lampu lalu lintas kontrol dari jarak jauh. Sistem yang dimaksud adalah pengontrolan lampu lalu lintas jarak jauh menggunakan *handphone* dengan fasilitas SMS.

Dalam pembuatan sistem kontrol ini menggunakan mikrokontroller AVR ATmega8535, *seven segment, lampu lalu lintas*, sedangkan untuk pengiriman pesan menggunakan *handphone siemens M35*. Sistem kontrol ini sangat efektif dan efisien baik waktu dan tenaga selama proses berlangsung. serta mempercepat dalam memberikan informasi apabila lampu lalu lintas mati sehingga dapat langsung diantisipasi.

## KATA PENGANTAR

Puji syukur penulis panjatkan atas Berkat Rahmat Allah Yang Maha Kuasa, berupa nikmat kesehatan dan kesempatan. Shalawat beserta Salam kepada Nabi Besar Muhammad SAW sebagai pembawa kebenaran dan pencerang di muka bumi ini, sehingga penulis dapat menyelesaikan laporan Skripsi dengan judul:

*"Perancangan dan Pembuatan Sistem Pengaturan Lampu Lalu Lintas Dilengkapi Peralatan Cellular Sebagai Pengendali"*

Pembuatan Skripsi ini disusun guna memenuhi syarat akhir kelulusan pendidikan jenjang Strata-1 di Institut Teknologi Nasional Malang. Laporan Skripsi ini merupakan tanggung jawab tertulis atas ilmu pengetahuan yang didapat selama penyusun mengikuti kuliah.

Atas terselesaikannya Skripsi ini, penulis mengucapkan terima kasih kepada:

- ❖ Bapak Dr.Ir.Abraham Lomi,MSEE selaku Rektor Institut Teknologi Nasional Malang.
- ❖ Bapak Ir.Sidik Noertjahjono,MT selaku Dekan Fakultas Teknologi Industri Institut Teknologi Nasional Malang
- ❖ Bapak Ir.Yudi Limpraptono,MT selaku Ketua Jurusan Teknik Elektro SI.
- ❖ Bapak Ir.Yudi Limpraptono,MT selaku Dosen Pembimbing yang telah banyak memberikan masukan, bimbingan, pengarahan, serta ilmu-ilmu yang sangat bermanfaat sehingga Skripsi ini dapat terselesaikan.
- ❖ Bapak dan Ibu dosen Institut Teknologi Nasional Malang atas ilmu yang telah diberikan selama perkuliahan.
- ❖ Kedua Orang tua tercinta terima kasih atas Do'a, semua cinta, kasih sayang, yang telah banyak membantu dan selalu memberikan dorongan, semangat dan pengorbanan besar yang tulus sehingga penulis bisa mempersembahkan setitik keberhasilan ini. Semoga saya selalu menjadi anak yang shaleh dan berbakti.

- ❖ Adik-adikku tercinta, terima kasih atas doa, *motivasi*, dan bantuannya baik *moril* maupun *material*, *All of my cousins* atas celotchan yang membuat hari-hariku lebih indah *I am proud be the part of our big family. A lot of love for you.*
- ❖ Riska Handayani yang tanpa henti-hentinya selalu memberikan bantuan, semangat serta dukungan sehingga penulis tetap optimis untuk tegak melangkah dalam menggapai cita-cita. *No body perfect, but you're the most perfect girl that i ever had. I love you.*
- ❖ Teman-teman Kuliah Teknik Elektronika 2002 yang telah membantu dalam penyelesaian Skripsi ini.
- ❖ Terima kasih untuk orang-orang yang telah mendukungku dan merasa direpotkan, yang tak dapat kusebutkan satu persatu.

Penulis menyadari bahwa laporan ini masih banyak yang perlu disempurnakan. Oleh sebab itu kritik dan saran yang membangun sangat diharapkan.

Akhir kata, penulis mohon maaf kepada semua pihak bila mana selama penyusunan Skripsi ini penyusun membuat kesalahan secara tidak sengaja dan semoga Skripsi ini dapat bermanfaat bagi kita semua.

Malang, 2009

Penulis

## DAFTAR ISI

<b>JUDUL</b> .....	<b>i</b>
<b>LEMBAR PERSETUJUAN</b> .....	<b>ii</b>
<b>BERITA ACARA</b> .....	<b>iii</b>
<b>ABSTRAKSI</b> .....	<b>iv</b>
<b>KATA PENGANTAR</b> .....	<b>v</b>
<b>DAFTAR ISI</b> .....	<b>vii</b>
<b>DAFTAR GAMBAR</b> .....	<b>x</b>
<b>DAFTAR TABEL</b> .....	<b>xii</b>
<b>BAB I PENDAHULUAN</b> .....	<b>1</b>
1.1. Latar Belakang .....	1
1.2. Tujuan .....	2
1.3. Rumusan Masalah .....	2
1.4. Batasan Masalah .....	3
1.5. Metodologi Penulisan .....	4
1.6. Sistematika Penulisan .....	4
<b>BAB II DASAR TEORI</b> .....	<b>6</b>
2.1. Umum .....	6
2.2. SMS ( Short Message Service ) .....	7
2.2.1. PDU (Protocol Data Unit) SMS .....	8
2.2.1.1. Delapan Header Untuk Kirim SMS .....	9
2.2.1.2. PDU Untuk SMS Terima Dari SMS Centre .....	18
2.3. Handphone Siemens M35 .....	19
2.3.1. Pin Out Handphone Siemens M35 .....	19

	2.3.2. Kabel Data Siemens M35 .....	20
	2.3.3. Perintah AT Command .....	21
	2.4. Mikrokontroler AVR (Alf and Vegard's Risc Processors) .....	22
	2.4.1. Arsitektur Atmega 8535 .....	23
	2.4.2. Konfigurasi Pin Atmega 8535 .....	26
	2.4.3. Memori Data dan Memori Program Atmega 8535 .....	28
	2.5. Seven Segment .....	29
	2.6. LED (Light Emiting Diode) .....	31
<b>BAB</b>	<b>III : PERANCANGAN ALAT .....</b>	<b>33</b>
	3.1. Blok Diagram Sistem .....	33
	3.2. Perencanaan Perangkat Keras (Hardware) .....	35
	3.2.1. Sistem Mikrokontroler AVR ATmega 8535 ....	35
	3.2.2. Antarmuka Komunikasi Serial .....	38
	3.2.3. Rangkaian Lampu .....	40
	3.2.4. Rangkaian Driver Seven segment .....	41
	3.2.4. Rangkaian Sensor Supply .....	42
	3.3. Perencanaan Perangkat Lunak (Software) .....	43
<b>BAB</b>	<b>IV PENGUJIAN ALAT .....</b>	<b>45</b>
	4.1. Pengujian Rangkaian ATmega 8535 .....	45
	4.2. Pengujian Antarmuka Komunikasi Serial .....	47
	4.3. Pengujian Rangkaian Lampu .....	50
	4.4. Pengujian Rangkaian Driver Seven segment .....	53
	4.5. Pengujian Rangkaian Keseluruhan .....	56



<b>BAB</b>	<b>V</b>	<b>PENUTUP</b> .....	<b>58</b>
		5.1. Kesimpulan .....	58
		5.2. Saran .....	58
<b>DAFTAR PUSTAKA</b> .....			<b>63</b>
<b>LAMPIRAN</b> .....			<b>64</b>

## DAFTAR GAMBAR

Gambar	2-1	Blok Diagram Arsitektur ATmega 8535 .....	24
Gambar	2-2	Konfigurasi Pin ATmega 8535 .....	27
Gambar	2-3	Memori Data ATmega 8535 .....	28
Gambar	2-4	Memori Program ATmega 8535 .....	29
Gambar	2-5	Peraga Seven segment .....	30
Gambar	2-6	Lambang Skematis dan Contoh Rangkaian Dari LED ..	32
Gambar	3-1	Blok Diagram Sistem .....	33
Gambar	3-2	Sistem Mikrokontroller AVR ATmega 8535 .....	36
Gambar	3-3	Konektor Handphone Siemens M35 .....	38
Gambar	3-4	Antarmuka Komunikasi Serial Mikrokontroller Dengan Handphone .....	39
Gambar	3-5	Rangkaian Lampu .....	40
Gambar	3-6	Rangkaian Driver Seven segment .....	41
Gambar	3-7	Rangkaian Sensor Supply .....	42
Gambar	3-8	Flowchart Sistem .....	44
Gambar	4-1	Rangkaian Pengujian ATmega 8535 .....	46
Gambar	4-2	Rangkaian Pengujian Antarmuka Serial .....	49
Gambar	4-3	Rangkaian Pengujian Lampu .....	51
Gambar	4-4	Rangkaian Pengujian Driver Seven segment .....	54
Gambar	4-5	Blok Diagram Pengujian Secara Keseluruhan .....	57
Gambar	4-6	Foto Alat Keseluruhan .....	57
Gambar	4-7	Foto Alat Tampak dari Dalam.....	58
Gambar	4-8	Foto Tampilan Jalur 1 ON Seven Segment tidak Bekerja .....	59

Gambar 4-9	Foto Tampilan Jalur 2 ON Seven Segment tidak Bekerja .....	59
Gambar 4-10	Foto Tampilan Jalur 3 ON Seven Segment tidak Bekerja .....	59
Gambar 4-11	Foto Tampilan Jalur 4 ON Seven Segment tidak Bekerja .....	60
Gambar 4-12	Foto Tampilan Jalur OFF Seven Segment Bekerja .....	60

## DAFTAR TABEL

Tabel	2-1	Skema 7 Bit .....	17
Tabel	2-2	Pin Out HP Siemens .....	20
Tabel	2-3	Kabel Data Siemens M35 .....	21
Tabel	3-1	Pin Out Konektor Handphone Siemens .....	39
Tabel	4-1	Hasil Pengujian Rangkaian ATmega 8535 .....	47
Tabel	4-2	Hasil Pengujian Antarmuka Serial .....	50
Tabel	4-3	Hasil Pengujian Rangkaian Lampu .....	52
Tabel	4-4	Hasil Pengujian Rangkaian Driver Seven segment .....	55
Tabel	4-5	Hasil Pengujian Secara Keseluruhan .....	60

# BAB I

## PENDAHULUAN

### 1.1. Latar Belakang

Semakin cepatnya perkembangan teknologi jaringan komputer yang ditandai dengan munculnya teknologi-teknologi baru. Dilihat dari banyaknya persaingan di bidang industri dan teknologi yang didukung oleh kualitas maupun kuantitas suatu produk menjadi salah satu acuan dalam mengembangkan teknologi. Kemampuan inilah yang menjadi syarat untuk mewujudkan modul sistem peralatan dalam pengaturan lampu lalu lintas berbasis mikrokontroller ATmega8535 yang memanfaatkan kerja sistem *cellular*. Dengan penerapan mikrokontroller ATmega8535 hal ini memungkinkan dilakukan pemanfaatan media elektronik seperti halnya dalam mengatur sistem kerja lampu lalu lintas. Dalam hal ini bagaimana cara memanfaatkan dan memfungsikan handphone menjadi peralatan yang mampu mengendalikan lampu lalu lintas yang menggunakan teknologi *via sms*.

Lampu lalu lintas memegang peranan penting dalam pengaturan kelancaran lalu lintas. Sistem pengendalian lampu lalu lintas yang baik akan secara otomatis menyesuaikan diri dengan kepadatan arus lalu lintas pada jalur yang diatur. Dengan penerapan Mikrokontroller ATmega8535 hal ini sangat memungkinkan untuk dilakukan.

Berdasarkan perkembangan teknologi komunikasi sekarang ini, semakin banyak sistem komunikasi yang diterapkan di lembaga pemerintahan akan sangat

bermanfaat bagi masyarakat pada umumnya. Dimana kita ketahui bahwa penerapan ilmu pengetahuan pada perguruan tinggi memberikan solusi dalam pengembangan ilmu pengetahuan yang selanjutnya dapat meningkatkan SDM (sumber daya manusia) Mengingat sumber daya manusia sangat rendah dan minimnya penerapan ilmu pengetahuan dan sangat penting untuk setiap generasi, maka perlu adanya pengembangan sistem pengetahuan teknologi. Pemakaian teknologi sekarang ini sangat minim bagi sebagian masyarakat, maka dari itu solusi yang tepat untuk penggunaan teknologi yang ada dengan memanfaatkan sebagian teknologi yang sudah ada. Mencoba untuk memberikan satu solusi dalam pengembangan ilmu pengetahuan yang berkaitan dengan teknologi dimana peran alat tersebut berfungsi dalam pengaturan lampu lalu lintas, Sehingga alat yang dirancang mampu bekerja secara optimal.

## **1.2. Tujuan**

Tujuan skripsi ini adalah untuk mengetahui sejauh mana kinerja sistem yang telah dibuat, yaitu sistem pengaturan lampu lalu lintas yang dilengkapi peralatan *cellular* sebagai pengendali melalui fasilitas *via sms*.

## **1.3. Rumusan Masalah**

Berdasarkan pada uraian latar belakang masalah, maka dapat disusun rumusan masalah. Adapun rumusan masalah dalam skripsi ini adalah:

1. Bagaimana merancang komunikasi serial antara *handphone* dengan mikrokontroler.

2. Bagaimana merancang sebuah sistem pengaturan lampu lalu lintas dengan peralatan *cellular* sebagai pengendali.
3. Bagaimana membuat *software* yang dapat mendeteksi sms datang dan mengolahnya ke mikrokontroller.
4. Bagaimana merancang dan membuat perangkat lunak atau *software* pada mikrokontroller yang mengendalikan semua kerja sistem agar dapat bekerja sesuai dengan yang telah direncanakan.

#### 1.4. Batasan Masalah

Agar permasalahan yang dibahas tidak melebihi dari hal-hal yang jauh dari bahasan masalah, maka diberi batasan-batasan masalah sebagai berikut :

1. Tidak membahas *hardware handphone siemens M35* secara mendetail.
2. Tidak membahas *handphone* dalam keadaan mati atau diluar jangkauan.
3. Hanya menggunakan *handphone siemens M35* dengan fasilitas sms sebagai informasi dan proses mengaktifkan dan mematikan system.
4. Menggunakan mikrokontroller AVR ATmega8535.
5. Menggunakan *seven segment* untuk tampilan waktu.
6. Tidak membahas prinsip kerja kerja lampu lalu lintas.
7. Tidak membahas *catu daya*.

## 1.5. Metodologi Penulisan

Metodologi penulisan skripsi ini adalah sebagai berikut:

### 1. Studi literatur

Memperoleh data dari internet dan membaca buku-buku literatur yang berhubungan dengan permasalahan dalam penyusunan skripsi ini.

### 2. Pengumpulan data

Melakukan tanya jawab dengan pihak-pihak yang berkompeten dengan teknik-teknik yang dipakai dalam pembuatan lampu lalu lintas.

### 3. Perancangan *hardware* dan *software*

*Hardware* meliputi perancangan dan pembuatan rangkaian secara keseluruhan berupa pembuatan PCB dan perakitan alat dapat bekerja, Melakukan desain aplikasi tampilan program dan desain *software* untuk lampu lalu lintas.

### 4. Pengolahan data

Menganalisa pola kerja dari alat tersebut dengan jalan membuat analisa dan menarik kesimpulan dari hasil pengujian yang ada.

## 1.6. Sistematika Penulisan

Sistematika dalam penulisan skripsi ini adalah sebagai berikut :

### **BAB I : PENDAHULUAN**

Membuat latar belakang, tujuan, rumusan masalah, batasan masalah dan sistematika penulisan.



## **BAB II : DASAR TEORI**

Membahas semua teori dasar yang berkaitan langsung dengan perencanaan sistem alat.

## **BAB III : PERANCANGAN DAN PEMBUATAN ALAT**

Membahas perancangan dan pembuatan sistem dari alat yang meliputi *hardware* dan *software*.

## **BAB IV : PENGUJIAN ALAT**

Membahas tentang pengujian dan pengukuran karakteristik dari hasil perancangan *software* dan *hardware* setelah pengambilan *sampel* dari hasil pengujian alat, setelah itu dilakukan analisa hasil pengujian alat tersebut.

## **BAB V : PENUTUP**

Berisi kesimpulan dari keseluruhan dari perancangan dan pembuatan alat, serta berisi saran untuk pengembangan alat.

## BAB II

### DASAR TEORI

#### 2.1. Umum

Pengiriman pesan menggunakan teknologi *celuler* dipercaya menjadi *trend* terbaru dan dapat diaplikasikan untuk berbagai kebutuhan. *Telepon celuler* dengan fasilitas sms mampu bertukar informasi berbasis teks secara jarak jauh (*remote*) dan tanpa kabel (*wireless*) sehingga dapat memberikan solusi yang tepat terhadap masalah pengontrolan lampu lalu lintas secara jarak jauh. Ditambah dengan dukungan teknologi mikrokontroller yang memungkinkan dibentuknya sebuah sistem yang memiliki efisiensi daya dan tempat, menjadikan telepon *celuler* sebagai sarana alternatif selain sebagai sarana komunikasi juga dapat dijadikan sebagai sarana pengendali jarak jauh.

Penggunaan mikrokontroller dalam berbagai aplikasi memang memberikan banyak keuntungan tapi juga tidak luput dari kekurangan. Keuntungan yang dapat diperoleh dengan menggunakan mikrokontroller antara lain, banyak pilihan yang ditawarkan tergantung kebutuhan, murah, bisa digunakan untuk bermacam-macam aplikasi, berdaya rendah, dan hanya memerlukan sedikit tambahan komponen luar dan proses penanganannya yang mudah baik dari segi operasi maupun aplikasinya. Sedangkan kekurangan dari mikrokontroller adalah keterbatasan memori didalamnya sehingga tidak mampu menangani program-program yang cukup besar dan rumit.

## 2.2. SMS (*Short Message Service*)

SMS merupakan salah satu layanan pesan teks yang dikembangkan dan distandarisasi oleh suatu badan yang bernama ETSI (*European Telecommunication Standards Institute*) sebagai dari pengembangan GSM *phase 2*, yang terdapat pada dokumentasi GSM 03.40 dan GSM 03.38. Dengan adanya fitur sms ini maka perangkat stasiun *celuler* digital (*Digital Cellular Terminal*, seperti *ponsel*) dapat mengirim dan menerima pesan-pesan teks dengan panjang sampai dengan 160 karakter melalui jaringan GSM.

Selama berada pada jangkauan pelayanan GSM, hanya dalam beberapa detik sms dapat dikirimkan ke perangkat stasiun *celuler* digital lainnya. Lebih dari sekedar pengiriman pesan biasa, layanan sms memberikan garansi sms akan sampai pada tujuan, meskipun perangkat yang dituju sedang tidak aktif yang dapat disebabkan karena sedang dalam kondisi mati atau berada di luar jangkauan layanan GSM. Pesan yang belum terkirim akan disimpan sementara oleh jaringan, dan akan segera dikirimkan ke perangkat yang dituju setelah adanya tanda kehadiran dari perangkat di jaringan tersebut.

Dengan fakta bahwa layanan sms (melalui jaringan GSM) hingga jangkauan nasional dan internasional dengan waktu keterlambatan yang sangat kecil, sehingga dimungkinkan layanan sms cocok untuk dikembangkan dalam aplikasi-aplikasi seperti: *pager*, *e-mail*, dan notifikasi *voice mail*, serta layanan pesan banyak pemakai (*multiple users*). Namun pengembangan aplikasi tersebut masih bergantung pada tingkat layanan yang disediakan oleh operator jaringan.

### 2.2.1. PDU (*Protocol Data Unit*) SMS

Dalam proses pengiriman atau penerimaan pesan pendek (sms), data yang dikirim maupun diterima oleh stasiun bergerak menggunakan salah satu dari 2 mode yang ada, yaitu: mode teks, atau mode PDU (*Protocol Data Unit*). Dalam mode PDU, pesan yang dikirim berupa informasi dalam bentuk data dengan beberapa kepala-kepala informasi. Hal ini dipermudah jika dalam pengiriman akan dilakukan kompresi data, atau akan dibentuk sistem penyandian data dari karakter dalam bentuk untaian bit-bit biner. Senarai PDU tidak hanya berisi pesan teks saja, tetapi terdapat beberapa meta informasi yang lainnya, seperti nomor pengirim, nomor sms *centre*, waktu pengiriman, dan sebagainya. Semua informasi yang terdapat dalam PDU, dituliskan dalam bentuk pasangan pasangan bilangan *heksadesimal* yang disebut dengan pasangan *oktet*.

Data yang mengalir ke/dari sms *centre* harus berbentuk PDU (*Protocol Data Unit*). PDU berisi bilangan-bilangan *heksadesimal* yang mencerminkan bahasa I/O. PDU terdiri atas beberapa *header*. *Header* untuk kirim sms ke sms *centre* berbeda dengan sms yang diterima dari sms *centre*.

Maksud dari bilangan *heksadesimal* adalah bilangan yang terdiri atas 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Sebagai contoh, untuk angka desimal 1000, bilangan heksa desimalnya adalah 3E8.

Cara mengkonversikanya :

$$1000 : 16 = 62 \text{ sisa } 8 = \underline{8}$$

$$62 : 16 = 3 \text{ sisa } 14 = \underline{E}$$

$$3 : 16 = 0 \text{ sisa } 3 = \underline{3}$$

### 2.2.1.1. Delapan *Header* Untuk Kirim SMS

PDU untuk mengirim sms terdiri atas delapan *header*, sebagai berikut:

1. Nomor sms *centre*

*Header* pertama ini terbagi atas tiga bagian *subheader*, yaitu:

- a. Jumlah pasangan *hexsadesimal* sms *centre* dalam bilangan *heksa*.
- b. National/International *code*
  - untuk National, kode *subheader*nya yaitu 81
  - untuk International, kode *subheader*nya yaitu 91
- c. No sms *centrenya* sendiri, dalam pasangan *heksa* dibalik-balik.

Jika tertinggal satu angka *heksa* yang tidak memiliki pasangan, angka tersebut akan dipasangkan dengan huruf F didepannya.

Contoh: untuk nomor sms *centre* Excelcom dapat ditulis dengan dua cara sebagai berikut:

*Cara pertama:*

0818445009 diubah menjadi:

- a. 06 à karena  $1 + 5 = 6$  pasang
- b. 81 à 1 pasang
- c. 80-81-44-05-90 à 5 pasang

Digabung menjadi: 06818081440590

*Cara kedua:*

0818445009 diubah menjadi:

a. 07 à karena  $1 + 6 = 7$  pasang

b. 91 à 1 pasang

c. 26-18-48-54-00-F9à 6 pasang

Digabung menjadi: 07912618485400F9

Berikut ini berupa beberapa no sms *centre* operator *celuler* di Indonesia.

Cara pertama:

No	Operator Selular	SMS Centre No	Kode PDU
1.	Telkomsel	0811000000	06818011000000
2.	Satelindo	0816124	0581806121F4
3	Excelcom	0818445009	06818081440590
4.	Indosat-M3	0855000000	06818055000000

Cara kedua:

No	Operator Selular	SMS Centre No	Kode PDU
1.	Telkomsel	62811000000	07912618010000F0
2.	Satelindo	62816124	059126181642
3	Excelcom	62818445009	07912618485400F9
4.	Indosat-M3	62855000000	07912658050000F0

2. Tipe sms

Untuk tipe SEND tipe SMS=1. Jadi bilangan heksanya adalah 01.

3. Nomor *referensi* sms

Nomor *referensi* ini dibiarkan dulu 0, jadi bilangan *heksanya* adalah 00. Nanti akan diberikan sebuah nomor *referensi* otomatis oleh *handphone* atau alat sms *getway*.

4. Nomor *handphone* penerima

Sama seperti cara menulis PDU *header* untuk sms *centre*, *header* ini juga terbagi atas tiga bagian, sebagai berikut:

1. Jumlah bilangan *desimal* nomor *ponsel* yang dituju dalam bilangan *heksa*
2. National/International *code*.
  - ✓ Untuk national, kode *subheadernya*: 81
  - ✓ Untuk international, kode *subheadernya*: 91
3. Nomor *handphone* yang dituju, dalam pasangan *heksa* dibalik-balik. Jika tertinggal satu angka *heksa* yang tidak memiliki pasangan, angka tersebut dipasangkan dengan huruf F didepannya.

Contoh:

Untuk nomor *handphone* yang dituju = 628129573337 dapat ditulis dengan dua cara sebagai berikut:

Cara pertama: 08129573337 diubah menjadi:

- a. 0B : ada 11 angka
- b. 81
- c. 80-21-59-37-33-F7

Digabung menjadi: 0B818021593733F7

Cara kedua: 628129573337 diubah menjadi:

- a. 0C : ada 11 angka
- b. 91
- c. 26-18-92-75-33-73

Digabung menjadi: 0C91261892753373

5. Bentuk sms, antara lain:

0 -> 00 -> dikirim sebagai sms

1 -> 01 -> dikirim sebagai *telex*

2 -> 02 -> dikirim sebagai *fax*

Dalam hal ini, untuk mengirim dalam bentuk sms tentu saja kita memakai 00.

6. *Skema encoding data I/O*



Ada dua skema, yaitu:

1. Skema 7 bit ditandai dengan angka 0 = 00
2. Skema 8 bit ditandai dengan angka lebih besar dari 0 diubah ke *heksa*

Kebanyakan *handphone/sms gateway* yang ada di pasaran sekarang menggunakan skema 7 bit sehingga kita menggunakan kode 00.

7. Jangka waktu sebelum sms expired

Jika bagian ini *diskip*, itu berarti kita tidak membatasi waktu berlakunya sms. Sedangkan jika kita isi dengan suatu bilangan *integer* yang kemudian diubah ke pasangan *heksa* tertentu, bilangan yang kita berikan tersebut akan mewakili jumlah waktu validitas sms tersebut.

Rumus untuk menghitung jangka waktu validasi sms adalah sebagai berikut:

Integer (INT)	Jangka Waktu Validasi SMS
0-143	(INT +1) menit (berarti:5 menit s/d 12 jam)
144-167	12 jam + (INT-143) x 30 menit
168-196	(INT - 166) x 1 hari
197-255	(INT - 192) x 1 minggu

Agar sms kita pastikan terkirim sampai ke *handphone* penerima, sebaiknya kita tidak memberikan batasan waktu *validnya*.

#### 8. Isi sms

*Header* ini terdiri atas 2 *subheader*, yaitu:

##### 1. Panjang isi (jumlah huruf dari isi)

Misalnya untuk kata "hello" ada 5 huruf -> 05

##### 2. Isi berupa pasangan bilangan heksa

Untuk *handphone/sms gateway* berskema encoding 7 bit, jika kita mengetikkan suatu huruf dari keypadnya, berarti kita telah membuat 7 angka 1/0 berurutan.

Skema 7 bit tersebut diperlihatkan dibawah ini:

Ada dua langkah yang harus kita lakukan untuk mengkonversi isi sms, yaitu:

Langkah pertama: mengubahnya menjadi kode 7 bit

Langkah kedua: mengubahnya kode 7 bit menjadi 8 bit, yang diwakili oleh pasangan *heksa*.

Contoh: untuk kata "hello"

Langkah pertama:

Bit    7    1

h 110 1000

e 110 0101

l 110 1100

l 110 1100

o 110 1111

Langkah kedua:

E 8

h 1110 1000

3 2

c 0011 0010 1

9 B

l 1001 1011 00

F D

l 1111 1101 100

0 6

o 0000 0110 1111

Oleh karena itu total 7 bit x 5 huruf=35 bit, sedangkan yang kita perlukan adalah 8 bit x 5 huruf=40 bit, maka diperlukan 5 bit dummy yang diisi dengan bilangan0.

Setiap 8 bit mewakili suatu pasangan *heksa*. Tiap 4 bit mewakili suatu angka *heksa*, tentu saja karena secara logika  $2^4 = 16$ .

Dengan demikian kata “hallo” hasil konversinya menjadi: **E8329BFD06**

#### 9. Menggabungkan kedelapan *header*

Setelah kita mempelajari masing-masing *header* maupun *subheader* untuk mengirim sms di atas, kini kita akan menggabungkannya menjadi PDU yang lengkap.

Contoh:

Untuk mengirimkan kata “hallo” ke nomor handphone 618129573337 lewat sms *centre* Excelcom, tanpa membatasi jangka waktu valid, maka PDU lengkapnya adalah:

07912618485400F901000C912618927533730000005E8329BFD06

( 07912618485400F9,01,00,0C91261892753373,00,00,05E8329BFD06)

Tabel 2.1.

Skema 7 bit<sup>(81)</sup>

				b7	0	0	0	0	1	1	1	1
				B6	0	0	1	1	0	0	1	1
				B5	0	1	0	1	0	1	0	1
b4	b3	b2	B1		0	1	2	3	4	5	6	7
0	0	0	0	0	@	D	SP	0	-	P		p
0	0	0	1	1			!	1	A	Q	a	q
0	0	1	0	2		F	"	2	B	R	b	r
0	0	1	1	3	\$	G	#	3	C	S	c	s
0	1	0	0	4		L		4	D	T	d	t
0	1	0	1	5		W	%	5	E	U	e	u
0	1	1	0	6		P	&	6	F	V	f	v
0	1	1	1	7		Y	'	7	G	W	g	w
1	0	0	0	8		S	(	8	H	X	h	x
1	0	0	1	9		Q	)	9	I	Y	i	y
1	0	1	0	10	LF	X	*	:	J	Z	j	z
1	0	1	1	11			+	;	K	Ä	k	ä
1	1	0	0	12			,	<	L	Ö	l	ö
1	1	0	1	13	CR		-	=	M		m	
1	1	1	0	14		b	.	>	N	Ü	n	ü
1	1	1	1	15			/	?	O		o	

### 2.2.1.2. PDU Untuk SMS Terima Dari SMS Centre

Delapan *header* untuk sms terima. *Header* dibawah ini telah dibahas sebelumnya, kecuali beberapa yang berbeda, dijelaskan di bawah ini:

1. No sms centre.
2. Tipe sms à untuk sms terima = 4 -> **04**
3. Nomor handphone pengirim.
4. Bentuk sms.
5. Skema encoding.
6. Tanggal dan waktu sms *distamo* di sms *centre*.

Diwakili oleh 12 bilangan *heksa* (6 pasang) yang berarti:yy/mm/dd  
hh:mm:ss

Contoh: 207022512380à 01/07/22 15:32:08à 22 Juli 2002 15:32:08  
Wib

7. Batas validasi waktu jika tidak dibatasi dilambangkan dengan **00**
8. Isi sms.

Membedah kedelapan *header*

Setelah mengupas satu demi satu *header* untuk sms terima ini, maka untuk PDU dibawah ini:

```
07912658050000F0040C9126581610739800002070225123800005E83  
29BFD06
```

07912658050000F0,04,0C91265816107398,00,00,207022512380,00,05

E8329BFD06

Dapat diartikan sebagai berikut:

1. SMS tersebut dikirim lewat sms *centre*:62855000000
2. SMS tersebut merupakan sms terima
3. SMS tersebut dikirim dari handphone dengan nomor *Sim Card*:628561013789
4. SMS tersebut diterima dalam bentuk sms
5. SMS tersebut memiliki skema encoding 7 bit
6. SMS tersebut sampai di sms centre pada tanggal 22-07-02, pukul: 15:32:08 Wib
7. SMS tersebut tidak memiliki batas waktu valid
8. SMS tersebut isinya adalah "hello".

### **2.3. Handphone Siemens M35**

#### **2.3.1. Pin Out Handphone Siemens M35**

Untuk melakukan koneksi dengan mikrokontroler, pada *handphone* siemens M35 terdapat *pin out* dengan susunan seperti yang ditunjukkan dalam tabel berikut ini.

**Tabel 2.2.**  
Pin out HP Siemens<sup>[7]</sup>

<b>Pin</b>	<b>Nama</b>	<b>Fungsi</b>	<b>In/Out</b>
1	GND	Ground	
2	Self Service	Recognition / control battery	In
3	Load	Charging Voltage	Out
4	Battery	Battery	In
5	Data Out	Data Sent	Out
6	Data In	Data Received	Out
7	Z_Clk	Recognition / control accessories	In
8	Z_Data	Recognition / control accessories	
9	MICG	Ground for microphone	
10	MIC	Microphone input	In
11	AUD	Loudspeaker	
12	AUDG	Ground for eksternal LS	Out

### 2.3.2. Kabel Data Siemens M35

Siemens dilengkapi dengan kabel data untuk melakukan komunikasi data secara serial dengan terminal lain. Kecepatan transfer data sebesar 19200. Untuk mengadakan komunikasi serial, pin-pin yang digunakan seperti tabel 2.3. berikut ini:



**Tabel 2.3.**  
Kabel Data Siemens M35<sup>[7]</sup>

Nomor Pin	Nama	Fungsi
1	GND	Ground
5	TFMS/DFMS – Terminal adaptor equipment From Mobile Station Data From Mobile Station	Serial Data Out (Out)
6	TTMS/DTMS – Terminal adaptor equipment To Mobile Station Data To Mobile Station	Serial Data In (RX)

### 2.3.3. Perintah *AT Command*

Perintah *AT (Hayes AT Command)* digunakan untuk berkomunikasi dengan terminal (*modem*) melalui gerbang serial pada komputer. Dengan menggunakan perintah *AT command* dapat diketahui atau dibaca kondisi dari terminal seperti mengetahui kondisi sinyal, kondisi baterai, mengirim pesan, membaca pesan, menambah item pada daftar telepon, dan sebagainya. Berikut beberapa jenis perintah *AT command* yang berhubungan dengan penanganan pesan-pesan SMS.

- ❖ *AT+CMGS*, perintah ini digunakan untuk mengirim pesan
- ❖ *AT+CMGR*, perintah ini digunakan untuk membaca pesan
- ❖ *AT+CMGF*, perintah ini digunakan untuk format pesan

- ❖ AT+CMGD, perintah ini digunakan untuk menghapus pesan
- ❖ AT+CNMI, perintah ini digunakan untuk prosedur indikasi pesan baru
- ❖ AT+CPMS, perintah ini digunakan untuk pemilihan target memori
- ❖ AT+CSMS, perintah ini digunakan untuk pemilihan layanan pesan

#### 2.4. Mikrokontroler AVR (*Alf and Vegard's Risc Processors*)

Atmel sebagai salah satu vendor yang mengembangkan dan memasarkan produk-produk mikroelektronika telah dijadikan sebagai suatu teknologi standar bagi para desainer masa kini. Dengan perkembangan terkini adalah generasi AVR (*Alf and Vegard's Risc Processors*).

Mikrokontroler AVR adalah mikrokontroler dengan arsitektur RISC 8 bit, dimana semua instruksi dikemas dalam kode 16 bit (*16 bits word*) dan sebagian besar instruksi dieksekusi dalam 1 siklus clock, berbeda dengan instruksi MCS51 yang dieksekusi dalam 12 siklus clock. Kedua jenis mikrokontroler tersebut dibedakan oleh perbedaan arsitektur yang dimiliki. AVR berteknologi RISC (*Reduced Instruction Set Computing*), sedangkan seri MCS51 berteknologi CISC (*Complex Instruction Set Computing*).

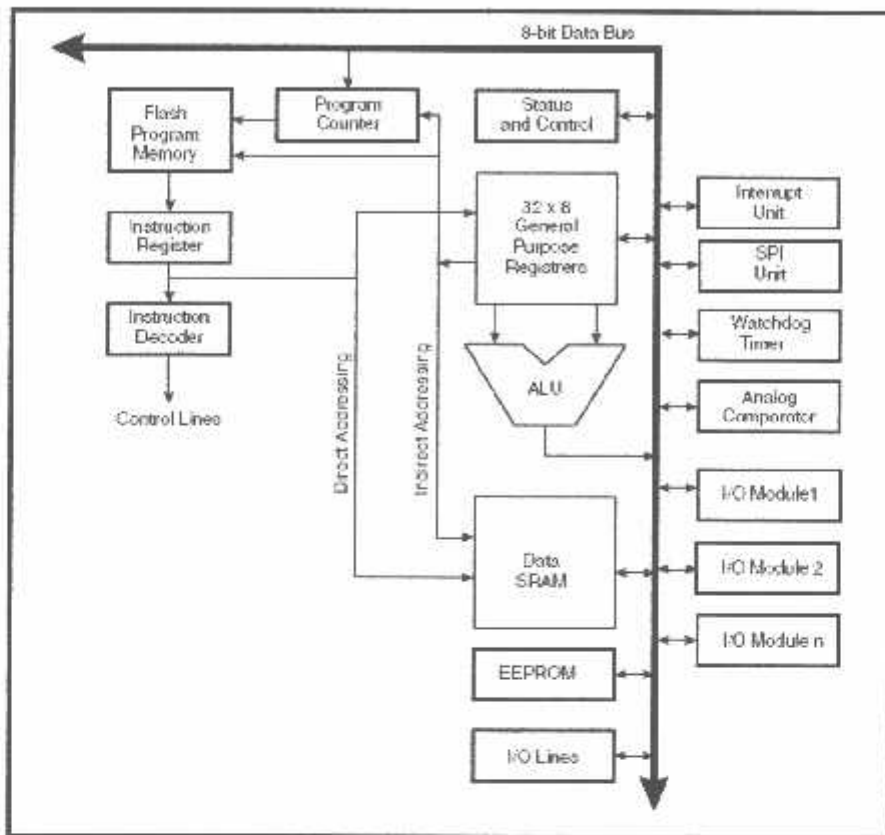
Secara umum, mikrokontroler jenis AVR dapat dikelompokkan menjadi 4 kelas, yaitu keluarga Atiny, keluarga AT90Sxx, keluarga ATmega, dan AT86RFxx. Pada dasarnya masing-masing kelas dibedakan oleh memori, peripheral, dan fungsinya. Dari segi arsitektur dan instruksi yang dipergunakan,

mereka bisa dikatakan hampir sama. Berdasarkan uraian diatas maka dalam penulisan skripsi ini digunakan mikrokontroler ATmega8535.

#### **2.4.1. Arsitektur ATmega8535**

ATmega8535 adalah mikrokontroller 8 bit berteknologi CMOS yang berdaya rendah berbasis AVR dengan arsitektur RISC. Dengan keuntungan eksekusi instruksi rata-rata satu siklus clock merupakan pendekatan 1 MIPS (*million instruction Per Second*) memberikan perancangan sistem dengan konsumsi daya optimal versus kecepatan proses.

Inti kombinasi AVR adalah kombinasi instruksi yang beraneka ragam dengan 32 register (*32 general purpose working register*). Ke 32 register tersebut langsung dihubungkan ke ALU (*Arithmetic Logic Unit*), melalui dua register independen, sehingga dapat diakses dalam satu instruksi yaitu satu siklus clock. Hasil dari arsitektur ini lebih efisien sepuluh kali dari mikrokontroler jenis CISC seperti AT89Cxxx atau AT89Sxxx.



**Gambar 2.1.**

Blok Diagram Arsitektur ATmega8535<sup>[1]</sup>

Mikrokontroler ATmega8535 memiliki model arsitektur *harvard*, dimana memori untuk data dan program terpisah, bus untuk program dan bus untuk data juga terpisah. Dalam arsitektur ATmega8535, seluruh GPR (*General Purpose Register [32-register]*) terhubung langsung ke ALU (*Arithmetic Logic Unit*)/*procesor*. Sehingga eksekusi instruksi lebih cepat. Dalam satu siklus clock terdapat dua register independen yang dapat diakses oleh satu instruksi. Teknik yang digunakan adalah *fetch during execution* atau memegang sambil mengerjakan. Hal ini berarti, dua operan dibaca dari satu register, dilakukan

eksekusi operasi, dan hasilnya disimpan kembali dalam satu register, semuanya dilakukan hanya dalam satu siklus clock.

Dari 32 register terdapat enam buah register yang dapat digunakan untuk pengalamatan tidak langsung 16 bit sebagai register *pointer* (pcnunjuk). Register tersebut memiliki nama khusus, yaitu X, Y, Z. Masing-masing terdiri dari sepasang register, seperti: X (R27:R26), Y (R29:R28), dan Z (R31:R30). ALU mendukung operasi bit, fungsi aritmetika dan logika antara register dengan register atau antara register dengan nilai konstan, atau hanya operasi satu register.

Untuk kontrol aliran program disediakan intruksi lompatan bersyarat dan tak bersyarat, intruksi *Call* (panggil), dapat ditempatkan di seluruh ruangan program. Kebanyakan instruksi ATmega8535 mempunyai format 16 bit *word*. Setiap alamat memori program mengandung sebuah instruksi 16 atau 32 bit.

Selama interupsi dan pemanggilan subrutin, alamat program counter (PC) di simpan kedalam Stack. Stack akan efektif diletakkan di SRAM, dan konsekuensinya ukuran stack dibatasi oleh ukuran SRAM dan penggunaan SRAM. Setiap pengguna dapat menginisialisasi SP (Stack pointer) dalam rutin RESET (*sebelum subrutin atau interupsi dieksekusi*). Stack pointer SP dapat ditulis dan di baca dalam ruangan I/O.

Kapabilitas detail dari ATmega8535 adalah sebagai berikut:

- Sistem mikroprosesor 8 bit berbasis RISC dengan kecepatan maksimal 16 MHz.

- Kapabilitas *memory flash* 8 KByte, SRAM sebesar 512 Byte, dan EEPROM (*Electrically Erasable Programmable Read Only Memory*) sebesar 512 Byte.
- ADC internal dengan fidelitas 10 bit sebanyak 8 channel.
- Portal komunikasi serial (USART) dengan kecepatan maksimal 2,5 Mbps.
- Enam pilihan mode *sleep* dapat digunakan untuk menghemat penggunaan daya listrik.

#### 2.4.2. Konfigurasi Pin ATmega8535

Konfigurasi pin pada ATmega8535 adalah sebagai berikut:

1. VCC, merupakan pin yang digunakan sebagai pin masukan catu daya 5V.
2. GND, merupakan pin ground.
3. Port A (PA0..PA7), merupakan pin I/O dua arah atau dapat juga digunakan sebagai pin masukan ADC.
4. Port B (PB0..PB7), merupakan pin I/O dua arah dan dapat digunakan sebagai pin fungsi khusus, *Timer/Counter*, komparator analog, dan SPI.
5. Port C (PC0..PC7), merupakan pin I/O dua arah dan dapat digunakan sebagai pin fungsi khusus TWI, komparator analog dan *Timer Oscillator*.

6. Port D (PD0..PD7), merupakan pin I/O dua arah dan dapat digunakan sebagai pin fungsi khusus komparator analog, interupsi eksternal, dan komunikasi serial.
7. RESET, merupakan pin yang digunakan untuk mereset mikrokontroller.
8. XTAL1 dan XTAL2, merupakan pin yang digunakan sebagai masukan untuk clock eksternal.
9. AVCC, merupakan pin yang digunakan sebagai masukan tegangan ADC.
10. AREF, merupakan pin yang digunakan sebagai masukan tegangan referensi ADC.
11. Masukan dan keluarannya adalah kondisi high (tegangan 5V) dan kondisi Low (tegangan 0V).

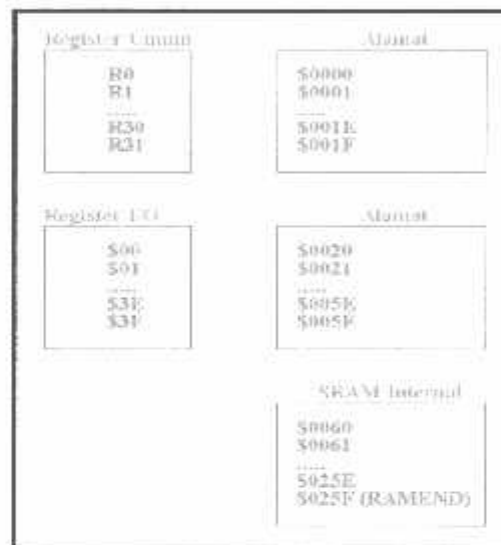


**Gambar 2.2.**

Konfigurasi pin ATmega8535<sup>[1]</sup>

### 2.4.3. Memori Data dan Memori Program ATmega8535

Ruang pengalamatan memori data dan memori program pada ATmega8535 di tempatkan secara terpisah. Memori data terbagi menjadi 3 bagian, yang terdiri dari 32 buah register umum, 64 buah register I/O, dan 512 *Byte* SRAM Internal. Register-register yang dapat digunakan untuk keperluan umum menempati *space* data pada alamat terbawah \$00 sampai \$1F.



Gambar 2.3.

Memori Data ATmega8535<sup>[1]</sup>

64 alamat berikutnya yaitu mulai \$20 hingga \$5F adalah register-register khusus untuk menangani I/O dan kontrol terhadap mikrokontroler. Register tersebut merupakan register yang khusus digunakan untuk mengatur fungsi terhadap berbagai *peripheral* mikrokontroler, seperti kontrol register, *timer/counter*, fungsi-fungsi I/O dan sebagainya. Alamat memori berikutnya pada lokasi \$60 sampai \$25F digunakan untuk SRAM 512 *Byte*.



Memori program yang terletak dalam *Flash PEROM* disusun dalam word atau 2 *Byte* karena setiap instruksi lebarnya 16 bit atau 32 bit. *Flash PEROM* yang dimiliki AVR ATmega8535 adalah 4 *Kbyte* X16 bit dengan alamat mulai dari \$000 sampai \$1FFF. AVR ATmega8535 dilengkapi dengan 12 bit *Program Counter* (PC) isi *Flash* dapat dialamati. Selain itu, AVR ATmega8535 juga memiliki memori data berupa EEPROM 8 bit sebanyak 512 *Byte*. Alamat EEPROM dimulai dari \$000 sampai \$1FF.

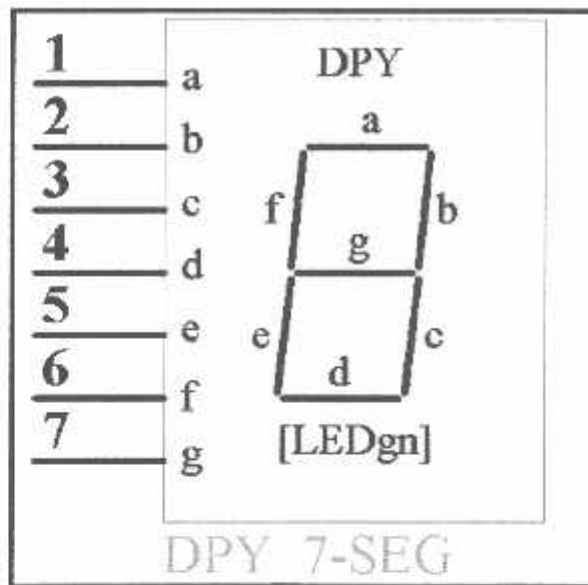


**Gambar 2.4.**

Memori Program ATmega8535<sup>[1]</sup>

## 2.5. Seven Segment

Peralatan *output* yang biasa digunakan untuk menampilkan bilangan *desimal* adalah peraga tujuh segmen (*Seven Segment Display*), yang ditunjukkan pada gambar 2.5. *Seven segment* tersebut dilabelkan dengan huruf a sampai g yang menyatakan digit desimal dari 0 sampai 9.



Gambar 2.5.

Peraga *Seven Segment* <sup>[6]</sup>

Pada dasarnya, *seven segment* merupakan kumpulan dari 7 LED yang disusun sedemikian rupa sehingga membentuk sebuah angka. *Seven segment* dapat di bedakan berdasarkan pin LED yang disatukan, yaitu:

1. *Seven Segment Common Anode*, Anoda dari semua LED disatukan sehingga *seven segment* akan menyala jika *commonnya* dihubungkan dengan positif (VCC) dan yang lainnya dihubungkan dengan nol (atau negatif)
2. *Seven Segment Common Cathode*, Katoda dari semua LED disatukan sehingga *seven segment* akan menyala jika *commonnya* dihubungkan dengan negatif (0 volt) dan pin yang lainnya dihubungkan dengan positif.

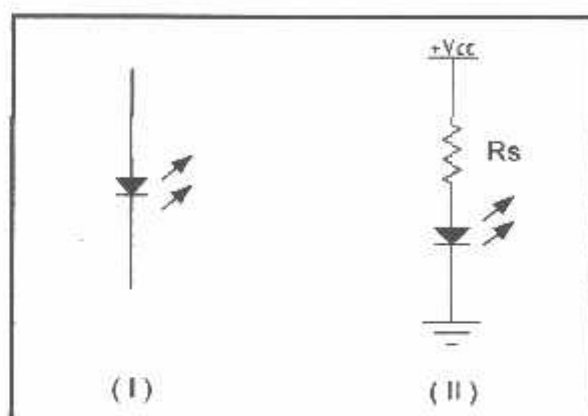
Display *seven segment* dapat dibuat dengan beberapa cara. Tiap *seven segment* tersebut dapat berupa filamen tipis yang berpijar. Jenis peraga ini disebut peraga pijar (*incandescent display*), dan sama dengan bola lampu biasa. Peraga jenis lain adalah tabung pelepas gas (*gas discharge*) yang beroperasi pada tegangan tinggi. Unit ini menghasilkan sinar jingga. Tabung yang berpendar (*fluorescent*) akan menghasilkan sinar hijau bila menyala dan beroperasi pada tegangan rendah. Dioda pemancar cahaya (LED) merupakan satu peraga *seven segment* yang dapat menampilkan beberapa warna kecuali warna merah. Oleh karena berupa dioda, LED sensitif terhadap polaritas. Katoda (K) harus dihubungkan ke negatif (GND) dari catu daya. Anoda (A) harus dihubungkan ke positif dari catu daya.

## 2.6. (LED) *Light Emitting Diode*

Dioda pemancar cahaya *light-emitting diode* (LED) merupakan sebuah dioda yang dapat memancarkan cahaya dengan memanfaatkan adanya penurunan elektron dari *level* konduksi ke *level valensi* yang menimbulkan pelepasan energi dalam bentuk panas atau cahaya. Sebuah elektron yang bergerak melintasi sembarang pertemuan PN akan bergerak ke daerah lubang (*hole*). Hal ini akan menyebabkan sebuah elektron konduksi yang berdekatan turun ke level valensinya, sambil meradiasikan energi. Dalam dioda dan transistor biasa, yang terbuat dari germanium, silikon, atau galium arsenida.

Arus maju berkisar antara 10 mA sampai dengan 20 mA untuk kecerahan maksimum. Sebuah resistor akan selalu dihubungkan seri dengan LED untuk

membatasi arus sesuai dengan yang dikehendaki. LED menggantikan lampu pilot yang dulu digunakan sebagai penunjuk visual dalam peralatan. Sebuah LED ditambah sebuah resistor dapat digunakan dalam sembarang tegangan dari 2 sampai 10 volt. LED menggunakan arus yang jauh lebih kecil daripada lampu pilot.



**Gambar 2.6.**

Lambang Skematis Dan Contoh Rangkaian Dari LED<sup>[6]</sup>

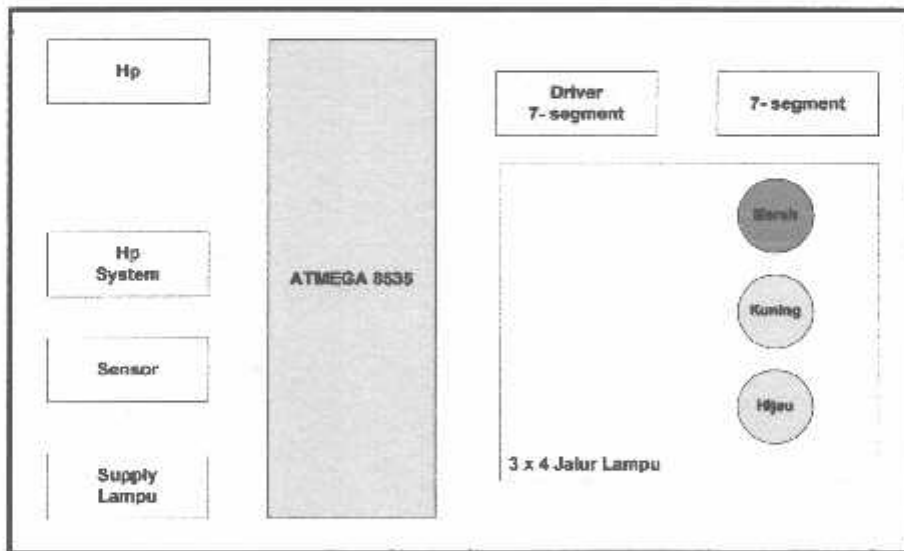
## BAB III

### PERANCANGAN ALAT

Pada bab ini akan dibahas tentang perancangan dan pembuatan sistem pengaturan lampu lalu lintas dilengkapi peralatan *cellular* sebagai pengendali beserta prinsip kerja keseluruhan sistem. Perancangan alat dalam sistem ini terdiri perancangan perangkat keras (*hardware*) dan perangkat lunak (*software*).

#### 3.1. Blok Diagram Sistem

Blok diagram sistem pengaturan lampu lalu lintas dilengkapi peralatan *cellular* sebagai pengendali yang direncanakan ditunjukkan pada gambar 3.1.



**Gambar 3.1.**  
Blok diagram sistem

### Fungsi blok diagram

- Mikrokontroler Atmega8535  
Berfungsi sebagai pengendali utama sistem alat lampu lalu lintas.
- Lampu lalu lintas  
Berfungsi sebagai media penyampai informasi bagi pengguna jalan.
- Handphone M35  
Berfungsi sebagai penerima data via sms.
- Seven segment  
Berfungsi sebagai display untuk menampilkan waktu (*Counter down*).
- Driver  
Berfungsi sebagai penerima data digital dari mikrokontroler, untuk mengendalikan *seven segment*.
- Sensor:  
Digunakan sebagai pendeteksi terjadinya lampu jika lampu dalam keadaan mati.

### Prinsip Kerja

Pada prinsipnya alat ini bekerja berdasarkan sistem pengontrolan dari mikrokontroler ATmega8535 sebagai pusat sistem lampu lalu lintas, yang proses pengolahan data dikendalikan oleh *handphone* dari dalam keadaan darurat. Dari sistem yang akan direncanakan penggunaan via sms sebagai pengontrol data eksternal sebagai sistem yang bersifat *mobile*.

Dalam pengolahan data untuk mengontrol lampu lalu lintas, media *handphone* bekerja sebagai *interface* data luar *via sms* dimana data yang diterima dianggap sebagai perintah untuk mengendalikan sistem lampu lalu lintas,. Sedangkan *seven segment* bekerja sebagai tampilan/display untuk durasi waktu.

### 3.2. Perencanaan Perangkat Keras (*Hardware*)

Perangkat keras yang digunakan dalam sistem ini terdiri dari beberapa bagian yaitu:

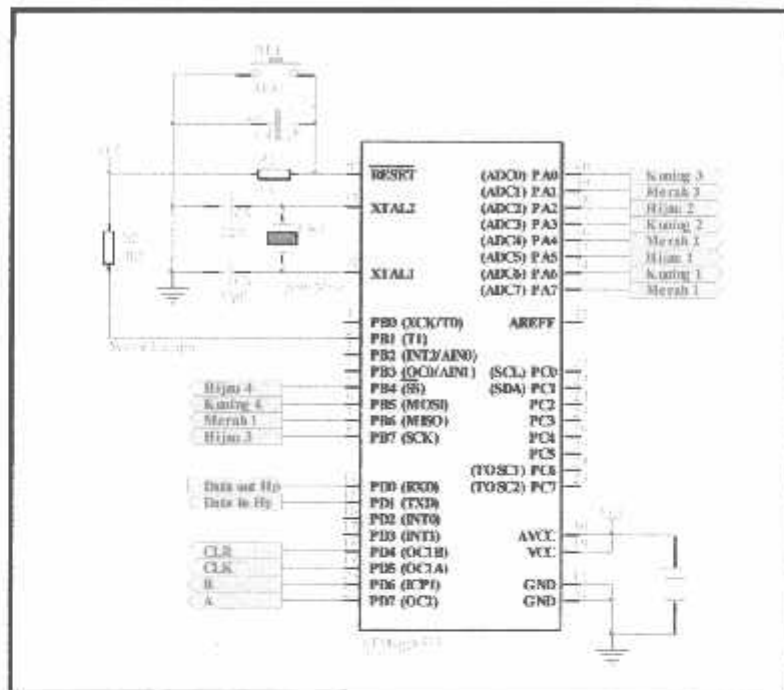
1. Sistem mikrokontroller AVR ATmega8535.
2. Handphone siemens M35 beserta kabel datanya.
3. Driver *seven segment*
4. Lampu LED sebagai simulasi lampu lalu lintas

#### 3.2.1 Sistem Mikrokontroller AVR ATmega8535

Mikrokontroller tipe ATmega8535 adalah produksi Atmel yang berbasis pada arsitektur AVR, yang memiliki arsitektur RISC 8 bit, dimana semua instruksi dikemas dalam kode 16 bit (*16 bits word*) dan sebagian besar instruksi dieksekusi dalam 1 siklus *clock* serta dilengkapi dengan EEPROM (*Electrically Erasable Programmable Read Only Memory*) sebesar 512 byte.

Dalam rangkaian sistem ini terdapat komponen yang paling penting yaitu rangkaian mikrokontroller AVR ATmega8535. Sistem mikrokontroller berfungsi sebagai pengendali utama dan pemroses data antara *handphone*, *lampu lalu lintas* dan *driver seven segment*. Perencanaan mikrokontroller dapat dilihat pada gambar

3.2



**Gambar 3.2.**

Sistem mikrokontroler AVR ATmega8535

Agar sebuah mikrokontroler dapat bekerja sebagai pengontrol, maka kaki-kaki/port mikrokontroler dihubungkan dalam rangkaian-rangkaian eksternal. Dalam perancangan ini, port yang digunakan adalah sebagai berikut:

- Port A0 dihubungkan dengan driver lampu merah 1
- Port A1 dihubungkan dengan driver lampu kuning 1
- Port A2 dihubungkan dengan driver lampu hijau 1
- Port A3 dihubungkan dengan driver lampu merah 2
- Port A4 dihubungkan dengan driver lampu kuning 2
- Port A5 dihubungkan dengan driver lampu hijau 2



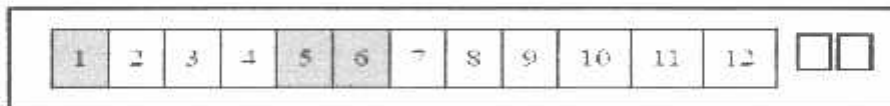
- Port A6 dihubungkan dengan driver lampu merah 3
- Port A7 dihubungkan dengan driver lampu kuning 3
- Port B7 dihubungkan dengan driver lampu hijau 3
- Port B6 dihubungkan dengan driver lampu merah 4
- Port B5 dihubungkan dengan driver lampu kuning 4
- Port B4 dihubungkan dengan driver lampu hijau 4
- Port B1 dihubungkan dengan pull up resistor 4K7 sebagai *sensor supply*. Jalur tersebut digunakan sebagai inputan yaitu sensor yang memantau *power supply*. Apabila pada jalur port ini tidak berlogika 1 maka sistem akan mengirimkan sms bahwa terjadi pemutusan listrik PLN ke *handphone* pengontrol.
- Port D0 (RXD) digunakan sebagai serial *input* yang dihubungkan dengan pin *transmitter* ponsel.
- Port D1 (TXD) digunakan sebagai serial *output* yang dihubungkan dengan pin *reciever* ponsel.
- Port D4 dihubungkan dengan port CLR IC 74LS164
- Port D5 dihubungkan dengan port CLK IC 74LS164
- Port D6 dihubungkan dengan port B IC 74LS164
- Port D7 dihubungkan dengan port A IC 74LS164
- Reset digunakan untuk mereset program kontrol mikrokontroler.
- XTAL1 dan XTAL2 digunakan sebagai input dari rangkaian osilator kristal. Rangkaian osilator kristal terdiri dari kristal osilator 11,0592 MHz, kapasitor C1 dan C2 yang masing-masing bernilai 22 pF, akan

membangkitkan pulsa clock yang menjadi penggerak bagi seluruh operasi internal mikrokontoller.

- VCC dihubungkan dengan tegangan sebesar +5V yang berasal dari *power supply* dan baterai.
- GND dihubungkan ke *ground* catu daya.

### 3.2.2. Antarmuka Komunikasi Serial

Koneksi dengan *handphone* siemens M35 pada dasarnya digunakan untuk dapat berkomunikasi dengan *handphone*, sehingga diperlukan sebuah kabel data sebagai konektor untuk menghubungkan *handphone* dengan mikrokontroller. Konektor *handphone* siemens M35 dapat dilihat pada gambar 3.3. Tidak semua pin *out* terhubung ke mikrokontroller, tetapi hanya pin nomor 1 (*ground*), 5 (*Tx/data out*) dan 6 (*Rx/data in*).



**Gambar 3.3.**

Konektor *Handphone* Siemens M35<sup>[7]</sup>

Untuk penggunaan dari masing-masing pin dari konektor pada gambar dapat dilihat pada tabel berikut.

**Tabel 3.1.**

Pin out Konektor *Handphone* Siemens M35<sup>[7]</sup>

PIN	NAMA	FUNGSI	IN/OUT
1	GND	Ground	
2	SELF SERVICE	Recognition control battery charger	In/Out
3	LOAD	Charging Voltage	In
4	BATTERY	Battery	Out
5	DATA OUT	Data sent	Out
6	DATA IN	Data received	In
7	Z_CLK	Recognition control accessories	
8	Z_DATA	Recognition control accessories	
9	MICG	Ground for microphone	In
10	MIC	Microphone input	
11	AUD	Loudspeaker	Out
12	AUDG	Ground for eksternal speaker	

Pada perancangan sistem ini rangkaian konektor telepon seluler dengan sistem pengontrol utama (rangkai mikrokontroller) ditunjukkan dalam gambar 3.4 pada perancangan ini, pin 5 yang berfungsi sebagai data out dihubungkan dengan PD0 (RXD), pin 6 yang berfungsi sebagai data *recieved/data in* dihubungkan pada PD1 (TXD) mikrokontroller, sedangkan pin 1 dihubungkan ke ground.

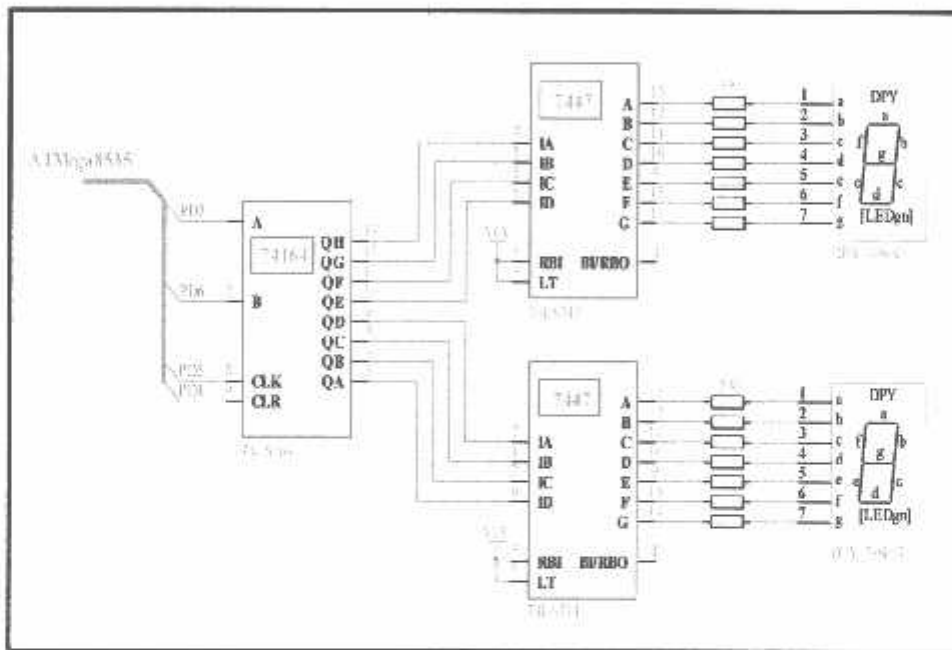


**Gambar 3.4.**

Antarmuka komunikasi serial Mikrokontroller dengan HP

### 3.2.4. Rangkaian *Driver Seven Segment*

Sistem pengaturan lampu lalu lintas dilengkapi peralatan *cellular* sebagai pengendali ini menggunakan 2 buah *seven segment* sebagai *display* durasi waktu nyala lampu lalu lintas. Rangkaian *driver seven segment* dibentuk dengan menggunakan 1 buah IC register geser 74LS164 dan 2 buah IC decoder BCD to *seven segment* 74LS47.



Gambar 3.6.

Rangkaian Driver Seven Segment

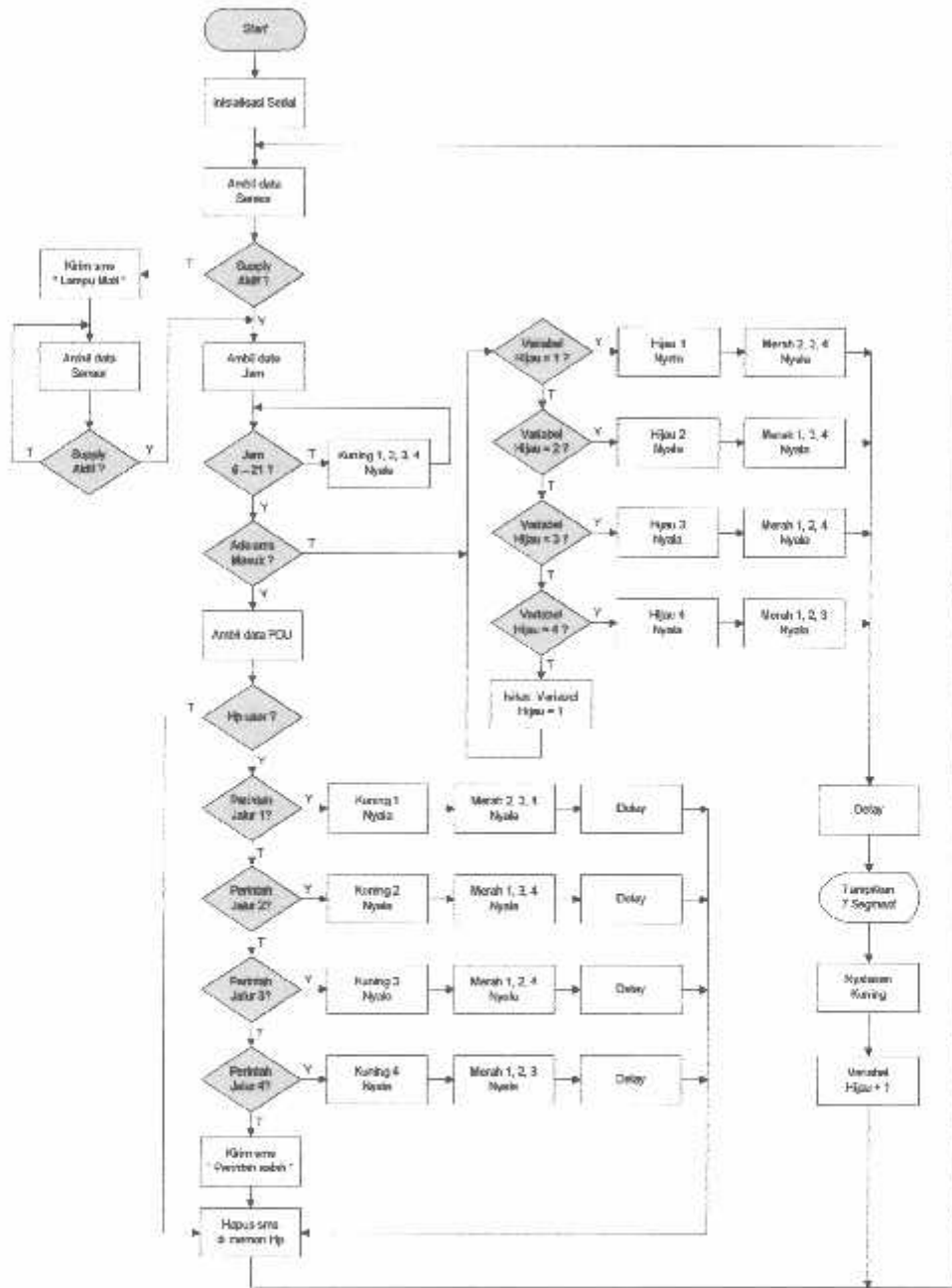
### 3.3. Perencanaan Perangkat Lunak (*software*)

Perangkat lunak dirancang dengan menggunakan bahasa *assembly* yang ditulis dengan text editor AVR Studio versi 4. Dengan menggunakan *software* AVR Studio tersebut pula program sumber *assembly* di compile menjadi program objek maupun hexa, kemudian di load ke dalam mikrokontroler menggunakan *software pony prog2000*.

Setiap SMS yang datang akan disimpan di SIM *card* pada lokasi memori mailbox dengan mengisi nomor mailbox yang paling rendah terlebih dahulu. Apabila seluruh lokasi mailbox sudah terisi, maka sms *center* tidak akan mengirimkan sms ke ponsel sampai tersedianya tempat kosong pada mailbox. Pada rutin pembacaan sms ini lokasi mailbox nomor 1 disiapkan untuk menampung sms yang datang, dimana sms tersebut dianggap sebagai perintah untuk pengendalian lampu lalu lintas. Setelah pembacaan sms oleh mikrokontroller selesai, lokasi tersebut dihapus untuk menampung sms yang lain.

Ponsel menerima sms tidak sekedar berupa pesan, tetapi data pesan sms tersebut merupakan bagian dari serangkaian data-data yang terbentuk saat pengiriman melalui sms *center*. Apabila rangkaian data seluruhnya ditampung oleh mikrokontroller, maka dibutuhkan jumlah lokasi memori RAM yang besar. Hal itu tidak mungkin dilakukan karena terbatasnya besar RAM yang dimiliki oleh mikrokontroler ATmega8535, yaitu sebesar 512 byte. Dengan demikian, maka data yang diterima akan langsung digunakan sebagai perintah untuk seting timer dan setelah itu langsung dihapus sehingga data tidak menumpuk.

Alur proses program mikrokontroler secara keseluruhan ditunjukkan pada diagram alir sebagai berikut:



Gambar 3.8.  
Flowchart Sistem

## **BAB IV**

### **PENGUJIAN ALAT**

Pada bab ini akan dibahas mengenai pengujian programmable sistem pengaturan lampu lalu lintas dilengkapi peralatan *cellular* sebagai pengendali jarak jauh *via sms*. Pengujian ini bertujuan untuk menguji apakah alat yang dibuat telah dapat bekerja sesuai perancangan. Pengujian dilakukan pada perangkat keras maupun perangkat lunak. Metode pengujian alat adalah sebagai berikut:

1. Pengujian Rangkaian ATmega8535
2. Pengujian Antarmuka Serial Handphone
3. Pengujian Rangkaian Lampu
4. Pengujian Rangkaian Driver Seven Segment
5. Pengujian rangkaian secara keseluruhan

#### **4.1. Pengujian Rangkaian ATmega8535**

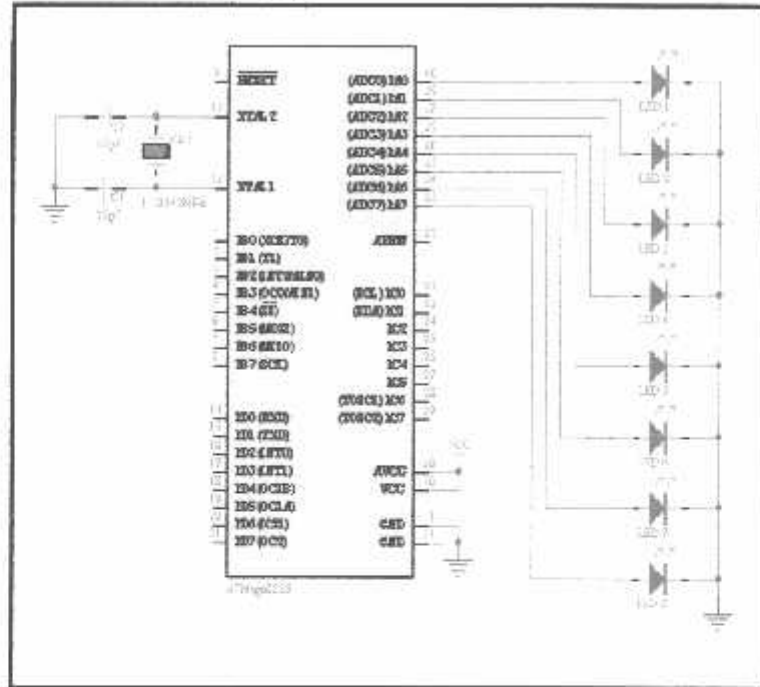
##### **A. Tujuan**

Untuk mengetahui apakah rangkaian mikrokontroler ATmega8535 dapat berfungsi dengan baik atau tidak.

##### **B. Peralatan yang digunakan**

1. 8 buah LED
2. Mikrokontroller AT mega8535.
3. Power supply

### C. Rangkaian pengujian



Gambar 4.1.

Rangkaian Pengujian ATmega8535

#### Langkah-langkah pengujian

1. Merangkai modul rangkaian seperti pada Gambar 4.1
2. Memberikan catu daya pada rangkaian dan mengaktifkannya.
3. Memasukkan program pengujian rangkaian mikrokontroler.
4. Mengamati lampu LED.



**D. Data hasil pengujian**

**Tabel 4.1.**  
Hasil Pengujian Rangkaian ATmega8535

No	PORT A (Biner)	LED 8	LED 7	LED 6	LED 5	LED 4	LED 3	LED 2	LED 1
1	0000 0000	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
2	0000 0001	OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON
3	0000 0010	OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF
4	0000 0100	OFF	OFF	OFF	OFF	OFF	ON	OFF	OFF
5	0000 1000	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF
6	0001 0000	OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF
7	0010 0000	OFF	OFF	ON	OFF	OFF	OFF	OFF	OFF
8	0100 0000	OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF
9	1000 0000	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF
10	1111 1111	ON	ON	ON	ON	ON	ON	ON	ON

**E. Analisis hasil**

LED 1 - 8 menyala ketika mikrokontroller diberi masukan high (1)  
dan mati ketika diberi masukan low (0).

## **F. Kesimpulan**

Mikrokontroler sebagai pengendali utama keseluruhan rangkaian berfungsi dengan baik.

### **4.2. Pengujian Antarmuka Komunikasi Serial**

*Interfacing* komunikasi serial antara mikrokontroler dengan *handphone* dihubungkan dengan menggunakan kabel data *handphone*. Kabel data yang ada di pasaran pada umumnya telah dilengkapi dengan rangkaian RS232 yang memang oleh *vendor* pembuat kabel data tersebut, bertujuan agar *handphone* dapat digunakan untuk *transfer* data atau berkomunikasi dengan PC (*personal computer*) pada level tegangan RS232. Dengan demikian, mengingat mikrokontroler bekerja dalam level tegangan TTL (*transistor-transistor logic*), maka disini ada dua solusi yang bisa digunakan agar mikrokontroler dapat berkomunikasi dengan *handphone*, yaitu dengan menambahkan rangkaian RS232 pada rangkaian mikrokontroler agar dapat berkomunikasi dalam level tegangan RS232, atau bisa juga dengan memotong ujung kabel data tersebut agar dapat berkomunikasi pada level tegangan TTL. Disini penulis menggunakan level tegangan TTL sehingga kabel data tersebut di potong ujungnya.

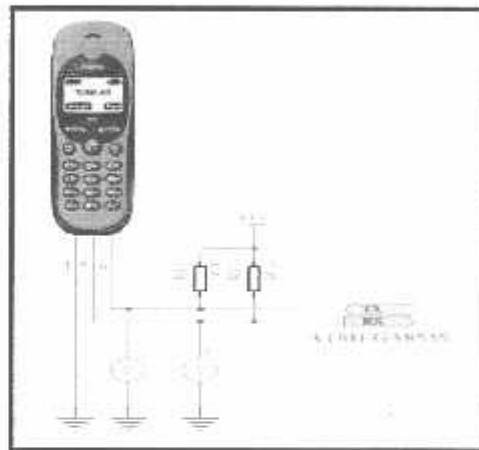
#### **A. Tujuan**

Untuk mengetahui apakah antarmuka serial yang dibangun, telah bekerja dalam level tegangan TTL.

**B. Peralatan yang digunakan**

1. *Handphone* siemens M35
2. Kabel data *handphone* siemens M35
3. Mikrokontroller AT mega8535
4. 2 buah multimeter digital tipe DT-830B
5. Power supply

**C. Rangkaian pengujian**



**Gambar 4.2.**

Rangkaian Pengujian Antarmuka Serial

Langkah-langkah pengujian:

1. Merangkai modul rangkaian seperti pada gambar 4.2
2. Menghubungkan rangkaian pada catu daya sebagai sumber tegangan dan mengaktifkannya.
3. Melakukan inisialisasi serial.

- Mengukur nilai tegangan keluaran pada pin Tx dan Rx pada saat kondisi *high* dan *low*.

**D. Data hasil pengujian**

**Tabel 4.2.**  
Hasil Pengujian Antarmuka Serial

<b>Tx (Volt)</b>	<b>Rx (Volt)</b>
Kondisi <i>high</i>	
4,9	4,9
Kondisi <i>low</i>	
0	0

**E. Analisis hasil**

Pada jalur Tx dan Rx diberikan resistor pull-up sehingga jalur data bersifat *open drain*, maka dalam kondisi tidak ada data, jalur tersebut dalam keadaan high. Dari hasil pengujian didapatkan logika high pada Rx dan Tx adalah 4,9 Volt dan logika low adalah 0 Volt.

**F. Kesimpulan**

Keluaran tegangan dari *handphone* sesuai dengan perencanaan, bekerja dalam level tegangan TTL.

**4.3. Pengujian Rangkaian Lampu**

Rangkaian lampu lalu lintas di simulasikan dengan menggunakan lampu LED warna merah, kuning, dan hijau. Rangkaian lampu LED di konfigurasi

dengan menggunakan *common anode* sehingga untuk menyalakan di butuhkan logika low dan logika high untuk mematikan.

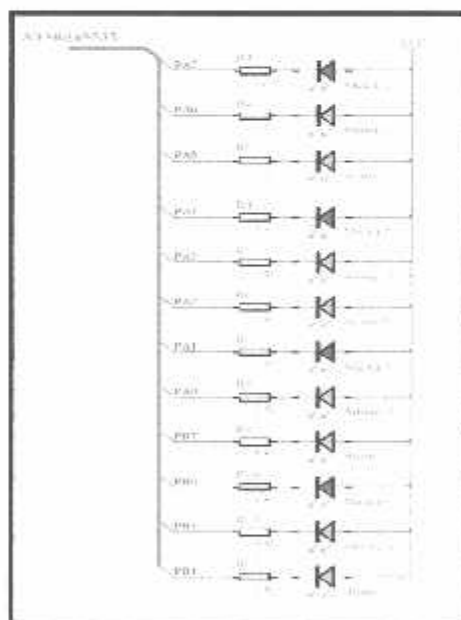
#### A. Tujuan

Untuk mengetahui apakah rangkaian lampu telah dapat bekerja sesuai dengan yang di harapkan yaitu di berikan logika low untuk mengaktifkan dan diberikan logika high untuk mematikan.

#### B. Peralatan yang digunakan

1. Mikrokontroller AT mega8535
2. Rangkaian LED
3. Power supply 5 Volt

#### C. Rangkaian pengujian



**Gambar 4.3.**

Rangkaian Pengujian Lampu

Langkah-langkah pengujian :

1. Merangkai modul rangkaian seperti pada Gambar 4.3
2. Memasukkan program tes lampu
3. Menghubungkan catu daya dengan rangkaian sebagai sumber tegangan dan mengaktifkannya.
4. Mencatat hasil pengujian ke dalam tabel

**D. Data hasil pengujian**

**Tabel 4.3.**

Hasil Pengujian Rangkaian lampu

No	PORT A (Biner)	LED K3	LED M3	LED H2	LED K2	LED M2	LED H1	LED K1	LED M1
1	1111 1110	OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON
2	1111 1101	OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF
3	1111 1011	OFF	OFF	OFF	OFF	OFF	ON	ON	OFF
4	1111 0111	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF
5	1110 1111	OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF
6	1101 1111	OFF	OFF	ON	ON	OFF	OFF	OFF	OFF
7	1011 1111	OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF
8	0111 1111	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF

No	PORT B (Biner)	-	-	-	-	LED H4	LED K4	LED M4	LED H3
9	1111 1110	-	-	-	-	OFF	OFF	OFF	ON
10	1111 1101	-	-	-	-	OFF	OFF	ON	OFF
11	1111 1011	-	-	-	-	OFF	ON	OFF	OFF
12	1111 0111	-	-	-	-	ON	OFF	OFF	OFF

#### E. Analisis hasil

Pengujian rangkaian lampu secara *hardware* dilakukan dengan cara memberikan suatu kondisi *'low'* (0 Volt) dan *'high'* (5 Volt) pada port yang terhubung dengan lampu bersangkutan. Dengan melihat tabel hasil pengujian dapat diketahui bahwa rangkaian lampu telah dapat bekerja sesuai yang diharapkan yaitu On pada saat diberi logika low, dan Off pada saat diberi logika high.

#### F. Kesimpulan

Rangkaian lampu menggunakan konfigurasi *common anode* sehingga dibutuhkan logika low untuk menyalakan dan logika high untuk mematikan.

#### 4.4. Pengujian Rangkaian *Driver Seven Segment*

*Seven segment* digunakan sebagai *display* durasi waktu nyala lampu lalu lintas. *Seven segment* yang digunakan mempunyai spesifikasi *common anode*,

sehingga digunakan IC 74LS47 sebagai decoder BCD to *seven segment*. IC 74LS164 digunakan sebagai register geser SIPO (*serial input paralel output*) yang mengontrol IC 74LS47. Dengan digunakannya register geser dan decoder BCD to *seven segment* ini maka hanya dibutuhkan 4 jalur data untuk dapat mengontrol display *seven segment* yaitu CLR, CLK, A dan B.

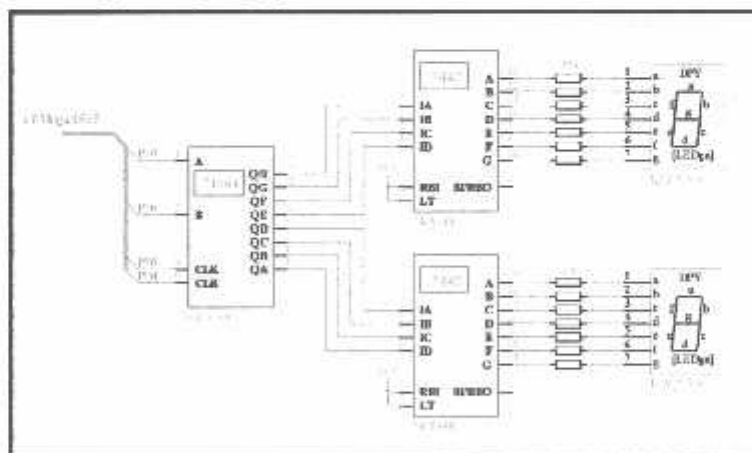
### A. Tujuan

Untuk mengetahui apakah rangkaian driver *seven segment* telah dapat bekerja sesuai dengan yang diharapkan yaitu dapat digunakan untuk menampilkan angka 0-9 pada *display seven segment*.

### B. Peralatan yang digunakan

1. Mikrokontroller AT mega8535
2. Rangkaian Driver Seven Segment
3. Power supply 5 Volt

### C. Rangkaian pengujian



Gambar 4.4.

Rangkaian Pengujian Driver Seven Segment



Langkah-langkah pengujian :

1. Merangkai modul rangkaian seperti pada Gambar 4.4
2. Memasukkan program tes driver seven segment
3. Menghubungkan catu daya dengan rangkaian sebagai sumber tegangan dan mengaktifkannya.
4. Mencatat hasil pengujian ke dalam tabel

**D. Data hasil pengujian**

**Tabel 4.4.**

Hasil Pengujian Rangkaian Driver Seven Segment

No	74LS164 Hexa	74LS47_1							Display_1	74LS47_2							Display_2
		a	b	c	d	e	f	g		a	b	c	d	e	f	g	
1	00	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
2	01	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1	1
3	02	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	2
4	03	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	3
5	04	0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	4
6	05	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	5
7	06	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	6
8	07	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	7
9	08	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	8
10	09	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	9
11	10	1	0	0	1	1	1	1	1	0	0	0	0	0	0	1	0
12	20	0	0	1	0	0	1	0	2	0	0	0	0	0	0	1	0

13	30	0	0	0	0	1	1	0	3	0	0	0	0	0	0	1	0
14	40	1	0	0	1	1	0	0	4	0	0	0	0	0	0	1	0
15	50	0	1	0	0	1	0	0	5	0	0	0	0	0	0	1	0
16	60	1	1	0	0	0	0	0	6	0	0	0	0	0	0	1	0
17	70	0	0	0	1	1	1	1	7	0	0	0	0	0	0	1	0
18	80	0	0	0	0	0	0	0	8	0	0	0	0	0	0	1	0
19	90	0	0	0	1	1	1	1	9	0	0	0	0	0	0	1	0

**E. Analisis hasil**

Pengujian rangkaian *driver seven segment* secara *hardware* dilakukan dengan cara mengirimkan data ke IC 74LS164 sehingga rangkaian decoder BCD to *sevent segment* 74LS47 mendapatkan inputan data yang akan di tampilkan pada *display seven segment*. Dengan melihat tabel hasil pengujian dapat diketahui bahwa rangkaian *driver seven segment* dapat bekerja sesuai yang diharapkan yaitu dapat menampilkan nilai angka pada *seven segment*.

**F. Kesimpulan**

Dengan menggunakan rangkaian *driver seven segment* ini maka dapat menghemat port pada mikrokontroller. Data dikirim secara serial berurutan satu persatu hingga 8 bit dimulai dari nilai bit yang paling kecil.

**4.5. Pengujian Rangkaian Keseluruhan**

**A. Tujuan**

Untuk mengetahui apakah keseluruhan rangkaian telah berfungsi dengan baik.

**B. Peralatan yang digunakan**

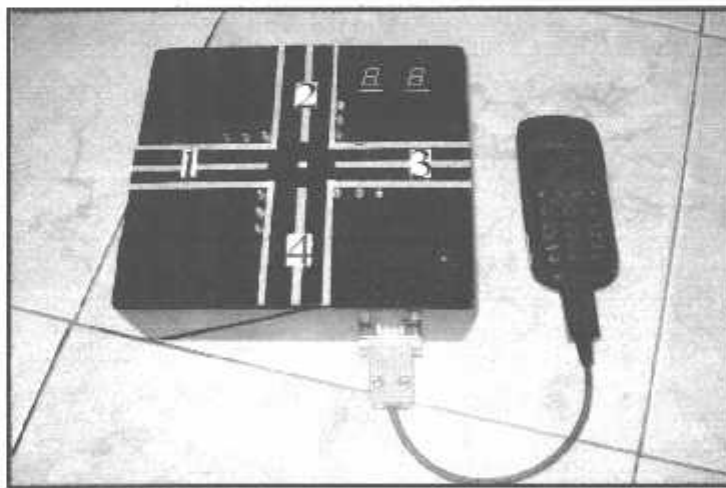
1. Handphone user terminal
2. Handphone base terminal (sistem)
3. Kabel data
4. Rangkaian keseluruhan system

**C. Blok diagram**



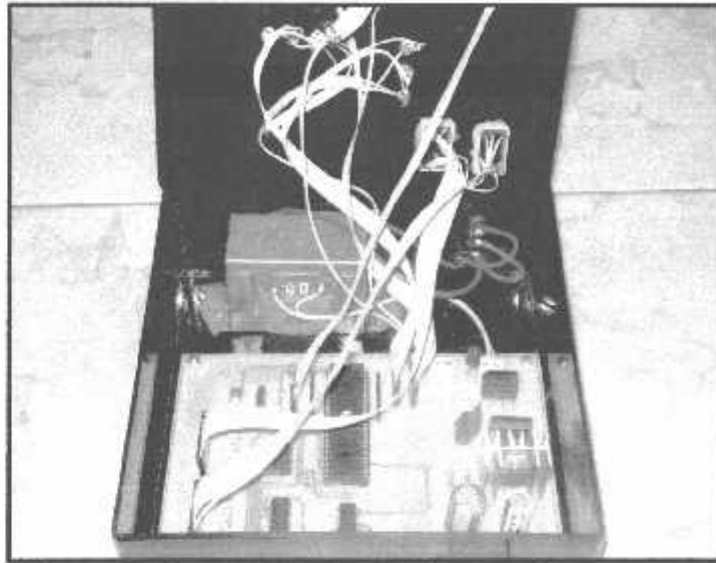
**Gambar 4.5.**

Diagram Blok Pengujian Secara Keseluruhan



**Gambar 4.6.**

Foto alat keseluruhan



**Gambar 4.7.**

Foto alat tampak dari dalam

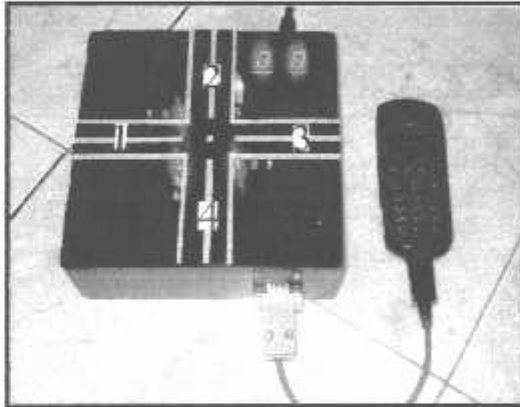
Langkah-langkah pengujian :

1. Menghubungkan seluruh blok rangkaian menjadi satu sistem seperti pada Gambar 4.5
2. Memberikan catu daya sebagai sumber tegangan pada sistem.
3. Menghubungkan sistem Mikrokontroller dengan Handphone
4. Memastikan semua bagian telah terhubung dengan benar.
5. Menggunakan *handphone* tipe sembarang untuk mengirimkan perintah sms, dengan format pengiriman perintah 1ON, 2ON, 3ON, 4ON, OFF untuk perintah normal.
6. Mengamati apakah sistem dapat bekerja sesuai dengan perintah sms.
7. Mencatat ke dalam tabel hasil pengujian.



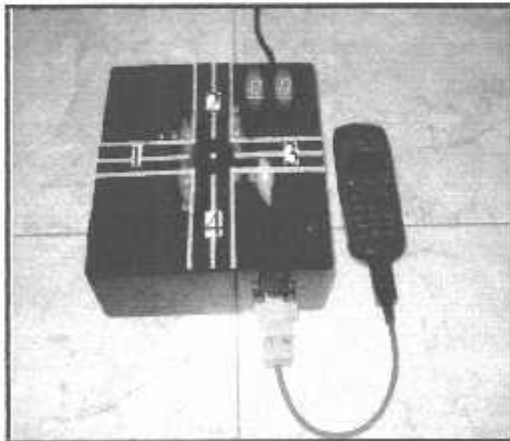
**Gambar 4.8.**

Foto tampilan jalur 1 ON Seven segment tidak bekerja



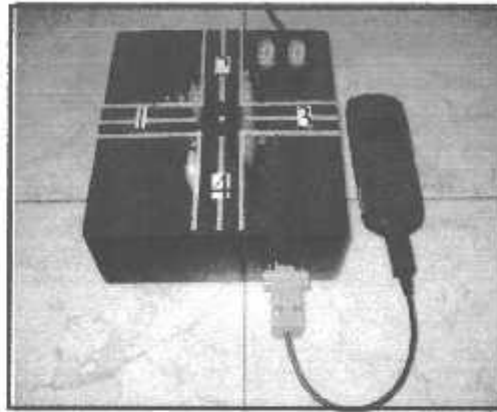
**Gambar 4.9.**

Foto tampilan jalur 2 ON Seven segment tidak bekerja



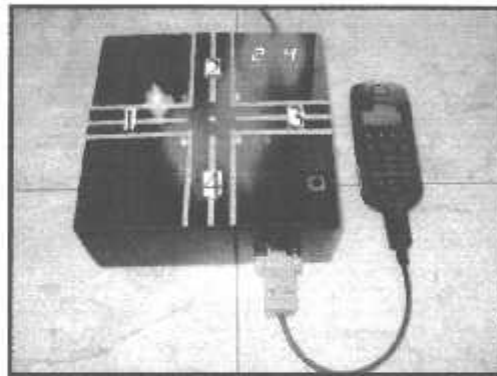
**Gambar 4.10.**

Foto tampilan jalur 3 ON Seven segment tidak bekerja



**Gambar 4.11.**

Foto tampilan jalur 4 ON Seven segment tidak bekerja



**Gambar 4.12.**

Foto tampilan jalur OFF Dilajur 1 Seven segment bekerja

**D. Data hasil pengujian**

**Tabel 4.5**

Hasil Pengujian Secara Keseluruhan

No	Perintah SMS	Jalur 1	Jalur 2	Jalur 3	Jalur 4
1	1ON	Hijau	Merah	Merah	Merah
2	2ON	Merah	Hijau	Merah	Merah
3	3ON	Merah	Merah	Hijau	Merah
4	4ON	Merah	Merah	Merah	Hijau
5	OFF	Kembali Normal			

**E. Analisis hasil**

Sistem merespon ketika ada sms perintah masuk. Jalur yang di kontrol nyala hijau sedangkan yang lain nyala merah.

**F. Kesimpulan**

Keseluruhan rangkaian berfungsi dengan baik. Perlu dikirimkan perintah "OFF" agar jalur kembali normal melanjutkan looping nyala lampu lalu lintas. Setiap perintah yang dikirim untuk mengaktifkan jalur, seven segment akan tampil angka 00, sedangkan untuk perintah OFF lampu akan normal begitu juga dengan tampilan seven segment.

## BAB V

### PENUTUP

#### 5.1. Kesimpulan

Kesimpulan yang didapat adalah sebagai berikut:

1. Untuk memastikan sistem komunikasi serial antara telepon *celuler* dengan mikrokontroller AVR ATmega8535 dapat berfungsi, pin-pin transmitter dan receiver harus bertegangan logika 1.
2. Data yang dikirimkan telepon *celuler* sebagai tanggapan atas instruksi yang diterima, dikirim menggunakan format PDU (*Protocol Data Unit*) dalam kode ASCII. Untuk dapat diambil informasinya diperlukan pengubahan dari kode Hex-Ascii ke kode Heksa, diteruskan dari kode Heksa ke kode Ascii.
3. Setiap melakukan pengiriman perintah ON hanya lampu hijau yang menyala, sesuai dengan jalur yang dipilih. Untuk mengembalikan dalam kondisi normal dengan mengirim perintah OFF.

#### 5.2. Saran

Beberapa kekurangan alat ini adalah harus membayar setiap kali mengirim sms, juga kartu memiliki masa aktif dan kadang sms terlambat atau malah tidak sampai, masalah ini dapat diatasi dengan menggunakan kartu yang sejenis, sehingga sms dapat segera sampai.



## DAFTAR PUSTAKA

1. Atmel. 2006. ATmega 8535 AVR. USA:
2. Atmel. 2006. AVR and Third Party Tools. USA:
3. Atmel. 2006. AVR Assembler User Guide. USA:
4. Atmel. 2006. AVR Hardware Design Considerations. USA:
5. Atmel. 2006. AVR Instruction Set. USA:
6. [www.electronicclub.com](http://www.electronicclub.com). 2006
7. Siemens AG. 2001. AT Command Set for Siemens Mobile Phones and Modems. Munich:
8. Wardana, Lingga., 2004, Belajar Sendiri mikrokontroler AVR Seri ATmega8535 Simulasi, Hardware, dan Aplikasi.



INSTITUT TEKNOLOGI NASIONAL MALANG  
FAKULTAS TEKNOLOGI INDUSTRI  
JURUSAN TEKNIK ELEKTRO S-1  
KONSENTRASI TEKNIK ELEKTRONIKA

---





### FORMULIR PERBAIKAN SKRIPSI

Nama : BAMBANG HUSONO  
NIM : 02.17.086  
Masa Bimbingan : 20 Januari 2009 s/d 20 Juni 2009  
Judul : "PERANCANGAN DAN PEMBUATAN SISTEM  
PENGATURAN LAMPU LALU LINTAS  
DILENGKAPI PERALATAN *CELLULAR* SEBAGAI  
PENGENDALI"

No	MATERI PERBAIKAN	PARAF
1.	Penulisan Sensor Lampu	
2.	Penulisan Kesimpulan	
3.	Sesuaikan Alat Yang Dibuat Dengan Tugas Skripsinya (Rangkaian-Rangkaiannya)	
4.	Tambahkan Foto-Foto Alat Pada Pengujian	
5.	Sesuaikan Kesimpulan Dengan Tujuan dan Batasan Masalah	

Disetujui:

Penguji I

( Ir. Eko Nurcahyo )  
NIP.Y.1028700172

Penguji II

( M.Ibrahim Ashari, ST, MT )  
NIP.P.1030100358

Mengetahui:

Dosen Pembimbing

( Ir.F.Yudi Limpraptono, MT )  
NIP.Y.1039500274



### Formulir Perbaikan Ujian Skripsi

Dalam pelaksanaan Ujian Skripsi Janjang Strata 1 Jurusan Teknik Elektro Konsentrasi T. Energi Listrik / T. Elektronika / T. Infokom, maka perlu adanya perbaikan skripsi untuk mahasiswa :

NAMA : BKTUBANG H  
NIM : 0217086  
Perbaikan meliputi :

- Sesuaikan alat yg di buat dg tugas & rumpahnya  
(rangkaian nya).

- Tambahkan foto alat pd pengujian. ✓

- Sesuaikan kesimpulan dg tujuan & babasan masalah

Malang, 23-03-2009

()  
Eko



INSTITUT TEKNOLOGI NASIONAL MALANG  
FAKULTAS TEKNOLOGI INDUSTRI  
JURUSAN TEKNIK ELEKTRO

### Formulir Perbaikan Ujian Skripsi

Dalam pelaksanaan Ujian Skripsi Janjang Strata 1 Jurusan Teknik Elektro Konsentrasi T. Energi Listrik / T. Elektronika / T. Infokom, maka perlu adanya perbaikan skripsi untuk mahasiswa :

NAMA


NIM

Perbaikan meliputi

Bambang Husaro  
0217086

Penulisan sensor energi  
Penulisan kesimpulan



Malang, 23 maret 2009.

  
M. Nurdin Ashari, ST, MT



## LEMBAR PENGAJUAN JUDUL SKRIPSI JURUSAN TEKNIK ELEKTRO S-1

Konsentrasi : Teknik Energi Listrik/Teknik Elektronika/Teknik Komputer & Informatika\*)

1.	Nama Mahasiswa: <b>BAMBANG HUSONO</b>	Nim: <b>0217086</b>		
2.	Waktu Pengajuan	Tanggal: <b>10</b>	Bulan: <b>OCTOBER</b>	Tahun: <b>2008</b>
3.	Spesifikasi Judul (berilah tanda silang)**)			
	a. Sistem Tenaga Elektrik	e. Elektronika & Komponen		
	b. Energi & Konversi Energi	<input checked="" type="checkbox"/> f. Elektronika Digital & Komputer		
	c. Tegangan Tinggi & Pengukuran	g. Elektronika Komunikasi		
	d. Sistem Kendali Industri	h. lainnya .....		
4.	Konsultasikan judul sesuai materi bidang ilmu kepada Dosen*)	Ketua Jurusan		
	<b>P. F. Eudi L, MT</b>	 Ir. F. Yudi Limpraptono MT NIP. P. 1039500274		
5.	Judul yang diajukan mahasiswa:	<b>PERANGANGAN DAN PEMBUATAN SISTEM PENGATURAN LAMPU LALULINTAS PELENGKAPI PERALATAN SELULAR SEBAGAI PENGENDALI</b>		
6.	Perubahan judul yang disetujui Dosen sesuai materi bidang ilmu	..... ..... .....		
7.	Catatan: ..... ..... .....			
	Persetujuan Judul skripsi yang dikonsultasikan kepada Dosen materi bidang ilmu	Disetujui <b>10 OKTOBER</b> 2008 Dosen  Ir. F. Yudi Limpraptono MT		

Perhatian:

1. Formulir pengajuan ini harap dikembalikan kepada jurusan paling lambat satu minggu setelah disetujui kelompok dosen keahlian dengan dilampirkan proposal skripsi beserta persyaratan skripsi sesuai form S-1
2. Keterangan: \*) Coret yang tidak perlu  
\*\*) dilingkari a, b, c, ..... atau g sesuai bidang keahlian

Lampiran : 1 (satu) berkas  
**Pembimbing Skripsi**

Cepada : Yth. Bapak Ir. F. Yudi Limpraptono, MT  
Dosen Institut Teknologi Nasional  
MALANG

Yang bertanda tangan di bawah ini :

Nama : Bambang Husono  
Nim : 02.17.086  
Jurusan : Teknik Elektro S-1  
Konsentrasi : Teknik Elektronika

Dengan ini mengajukan permohonan, kiranya Bapak bersedia menjadi Dosen Pembimbing Utama, untuk penyusunan Skripsi dengan judul (proposal terlampir) :

**Perancangan Dan Pembuatan Sistem Pengaturan Lampu Lalulintas Dilengkapi Peralatan Cellular Sebagai Pengendali**


Adapun tugas tersebut sebagai salah satu syarat untuk menempuh Tugas Akhir Sarjana Teknik.  
Demikian permohonan kami dan atas kesediaan Bapak kami ucapkan terima kasih.

Malang, Desember 2008

Mengetahui  
Ketua Jurusan Teknik Elektro S-1

  
Ir. F. Yudi Limpraptono, MT  
Nip. Y. 1039500274

Hormat Kami,

  
Bambang Husono  
NIM. 02.17.086

Form S-3a

**PERNYATAAN KESEDIAAN DALAM PEMBIMBINGAN SKRIPSI**

Sesuai permohonan dari Mahasiswa :

Nama : Bambang Husono

Nim : 02.17.086

Semester : 13

Jurusan : Teknik Elektro S-1

Konsentrasi : Teknik Elektronika

Dengan ini menyatakan bersedia / ~~tidak bersedia~~ \*) Membimbing skripsi dari Mahasiswa tersebut, dengan judul :

**Perancangan Dan Pembuatan Suster : Pengaturan Lampu Lalulintas Dilengkapi Peralatan Cellular Sebagai Pengendali**

Demikian Surat Pernyataan ini kami buat agar dapat dipergunakan seperlunya.

Malang, Desember 2008

**Kami Yang Membuat Pernyataan,**



Ir. F. Yudi Limpraptono, MT  
NIP. Y. 1039500274

Catatan :

Setelah disetujui agar Formulir ini  
disebutkan mahasiswa/i yang bersangkutan  
kepada Jurusan untuk diproses lebih lanjut  
)Coret yang tidak perlu

Form S-3b





FORMULIR BIMBINGAN SKRIPSI

Nama : Bambang Husono  
Nim : 02.17.086  
Masa Bimbingan : 20-Januari-2009 s/d 20-Juni-2009  
Judul Skripsi : Perancangan Dan Pembuatan Sistem Pengaturan Lampu  
Lalulintas Dilengkapi Peralatan Cellular Sebagai Pengendali

No	Tanggal	Uraian	Paraf Pembimbing
1		Bab I	
2		Bab II	
3		Bab III	
4		Bab III	
5		Bab IV	
6		Penutup	
7	20/Jan/09	Hasil Revisi	
8			
9			
10			

Malang, Januari 2009

Dosen pembimbing

**Ir. F. Yudi Limpraptono, MT**

NIP. 1039500274

Form S-4a



```
.include "m8535def.inc"

.def    flag_ON      =R1
.def    cdown1=R2
.def    cdown2=R3
.def    cdown3=R4
.def    cdown4=R5
.def    sec1    =R6
.def    min1    =R7
.def    hour1   =R8
.def    ds7      =R9
.def    flag_h   =R12
.def    S2       =R13
.def    A1       =R14
.def    MH       =R25
.def    BCD      =R24
.def    BCDL     =R23
.def    tmp      =r16
.def    txbyte   =r17
.def    rxbyte   =r18
.def    tmp_1    =r19

.equ    fclock    =11059200
.equ    baud_rate =19200
.equ    ubbr_value = (fclock/(16*baud_rate))-1
.equ    timer_value1=0xD5D0
.equ    timer_value = 0xffca
```

---

```
.equ timer_value100 = 0xfba
```

```
;VARIABLE PORT
```

```
;PORT A-----
```

```
.EQU RED1 = 0
```

```
.EQU YELLOW1 = 1
```

```
.EQU GREEN1 = 2
```

```
.EQU RED2 = 3
```

```
.EQU YELLOW2 = 4
```

```
.EQU GREEN2 = 5
```

```
.EQU RED3 = 6
```

```
.EQU YELLOW3 = 7
```

```
;PORT B-----
```

```
.EQU GREEN3 = 0
```

```
.EQU RED4 = 1
```

```
.EQU YELLOW4 = 2
```

```
.EQU GREEN4 = 3
```

```
.EQU SENSOR= 4
```

---

```
;PORT D-----  
  
;SERIAL SHIFT REGISTER  
  
.EQU CLR1 = 4  
.EQU CLK1 = 5  
.EQU SR_B = 6  
.EQU SR_A = 7  
  
  
.org 0x0000  
rjmp main  
  
;org 0x000B  
;rjmp usart_rxc  
  
  
main:  
ldi r16,low(RAMEND)  
out SPL,r16  
ldi r16,high(RAMEND)  
out SPH,r16  
ldi r16,0xff  
out ddra,r16  
out porta,r16  
out ddrC,r16  
out portc,r16  
out ddrb,r16  
out portb,r16  
ldi tmp,0b01111111
```

---

```
out        ddrd,tmp
ldi        tmp,0b11111111
out        portd,tmp
rcall     init_usart
rcall     delay1d
rcall     reset_all
ldi        tmp,0
out        portc,tmp
```

tes\_lampu:

```
ldi        tmp,0
out        porta,tmp
out        portb,tmp
rcall     delay1d
ldi        tmp,255
out        porta,tmp
out        portb,tmp
rcall     delay1d
ldi        tmp,0
out        porta,tmp
out        portb,tmp
rcall     delay1d
ldi        tmp,255
out        porta,tmp
out        portb,tmp
rcall     delay1d
```

---

```
ldi      tmp,0
out      porta,tmp
out      portb,tmp
rcall    delay1d
ldi      tmp,255
out      porta,tmp
out      portb,tmp
rcall    delay1d
rcall    cek_sms
;rjmp    tes_lampu
```

Hijau1:

```
ldi      tmp,1
mov      flag_h,tmp
ldi      tmp,49
mov      cdown1,tmp
sbi      portb,GREEN4
cbi      porta,GREEN1
cbi      porta,RED2
cbi      porta,RED3
cbi      portb,RED4
```

loop\_H1:

```
mov      ds7,cdown1
rcall    dis_7
rcall    delay1d
```

---

```
rcall cek_sms
dec cdown1
mov tmp,cdown1
cpi tmp,1
brne loop_H1
mov ds7,cdown1
rcall dis_7
```

loop\_K1:

```
cbi porta,YELLOW1
cbi porta,YELLOW2
rcall delay1d
rcall cek_sms
sbi porta,YELLOW1
sbi porta,YELLOW2
dec cdown1
mov ds7,cdown1
rcall dis_7
rcall delay1d
rcall cek_sms
```

Hijau2:

```
ldi tmp,2
mov flag_h,tmp
ldi tmp,49
mov cdown2,tmp
```

---



```
sbi          porta, GREEN1
cbi          porta, GREEN2
cbi          porta, RED1
cbi          porta, RED3
cbi          portb, RED4
```

loop\_H2:

```
mov          ds7, cdown2
rcall       dis_7
rcall       delay1d
rcall       cek_sms
dec          cdown2
mov          tmp, cdown2
cpi         tmp, 1
brne       loop_H2
mov          ds7, cdown2
rcall       dis_7
```

loop\_K2:

```
cbi          porta, YELLOW2
cbi          porta, YELLOW3
rcall       delay1d
rcall       cek_sms
sbi          porta, YELLOW2
sbi          porta, YELLOW3
dec          cdown2
```

---

```
mov          ds7,cdown2
rcall   dis_7
rcall   delay1d
rcall   cek_sms
```

Hijau3:

```
ldi          tmp,3
mov          flag_h,tmp
ldi          tmp,49
mov          cdown3,tmp
sbi          porta,GREEN2
cbi          portb,GREEN3
cbi          porta,RED1
cbi          porta,RED2
cbi          portb,RED4
```

loop\_H3:

```
mov          ds7,cdown3
rcall   dis_7
rcall   delay1d
rcall   cek_sms
dec          cdown3
mov          tmp,cdown3
cpi          tmp,1
brne  loop_H3
mov          ds7,cdown3
```

---

```
rcall    dis_7
```

```
loop_K3:
```

```
cbi      porta,YELLOW3
```

```
cbi      portb,YELLOW4
```

```
rcall    delay1d
```

```
rcall    cek_sms
```

```
sbi      porta,YELLOW3
```

```
sbi      portb,YELLOW4
```

```
dec      cdown4
```

```
mov      ds7,cdown4
```

```
rcall    dis_7
```

```
rcall    delay1d
```

```
rcall    cek_sms
```

```
Hijau4:
```

```
ldi      tmp,4
```

```
mov      flag_h,tmp
```

```
ldi      tmp,49
```

```
mov      cdown4,tmp
```

```
sbi      portb,GREEN3
```

```
cbi      portb,GREEN4
```

```
cbi      porta,RED1
```

```
cbi      porta,RED2
```

```
cbi      portb,RED3
```

---

```
loop_H4:
mov     ds7,cdown4
rcall  dis_7
rcall  delay1d
rcall  cek_sms
dec     cdown4
mov     tmp,cdown4
cpi     tmp,1
brne   loop_H4
mov     ds7,cdown4
rcall  dis_7
```

```
loop_K4:
cbi     portb,YELLOW4
cbi     porta,YELLOW1
rcall  delay1d
rcall  cek_sms
sbi     portb,YELLOW4
sbi     porta,YELLOW1
dec     cdown4
mov     ds7,cdown4
rcall  dis_7
rcall  delay1d
rcall  cek_sms
rjmp   Hijau1
```

---

```

cek_perintah:
;*****

cek_1_ON:
;*****

clr          r27

ldi         r26,$60

ldi         zl,low(2*msg1_ON)

ldi         zh,high(2*msg1_ON)

adiw   zh:zl,2

Loop_1_ON:

lpm

mov         r29,r0

cpi         r29,0

breq   ON_1

ld         tmp_1,x+

cp         tmp_1,r0

brne   cek_2_ON

adiw   zh:zl,1

rjmp   Loop_1_ON

ON_1:

ldi         tmp,0

mov         ds7,tmp

rcall   dis_7

```

---

```
ldi      tmp,1
mov      flag_ON,tmp
rcall   warning
cbi      porta,GREEN1
cbi      porta,RED2
cbi      porta,RED3
cbi      portb,RED4
SBI      porta,RED1
sbi      porta,GREEN2
sbi      portB,GREEN3
sbi      portB,GREEN4
;rcall   kirim_cmgd
RCALL   _report
```

```
Loop_ON_ON:
rcall   cek_sms
rcall   delay1d
rcall   delay1d
rcall   delay1d
rjmp    Loop_ON_ON
```

```
,*****
,
cek_2_ON:
,*****
clr      r27
ldi      r26,$60
```

---

```
ldi          zl,low(2*msg2_ON)
ldi          zh,high(2*msg2_ON)
adiw  zh:zl,2
```

Loop\_2\_ON:

```
lpm
mov          r29,r0
cpi          r29,0
breq  ON2
ld           tmp_1,x+
cp           tmp_1,r0
brne  cek_3_ON
adiw  zh:zl,1
rjmp  Loop_2_ON
```

ON2:

```
ldi          tmp,0
mov          ds7,tmp
rcall  dis_7
ldi          tmp,2
mov          flag_ON,tmp
rcall  warning
cbi          porta,GREEN2
cbi          porta,RED1
cbi          porta,RED3
cbi          portb,RED4
```

---

```

sbi          porta,RED2
sbi          porta,GREEN1
sbi          portB,GREEN3
sbi          portB,GREEN4
;rcall  kirim_cmgd
RCALL  _report
rjmp  Loop_ON_ON

;*****
cek_3_ON:
;*****
clr          r27
ldi          r26,$60
ldi          zl,low(2*msg3_ON)
ldi          zh,high(2*msg3_ON)
adiw  zh:zl,2

Loop_3_ON:
lpm
mov          r29,r0
cpi          r29,0
breq  ON3
ld          tmp_1,x+
cp          tmp_1,r0
brne  cek_4_ON
adiw  zh:zl,1

```

---



```
rjmp Loop_3_ON
```

```
ON3:
```

```
ldi tmp,0
```

```
mov ds7,tmp
```

```
rcall dis_7
```

```
ldi tmp,3
```

```
mov flag_ON,tmp
```

```
rcall warning
```

```
cbi portb,GREEN3
```

```
SBI PORTA,RED3
```

```
cbi porta,RED1
```

```
cbi porta,RED2
```

```
cbi portb,RED4
```

```
sbi porta,GREEN1
```

```
sbi porta,GREEN2
```

```
sbi portB,GREEN4
```

```
;rcall kirim_cmgd
```

```
RCALL _report
```

```
rjmp Loop_ON_ON
```

```
,*****
```

```
cek_4_ON:
```

```
,*****
```

```
clr r27
```

```
ldi r26,$60
```

---

```
ldi        zl,low(2*msg4_ON)
ldi        zh,high(2*msg4_ON)
adiw      zh:zl,2
```

Loop\_4\_ON:

```
lpm
mov        r29,r0
cpi        r29,0
breq      ON4
ld         tmp_1,x+
cp         tmp_1,r0
brne      cek_OFF
adiw      zh:zl,1
rjmp      Loop_4_ON
```

ON4:

```
ldi        tmp,0
mov        ds7,tmp
rcall      dis_7
ldi        tmp,4
mov        flag_ON,tmp
rcall      warning
cbi        portb,GREEN4
SBI        PORTB,RED4
cbi        porta,RED1
cbi        porta,RED2
```

```

cbi          porta,RED3
sbi          porta,GREEN1
sbi          porta,GREEN2
sbi          portB,GREEN3

;rcall  kirim_cmgd
RCALL  _report
rjmp   Loop_ON_ON

,*****
cek_OFF:
,*****

clr          r27
ldi          r26,$60
ldi          zl,low(2*msg_OFF)
ldi          zh,high(2*msg_OFF)
adiw  zh:zl,2

Loop_OFF:
lpm
mov          r29,r0
cpi          r29,0
breq  OFF1
ld          tmp_1,x+
cp          tmp_1,r0
brne  kirim_false1
adiw  zh:zl,1

```

---

```
rjmp Loop_OFF
```

```
OFF1:
```

```
mov     tmp,flag_ON
```

```
cpi     tmp,1
```

```
breq   to_hijau2
```

```
cpi     tmp,2
```

```
breq   to_hijau2
```

```
cpi     tmp,3
```

```
breq   to_hijau3
```

```
cpi     tmp,4
```

```
breq   to_hijau4
```

```
RET
```

```
kirim_false1:
```

```
ret
```

```
to_hijau1:
```

```
LDI     TMP,0
```

```
MOV     flag_ON,TMP
```

```
RCALL  _report
```

```
ldi     tmp,5
```

```
mov     ds7,tmp
```

```
rcall  dis_7
```

```
loop_to_1:
```

---

```
cbi      portb,YELLOW4
cbi      porta,YELLOW1
rcall    delay1d
sbi      portb,YELLOW4
sbi      porta,YELLOW1
dec      ds7
rcall    dis_7
mov      tmp,ds7
cpi      tmp,0
brne    loop_to_1
rjmp     hijau1
```

```
;=====
```

```
to_hijau2:
```

```
LDI      TMP,0
MOV      flag_ON,TMP
RCALL    _report
ldi      tmp,5
mov      ds7,tmp
rcall    dis_7
```

```
loop_to_2:
```

```
cbi      porta,YELLOW2
cbi      porta,YELLOW1
rcall    delay1d
```

---

```
sbi        porta,YELLOW2
sbi        porta,YELLOW1
dec        ds7
rcall     dis_7
mov        tmp,ds7
cpi        tmp,0
brne     loop_to_2
rjmp     hijau2
```

```
;=====
```

```
to_hijau3:
```

```
LDI        TMP,0
MOV        flag_ON,TMP
RCALL     _report
ldi        tmp,5
mov        ds7,tmp
rcall     dis_7
```

```
loop_to_3:
```

```
cbi        porta,YELLOW2
cbi        porta,YELLOW3
rcall     delay1d
sbi        porta,YELLOW2
sbi        porta,YELLOW3
dec        ds7
```

---

```
rcall dis_7
mov     tmp,ds7
cpi     tmp,0
brne   loop_to_3
rjmp   hijau3
```

```
;=====
```

```
to_hijau4:
```

```
LDI     TMP,0
MOV     flag_ON,TMP
RCALL  _report
ldi     tmp,5
mov     ds7,tmp
rcall  dis_7
```

```
loop_to_4:
```

```
cbi     porta,YELLOW3
cbi     portb,YELLOW4
rcall  delay1d
sbi     porta,YELLOW3
sbi     portb,YELLOW4
dec     ds7
rcall  dis_7
mov     tmp,ds7
cpi     tmp,0
```

---

```
brne loop_to_embuh
rjmp hijau4
```

```
loop_to_embuh:
rjmp hijau1
```

```
warning:
```

```
cbi      porta,YELLOW1
cbi      porta,YELLOW2
cbi      porta,YELLOW3
cbi      portb,YELLOW4
rcall    delay1d
sbi      porta,YELLOW1
sbi      porta,YELLOW2
sbi      porta,YELLOW3
sbi      portb,YELLOW4
ret
```

```
msg1_ON:
```

```
.db "03B1A713",0 ;10N
```

```
msg2_ON:
```

```
.db "03B2A713",0 ;20N
```

```
msg3_ON:
```

```
.db "03B3A713",0 ;30N
```

---



msg4\_ON:

.db "03B4A713",0 ;4ON

msg\_OFF:

.db "034FA311",0 ;OFF

report\_lampu:

rcall kirim\_cmgs

ldi zl,low(2\*msg\_OFF)

ldi zh,high(2\*msg\_OFF)

S\_T1\_Of:

lpm

mov txbyte,r0

cpi txbyte,0

breq Out\_T1\_Of

rcall usart\_tx

adiw zh:zl,1

rjmp S\_T1\_ON

out\_T1\_Of:

rcall kirim1A

ret

lampu\_mati:

---

```
rcall report_lampu
tunggu_lampu:
sbic portd,7
rjmp tunggu_lampu
ret
```

```
delay1d:
;sbis portd,7 ; jika PLN lampu mati, tunggu sampai nyala
;rcall lampu_mati
;RCALL timing_loop ; menambahkan nilai waktu, awal inisialisasi
;rcall cek_time_21 ; pada awal nilai waktu telah di isi 06.00.00
RCALL DELAY1D1
;
RET
```

```
cek_time_21:
mov tmp,hour1
cpi tmp,21
breq kuning_semua
ret
```

```
kuning_semua:
rcall warning
rcall delay1d
mov tmp,hour1
cpi tmp,6
```

---

```
brne kuning_semua
```

```
ret
```

```
timing_loop:
```

```
second:
```

```
inc sec1
```

```
mov tmp,sec1
```

```
cpi tmp,60
```

```
breq minute
```

```
ret
```

```
minute:
```

```
inc min1
```

```
mov tmp,min1
```

```
cpi tmp,60
```

```
breq hour
```

```
ret
```

```
hour:
```

```
inc hour1
```

```
mov tmp,hour1
```

```
cpi tmp,24
```

```
breq out_cek
```

```
ret
```

```
out_cek:
```

---

out\_T1\_ON:

```
rcall  kirim1A
rcall  delay1d
rcall  delay1d
rcall  kirim_cmgsd
rcall  delay1d
rcall  delay1d
rcall  delay1d
ret
```

kirim1A:

```
ldi    txbyte,$1A
rcall  usart_tx
ret
```

kirim\_cmgs:

```
ldi    zl,low(2*msg_cmgs)
ldi    zh,high(2*msg_cmgs)
```

cmgs:

```
lpm
mov    txbyte,r0
cpi    txbyte,0
breq  tunggu
rcall  usart_tx
```

---

```
adiw zh:zl,1
```

```
rjmp cmgs
```

```
msg_cmgs:
```

```
.db "AT+CMGS=27",13,10,0
```

```
;-----
```

```
;tunggu
```

```
;-----
```

```
tunggu:
```

```
rcall usart_rx
```

```
cpi rxbyte,'>'
```

```
breq kirim_header
```

```
rjmp tunggu
```

```
;-----
```

```
;Kirim Header
```

```
;-----
```

```
kirim_header:
```

```
ldi zl,low(2*msgheader)
```

```
ldi zh,high(2*msgheader)
```

```
header:
```

```
lpm
```

```
mov txbyte,r0
```

```
cpi txbyte,0
```

---

```
breq kirim_pesan
rcall usart_tx
adiw zh:zl,1
rjmp header
```

msgheader:

```
.db "07912658050000F001000D91265846467744F40000",0
;.db "07912658050000F001000D91265837376724F50000",0
;-----
```

kirim\_pesan:

```
ret
```

```
;-----
;SUBROUTIN SET TERIMA SMS
```

```
;-----
cek_sms:
```

periksa\_sms:

```
rcall KIRIM_CMGR
mov     tmp,A1
cpi     tmp,1
breq   pesanterima
ret
```

pesanterima:

```
rcall    usart_rx
```

```
rcall    usart_rx
```

```
cpi      rxbyte,'7'
```

```
brne    cek_06
```

```
ldi     tmp,0
```

loop\_07:

```
rcall    usart_rx
```

```
cpi      tmp,15
```

```
breq    cek_nomor
```

```
inc     tmp
```

```
rjmp    loop_07
```

cek\_06:

```
cpi      rxbyte,'6'
```

```
brne    out_cek_SMSC
```

```
ldi     tmp,0
```

loop\_06:

```
rcall    usart_rx
```

```
cpi      tmp,13
```

```
breq    cek_nomor
```

```
inc     tmp
```

```
rjmp    loop_06
```

---

out\_cek\_SMSC:

rcall kirim\_cmgd

ret

cek\_nomor1:

ldi tmp,0

loop\_cek\_N:

rcall usart\_rx

cpi tmp,21

breq skipwaktu

inc tmp

rjmp loop\_cek\_N

cek\_nomor:

ldi zl,low(2\*msg\_user)

ldi zh,high(2\*msg\_user)

loadperintah:

lpm

mov r29,r0

cpi r29,0 ;apakah pembandingan sudah nol

breq skipwaktu ;jika sudah lompat ke skip waktu

rcall usart\_rx

cp rxbyte,r0 ;cocokkan karakter sms

brne bukanuser ;jika tidak cocok bukan hp user

adiw zh:zl,1

---



```
rjmp loadperintah
```

```
msg_user:
```

```
.db "0D91265846467744F40000",0 ;sms terima
```

```
bukanuser:
```

```
rcall kirim_cmgd ; kirim AT+CMGD=1
```

```
ret
```

```
skipwaktu:
```

```
ldi r16,13 ;skip 16 bit data waktu
```

```
loopskip:
```

```
rcall usart_rx
```

```
cpi r16,0
```

```
breq end_skip
```

```
dec r16
```

```
rjmp loopskip
```

```
end_skip:
```

```
rcall usart_rx
```

```
cpi rxbyte,'0'
```

```
breq cek_03
```

```
rjmp kirim_false
```

```
cek_03:
```

---

```
rcall    usart_rx
cpi      rxbyte,'3'
brne    cek_04
clr      r27
ldi      r26,$60
ldi      tmp,0
```

Loop\_cek03:

```
rcall    usart_rx
st       x+,rxbyte
cpi      tmp,5
breq    to_cek_perintah
inc      tmp
rjmp    loop_cek03
```

cek\_04:

```
cpi      rxbyte,'4'
brne    kirim_false
clr      r27
ldi      r26,$60
ldi      tmp,0
```

Loop\_cek04:

```
rcall    usart_rx
st       x+,rxbyte
cpi      tmp,7
```

---

```
breq to_cek_perintah
inc tmp
rjmp loop_cek04
```

```
 kirim_false:
rcall kirim_cmgd
ret
```

```
to_cek_perintah:
rjmp cek_perintah
ret
```

```
;-----
;-----
;KIRIM AT+CMGR=1
;-----
```

```
KIRIM_CMGR:
ldi zl,low(2*msgcmgr)
ldi zh,high(2*msgcmgr)
```

```
CMGR:
lpm
mov txbyte,r0
cpi txbyte,0
```

---

```
breq  tunggunol1
```

```
rcall usart_tx
```

```
adiw  zh:zl,1
```

```
rjmp  CMGR
```

```
msgcmgr:
```

```
.db  "at+CMGR=1",13,10,0      ;(13+jml karakter)
```

```
tunggunol1:
```

```
loop_skip:
```

```
rcall usart_rx
```

```
cpi   rxbyte,':'
```

```
brne  loop_skip
```

```
rcall usart_rx
```

```
rcall usart_rx
```

```
cpi   rxbyte,'0'
```

```
brne  menuju_data
```

```
ldi   tmp,0
```

```
;mov  A1,tmp
```

```
ret
```

```
menuju_data:
```

```
rcall usart_rx
```

```
cpi   rxbyte,$0D
```

---

```

brne  menuju_data
rcall  usart_rx                ;setelah $0D masih ada data satu lagi
ldi    tmp,1
mov    A1,tmp
ret

;-----
;KIRIM AT+CMGD=1
;-----

 kirim_cmgd:
rcall  delay1d
rcall  delay1d
rcall  delay1d
ldi    zl,low(2*msgcmd)
ldi    zh,high(2*msgcmd)

load10:
lpm
mov    txbyte,r0
cpi    txbyte,0
breq  tunggu_cmgd
rcall  usart_tx
adiw  zh:zl,1
rjmp  load10

msgcmd:
.db   "at+cmd=1",13,10,0

```

---

tunggu\_cmgd:

rcall usart\_rx

cpi rxbyte,\$4B

breq kembali\_cmgd

rjmp tunggu\_cmgd

kembali\_cmgd:

ldi tmp,0

mov A1,tmp

rcall delay1d

ret

;

;Subrutin inisialisasi usart

;

init\_usart:

ldi tmp,high(ubbr\_value) ;set baud rate

out UBRRH,tmp

ldi tmp,low(ubbr\_value)

out UBRRL,tmp

ldi tmp,(1<<RXEN)|(1<<TXEN)|(1<<RXCIE) ;enable receiver and transmitter

out UCSRB,tmp

ldi tmp,(1<<URSEL)|(3<<UCSZ0) ;set frame format: 8data, 1 stop bit

out UCSRC,tmp

ret

---

```
;-----
```

```
;uart transmit data
```

```
;-----
```

```
uart_tx:
```

```
sbis UCSRA,UDRE ;wait for empty transmit buffer
```

```
rjmp uart_tx
```

```
out UDR,txbyte ;Put data into buffer, sends the data
```

```
ret
```

```
;-----
```

```
;uart receive data
```

```
;-----
```

```
uart_rx:
```

```
sbis UCSRA,RXC ;wait for data to be received
```

```
rjmp uart_rx
```

```
in rxbyte,UDR ;get and return received data from buffer
```

```
ret
```

```
;-----
```

```
;Subrutin delay 5ms
```

```
;-----
```



```

out    TCNT1H,r16

ldi    r16,LOW(timer_value100)

out    TCNT1L,r16

ldi    r16,0b00000001           ;masukkan prescaler utk timer disini
1024

out    TCCR1B,R16

looptimer3:

in     R17,TIFR

sbrs  r17,TOV1                 ;tunggu sampai timer1 overflow flag set

rjmp  looptimer3

ldi    r16,0b00000100         ;timer 1 overflow flag dinolkan dg beri
logik 1

out    TIFR,r16

ret

;=====

;-----

;Subrutin delay 1d

;-----

delay1d1:

ldi    r16,0b00000100

out    TIMSK,r16

ldi    r16,high(timer_value1)

out    TCNT1H,r16

ldi    r16,low(timer_value1)

out    TCNT1L,r16

```

---



```
ldi    r16,0b00000101
```

```
out    TCCR1B,r16
```

```
looptimer2:
```

```
in     r17,TIFR
```

```
sbrs  r17,TOV1
```

```
rjmp  looptimer2
```

```
ldi     r16,0b000000100
```

```
out    TIFR,r16
```

```
ret
```

```
usart_rxc:
```

```
CBI    PORTB,3
```

```
rcall  usart_rx
```

```
mov    tmp,rxbyte
```

```
;rcall kirim_karakter
```

```
rcall  delay1d
```

```
reti
```

```
seting:
```

```
ldi    r21,0b10000000
```

```
out    UCSRA,r21
```

```
sei
```

```
rjmp  usart_rxc
```

```
kirim_csms:
```

---

```
ldi    zl,low(2*msg_csms)
ldi    zh,high(2*msg_csms)
```

```
csms:
```

```
lpm
```

```
mov    txbyte,r0
```

```
cpi    txbyte,0
```

```
breq  tunggus
```

```
rcall  usart_tx
```

```
;rcall  usart_rx ;>>>test loopback
```

```
adiw  zh:zl,1
```

```
rjmp  csms
```

```
msg_csms:
```

```
.db   "AT+CSMS=1",13,10,0      ;(13+jml karakter)
```

```
;-----
```

```
;tunggu
```

```
;-----
```

```
tunggus:
```

```
rcall  usart_rx
```

```
cpi    rxbyte,'K'
```

```
brne  tunggus
```

```
rcall  usart_rx
```

---

```
rcall    usart_rx
```

```
;=====
```

```
kirim_cnmi:
```

```
ldi     zl,low(2*msg_cnmi)
```

```
ldi     zh,high(2*msg_cnmi)
```

```
cnmi:
```

```
lpm
```

```
mov     txbyte,r0
```

```
cpi     txbyte,0
```

```
breq    tungguss
```

```
rcall   usart_tx
```

```
;rcall  usart_rx ;>>>test loopback
```

```
adiw   zh:zl,1
```

```
rjmp   cnmi
```

```
msg_cnmi:
```

```
.db    "AT+CNMI=1,1,0,1,1",13,10,0 ;(13+jml karakter)
```

```
;-----
```

```
;tunggu
```

```
;-----
```

```
tungguss:
```

---

```
rcall    usart_rx
cpi      rxbyte,'K'
brne    tungguss
rcall    usart_rx
rcall    usart_rx
ret
```

```
dis_7:
mov      tmp,ds7
rcall    bin2bcd8
out      portc,BCD
ret
```

```
;=====
```

```
dis_7_1:
mov      tmp,ds7

rcall    bin2bcd8
ldi      tmp,8
sec
```

```
to_sipo:
ror      BCD
brcc    putchar1          ;if carry set
sbi      PORTD,SR_A
```

---

s\_clock:

cbi PORTD,CLK1

sbi PORTD,CLK1

dec tmp

brne to\_sipo

ret

putchar1:

cbi PORTD,SR\_A ; send a '0'

rjmp s\_clock

bin2bcd8:

clr BCD

bBCD8\_1:

subi tmp,10

brcs bBCD8\_2

inc BCD

rjmp bBCD8\_1 ;loop again

bBCD8\_2:

subi tmp,-10 ;compensate extra subtraction

swap BCD

add BCD,tmp

ret

---

```
reset_all:
ldi    tmp,1
mov    MH,tmp
ldi    tmp,0
mov    A1,tmp
mov    txbyte,tmp
mov    rxbyte,tmp
mov    sec1,tmp
mov    min1,tmp
mov    bcd,tmp
mov    bcdl,tmp
mov    ds7,tmp
ldi    tmp,6
mov    hour1,tmp
ret
```

---

## Features

High-performance, Low-power AVR<sup>®</sup> 8-bit Microcontroller

Advanced RISC Architecture

- 130 Powerful Instructions – Most Single Clock Cycle Execution
- 32 x 8 General Purpose Working Registers
- Fully Static Operation
- Up to 16 MIPS Throughput at 16 MHz
- On-chip 2-cycle Multiplier

Nonvolatile Program and Data Memories

- 8K Bytes of In-System Self-Programmable Flash  
Endurance: 10,000 Write/Erase Cycles
- Optional Boot Code Section with Independent Lock Bits  
In-System Programming by On-chip Boot Program  
True Read-While-Write Operation
- 512 Bytes EEPROM  
Endurance: 100,000 Write/Erase Cycles
- 512 Bytes Internal SRAM
- Programming Lock for Software Security

Peripheral Features

- Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Real Time Counter with Separate Oscillator
- Four PWM Channels
- 8-channel, 10-bit ADC
  - 8 Single-ended Channels
  - 7 Differential Channels for TQFP Package Only
  - 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x for TQFP Package Only
- Byte-oriented Two-wire Serial Interface
- Programmable Serial USART
- Master/Slave SPI Serial Interface
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator

Special Microcontroller Features

- Power-on Reset and Programmable Brown-out Detection
- Internal Calibrated RC Oscillator
- External and Internal Interrupt Sources
- Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby

I/O and Packages

- 32 Programmable I/O Lines
- 40-pin PDIP, 44-lead TQFP, 44-lead PLCC, and 44-pad QFN/MLF

Operating Voltages

- 2.7 - 5.5V for ATmega8535L
- 4.5 - 5.5V for ATmega8535

Speed Grades

- 0 - 8 MHz for ATmega8535L
- 0 - 16 MHz for ATmega8535



8-bit AVR<sup>®</sup>  
Microcontroller  
with 8K Bytes  
In-System  
Programmable  
Flash

ATmega8535  
ATmega8535L

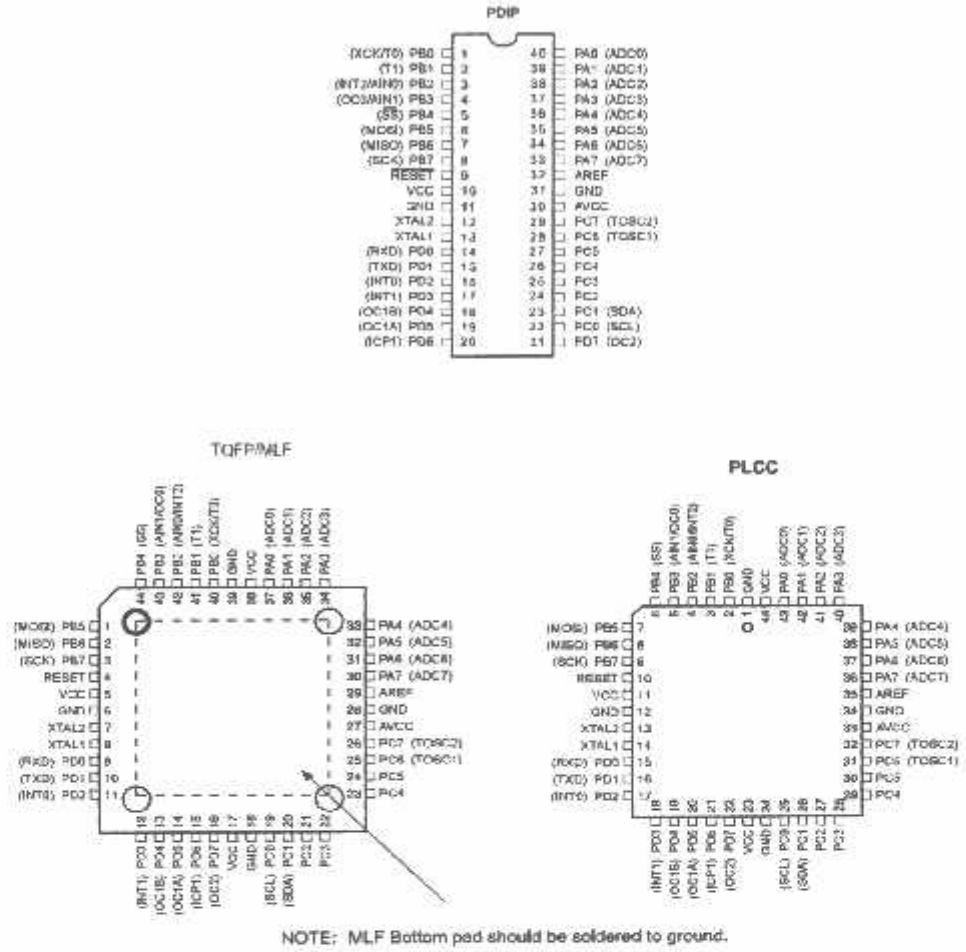
Rev. 2502G-AVR-04/05





## n Configurations

Figure 1. Pinout ATmega8535



## Disclaimer

Typical values contained in this data sheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

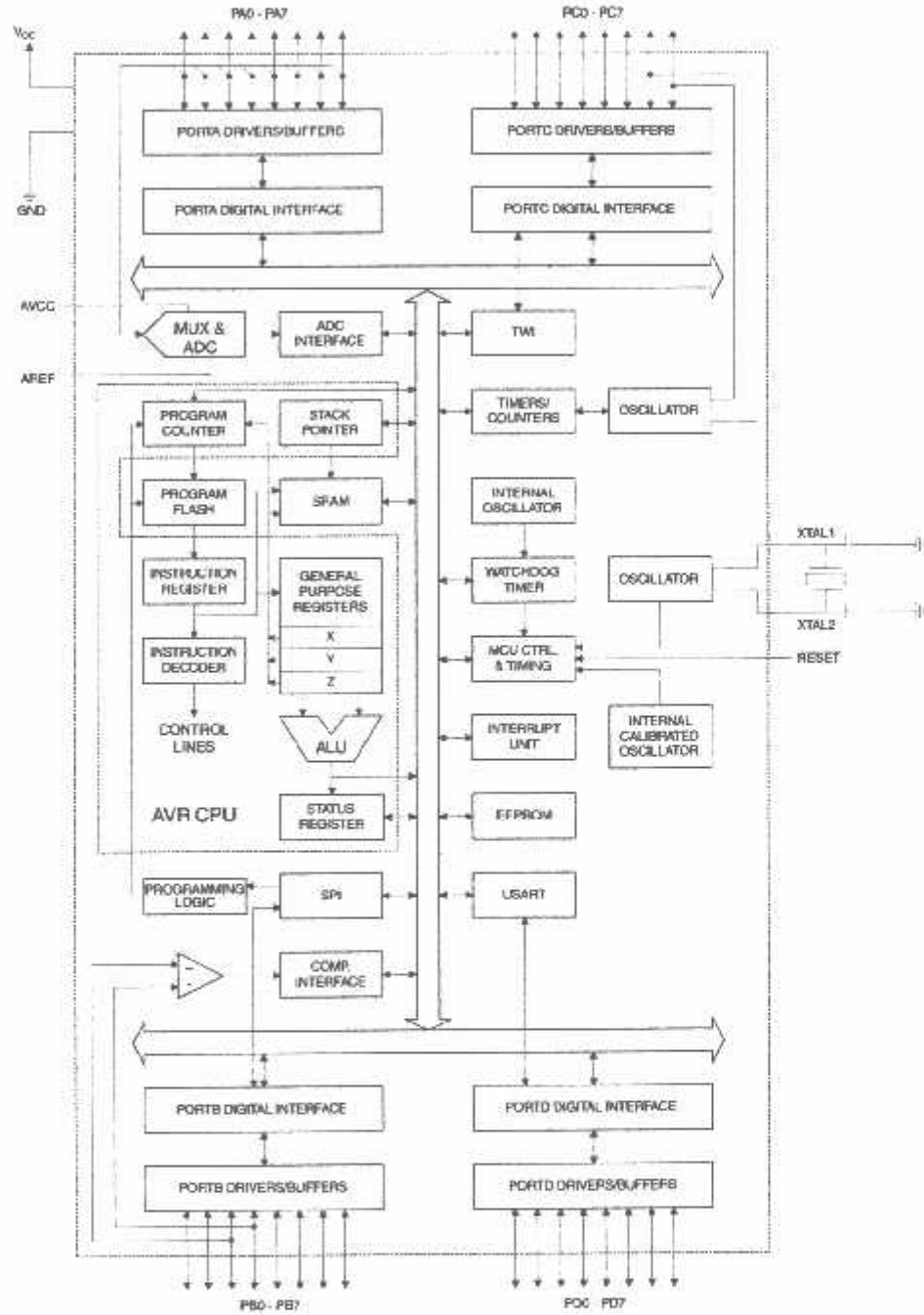


## Overview

The ATmega8535 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing instructions in a single clock cycle, the ATmega8535 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

## Block Diagram

Figure 2. Block Diagram





The AVR core combines a rich instruction set with 32 general purpose working registers. All 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega8535 provides the following features: 8K bytes of In-System Programmable Flash with Read-While-Write capabilities, 512 bytes EEPROM, 512 bytes SRAM, 32 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain in TQFP package, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or Hardware Reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption. In Extended Standby mode, both the main Oscillator and the asynchronous timer continue to run.

The device is manufactured using Atmel's high density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega8535 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega8535 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, In-Circuit Emulators, and evaluation kits.

## AT90S8535 Compatibility

The ATmega8535 provides all the features of the AT90S8535. In addition, several new features are added. The ATmega8535 is backward compatible with AT90S8535 in most cases. However, some incompatibilities between the two microcontrollers exist. To solve this problem, an AT90S8535 compatibility mode can be selected by programming the S8535C fuse. ATmega8535 is pin compatible with AT90S8535, and can replace the AT90S8535 on current Printed Circuit Boards. However, the location of fuse bits and the electrical characteristics differs between the two devices.

## AT90S8535 Compatibility Mode

Programming the S8535C fuse will change the following functionality:

- The timed sequence for changing the Watchdog Time-out period is disabled. See "Timed Sequences for Changing the Configuration of the Watchdog Timer" on page 44 for details.
- The double buffering of the USART Receive Register is disabled. See "AVR USART vs. AVR UART – Compatibility" on page 144 for details.

## ATmega8535(L)

25020-AVR-04/05

## Pin Descriptions

<b>V<sub>DD</sub></b>	Digital supply voltage.
<b>TD</b>	Ground.
<b>Port A (PA7..PA0)</b>	<p>Port A serves as the analog inputs to the A/D Converter.</p> <p>Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.</p>
<b>Port B (PB7..PB0)</b>	<p>Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.</p> <p>Port B also serves the functions of various special features of the ATmega8535 as listed on page 59.</p>
<b>Port C (PC7..PC0)</b>	<p>Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.</p>
<b>Port D (PD7..PD0)</b>	<p>Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.</p> <p>Port D also serves the functions of various special features of the ATmega8535 as listed on page 63.</p>
<b><math>\overline{\text{RESET}}</math></b>	Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 15 on page 36. Shorter pulses are not guaranteed to generate a reset.
<b>AL1</b>	Input to the Inverting Oscillator amplifier and input to the internal clock operating circuit.
<b>AL2</b>	Output from the inverting Oscillator amplifier.
<b>V<sub>CC</sub></b>	AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to V <sub>CC</sub> , even if the ADC is not used. If the ADC is used, it should be connected to V <sub>CC</sub> through a low-pass filter.
<b>AREF</b>	AREF is the analog reference pin for the A/D Converter.



## Source Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Please confirm with the C Compiler documentation for more details.

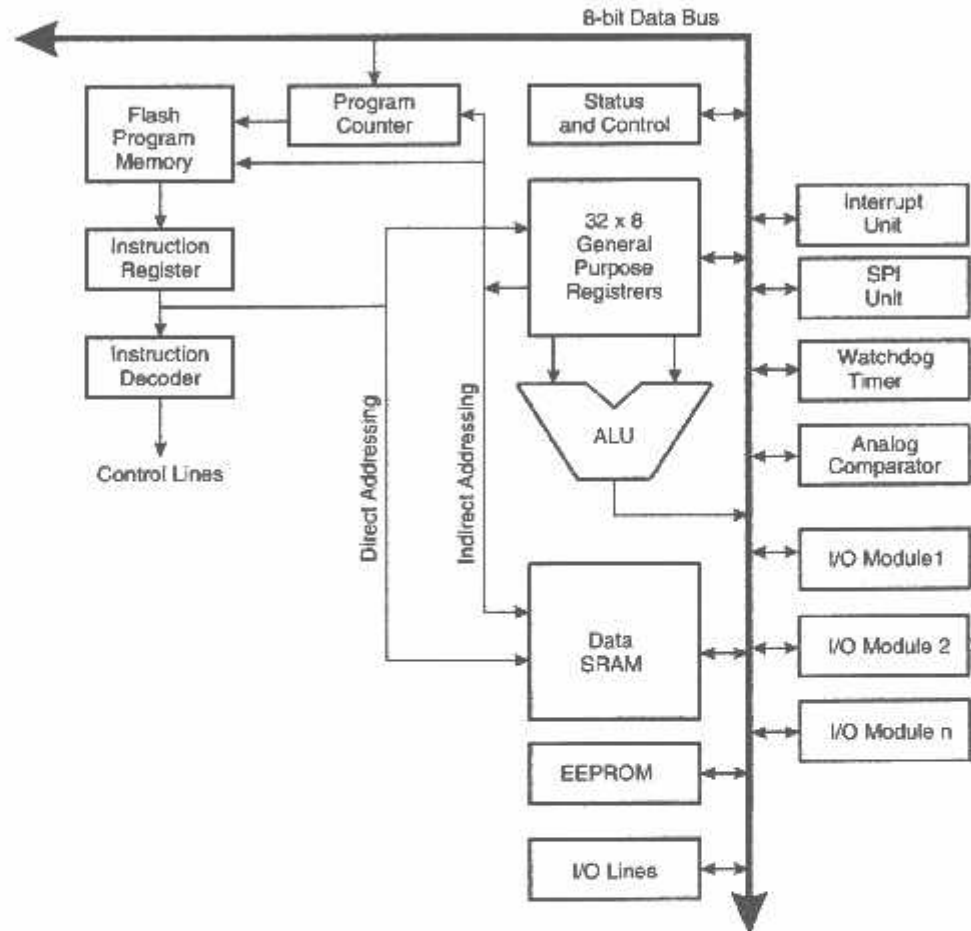
/R CPU Core

roduction

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

hitectural Overview

Figure 3. Block Diagram of the AVR MCU Architecture



In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Re-Programmable Flash memory.

The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle.



Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing – enabling efficient address calculations. One of these address pointers can also be used as an address pointer for look up tables in Flash program memory. These added function registers are the 16-bit X-, Y-, and Z-registers, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the reset routine (before subroutines or interrupts are executed). The Stack Pointer SP is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, 0x20 - 0x5F.

## LU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an Immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See the “Instruction Set” section for a detailed description.

## Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will, in many cases, remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

The AVR Status Register – SREG – is defined as:

BIT	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

- **Bit 6 – T: Bit Copy Storage**

The Bit Copy instructions BLD (Bit Load) and BST (Bit Store) use the T-bit as source or destination for the operated bit. A bit from a register in the Register file can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

- **Bit 5 – H: Half Carry Flag**

The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half carry is useful in BCD arithmetic. See the "Instruction Set Description" for detailed information.

- **Bit 4 – S: Sign Bit,  $S = N \oplus V$**

The S-bit is always an exclusive or between the Negative Flag N and the Two's Complement Overflow Flag V. See the "Instruction Set Description" for detailed information.

- **Bit 3 – V: Two's Complement Overflow Flag**

The Two's Complement Overflow Flag V supports two's complement arithmetics. See the "Instruction Set Description" for detailed information.

- **Bit 2 – N: Negative Flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

- **Bit 1 – Z: Zero Flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

- **Bit 0 – C: Carry Flag**

The Carry Flag C indicates a carry in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

## General Purpose Register File

The Register File is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the Register File:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

Figure 4 shows the structure of the 32 general purpose working registers in the CPU.

**Figure 4.** AVR CPU General Purpose Working Registers

	7	0	Addr.	
General Purpose Working Registers	R0		0x00	
	R1		0x01	
	R2		0x02	
	...			
	R13		0x0D	
	R14		0x0E	
	R15		0x0F	
	R16		0x10	
	R17		0x11	
	...			
	R26		0x1A	X-register Low Byte
	R27		0x1B	X-register High Byte
	R28		0x1C	Y-register Low Byte
	R29		0x1D	Y-register High Byte
	R30		0x1E	Z-register Low Byte
	R31		0x1F	Z-register High Byte

Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

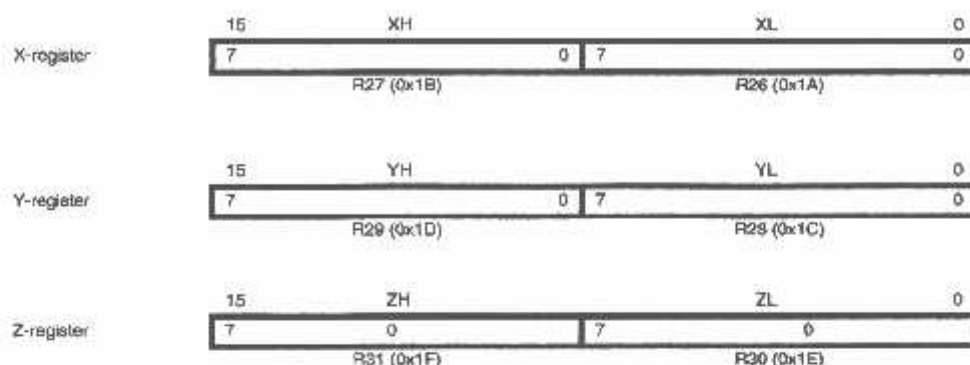
As shown in Figure 4, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y-, and Z-pointer Registers can be set to index any register in the file.



## X-register, Y-register, and Z-register

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the Data Space. The three indirect address registers X, Y, and Z are defined as described in Figure 5.

Figure 5. The X-, Y-, and Z-registers



In the different addressing modes, these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the instruction set reference for details).

## Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above 0x60. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

Bit	15	14	13	12	11	10	9	8	
	-	-	-	-	-	-	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0	

## Instruction Execution Timing

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock  $clk_{CPU}$ , directly generated from the selected clock source for the chip. No internal clock division is used.

Figure 6 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

**Figure 6. The Parallel Instruction Fetches and Instruction Executions**

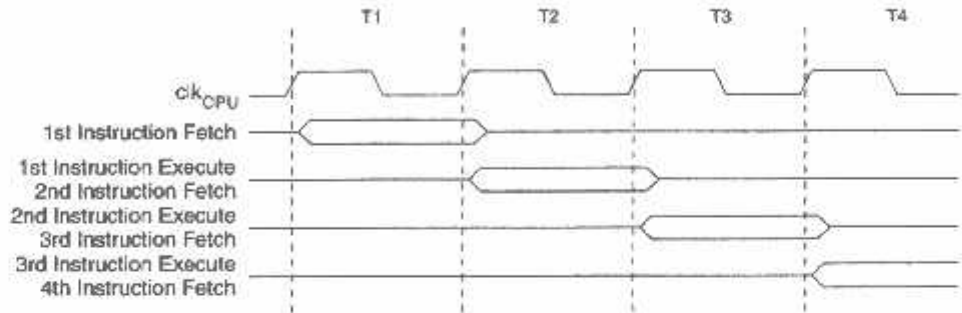
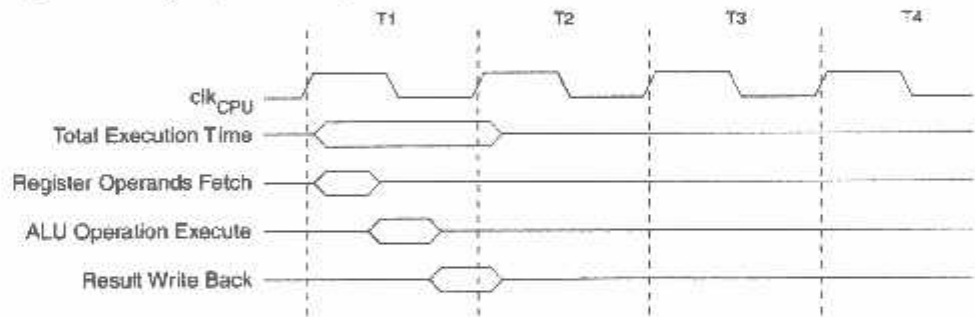


Figure 7 shows the internal timing concept for the Register file. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

**Figure 7. Single Cycle ALU Operation**



## Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate Program Vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security. See the section "Memory Programming" on page 235 for details.

The lowest addresses in the program memory space are, by default, defined as the Reset and Interrupt Vectors. The complete list of vectors is shown in "Interrupts" on page 45. The list also determines the priority levels of the different interrupts. The lower the address, the higher the priority level is. RESET has the highest priority, and next is INTO – the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the General Interrupt Control Register (GICR). Refer to "Interrupts" on page 45 for more information. The Reset Vector can

also be moved to the start of the Boot Flash section by programming the BOOTRST Fuse, see "Boot Loader Support – Read-While-Write Self-Programming" on page 222.

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the interrupt flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding interrupt flag. Interrupt flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the interrupt flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding interrupt flag(s) will be set and remembered until the Global Interrupt Enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have interrupt flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Note that the Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Assembly Code Example	
<b>in</b> r16, SREG	; store SREG value
<b>cli</b>	; disable interrupts during timed sequence
<b>sbi</b> EECR, EEMWE	; start EEPROM write
<b>sbi</b> EECR, EEWE	
<b>out</b> SREG, r16	; restore SREG value (I-bit)
C Code Example	
<b>char</b> cSREG;	
cSREG = SREG; /* store SREG value */	
/* disable interrupts during timed sequence */	
_CLI();	
EECR  = (1<<EEMWE); /* start EEPROM write */	
EECR  = (1<<EEWE);	
SREG = cSREG; /* restore SREG value (I-bit) */	



When using the SEI instruction to enable interrupts, the instruction following SEI will be executed before any pending interrupts, as shown in this example.

Assembly Code Example
<pre>sei ; set global interrupt enable sleep ; enter sleep, waiting for interrupt ; note: will enter sleep before any pending ; interrupt(s)</pre>
C Code Example
<pre>_SEI(); /* set global interrupt enable */ _SLEEP(); /* enter sleep, waiting for interrupt */ /* note: will enter sleep before any pending interrupt(s) */</pre>

#### Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum. After four clock cycles, the Program Vector address for the actual interrupt handling routine is executed. During this four clock cycle period, the Program Counter is pushed onto the Stack. The Vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes four clock cycles. During these four clock cycles, the Program Counter (two bytes) is popped back from the Stack, the Stack Pointer is incremented by two, and the I-bit in SREG is set.

## Memory

### System programmable Flash Program Memory

This section describes the different memories in the ATmega8535. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the ATmega8535 features an EEPROM Memory for data storage. All three memory spaces are linear and regular.

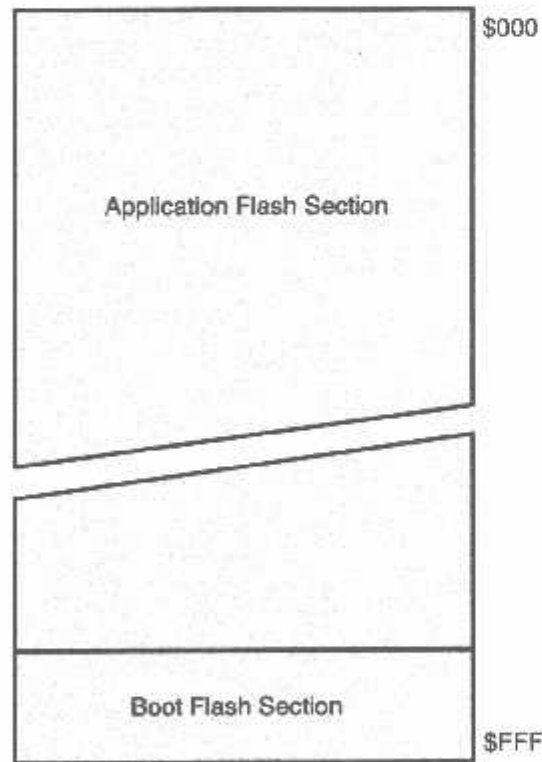
The ATmega8535 contains 8K bytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 4K x 16. For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section.

The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega8535 Program Counter (PC) is 12 bits wide, thus addressing the 4K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in "Boot Loader Support – Read-While-Write Self-Programming" on page 222. "Memory Programming" on page 235 contains a detailed description on Flash Programming in SPI or Parallel Programming mode.

Constant tables can be allocated within the entire program memory address space (see the LPM – Load Program Memory Instruction description).

Timing diagrams for Instruction fetch and execution are presented in "Instruction Execution Timing" on page 12.

**Figure 8.** Program Memory Map



## RAM Data Memory

Figure 9 shows how the ATmega8535 SRAM Memory is organized.

The 608 Data Memory locations address the Register File, the I/O Memory, and the internal data SRAM. The first 96 locations address the Register File and I/O Memory, and the next 512 locations address the internal data SRAM.

The five different addressing modes for the data memory cover: Direct, Indirect with Displacement, Indirect, Indirect with Pre-decrement, and Indirect with Post-increment. In the Register File, registers R26 to R31 feature the indirect addressing pointer registers.

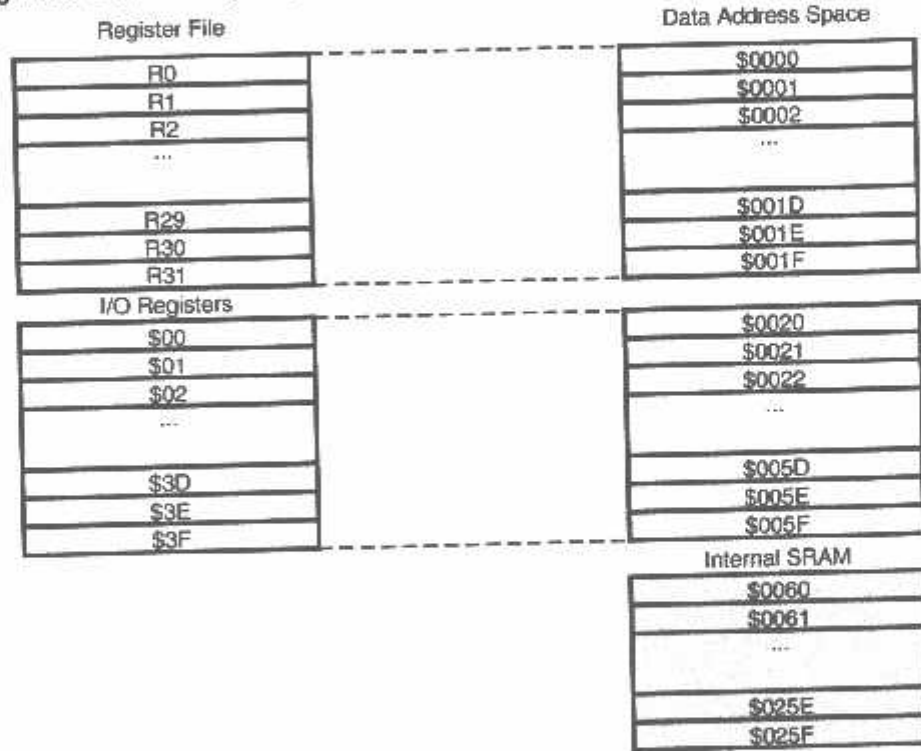
The direct addressing reaches the entire data space.

The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.

When using register Indirect addressing modes with automatic pre-decrement and post-increment, the address registers X, Y, and Z are decremented or incremented.

The 32 general purpose working registers, 64 I/O Registers, and the 512 bytes of internal data SRAM in the ATmega8535 are all accessible through all these addressing modes. The Register File is described in "General Purpose Register File" on page 10.

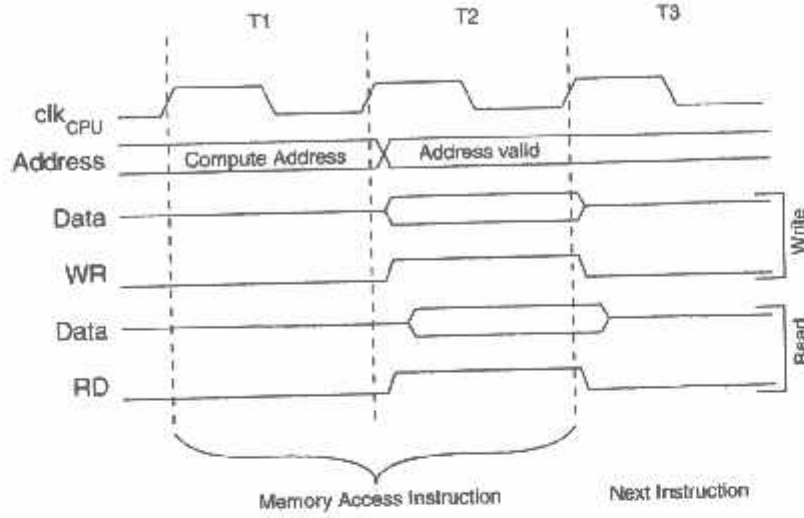
**Figure 9. Data Memory Map**



## Memory Access Times

This section describes the general access timing concepts for internal memory access. The internal data SRAM access is performed in two  $clk_{CPU}$  cycles as described in Figure 10.

**Figure 10.** On-chip Data SRAM Access Cycles



## EEPROM Data Memory

The ATmega8535 contains 512 bytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

"Memory Programming" on page 235 contains a detailed description on EEPROM Programming in SPI or Parallel Programming mode.

## EEPROM Read/Write Access

The EEPROM Access Registers are accessible in the I/O space.

The write access time for the EEPROM is given in Table 1. A self-timing function, however, lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies,  $V_{CC}$  is likely to rise or fall slowly on Power-up/down. This causes the device, for some period of time, to run at a voltage lower than specified as minimum for the clock frequency used, see "Preventing EEPROM Corruption" on page 21 for details on how to avoid problems in these situations.

In order to prevent unintentional EEPROM writes, a specific write procedure must be followed. Refer to the description of the EEPROM Control Register for details on this.

When the EEPROM is read, the CPU is halted for four clock cycles before the next instruction is executed. When the EEPROM is written, the CPU is halted for two clock cycles before the next instruction is executed.

### EEPROM Address Register – EEARH and EEARL

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	–	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	X	

- **Bits 15..9 – Res: Reserved Bits**

These bits are reserved bits in the ATmega8535 and will always read as zero.

- **Bits 8..0 – EEAR8..0: EEPROM Address**

The EEPROM Address Registers – EEARH and EEARL specify the EEPROM address in the 512 bytes EEPROM space. The EEPROM data bytes are addressed linearly between 0 and 511. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

### EEPROM Data Register – EEDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..0 – EEDR7..0: EEPROM Data**

For the EEPROM write operation, the EEDR Register contains the data to be written to the EEPROM in the address given by the EEAR Register. For the EEPROM read operation, the EEDR contains the data read out from the EEPROM at the address given by EEAR.

### EEPROM Control Register – EECR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	X	0	

- **Bits 7..4 – Res: Reserved Bits**

These bits are reserved bits in the ATmega8535 and will always read as zero.

- **Bit 3 – EERIE: EEPROM Ready Interrupt Enable**

Writing EERIE to one enables the EEPROM Ready Interrupt if the I-bit in SREG is set. Writing EERIE to zero disables the interrupt. The EEPROM Ready Interrupt generates a constant interrupt when EEWE is cleared.

- **Bit 2 – EEMWE: EEPROM Master Write Enable**

The EEMWE bit determines whether setting EEWE to one causes the EEPROM to be written. When EEMWE is set, setting EEWE within four clock cycles will write data to the EEPROM at the selected address. If EEMWE is zero, setting EEWE will have no effect. When EEMWE has been written to one by software, hardware clears the bit to zero after four clock cycles. See the description of the EEWE bit for an EEPROM write procedure.

- **Bit 1 – EEWE: EEPROM Write Enable**

The EEPROM Write Enable Signal EEWE is the write strobe to the EEPROM. When address and data are correctly set up, the EEWE bit must be written to one to write the



value into the EEPROM. The EEMWE bit must be written to one before a logical one is written to EEW, otherwise no EEPROM write takes place. The following procedure should be followed when writing the EEPROM (the order of steps 3 and 4 is not essential):

1. Wait until EEW becomes zero.
2. Wait until SPEN in SPMCR becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a logical one to the EEMWE bit while writing a zero to EEW in EECR.
6. Within four clock cycles after setting EEMWE, write a logical one to EEW.

The EEPROM can not be programmed during a CPU write to the Flash memory. The software must check that the Flash programming is completed before initiating a new EEPROM write. Step 2 is only relevant if the software contains a Boot Loader allowing the CPU to program the Flash. If the Flash is never updated by the CPU, step 2 can be omitted. See "Boot Loader Support – Read-While-Write Self-Programming" on page 222 for details about Boot programming.

**Caution:** An interrupt between step 5 and step 6 will make the write cycle fail, since the EEPROM Master Write Enable will time-out. If an interrupt routine accessing the EEPROM is interrupting another EEPROM access, the EEAR or EEDR Register will be modified, causing the interrupted EEPROM access to fail. It is recommended to have the Global Interrupt Flag cleared during all the steps to avoid these problems.

When the write access time has elapsed, the EEW bit is cleared by hardware. The user software can poll this bit and wait for a zero before writing the next byte. When EEW has been set, the CPU is halted for two cycles before the next instruction is executed.

• **Bit 0 – EERE: EEPROM Read Enable**

The EEPROM Read Enable Signal EERE is the read strobe to the EEPROM. When the correct address is set up in the EEAR Register, the EERE bit must be written to a logic one to trigger the EEPROM read. The EEPROM read access takes one instruction, and the requested data is available immediately. When the EEPROM is read, the CPU is halted for four cycles before the next instruction is executed.

The user should poll the EEW bit before starting the read operation. If a write operation is in progress, it is neither possible to read the EEPROM, nor to change the EEAR Register.

The calibrated Oscillator is used to time the EEPROM accesses. Table 1 lists the typical programming time for EEPROM access from the CPU.

**Table 1. EEPROM Programming Time**

Symbol	Number of Calibrated RC Oscillator Cycles <sup>(1)</sup>	Typ Programming Time
EEPROM Write (from CPU)	8448	8.4 ms

Note: 1. Uses 1 MHz clock, independent of CKSEL Fuse settings.



The following code examples show one assembly and one C function for writing to the EEPROM. The examples assume that interrupts are controlled (e.g., by disabling interrupts globally) so that no interrupts will occur during execution of these functions. The examples also assume that no Flash Boot Loader is present in the software. If such code is present, the EEPROM write function must also wait for any ongoing SPM command to finish.

#### Assembly Code Example

```

EEPROM_write
; Wait for completion of previous write
sbic EECR,EEWE
rjmp EEPROM_write
; Set up address (r18:r17) in address register
out EEARH,r18
out EEARL,r17
; Write data (r16) to Data Register
out EEDR,r16
; Write logical one to EEMWE
sbi EECR,EEMWE
; Start eeprom write by setting EEWE
sbi EECR,EEWE
ret
    
```

#### C Code Example

```

void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
    /* Wait for completion of previous write */
    while((EECR & (1<<EEWE))
        ;

    /* Set up Address and Data Registers */
    EEAR = uiAddress;
    EEDR = ucData;
    /* Write logical one to EEMWE */
    EECR |= (1<<EEMWE);
    /* Start eeprom write by setting EEWE */
    EECR |= (1<<EEWE);
}
    
```

The next code examples show assembly and C functions for reading the EEPROM. The examples assume that interrupts are controlled so that no interrupts will occur during execution of these functions.

### Assembly Code Example

```

EEPROM_read:
    ; Wait for completion of previous write
    sbic EECR,EEWE
    rjmp EEPROM_read
    ; Set up address (r18:r17) in Address Register
    out  EEARH,r18
    out  EEARL,r17
    ; Start eeprom read by writing EERE
    sbi  EECR,EERE
    ; Read data from Data Register
    in   r16,EEDR
    ret
    
```

### C Code Example

```

unsigned char EEPROM_read(unsigned int uiAddress)
{
    /* Wait for completion of previous write */
    while(EECR & (1<<EEWE))
        ;
    /* Set up Address Register */
    EEAR = uiAddress;
    /* Start eeprom read by writing EERE */
    EECR |= (1<<EERE);
    /* Return data from Data Register */
    return EEDR;
}
    
```

### PROM Write During Power-down Sleep Mode

When entering Power-down sleep mode while an EEPROM write operation is active, the EEPROM write operation will continue, and will complete before the write access time has passed. However, when the write operation is completed, the Oscillator continues running, and as a consequence, the device does not enter Power-down entirely. It is therefore recommended to verify that the EEPROM write operation is completed before entering Power-down.

### Preventing EEPROM Corruption

During periods of low  $V_{CC}$ , the EEPROM data can be corrupted because the supply voltage is too low for the CPU and the EEPROM to operate properly. These issues are the same as for board level systems using EEPROM and the same design solutions should be applied.

An EEPROM data corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the EEPROM requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage is too low.





EEPROM data corruption can easily be avoided by following this design recommendation:

Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD). If the detection level of the internal BOD does not match the needed detection level, an external low  $V_{CC}$  Reset Protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.

## Memory

The I/O space definition of the ATmega8535 is shown in page 297.

All ATmega8535 I/Os and peripherals are placed in the I/O space. The I/O locations are accessed by the IN and OUT instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions. Refer to the instruction set section for more details. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses.

For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.

Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

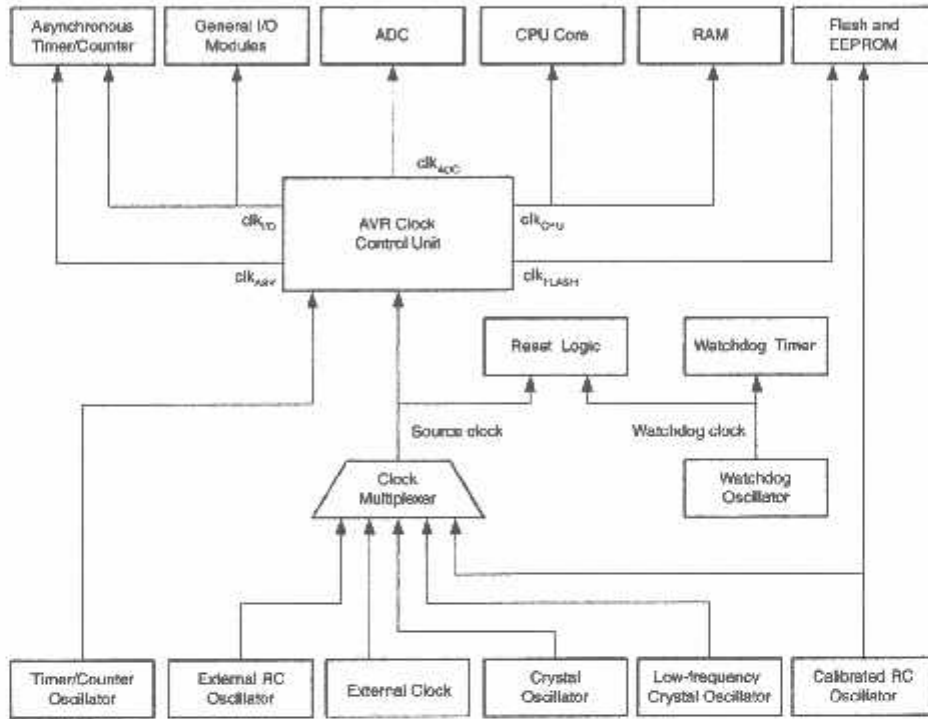
The I/O and peripherals control registers are explained in later sections.

System Clock and Clock Options

Clock Systems and their Distribution

Figure 11 presents the principal clock systems in the AVR and their distribution. All of the clocks need not be active at a given time. In order to reduce power consumption, the clocks to modules not being used can be halted by using different sleep modes, as described in "Power Management and Sleep Modes" on page 31. The clock systems are detailed below.

Figure 11. Clock Distribution



CPU Clock –  $clk_{CPU}$

The CPU clock is routed to parts of the system concerned with operation of the AVR core. Examples of such modules are the General Purpose Register File, the Status Register and the data memory holding the Stack Pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

I/O Clock –  $clk_{I/O}$

The I/O clock is used by the majority of the I/O modules, like Timer/Counters, SPI, and USART. The I/O clock is also used by the External Interrupt module, but note that some external interrupts are detected by asynchronous logic, allowing such interrupts to be detected even if the I/O clock is halted. Also note that address recognition in the TWI module is carried out asynchronously when  $clk_{I/O}$  is halted, enabling TWI address reception in all sleep modes.

Flash Clock –  $clk_{FLASH}$

The Flash clock controls operation of the Flash interface. The Flash clock is usually active simultaneously with the CPU clock.

**Asynchronous Timer Clock – ASY** The Asynchronous Timer clock allows the Asynchronous Timer/Counter to be clocked directly from an external 32 kHz clock crystal. The dedicated clock domain allows using this Timer/Counter as a real-time counter even when the device is in sleep mode.

**ADC Clock – clk<sub>ADC</sub>** The ADC is provided with a dedicated clock domain. This allows halting the CPU and I/O clocks in order to reduce noise generated by digital circuitry. This gives more accurate ADC conversion results.

**Clock Sources** The device has the following clock source options, selectable by Flash Fuse bits as shown below. The clock from the selected source is input to the AVR clock generator, and routed to the appropriate modules.

**Table 2.** Device Clocking Options Select<sup>(1)</sup>

Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

Note: 1. For all fuses "1" means unprogrammed while "0" means programmed.

The various choices for each clocking option is given in the following sections. When the CPU wakes up from Power-down or Power-save, the selected clock source is used to time the start-up, ensuring stable Oscillator operation before instruction execution starts. When the CPU starts from Reset, there is an additional delay allowing the power to reach a stable level before commencing normal operation. The Watchdog Oscillator is used for timing this real-time part of the start-up time. The number of WDT Oscillator cycles used for each time-out is shown in Table 3. The frequency of the Watchdog Oscillator is voltage dependent as shown in "ATmega8535 Typical Characteristics – Preliminary Data" on page 264.

**Table 3.** Number of Watchdog Oscillator Cycles

Typ Time-out (V <sub>CC</sub> = 5.0V)	Typ Time-out (V <sub>CC</sub> = 3.0V)	Number of Cycles
4.1 ms	4.3 ms	4K (4,096)
66 ms	69 ms	64K (65,536)

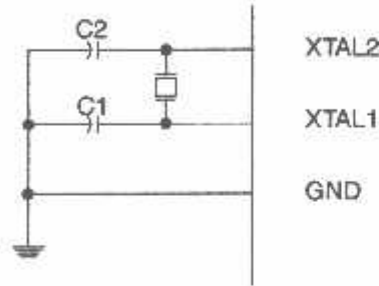
**Default Clock Source** The device is shipped with CKSEL = "0001" and SUT = "10". The default clock source setting is therefore the Internal RC Oscillator with longest startup time. This default setting ensures that all users can make their desired clock source setting using an In-System or Parallel Programmer.

**Crystal Oscillator** XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator, as shown in Figure 12. Either a quartz crystal or a ceramic resonator may be used. The CKOPT Fuse selects between two different oscillator amplifier modes. When CKOPT is programmed, the Oscillator output will oscillate with a full rail-to-rail swing on the output. This mode is suitable when operating in a very noisy environment or when the output from XTAL2 drives a second clock buffer. This mode has a wide frequency range. When CKOPT is unprogrammed, the Oscillator has a smaller output swing. This reduces power consumption considerably.

This mode has a limited frequency range and it can not be used to drive other clock buffers.

For resonators, the maximum frequency is 8 MHz with CKOPT unprogrammed and 16 MHz with CKOPT programmed. C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Some initial guidelines for choosing capacitors for use with crystals are given in Table 4. For ceramic resonators, the capacitor values given by the manufacturer should be used.

**Figure 12.** Crystal Oscillator Connections



The Oscillator can operate in three different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL3..1 as shown in Table 4.

**Table 4.** Crystal Oscillator Operating Modes

CKOPT	CKSEL3..1	Frequency Range <sup>(1)</sup> (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
1	101 <sup>(2)</sup>	0.4 - 0.9	-
1	110	0.9 - 3.0	12 - 22
1	111	3.0 - 8.0	12 - 22
0	101, 110, 111	1.0 - 16.0	12 - 22

- Notes:
1. The frequency ranges are preliminary values.
  2. This option should not be used with crystals, only with ceramic resonators.



The CKSEL0 fuse together with the SUT1..0 Fuses select the start-up times as shown in Table 5.

**Table 5.** Start-up Times for the Crystal Oscillator Clock Selection

CKSEL0	SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset ( $V_{CC} = 5.0V$ )	Recommended Usage
0	00	258 CK <sup>(1)</sup>	4.1 ms	Ceramic resonator, fast rising power
0	01	258 CK <sup>(1)</sup>	65 ms	Ceramic resonator, slowly rising power
0	10	1K CK <sup>(2)</sup>	–	Ceramic resonator, BOD enabled
0	11	1K CK <sup>(2)</sup>	4.1 ms	Ceramic resonator, fast rising power
1	00	1K CK <sup>(2)</sup>	65 ms	Ceramic resonator, slowly rising power
1	01	16K CK	–	Crystal Oscillator, BOD enabled
1	10	16K CK	4.1 ms	Crystal Oscillator, fast rising power
1	11	16K CK	65 ms	Crystal Oscillator, slowly rising power

- Notes:
1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
  2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device, and if frequency stability at start-up is not important for the application.





**Table 7. External RC Oscillator Operating Modes**

CKSEL3..0	Frequency Range (MHz)
0101	- 0.9
0110	0.9 - 3.0
0111	3.0 - 8.0
1000	8.0 - 12.0

When this Oscillator is selected, start-up times are determined by the SUT fuses as shown in Table 8.

**Table 8. Start-up Times for the External RC Oscillator Clock Selection**

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset ( $V_{CC} = 5.0V$ )	Recommended Usage
00	18 CK	–	BOD enabled
01	18 CK	4.1 ms	Fast rising power
10	18 CK	65 ms	Slowly rising power
11	6 CK <sup>(1)</sup>	4.1 ms	Fast rising power or BOD enabled

Note: 1. This option should not be used when operating close to the maximum frequency of the device.

## Calibrated Internal RC Oscillator

The calibrated internal RC Oscillator provides a fixed 1.0, 2.0, 4.0, or 8.0 MHz clock. All frequencies are nominal values at 5V and 25°C. This clock may be selected as the system clock by programming the CKSEL Fuses as shown in Table 9. If selected, it will operate with no external components. The CKOPT Fuse should always be unprogrammed when using this clock option. During Reset, hardware loads the calibration byte into the OSCCAL Register and thereby automatically calibrates the RC Oscillator. At 5V, 25°C and 1.0 MHz Oscillator frequency selected, this calibration gives a frequency within  $\pm 3\%$  of the nominal frequency. Using run-time calibration methods as described in application notes available at [www.atmel.com/avr](http://www.atmel.com/avr) it is possible to achieve  $\pm 1\%$  accuracy at any given  $V_{CC}$  and Temperature. When this Oscillator is used as the chip clock, the Watchdog Oscillator will still be used for the Watchdog Timer and for the Reset Time-out. For more information on the pre-programmed calibration value, see the section "Calibration Byte" on page 237.

**Table 9. Internal Calibrated RC Oscillator Operating Modes**

CKSEL3..0	Nominal Frequency (MHz)
0001 <sup>(1)</sup>	1.0
0010	2.0
0011	4.0
0100	8.0

Note: 1. The device is shipped with this option selected.

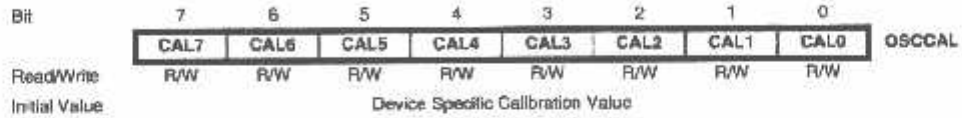
When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in Table 10. XTAL1 and XTAL2 should be left unconnected (NC).

**Table 10.** Start-up Times for the Internal Calibrated RC Oscillator Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset ( $V_{CC} = 5.0V$ )	Recommended Usage
00	6 CK	—	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10 <sup>(1)</sup>	6 CK	65 ms	Slowly rising power
11	Reserved		

Note: 1. The device is shipped with this option selected.

## Oscillator Calibration Register OSCCAL



### • Bits 7..0 – CAL7..0: Oscillator Calibration Value

Writing the calibration byte to this address will trim the Internal Oscillator to remove process variations from the Oscillator frequency. During Reset, the 1 MHz calibration value which is located in the signature row high byte (address 0x00) is automatically loaded into the OSCCAL Register. If the internal RC is used at other frequencies, the calibration values must be loaded manually. This can be done by first reading the signature row by a programmer, and then store the calibration values in the Flash or EEPROM. Then the value can be read by software and loaded into the OSCCAL Register.

When OSCCAL is zero, the lowest available frequency is chosen. Writing non-zero values to this register will increase the frequency of the Internal Oscillator. Writing 0xFF to the register gives the highest available frequency. The calibrated Oscillator is used to time EEPROM and Flash access. If EEPROM or Flash is written, do not calibrate to more than 10% above the nominal frequency. Otherwise, the EEPROM or Flash write may fail. Note that the Oscillator is intended for calibration to 1.0, 2.0, 4.0, or 8.0 MHz. Tuning to other values is not guaranteed, as indicated in Table 11.

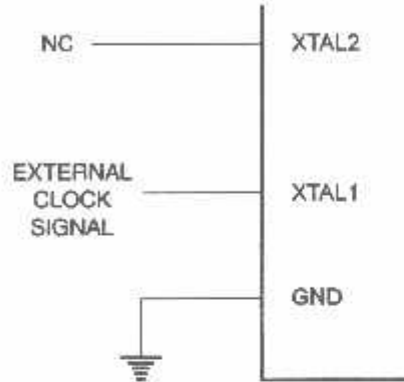
**Table 11.** Internal RC Oscillator Frequency Range.

OSCCAL Value	Min Frequency in Percentage of Nominal Frequency (%)	Max Frequency in Percentage of Nominal Frequency (%)
0x00	50	100
0x7F	75	150
0xFF	100	200

## External Clock

To drive the device from an external clock source, XTAL1 should be driven as shown in Figure 14. To run the device on an external clock, the CKSEL Fuses must be programmed to "0000". By programming the CKOPT Fuse, the user can enable an internal 36 pF capacitor between XTAL1 and GND.

**Figure 14.** External Clock Drive Configuration



When this clock source is selected, start-up times are determined by the SUT Fuses as shown in Table 12.

**Table 12.** Start-up Times for the External Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset ( $V_{CC} = 5.0V$ )	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10	6 CK	65 ms	Slowly rising power
11	Reserved		

When applying an external clock, it is required to avoid sudden changes in the applied clock frequency to ensure stable operation of the MCU. A variation in frequency of more than 2% from one clock cycle to the next can lead to unpredictable behavior. It is required to ensure that the MCU is kept in Reset during such changes in the clock frequency.

## Timer/Counter Oscillator

For AVR microcontrollers with Timer/Counter Oscillator pins (TOSC1 and TOSC2), the crystal is connected directly between the pins. No external capacitors are needed. The Oscillator is optimized for use with a 32.768 kHz watch crystal. Applying an external clock source to TOSC1 is not recommended.

## Power Management and Sleep Modes

Sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application's requirements.

To enter any of the six sleep modes, the SE bit in MCUCR must be written to logic one and a SLEEP instruction must be executed. The SM2, SM1, and SM0 bits in the MCUCR Register select which sleep mode (Idle, ADC Noise Reduction, Power-down, Power-save, Standby, or Extended Standby) will be activated by the SLEEP instruction. See Table 13 for a summary. If an enabled interrupt occurs while the MCU is in a sleep mode, the MCU wakes up. The MCU is then halted for four cycles in addition to the start-up time, it executes the interrupt routine, and resumes execution from the instruction following SLEEP. The contents of the Register File and SRAM are unaltered when the device wakes up from sleep. If a Reset occurs during sleep mode, the MCU wakes up and executes from the Reset Vector.

Figure 11 on page 23 presents the different clock systems in the ATmega8535, and their distribution. The figure is helpful in selecting an appropriate sleep mode.

### MCU Control Register – MCUCR

The MCU Control Register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7, 5, 4 – SM2..0: Sleep Mode Select Bits 2, 1, and 0**

These bits select between the six available sleep modes as shown in Table 13.

**Table 13. Sleep Mode Select**

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby <sup>(1)</sup>
1	1	1	Extended Standby <sup>(1)</sup>

Note: 1. Standby mode and Extended Standby mode are only available with external crystals or resonators.

- **Bit 6 – SE: Sleep Enable**

The SE bit must be written to logic one to make the MCU enter the sleep mode when the SLEEP instruction is executed. To avoid the MCU entering the sleep mode unless it is the programmer's purpose, it is recommended to write the Sleep Enable (SE) bit to one just before the execution of the SLEEP instruction and to clear it immediately after waking up.



### Idle Mode

When the SM2..0 bits are written to 000, the SLEEP instruction makes the MCU enter Idle mode, stopping the CPU but allowing SPI, USART, Analog Comparator, ADC, Two-wire Serial Interface, Timer/Counters, Watchdog, and the interrupt system to continue operating. This sleep mode basically halts  $clk_{CPU}$  and  $clk_{FLASH}$ , while allowing the other clocks to run.

Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the Timer Overflow and USART Transmit Complete interrupts. If wake-up from the Analog Comparator interrupt is not required, the Analog Comparator can be powered down by setting the ADC bit in the Analog Comparator Control and Status register – ACSR. This will reduce power consumption in Idle mode. If the ADC is enabled, a conversion starts automatically when this mode is entered.

### ADC Noise Reduction Mode

When the SM2..0 bits are written to 001, the SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the ADC, the External Interrupts, the Two-wire Serial Interface address watch, Timer/Counter2 and the Watchdog to continue operating (if enabled). This sleep mode basically halts  $clk_{I/O}$ ,  $clk_{CPU}$ , and  $clk_{FLASH}$ , while allowing the other clocks to run.

This improves the noise environment for the ADC, enabling higher resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered. Apart from the ADC Conversion Complete interrupt, only an External Reset, a Watchdog Reset, a Brown-out Reset, a Two-wire Serial Interface address match interrupt, a Timer/Counter2 interrupt, an SPM/EEPROM ready interrupt, an external level interrupt on INT0 or INT1, or an external interrupt on INT2 can wake up the MCU from ADC Noise Reduction mode.

### Power-down Mode

When the SM2..0 bits are written to 010, the SLEEP instruction makes the MCU enter Power-down mode. In this mode, the External Oscillator is stopped, while the External Interrupts, the Two-wire Serial Interface address watch, and the Watchdog continue operating (if enabled). Only an External Reset, a Watchdog Reset, a Brown-out Reset, a Two-wire Serial Interface address match interrupt, an external level interrupt on INT0 or INT1, or an external interrupt on INT2 can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

Note that if a level triggered interrupt is used for wake-up from Power-down mode, the changed level must be held for some time to wake up the MCU. Refer to "External Interrupts" on page 67 for details.

When waking up from Power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL fuses that define the Reset Time-out period, as described in "Clock Sources" on page 24.

### Power-save Mode

When the SM2..0 bits are written to 011, the SLEEP instruction makes the MCU enter Power-save mode. This mode is identical to Power-down, with one exception:

If Timer/Counter2 is clocked asynchronously, i.e., the AS2 bit in ASSR is set, Timer/Counter2 will run during sleep. The device can wake up from either Timer Overflow or Output Compare event from Timer/Counter2 if the corresponding Timer/Counter2 interrupt enable bits are set in TIMSK, and the Global Interrupt Enable bit in SREG is set.

If the asynchronous timer is NOT clocked asynchronously, Power-down mode is recommended instead of Power-save mode because the contents of the registers in the

asynchronous timer should be considered undefined after wake-up in Power-save mode if AS2 is 0.

This sleep mode basically halts all clocks except  $clk_{ASY}$ , allowing operation only of asynchronous modules, including Timer/Counter2 if clocked asynchronously.

## Standby Mode

When the SM2..0 bits are 110 and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Standby mode. This mode is identical to Power-down with the exception that the Oscillator is kept running. From Standby mode, the device wakes up in six clock cycles.

## Extended Standby Mode

When the SM2..0 bits are 111 and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Extended Standby mode. This mode is identical to Power-save mode with the exception that the Oscillator is kept running. From Extended Standby mode, the device wakes up in six clock cycles.

Table 14. Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

Sleep Mode	Active Clock domains					Oscillators		Wake up sources					
	$clk_{CPU}$	$clk_{FLASH}$	$clk_{IO}$	$clk_{ADC}$	$clk_{ASY}$	Main Clock Source Enabled	Timer Osc Enabled	INT2 INT1 INT0	TWI Address Match	Timer 2	SPM/EEPROM Ready	ADC	Other I/O
Power-down			X	X	X	X	X <sup>(2)</sup>	X	X	X	X	X	X
Power-save				X	X	X	X <sup>(2)</sup>	X <sup>(3)</sup>	X	X	X	X	
Standby								X <sup>(3)</sup>	X				
Extended Standby					X <sup>(2)</sup>	X	X <sup>(2)</sup>	X <sup>(3)</sup>	X	X <sup>(2)</sup>			

- Notes:
1. External Crystal or resonator selected as clock source
  2. If AS2 bit in ASSR is set
  3. Only INT2 or level interrupt INT1 and INT0





## Minimizing Power Consumption

There are several issues to consider when trying to minimize the power consumption in an AVR controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as possible of the device's functions are operating. All functions not needed should be disabled. In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption.

### Analog-to-Digital Converter

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion. Refer to "Analog-to-Digital Converter" on page 204 for details on ADC operation.

### Analog Comparator

When entering Idle mode, the Analog Comparator should be disabled if not used. When entering ADC Noise Reduction mode, the Analog Comparator should be disabled. In the other sleep modes, the Analog Comparator is automatically disabled. However, if the Analog Comparator is set up to use the Internal Voltage Reference as input, the Analog Comparator should be disabled in all sleep modes. Otherwise, the Internal Voltage Reference will be enabled, independent of sleep mode. Refer to "Analog Comparator" on page 201 for details on how to configure the Analog Comparator.

### Brown-out Detector

If the Brown-out Detector is not needed in the application, this module should be turned off. If the Brown-out Detector is enabled by the BODEN Fuse, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to "Brown-out Detection" on page 38 for details on how to configure the Brown-out Detector.

### Internal Voltage Reference

The Internal Voltage Reference will be enabled when needed by the Brown-out Detector, the Analog Comparator or the ADC. If these modules are disabled as described in the sections above, the internal voltage reference will be disabled and it will not be consuming power. When turned on again, the user must allow the reference to start up before the output is used. If the reference is kept on in sleep mode, the output can be used immediately. Refer to "Internal Voltage Reference" on page 40 for details on the start-up time.

### Watchdog Timer

If the Watchdog Timer is not needed in the application, this module should be turned off. If the Watchdog Timer is enabled, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to "Watchdog Timer" on page 40 for details on how to configure the Watchdog Timer.

### Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important thing is then to ensure that no pins drive resistive loads. In sleep modes where both the I/O clock ( $clk_{I/O}$ ) and the ADC clock ( $clk_{ADC}$ ) are stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. Refer to the section "Digital Input Enable and Sleep Modes" on page 54 for details on which pins are enabled. If the input buffer is enabled and the input signal is left floating or have an analog signal level close to  $V_{CC}/2$ , the input buffer will use excessive power.

## System Control and Reset

### Resetting the AVR

During Reset, all I/O Registers are set to their initial values, and the program starts execution from the Reset Vector. The instruction placed at the Reset Vector must be an RJMP instruction to the reset handling routine. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa. The circuit diagram in Figure 15 shows the reset logic. Table 15 defines the electrical parameters of the reset circuitry.

The I/O ports of the AVR are immediately reset to their initial state when a reset source goes active. This does not require any clock source to be running.

After all reset sources have gone inactive, a delay counter is invoked, stretching the internal reset. This allows the power to reach a stable level before normal operation starts. The time-out period of the delay counter is defined by the user through the CKSEL Fuses. The different selections for the delay period are presented in "Clock Sources" on page 24.

### Reset Sources

The ATmega8535 has four sources of Reset:

- Power-on Reset. The MCU is reset when the supply voltage is below the Power-on Reset threshold ( $V_{POR}$ ).
- External Reset. The MCU is reset when a low level is present on the  $\overline{RESET}$  pin for longer than the minimum pulse length.
- Watchdog Reset. The MCU is reset when the Watchdog Timer period expires and the Watchdog is enabled.
- Brown-out Reset. The MCU is reset when the supply voltage  $V_{CC}$  is below the Brown-out Reset threshold ( $V_{B0R}$ ) and the Brown-out Detector is enabled.



Figure 15. Reset Logic

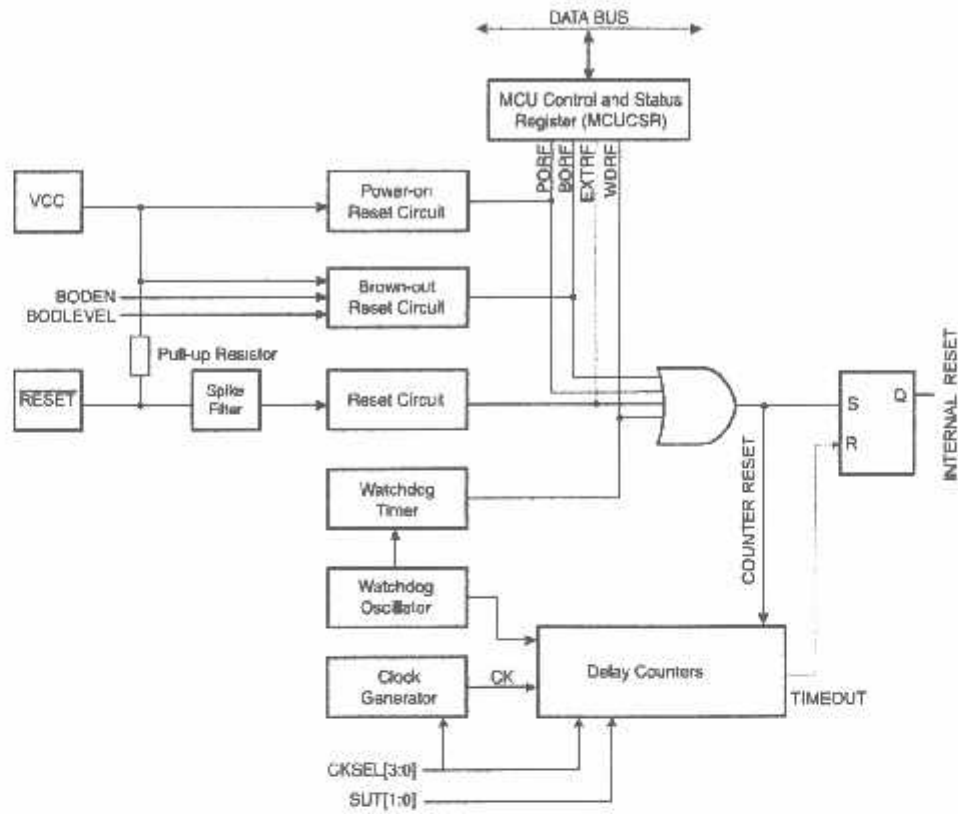


Table 15. Reset Characteristics

Symbol	Parameter	Condition	Min <sup>(1)</sup>	Typ <sup>(1)</sup>	Max <sup>(1)</sup>	Units
$V_{POR}$	Power-on Reset Threshold Voltage (rising)			1.4	2.3	V
	Power-on Reset Threshold Voltage (falling) <sup>(2)</sup>			1.3	2.3	V
$V_{RST}$	RESET Pin Threshold Voltage		0.2		0.9	V
$t_{RST}$	Minimum pulse width on RESET Pin				1.5	$\mu$ s
$V_{BOT}$	Brown-out Reset Threshold Voltage <sup>(3)</sup>	BODLEVEL = 1	2.5	2.7	2.9	V
		BODLEVEL = 0	3.6	4.0	4.2	
$t_{BOD}$	Minimum low voltage period for Brown-out Detection	BODLEVEL = 1		2		$\mu$ s
		BODLEVEL = 0		2		$\mu$ s
$V_{HYST}$	Brown-out Detector hysteresis			130		mV

- Notes: 1. Values are guidelines only.  
 2. The Power-on Reset will not work unless the supply voltage has been below  $V_{POR}$  (falling).  
 3. The Power-on Reset will not work unless the supply voltage has been below  $V_{POR}$  (falling).

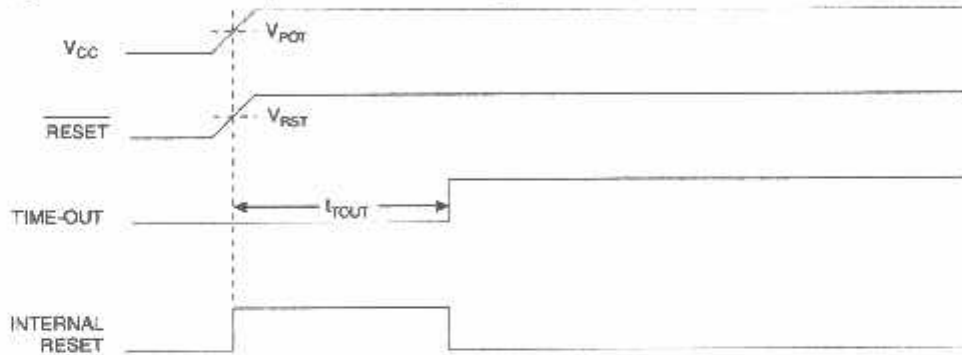
- $V_{BOT}$  may be below nominal minimum operating voltage for some devices. For devices where this is the case, the device is tested down to  $V_{CC} = V_{BOT}$  during the production test. This guarantees that a Brown-out Reset will occur before  $V_{CC}$  drops to a voltage where correct operation of the microcontroller is no longer guaranteed. The test is performed using  $BODLEVEL = 1$  for ATmega8535L and  $BODLEVEL = 0$  for ATmega8535.  $BODLEVEL = 1$  is not applicable for ATmega8535.

## Power-on Reset

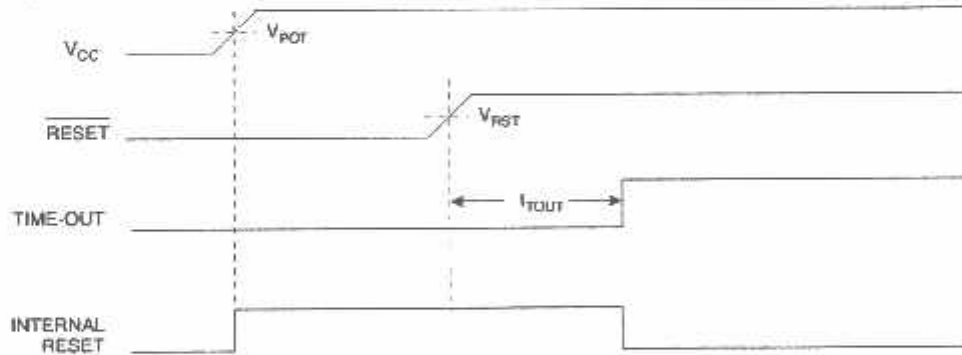
A Power-on Reset (POR) pulse is generated by an On-chip detection circuit. The detection level is defined in Table 15. The POR is activated whenever  $V_{CC}$  is below the detection level. The POR circuit can be used to trigger the Start-up Reset, as well as to detect a failure in supply voltage.

A Power-on Reset (POR) circuit ensures that the device is reset from Power-on. Reaching the Power-on Reset threshold voltage invokes the delay counter, which determines how long the device is kept in RESET after  $V_{CC}$  rise. The RESET signal is activated again, without any delay, when  $V_{CC}$  decreases below the detection level.

**Figure 16.** MCU Start-up,  $\overline{RESET}$  Tied to  $V_{CC}$



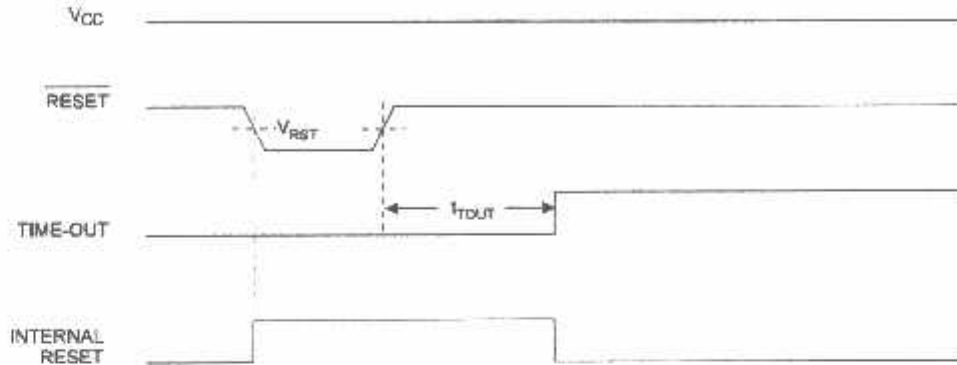
**Figure 17.** MCU Start-up,  $\overline{RESET}$  Extended Externally



### External Reset

An External Reset is generated by a low level on the  $\overline{\text{RESET}}$  pin. Reset pulses longer than the minimum pulse width (see Table 15) will generate a reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset. When the applied signal reaches the Reset Threshold Voltage –  $V_{\text{RST}}$  on its positive edge, the delay counter starts the MCU after the Time-out period  $t_{\text{TOUT}}$  has expired.

**Figure 18.** External Reset During Operation



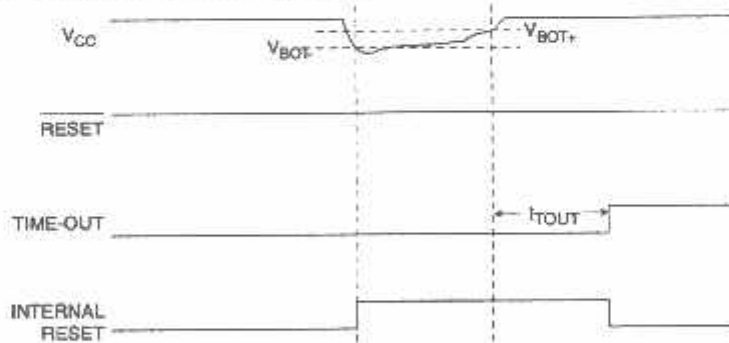
### Brown-out Detection

ATmega8535 has an On-chip Brown-out Detection (BOD) circuit for monitoring the  $V_{\text{CC}}$  level during operation by comparing it to a fixed trigger level. The trigger level for the BOD can be selected by the fuse BODLEVEL to be 2.7V (BODLEVEL unprogrammed), or 4.0V (BODLEVEL programmed). The trigger level has a hysteresis to ensure spike free Brown-out Detection. The hysteresis on the detection level should be interpreted as  $V_{\text{BOT+}} = V_{\text{BOT}} + V_{\text{HYST}}/2$  and  $V_{\text{BOT-}} = V_{\text{BOT}} - V_{\text{HYST}}/2$ .

The BOD circuit can be enabled/disabled by the fuse BODEN. When the BOD is enabled (BODEN programmed), and  $V_{\text{CC}}$  decreases to a value below the trigger level ( $V_{\text{BOT-}}$  in Figure 19), the Brown-out Reset is immediately activated. When  $V_{\text{CC}}$  increases above the trigger level ( $V_{\text{BOT+}}$  in Figure 19), the delay counter starts the MCU after the time-out period  $t_{\text{TOUT}}$  has expired.

The BOD circuit will only detect a drop in  $V_{\text{CC}}$  if the voltage stays below the trigger level for longer than  $t_{\text{BOD}}$  given in Table 15.

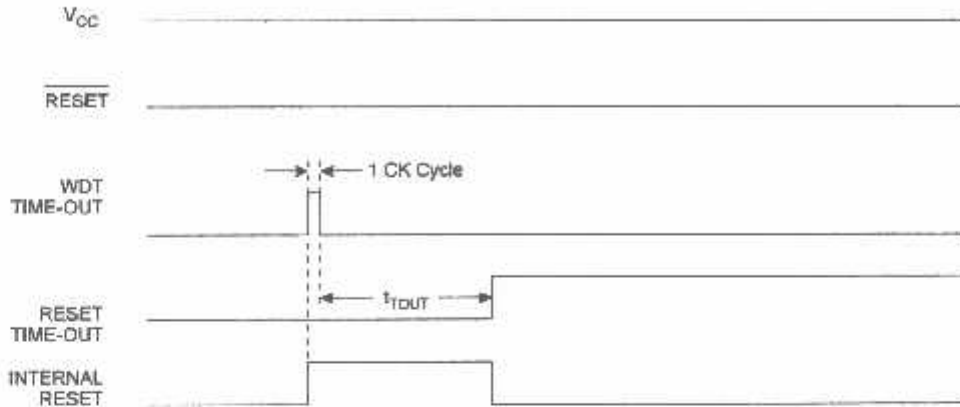
**Figure 19.** Brown-out Reset During Operation



## Watchdog Reset

When the Watchdog times out, it will generate a short reset pulse of one CK cycle duration. On the falling edge of this pulse, the delay timer starts counting the Time-out period  $t_{TOUT}$ . Refer to page 40 for details on operation of the Watchdog Timer.

**Figure 20.** Watchdog Reset During Operation



## MCU Control and Status Register – MCUCSR

The MCU Control and Status Register provides information on which reset source caused an MCU Reset.

Bit	7	6	5	4	3	2	1	0	
	–	ISC2	–	–	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0						See Bit Description

- **Bit 3 – WDRF: Watchdog Reset Flag**

This bit is set if a Watchdog Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 2 – BORF: Brown-out Reset Flag**

This bit is set if a Brown-out Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 1 – EXTRF: External Reset Flag**

This bit is set if an External Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 0 – PORF: Power-on Reset Flag**

This bit is set if a Power-on Reset occurs. The bit is reset only by writing a logic zero to the flag.

To make use of the Reset Flags to identify a reset condition, the user should read and then reset the MCUCSR as early as possible in the program. If the register is cleared before another reset occurs, the source of the reset can be found by examining the Reset Flags.

## Internal Voltage Reference

ATmega8535 features an internal bandgap reference. This reference is used for Brown-out Detection, and it can be used as an input to the Analog Comparator or the ADC. The 2.56V reference to the ADC is generated from the internal bandgap reference.

## Bandgap Reference Enable Conditions and Start-up Time

The voltage reference has a start-up time that may influence the way it should be used. The start-up time is given in Table 16. To save power, the reference is not always turned on. The reference is on during the following situations:

1. When the BOD is enabled (by programming the BODEN Fuse).
2. When the bandgap reference is connected to the Analog Comparator (by setting the ACBG bit in ACSR).
3. When the ADC is enabled.

Thus, when the BOD is not enabled, after setting the ACBG bit or enabling the ADC, the user must always allow the reference to start up before the output from the Analog Comparator or ADC is used. To reduce power consumption in Power-down mode, the user can avoid the three conditions above to ensure that the reference is turned off before entering Power-down mode.

**Table 16. Internal Voltage Reference Characteristics<sup>(1)</sup>**

Symbol	Parameter	Min	Typ	Max	Units
$V_{BG}$	Bandgap reference voltage	1.15	1.23	1.35	V
$t_{BG}$	Bandgap reference start-up time		40	70	$\mu$ s
$I_{BG}$	Bandgap reference current consumption		10		$\mu$ A

Note: 1. Values are guidelines only.

## Watchdog Timer

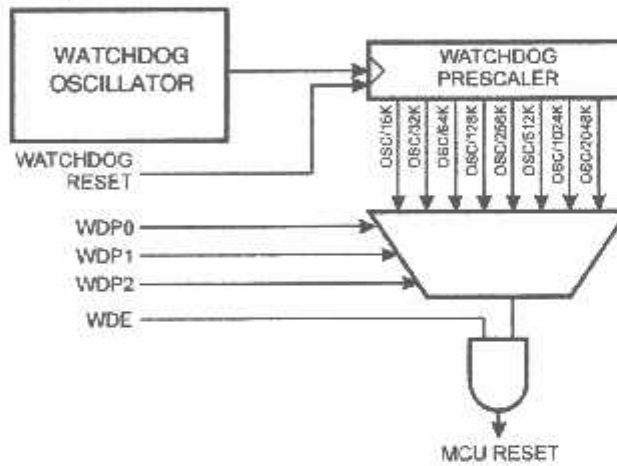
The Watchdog Timer is clocked from a separate On-chip Oscillator which runs at 1 MHz. This is the typical value at  $V_{CC} = 5V$ . See characterization data for typical values at other  $V_{CC}$  levels. By controlling the Watchdog Timer prescaler, the Watchdog Reset interval can be adjusted as shown in Table 18 on page 42. The WDR – Watchdog Reset – instruction resets the Watchdog Timer. The Watchdog Timer is also reset when it is disabled and when a Chip Reset occurs. Eight different clock cycle periods can be selected to determine the reset period. If the reset period expires without another Watchdog Reset, the ATmega8535 resets and executes from the Reset Vector. For timing details on the Watchdog Reset, refer to page 39.

To prevent unintentional disabling of the Watchdog or unintentional change of Time-out period, three different safety levels are selected by the Fuses S8535C and WDTON as shown in Table 17. Safety level 0 corresponds to the setting in AT90S8535. There is no restriction on enabling the WDT in any of the safety levels.

**Table 17.** WDT Configuration as a Function of the Fuse Settings of M103C and WDTON

S8535C	WDTON	Safety Level	WDT Initial State	How to Disable the WDT	How to Change Time-out
Unprogrammed	Unprogrammed	1	Disabled	Timed sequence	Timed sequence
Unprogrammed	Programmed	2	Enabled	Always enabled	Timed sequence
Programmed	Unprogrammed	0	Disabled	Timed sequence	No restriction
Programmed	Programmed	2	Enabled	Always enabled	Timed sequence

**Figure 21.** Watchdog Timer



## Watchdog Timer Control Register – WDTCSR

Bit	7	6	5	4	3	2	1	0	WDTCSR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..5 – Res: Reserved Bits**

These bits are reserved bits in the ATmega8535 and will always read as zero.

- **Bit 4 – WDCE: Watchdog Change Enable**

This bit must be set when the WDE bit is written to logic zero. Otherwise, the Watchdog will not be disabled. Once written to one, hardware will clear this bit after four clock cycles. Refer to the description of the WDE bit for a Watchdog disable procedure. In Safety Level 1 and 2, this bit must also be set when changing the prescaler bits. See “Timed Sequences for Changing the Configuration of the Watchdog Timer” on page 44.

- **Bit 3 – WDE: Watchdog Enable**

When the WDE is written to logic one, the Watchdog Timer is enabled, and if the WDE is written to logic zero, the Watchdog Timer function is disabled. WDE can only be cleared



If the WDCE bit has logic level one. To disable an enabled Watchdog Timer, the following procedure must be followed:

1. In the same operation, write a logic one to WDCE and WDE. A logic one must be written to WDE even though it is set to one before the disable operation starts.
2. Within the next four clock cycles, write a logic 0 to WDE. This disables the watchdog.

In safety level 2, it is not possible to disable the Watchdog Timer, even with the algorithm described above. See "Timed Sequences for Changing the Configuration of the Watchdog Timer" on page 44.

• **Bits 2..0 – WDP2, WDP1, WDP0: Watchdog Timer Prescaler 2, 1, and 0**

The WDP2, WDP1, and WDP0 bits determine the Watchdog Timer prescaling when the Watchdog Timer is enabled. The different prescaling values and their corresponding Timeout Periods are shown in Table 18.

**Table 18.** Watchdog Timer Prescale Select<sup>(1)</sup>

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V <sub>CC</sub> = 3.0V	Typical Time-out at V <sub>CC</sub> = 5.0V
0	0	0	16K (16,384)	17.1 ms	16.3 ms
0	0	1	32K (32,768)	34.3 ms	32.5 ms
0	1	0	64K (65,536)	68.5 ms	65 ms
0	1	1	128K (131,072)	0.14 s	0.13 s
1	0	0	256K (262,144)	0.27 s	0.26 s
1	0	1	512K (524,288)	0.55 s	0.52 s
1	1	0	1,024K (1,048,576)	1.1 s	1.0 s
1	1	1	2,048K (2,097,152)	2.2 s	2.1 s

Note: 1. Values are guidelines only.

The following code example shows one assembly and one C function for turning off the WDT. The example assumes that interrupts are controlled (e.g., by disabling interrupts globally) so that no interrupts will occur during execution of these functions.

Assembly Code Example
<pre> WDT_off: ; Reset WDT wdr ; Write logical one to WDCE and WDE in r16, WDTCR ori r16, (1&lt;&lt;WDCE) (1&lt;&lt;WDE) out WDTCR, r16 ; Turn off WDT ldi r16, (0&lt;&lt;WDE) out WDTCR, r16 ret         </pre>
C Code Example
<pre> void WDT_off(void) {     /* Reset WDT */     _WDR();     /* Write logical one to WDCE and WDE */     WDTCR  = (1&lt;&lt;WDCE)   (1&lt;&lt;WDE);     /* Turn off WDT */     WDTCR = 0x00; }         </pre>



## Interrupts

This section describes the specifics of the Interrupt handling as performed in ATmega8535. For a general explanation of the AVR interrupt handling, refer to "Reset and Interrupt Handling" on page 12.

### Interrupt Vectors in ATmega8535

**Table 19.** Reset and Interrupt Vectors

Vector No.	Program Address <sup>(2)</sup>	Source	Interrupt Definition
1	0x000 <sup>(1)</sup>	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	INT2	External Interrupt Request 2
20	0x013	TIMER0 COMP	Timer/Counter0 Compare Match
21	0x014	SPM_RDY	Store Program Memory Ready

- Notes:
1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see "Boot Loader Support – Read-While-Write Self-Programming" on page 222.
  2. When the IVSEL bit in GICR is set, Interrupt Vectors will be moved to the start of the Boot Flash section. The address of each Interrupt Vector will then be the address in this table added to the start address of the Boot Flash section.

Table 20 shows reset and Interrupt Vectors placement for the various combinations of BOOTRST and IVSEL settings. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa.

**Table 20.** Reset and Interrupt Vectors Placement

BOOTRST <sup>(1)</sup>	IVSEL	Reset Address	Interrupt Vectors Start Address
1	0	0x0000	0x0001
1	1	0x0000	Boot Reset Address + 0x0001
0	0	Boot Reset Address	0x0001
0	1	Boot Reset Address	Boot Reset Address + 0x0001

Note: 1. The Boot Reset Address is shown in Table 93 on page 233. For the BOOTRST Fuse "1" means unprogrammed while "0" means programmed.

The most typical and general program setup for the Reset and Interrupt Vector Addresses in ATmega8535 is:

```

AddressLabels Code Comments
0x000 rjmp RESET ; Reset Handler
0x001 rjmp EXT_INT0 ; IRQ0 Handler
0x002 rjmp EXT_INT1 ; IRQ1 Handler
0x003 rjmp TIM2_COMP ; Timer2 Compare Handler
0x004 rjmp TIM2_OVF ; Timer2 Overflow Handler
0x005 rjmp TIM1_CAPT ; Timer1 Capture Handler
0x006 rjmp TIM1_COMPA ; Timer1 Compare A Handler
0x007 rjmp TIM1_COMPB ; Timer1 Compare B Handler
0x008 rjmp TIM1_OVF ; Timer1 Overflow Handler
0x009 rjmp TIM0_OVF ; Timer0 Overflow Handler
0x00A rjmp SPI_STC ; SPI Transfer Complete Handler
0x00B rjmp USART_RXC ; USART RX Complete Handler
0x00C rjmp USART_UDRE ; UDR Empty Handler
0x00D rjmp USART_TXC ; USART TX Complete Handler
0x00E rjmp ADC ; ADC Conversion Complete Handler
0x00F rjmp EE_RDY ; EEPROM Ready Handler
0x010 rjmp ANA_COMP ; Analog Comparator Handler
0x011 rjmp TWSI ; Two-wire Serial Interface Handler
0x012 rjmp EXT_INT2 ; IRQ2 Handler
0x013 rjmp TIM0_COMP ; Timer0 Compare Handler
0x014 rjmp SPM_RDY ; Store Program Memory Ready Handler
;
0x015 RESET: ldi r16,high(RAMEND) ; Main program start
0x016 out SPH,r16 ; Set Stack Pointer to top of RAM
0x017 ldi r16,low(RAMEND)
0x018 out SPL,r16
0x019 sei ; Enable interrupts
0x020 <instr> xxx
... ..

```

When the BOOTRST Fuse is unprogrammed, the Boot section size set to 2K bytes and the IVSEL bit in the GICR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

AddressLabels	Code	Comments
0x000 RESET:	ldi r16,high(RAMEND)	; Main program start
0x001	out SPH,r16	; Set Stack Pointer to top of RAM
0x002	ldi r16,low(RAMEND)	
0x003	out SPL,r16	
0x004	sei	; Enable interrupts
0x005	<instr> xxx	
;		
.org 0xC01		
0xC01	rjmp EXT_INT0	; IRQ0 Handler
0xC02	rjmp EXT_INT1	; IRQ1 Handler
... ..	..	;
0xC14	rjmp SPM_RDY	; Store Program Memory Ready Handler

When the BOOTRST Fuse is programmed and the Boot section size set to 2K bytes, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

AddressLabels	Code	Comments
.org 0x001		
0x001	rjmp EXT_INT0	; IRQ0 Handler
0x002	rjmp EXT_INT1	; IRQ1 Handler
... ..	..	;
0x014	rjmp SPM_RDY	; Store Program Memory Ready Handler
;		
.org 0xC00		
0xC00 RESET:	ldi r16,high(RAMEND)	; Main program start
0xC01	out SPH,r16	; Set Stack Pointer to top of RAM
0xC02	ldi r16,low(RAMEND)	
0xC03	out SPL,r16	
0xC04	sei	; Enable interrupts
0xC05	<instr> xxx	

When the BOOTRST Fuse is programmed, the Boot section size set to 2K bytes and the IVSEL bit in the GICR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

AddressLabels	Code	Comments
.org 0xC00		
0xC00	rjmp RESET	; Reset handler
0xC01	rjmp EXT_INT0	; IRQ0 Handler
0xC02	rjmp EXT_INT1	; IRQ1 Handler
... ..	..	;
0xC14	rjmp SPM_RDY	; Store Program Memory Ready Handler
;		
0xC15 RESET:	ldi r16,high(RAMEND)	; Main program start
0xC16	out SPH,r16	; Set Stack Pointer to top of RAM
0xC17	ldi r16,low(RAMEND)	
0xC18	out SPL,r16	
0xC19	sei	; Enable interrupts
0xC20	<instr> xxx	



## Placing Interrupts Between Application and Boot Space

### General Interrupt Control Register – GICR

The General Interrupt Control Register controls the placement of the Interrupt Vector table.

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

#### • Bit 1 – IVSEL: Interrupt Vector Select

When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash section is determined by the BOOTSZ Fuses. Refer to the section “Boot Loader Support – Read-While-Write Self-Programming” on page 222 for details. To avoid unintentional changes of Interrupt Vector tables, a special write procedure must be followed to change the IVSEL bit:

1. Write the Interrupt Vector Change Enable (IVCE) bit to one.
2. Within four cycles, write the desired value to IVSEL while writing a zero to IVCE.

Interrupts will automatically be disabled while this sequence is executed. Interrupts are disabled in the cycle IVCE is set, and they remain disabled until after the instruction following the write to IVSEL. If IVSEL is not written, interrupts remain disabled for four cycles. The I-bit in the Status Register is unaffected by the automatic disabling.

**Note:** If Interrupt Vectors are placed in the Boot Loader section and Boot Lock bit BLB02 is programmed, interrupts are disabled while executing from the Application section. If Interrupt Vectors are placed in the Application section and Boot Lock bit BLB12 is programmed, interrupts are disabled while executing from the Boot Loader section. Refer to the section “Boot Loader Support – Read-While-Write Self-Programming” on page 222 for details on Boot Lock bits.

#### • Bit 0 – IVCE: Interrupt Vector Change Enable

The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts, as explained in the IVSEL description above. See Code Example below.

## Assembly Code Example

```
Move_interrupts:
; Enable change of interrupt vectors
ldi r16, (1<<IVCE)
out GICR, r16
; Move interrupts to boot Flash section
ldi r16, (1<<IVSEL)
out GICR, r16
ret
```

## C Code Example

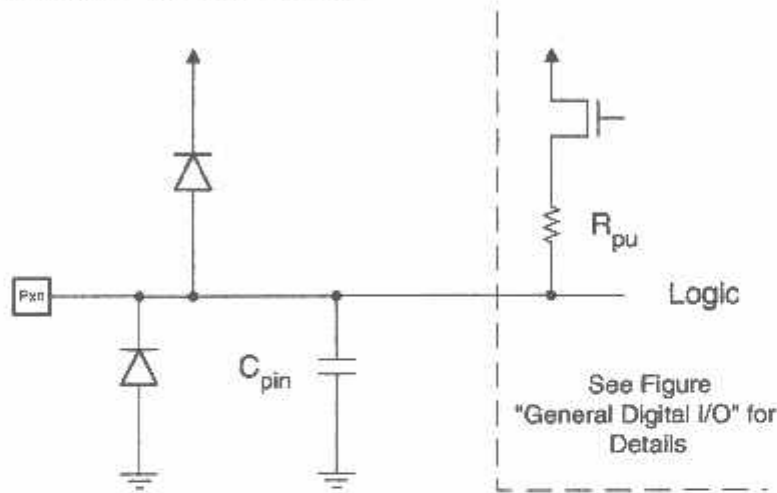
```
void Move_interrupts(void)
{
/* Enable change of interrupt vectors */
GICR = (1<<IVCE);
/* Move interrupts to boot Flash section */
GICR = (1<<IVSEL);
}
```

## -Ports

### Introduction

All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions. The same applies when changing drive value (if configured as output) or enabling/disabling of pull-up resistors (if configured as input). Each output buffer has symmetrical drive characteristics with both high sink and source capability. The pin driver is strong enough to drive LED displays directly. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both  $V_{CC}$  and Ground as indicated in Figure 22. Refer to "Electrical Characteristics" on page 253 for a complete list of parameters.

**Figure 22.** I/O Pin Equivalent Schematic



All registers and bit references in this section are written in general form. A lower case "x" represents the numbering letter for the port, and a lower case "n" represents the bit number. However, when using the register or bit defines in a program, the precise form must be used. For example, PORTB3 for bit no. 3 in Port B, here documented generally as PORTxn. The physical I/O Registers and bit locations are listed in "Register Description for I/O-Ports" on page 65.

Three I/O memory address locations are allocated for each port, one each for the Data Register – PORTx, Data Direction Register – DDRx, and the Port Input Pins – PINx. The Port Input Pins I/O location is read only, while the Data Register and the Data Direction Register are read/write. In addition, the Pull-up Disable – PUD bit in SFIOR disables the pull-up function for all pins in all ports when set.

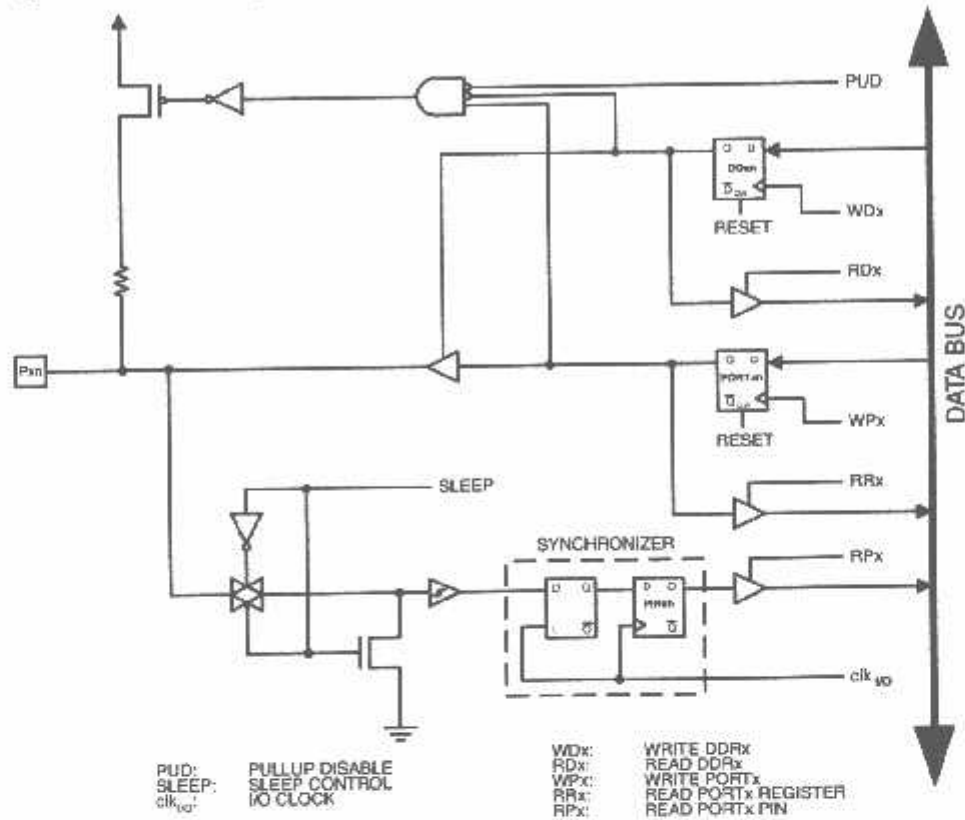
Using the I/O port as General Digital I/O is described in "Ports as General Digital I/O" on page 51. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in "Alternate Port Functions" on page 56. Refer to the individual module sections for a full description of the alternate functions.

Note that enabling the alternate function of some of the port pins does not affect the use of the other pins in the port as general digital I/O.

## Ports as General Digital

The ports are bi-directional I/O ports with optional internal pull-ups. Figure 23 shows a functional description of one I/O-port pin, here generically called Pxn.

**Figure 23.** General Digital I/O<sup>(1)</sup>



Note: 1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. clk<sub>IO</sub>, SLEEP, and PUD are common to all ports.

## Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in "Register Description for I/O-Ports" on page 65, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

If PORTxn is written a logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when a reset condition becomes active, even if no clocks are running.

If PORTxn is written a logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written a logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

When switching between tri-state ((DDxn, PORTxn) = 0b00) and output high ((DDxn, PORTxn) = 0b11), an intermediate state with either pull-up enabled ((DDxn, PORTxn) =

0b01) or output low ( $\{DDxn, PORTxn\} = 0b10$ ) must occur. Normally, the pull-up enabled state is fully acceptable, as a high-impedant environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the SFIOR Register can be set to disable all pull-ups in all ports.

Switching between input with pull-up and output low generates the same problem. The user must use either the tri-state ( $\{DDxn, PORTxn\} = 0b00$ ) or the output high state ( $\{DDxn, PORTxn\} = 0b10$ ) as an intermediate step.

Table 21 summarizes the control signals for the pin value.

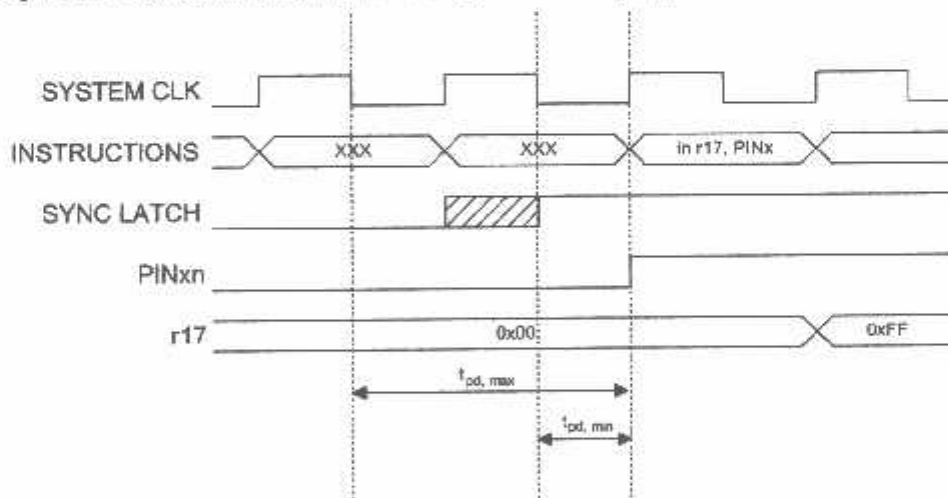
**Table 21.** Port Pin Configurations

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

### ding the Pin Value

Independent of the setting of Data Direction bit DDxn, the port pin can be read through the PINxn Register bit. As shown in Figure 23, the PINxn Register bit and the preceding latch constitute a synchronizer. This is needed to avoid metastability if the physical pin changes value near the edge of the internal clock, but it also introduces a delay. Figure 24 shows a timing diagram of the synchronization when reading an externally applied pin value. The maximum and minimum propagation delays are denoted  $t_{pd,max}$  and  $t_{pd,min}$  respectively.

**Figure 24.** Synchronization when Reading an Externally Applied Pin Value



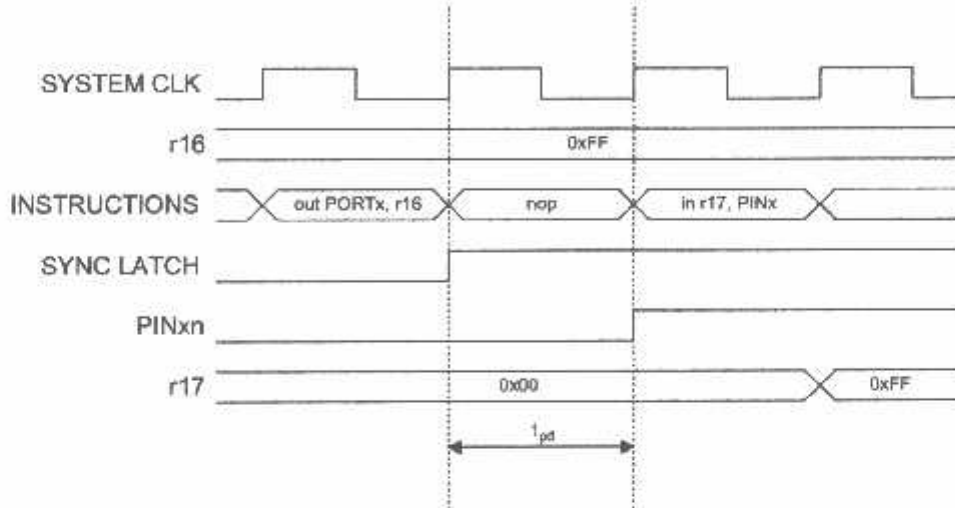
Consider the clock period starting shortly *after* the first falling edge of the system clock. The latch is closed when the clock is low, and goes transparent when the clock is high, as indicated by the shaded region of the "SYNC LATCH" signal. The signal value is latched when the system clock goes low. It is clocked into the PINxn Register at the suc-



ceeding positive clock edge. As indicated by the two arrows  $t_{pd,max}$  and  $t_{pd,min}$ , a single signal transition on the pin will be delayed between  $\frac{1}{2}$  and  $1\frac{1}{2}$  system clock period depending upon the time of assertion.

When reading back a software assigned pin value, a *nop* instruction must be inserted as indicated in Figure 25. The *out* instruction sets the "SYNC LATCH" signal at the positive edge of the clock. In this case, the delay  $t_{pd}$  through the synchronizer is one system clock period.

**Figure 25. Synchronization when Reading a Software Assigned Pin Value**



The following code example shows how to set port B pins 0 and 1 high, 2 and 3 low, and define the port pins from 4 to 7 as input with pull-ups assigned to port pins 6 and 7. The resulting pin values are read back again, but as previously discussed, a *nop* instruction is included to be able to read back the value recently assigned to some of the pins.

Assembly Code Example <sup>(1)</sup>
<pre> ... ; Define pull-ups and set outputs high ; Define directions for port pins ldi r16,(1&lt;&lt;PB7) (1&lt;&lt;PB6) (1&lt;&lt;PB1) (1&lt;&lt;PB0) ldi r17,(1&lt;&lt;DDB3) (1&lt;&lt;DDB2) (1&lt;&lt;DDB1) (1&lt;&lt;DDB0) out PORTB,r16 out DDRB,r17 ; Insert nop for synchronization nop ; Read port pins in r16,PINB ... </pre>
C Code Example
<pre> unsigned char i; ... /* Define pull-ups and set outputs high */ /* Define directions for port pins */ PORTB = (1&lt;&lt;PB7) (1&lt;&lt;PB6) (1&lt;&lt;PB1) (1&lt;&lt;PB0); DDRB = (1&lt;&lt;DDB3) (1&lt;&lt;DDB2) (1&lt;&lt;DDB1) (1&lt;&lt;DDB0); /* Insert nop for synchronization */ _NOP(); /* Read port pins */ i = PINB; ... </pre>

Note: 1. For the assembly program, two temporary registers are used to minimize the time from pull-ups are set on pins 0, 1, 6, and 7, until the direction bits are correctly set, defining bits 2 and 3 as low and redefining bits 0 and 1 as strong high drivers.

### Input Enable and Sleep

As shown in Figure 23, the digital input signal can be clamped to ground at the input of the Schmitt Trigger. The signal denoted SLEEP in the figure, is set by the MCU sleep controller in Power-down mode, Power-save mode, Standby mode, and Extended Standby mode to avoid high power consumption if some input signals are left floating, or have an analog signal level close to  $V_{CC}/2$ .

SLEEP is overridden for port pins enabled as External Interrupt pins. If the External Interrupt Request is not enabled, SLEEP is active also for these pins. SLEEP is also overridden by various other alternate functions as described in "Alternate Port Functions" on page 56.

If a logic high level ("one") is present on an Asynchronous External Interrupt pin configured as "Interrupt on Rising Edge, Falling Edge, or Any Logic Change on Pin" while the external interrupt is *not* enabled, the corresponding External Interrupt Flag will be set

when resuming from the above mentioned sleep modes, as the clamping in these sleep modes produces the requested logic change.

**Unconnected pins**

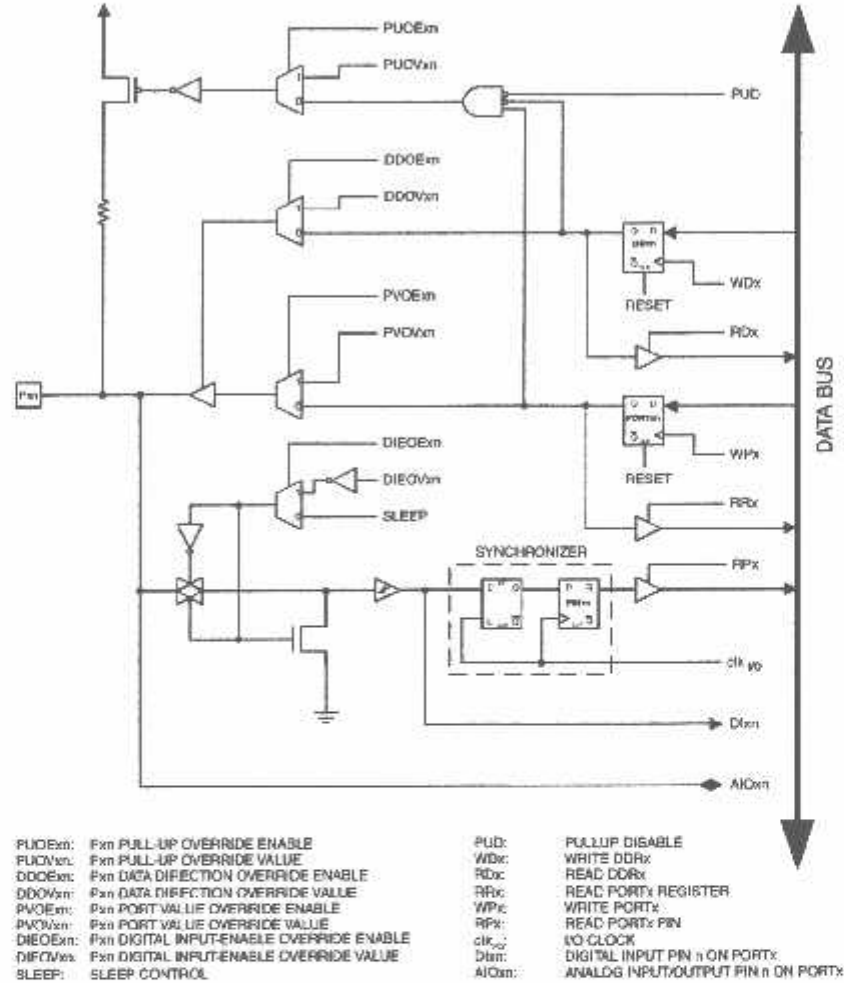
If some pins are unused, it is recommended to ensure that these pins have a defined level. Even though most of the digital inputs are disabled in the deep sleep modes as described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled (Reset, Active mode and Idle mode).

The simplest method to ensure a defined level of an unused pin, is to enable the internal pull-up. In this case, the pull-up will be disabled during reset. If low power consumption during reset is important, it is recommended to use an external pull-up or pull-down. Connecting unused pins directly to  $V_{CC}$  or GND is not recommended, since this may cause excessive currents if the pin is accidentally configured as an output.

### Alternate Port Functions

Most port pins have alternate functions in addition to being general digital I/Os. Figure 26 shows how the port pin control signals from the simplified Figure 23 can be overridden by alternate functions. The overriding signals may not be present in all port pins, but the figure serves as a generic description applicable to all port pins in the AVR microcontroller family.

Figure 26. Alternate Port Functions<sup>(1)</sup>



Note: 1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. clk<sub>IO</sub>, SLEEP, and PUD are common to all ports. All other signals are unique for each pin.

Table 22 summarizes the function of the overriding signals. The pin and port indexes from Figure 26 are not shown in the succeeding tables. The overriding signals are generated internally in the modules having the alternate function.

**Table 22.** Generic Description of Overriding Signals for Alternate Functions

Signal Name	Full Name	Description
PUOE	Pull-up Override Enable	If this signal is set, the pull-up enable is controlled by the PUOV signal. If this signal is cleared, the pull-up is enabled when {DDxn, PORTxn, PUD} = 0b010.
PUOV	Pull-up Override Value	If PUOE is set, the pull-up is enabled/disabled when PUOV is set/cleared, regardless of the setting of the DDxn, PORTxn, and PUD Register bits.
DDOE	Data Direction Override Enable	If this signal is set, the Output Driver Enable is controlled by the DDOV signal. If this signal is cleared, the Output driver is enabled by the DDxn Register bit.
DDOV	Data Direction Override Value	If DDOE is set, the Output Driver is enabled/disabled when DDOV is set/cleared, regardless of the setting of the DDxn Register bit.
PVOE	Port Value Override Enable	If this signal is set and the Output Driver is enabled, the port value is controlled by the PVOV signal. If PVOE is cleared, and the Output Driver is enabled, the port Value is controlled by the PORTxn Register bit.
PVOV	Port Value Override Value	If PVOE is set, the port value is set to PVOV, regardless of the setting of the PORTxn Register bit.
DIEOE	Digital Input Enable Override Enable	If this bit is set, the Digital Input Enable is controlled by the DIEOV signal. If this signal is cleared, the Digital Input Enable is determined by MCU-state (Normal mode, sleep modes).
DIEOV	Digital Input Enable Override Value	If DIEOE is set, the Digital Input is enabled/disabled when DIEOV is set/cleared, regardless of the MCU state (Normal mode, sleep modes).
DI	Digital Input	This is the Digital Input to alternate functions. In the figure, the signal is connected to the output of the schmitt trigger but before the synchronizer. Unless the Digital Input is used as a clock source, the module with the alternate function will use its own synchronizer.
AIO	Analog Input/output	This is the Analog Input/Output to/from alternate functions. The signal is connected directly to the pad, and can be used bi-directionally.

The following subsections shortly describe the alternate functions for each port, and relate the overriding signals to the alternate function. Refer to the alternate function description for further details.

Special Function IO Register –  
SFIOA

Bit	7	6	5	4	3	2	1	0	SFIOA
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 2 – PUD: Pull-up disable

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See “Configuring the Pin” on page 51 for more details about this feature.

Alternate Functions of Port A

Port A has an alternate function as analog input for the ADC as shown in Table 23. If some Port A pins are configured as outputs, it is essential that these do not switch when a conversion is in progress. This might corrupt the result of the conversion.

Table 23. Port A Pins Alternate Functions

Port Pin	Alternate Function
PA7	ADC7 (ADC input channel 7)
PA6	ADC6 (ADC input channel 6)
PA5	ADC5 (ADC input channel 5)
PA4	ADC4 (ADC input channel 4)
PA3	ADC3 (ADC input channel 3)
PA2	ADC2 (ADC input channel 2)
PA1	ADC1 (ADC input channel 1)
PA0	ADC0 (ADC input channel 0)

Table 24 and Table 25 relate the alternate functions of Port A to the overriding signals shown in Figure 26 on page 56.

Table 24. Overriding Signals for Alternate Functions in PA7..PA4

Signal Name	PA7/ADC7	PA6/ADC6	PA5/ADC5	PA4/ADC4
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	ADC7 INPUT	ADC6 INPUT	ADC5 INPUT	ADC4 INPUT

**Table 25.** Overriding Signals for Alternate Functions in PA3..PA0

Signal Name	PA3/ADC3	PA2/ADC2	PA1/ADC1	PA0/ADC0
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	-	-	-	-
AIO	ADC3 INPUT	ADC2 INPUT	ADC1 INPUT	ADC0 INPUT

**Alternate Functions Of Port B**

The Port B pins with alternate functions are shown in Table 26.

**Table 26.** Port B Pins Alternate Functions

Port Pin	Alternate Functions
PB7	SCK (SPI Bus Serial Clock)
PB6	MISO (SPI Bus Master Input/Slave Output)
PB5	MOSI (SPI Bus Master Output/Slave Input)
PB4	$\overline{SS}$ (SPI Slave Select Input)
PB3	AIN1 (Analog Comparator Negative Input) OC0 (Timer/Counter0 Output Compare Match Output)
PB2	AIN0 (Analog Comparator Positive Input) INT2 (External Interrupt 2 Input)
PB1	T1 (Timer/Counter1 External Counter Input)
PB0	T0 (Timer/Counter0 External Counter Input) XCK (USART External Clock Input/Output)

The alternate pin configuration is as follows:

- **SCK – Port B, Bit 7**

SCK: Master Clock output, Slave Clock input pin for SPI channel. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB7. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB7. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB7 bit.

- **MISO – Port B, Bit 6**

MISO: Master Data Input, Slave Data output pin for SPI channel. When the SPI is enabled as a Master, this pin is configured as an input regardless of the setting of DDB6. When the SPI is enabled as a Slave, the data direction of this pin is controlled by DDB6. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB6 bit.



- **MOSI – Port B, Bit 5**

MOSI: SPI Master Data output, Slave Data input for SPI channel. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB5. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB5. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB5 bit.

- **$\overline{SS}$  – Port B, Bit 4**

$\overline{SS}$ : Slave Select input. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB4. As a Slave, the SPI is activated when this pin is driven low. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB4. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB4 bit.

- **AIN1/OC0 – Port B, Bit 3**

AIN1, Analog Comparator Negative input. Configure the port pin as input with the internal pull-up switched off to avoid the digital port function from interfering with the function of the Analog Comparator.

OC0, Output Compare Match output: The PB3 pin can serve as an external output for the Timer/Counter0 Compare Match. The PB3 pin has to be configured as an output (DDB3 set (one)) to serve this function. The OC0 pin is also the output pin for the PWM mode timer function.

- **AIN0/INT2 – Port B, Bit 2**

AIN0, Analog Comparator Positive Input. Configure the port pin as input with the internal pull-up switched off to avoid the digital port function from interfering with the function of the Analog Comparator.

INT2, External Interrupt Source 2: The PB2 pin can serve as an external interrupt source to the MCU.

- **T1 – Port B, Bit 1**

T1, Timer/Counter1 Counter Source.

- **T0/XCK – Port B, Bit 0**

T0, Timer/Counter0 Counter Source.

XCK, USART External Clock. The Data Direction Register (DDB0) controls whether the clock is output (DDB0 set) or input (DDB0 cleared). The XCK pin is active only when the USART operates in synchronous mode.

Table 27 and Table 28 relate the alternate functions of Port B to the overriding signals shown in Figure 26 on page 56. SPI MSTR INPUT and SPI SLAVE OUTPUT constitute the MISO signal, while MOSI is divided into SPI MSTR OUTPUT and SPI SLAVE INPUT.



**Table 27.** Overriding Signals for Alternate Functions in PB7..PB4

Signal Name	PB7/SCK	PB6/MISO	PB5/MOSI	PB4/SS
PUOE	SPE • MSTR	SPE • MSTR	SPE • MSTR	SPE • MSTR
PUOV	PORTB7 • PUD	PORTB6 • PUD	PORTB5 • PUD	PORTB4 • PUD
DDOE	SPE • MSTR	SPE • MSTR	SPE • MSTR	SPE • MSTR
DDOV	0	0	0	0
PVOE	SPE • MSTR	SPE • MSTR	SPE • MSTR	0
PVOV	SCK OUTPUT	SPI SLAVE OUTPUT	SPI MSTR OUTPUT	0
DIOE	0	0	0	0
DIOV	0	0	0	0
DI	SCK INPUT	SPI MSTR INPUT	SPI SLAVE INPUT	SPI SS
AIO	–	–	–	–

**Table 28.** Overriding Signals for Alternate Functions in PB3..PB0

Signal Name	PB3/OC0/AIN1	PB2/INT2/AIN0	PB1/T1	PB0/T0/XCK
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC0 ENABLE	0	0	UMSEL
PVOV	OC0	0	0	XCK OUTPUT
DIOE	0	INT2 ENABLE	0	0
DIOV	0	1	0	0
DI	–	INT2 INPUT	T1 INPUT	XCK INPUT/T0 INPUT
AIO	AIN1 INPUT	AIN0 INPUT	–	–

## Alternate Functions of Port C

The Port C pins with alternate functions are shown in Table 29.

**Table 29.** Port C Pins Alternate Functions

Port Pin	Alternate Function
PC7	TOSC2 (Timer Oscillator Pin 2)
PC6	TOSC1 (Timer Oscillator Pin 1)
PC1	SDA (Two-wire Serial Bus Data Input/Output Line)
PC0	SCL (Two-wire Serial Bus Clock Line)

The alternate pin configuration is as follows:

- TOSC2 – Port C, Bit 7



**TOSC2, Timer Oscillator pin 2:** When the AS2 bit in ASSR is set (one) to enable asynchronous clocking of Timer/Counter2, pin PC7 is disconnected from the port, and becomes the inverting output of the Oscillator amplifier. In this mode, a crystal Oscillator is connected to this pin, and the pin can not be used as an I/O pin.

• **TOSC1 – Port C, Bit 6**

**TOSC1, Timer Oscillator pin 1:** When the AS2 bit in ASSR is set (one) to enable asynchronous clocking of Timer/Counter2, pin PC6 is disconnected from the port, and becomes the input of the inverting Oscillator amplifier. In this mode, a crystal Oscillator is connected to this pin, and the pin can not be used as an I/O pin.

• **SDA – Port C, Bit 1**

**SDA, Two-wire Serial Interface Data:** When the TWEN bit in TWCR is set (one) to enable the Two-wire Serial Interface, pin PC1 is disconnected from the port and becomes the Serial Data I/O pin for the Two-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation. When this pin is used by the Two-wire Serial Interface, the pull-up can still be controlled by the PORTC1 bit.

• **SCL – Port C, Bit 0**

**SCL, Two-wire Serial Interface Clock:** When the TWEN bit in TWCR is set (one) to enable the Two-wire Serial Interface, pin PC0 is disconnected from the port and becomes the Serial Clock I/O pin for the Two-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation. When this pin is used by the Two-wire Serial Interface, the pull-up can still be controlled by the PORTC0 bit.

Table 30 and Table 31 relate the alternate functions of Port C to the overriding signals shown in Figure 26 on page 56.

**Table 30. Overriding Signals for Alternate Functions in PC7..PC6**

Signal Name	PC7/TOSC2	PC6/TOSC1
PUOE	AS2	AS2
PUOV	0	0
DDOE	AS2	AS2
DDOV	0	0
PVOE	0	0
PVOV	0	0
DIEOE	AS2	AS2
DIEOV	0	0
DI	–	–
AIO	T/C2 OSC OUTPUT	T/C2 OSC INPUT

**Table 31.** Overriding Signals for Alternate Functions in PC1..PC0<sup>(1)</sup>

Signal Name	PC1/SDA	PC0/SCL
PUOE	TWEN	TWEN
PUOV	PORTC1 - PUD	PORTC0 - PUD
DDOE	TWEN	TWEN
DDOV	SDA_OUT	SCL_OUT
PVOE	TWEN	TWEN
PVOV	0	0
DIEOE	0	0
DIEOV	0	0
DI	-	-
AIO	SDA INPUT	SCL INPUT

Note: 1. When enabled, the Two-wire Serial Interface enables slew-rate controls on the output pins PC0 and PC1. This is not shown in the figure. In addition, spike filters are connected between the AIO outputs shown in the port figure and the digital logic of the TWI module.

## Alternate Functions of Port D

The Port D pins with alternate functions are shown in Table 32.

**Table 32.** Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	OC2 (Timer/Counter2 Output Compare Match Output)
PD6	ICP1 (Timer/Counter1 Input Capture Pin)
PD5	OC1A (Timer/Counter1 Output Compare A Match Output)
PD4	OC1B (Timer/Counter1 Output Compare B Match Output)
PD3	INT1 (External Interrupt 1 Input)
PD2	INT0 (External Interrupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)

The alternate pin configuration is as follows:

- **OC2 – Port D, Bit 7**

OC2, Timer/Counter2 Output Compare Match output: The PD7 pin can serve as an external output for the Timer/Counter2 Output Compare. The pin has to be configured as an output (DDD7 set (one)) to serve this function. The OC2 pin is also the output pin for the PWM mode timer function.

- **ICP1 – Port D, Bit 6**

ICP1 – Input Capture Pin: The PD6 pin can act as an Input Capture pin for Timer/Counter1.

- **OC1A – Port D, Bit 5**

OC1A, Output Compare Match A output: The PD5 pin can serve as an external output for the Timer/Counter1 Output Compare A. The pin has to be configured as an output

(DDD5 set (one)) to serve this function. The OC1A pin is also the output pin for the PWM mode timer function.

- **OC1B – Port D, Bit 4**

OC1B, Output Compare Match B output: The PD4 pin can serve as an external output for the Timer/Counter1 Output Compare B. The pin has to be configured as an output (DDD4 set (one)) to serve this function. The OC1B pin is also the output pin for the PWM mode timer function.

- **INT1 – Port D, Bit 3**

INT1, External Interrupt Source 1: The PD3 pin can serve as an external interrupt source.

- **INT0 – Port D, Bit 2**

INT0, External Interrupt Source 0: The PD2 pin can serve as an external interrupt source.

- **TXD – Port D, Bit 1**

TXD, Transmit Data (Data output pin for the USART). When the USART Transmitter is enabled, this pin is configured as an output regardless of the value of DDD1.

- **RXD – Port D, Bit 0**

RXD, Receive Data (Data input pin for the USART). When the USART Receiver is enabled this pin is configured as an input regardless of the value of DDD0. When the USART forces this pin to be an input, the pull-up can still be controlled by the PORTD0 bit.

Table 33 and Table 34 relate the alternate functions of Port D to the overriding signals shown in Figure 26 on page 56.

**Table 33. Overriding Signals for Alternate Functions PD7..PD4**

Signal Name	PD7/OC2	PD6/ICP1	PD5/OC1A	PD4/OC1B
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC2 ENABLE	0	OC1A ENABLE	OC1B ENABLE
PVOV	OC2	0	OC1A	OC1B
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	ICP1 INPUT	–	–
AIO	–	–	–	–

**Table 34. Overriding Signals for Alternate Functions in PD3..PD0**

Signal Name	PD3/INT1	PD2/INT0	PD1/TXD	PD0/RXD
PUOE	0	0	TXEN	RXEN
PUOV	0	0	0	PORTD0 • PUD
DDOE	0	0	TXEN	RXEN
DDOV	0	0	1	0
PVOE	0	0	TXEN	0
PVOV	0	0	TXD	0
DIEOE	INT1 ENABLE	INT0 ENABLE	0	0
DIEOV	1	1	0	0
DI	INT1 INPUT	INT0 INPUT	–	RXD
AIO	–	–	–	–

## Register Description for Ports

### Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

### Port B Data Register – PORTB

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### Port B Data Direction Register – DDRB

Bit	7	6	5	4	3	2	1	0	
	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



**B Input Pins Address – 3**

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

**C Data Register – PORTC**

Bit	7	6	5	4	3	2	1	0	
	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**C Data Direction Register – DDRC**

Bit	7	6	5	4	3	2	1	0	
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**C Input Pins Address – 2**

Bit	7	6	5	4	3	2	1	0	
	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

**D Data Register – PORTD**

Bit	7	6	5	4	3	2	1	0	
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**D Data Direction Register – DDRD**

Bit	7	6	5	4	3	2	1	0	
	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**D Input Pins Address – D**

Bit	7	6	5	4	3	2	1	0	
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

External Interrupts

The External Interrupts are triggered by the INT0, INT1, and INT2 pins. Observe that, if enabled, the interrupts will trigger even if the INT0..2 pins are configured as outputs. This feature provides a way of generating a software interrupt. The External Interrupts can be triggered by a falling or rising edge or a low level (INT2 is only an edge triggered interrupt). This is set up as indicated in the specification for the MCU Control Register – MCUCR and MCU Control and Status Register – MCUCSR. When the External Interrupt is enabled and is configured as level triggered (only INT0/INT1), the interrupt will trigger as long as the pin is held low. Note that recognition of falling or rising edge interrupts on INT0 and INT1 requires the presence of an I/O clock, described in “Clock Systems and their Distribution” on page 23. Low level interrupts on INT0/INT1 and the edge interrupt on INT2 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode. The I/O clock is halted in all sleep modes except Idle mode.

Note that if a level triggered interrupt is used for wake-up from Power-down mode, the changed level must be held for some time to wake up the MCU. This makes the MCU less sensitive to noise. The changed level is sampled twice by the Watchdog Oscillator clock. The period of the Watchdog Oscillator is 1 μs (nominal) at 5.0V and 25°C. The frequency of the Watchdog Oscillator is voltage dependent as shown in “Electrical Characteristics” on page 253. The MCU will wake up if the input has the required level during this sampling or if it is held until the end of the start-up time. The start-up time is defined by the SUT Fuses as described in “System Clock and Clock Options” on page 23. If the level is sampled twice by the Watchdog Oscillator clock but disappears before the end of the start-up time, the MCU will still wake up, but no interrupt will be generated. The required level must be held long enough for the MCU to complete the wake up to trigger the level interrupt.

The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

J Control Register – JCR

Bit	7	6	5	4	3	2	1	0	MCUCR
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	C	0	0	0	0	0	0	

• Bit 3, 2 – ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0

The External Interrupt 1 is activated by the external pin INT1 if the SREG I-bit and the corresponding interrupt mask in the GICR are set. The level and edges on the external INT1 pin that activate the interrupt are defined in Table 35. The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

Table 35. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

• Bit 1, 0 – ISC01, ISC00: Interrupt Sense Control 0 Bit 1 and Bit 0



The External Interrupt 0 is activated by the external pin INT0 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT0 pin that activate the interrupt are defined in Table 36. The value on the INT0 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

**Table 36.** Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

**I Control and Status Register – MCUCSR**

Bit	7	6	5	4	3	2	1	0	MCUCSR
	-	ISC2	-	-	WDRF	BORF	EXTRF	PORF	
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0		See Bit Description				

• **Bit 6 – ISC2: Interrupt Sense Control 2**

The asynchronous External Interrupt 2 is activated by the external pin INT2 if the SREG I-bit and the corresponding interrupt mask in GICR are set. If ISC2 is written to zero, a falling edge on INT2 activates the interrupt. If ISC2 is written to one, a rising edge on INT2 activates the interrupt. Edges on INT2 are registered asynchronously. Pulses on INT2 wider than the minimum pulse width given in Table 37 will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. When changing the ISC2 bit, an interrupt can occur. Therefore, it is recommended to first disable INT2 by clearing its Interrupt Enable bit in the GICR Register. Then, the ISC2 bit can be changed. Finally, the INT2 Interrupt Flag should be cleared by writing a logical one to its Interrupt Flag bit (INTF2) in the GIFR Register before the interrupt is re-enabled.

**Table 37.** Asynchronous External Interrupt Characteristics

Symbol	Parameter	Min	Typ	Max	Units
$t_{INT}$	Minimum pulse width for asynchronous external interrupt		50		ns

**General Interrupt Control Register – GICR**

Bit	7	6	5	4	3	2	1	0	GICR
	INT1	INT0	INT2	-	-	-	IVSEL	IVCE	
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 7 – INT1: External Interrupt Request 1 Enable**

When the INT1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control1 bits 1/0 (ISC11 and ISC10) in the MCU General Control Register (MCUCR) define whether the external interrupt is activated on the rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output.



The corresponding interrupt of External Interrupt Request 1 is executed from the INT1 Interrupt Vector.

• **Bit 6 – INT0: External Interrupt Request 0 Enable**

When the INT0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control0 bits 1/0 (ISC01 and ISC00) in the MCU General Control Register (MCUCR) define whether the external interrupt is activated on rising and/or falling edge of the INT0 pin or level sensed. Activity on the pin will cause an interrupt request even if INT0 is configured as an output. The corresponding interrupt of External Interrupt Request 0 is executed from the INT0 Interrupt Vector.

• **Bit 5 – INT2: External Interrupt Request 2 Enable**

When the INT2 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control2 bit (ISC2) in the MCU Control and Status Register (MCUCSR) defines whether the external interrupt is activated on the rising or falling edge of the INT2 pin. Activity on the pin will cause an interrupt request even if INT2 is configured as an output. The corresponding interrupt of External Interrupt Request 2 is executed from the INT2 Interrupt Vector.

**External Interrupt Flag Register – GIFR**

Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	INTF2	-	-	-	-	-	GIFR
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 7 – INTF1: External Interrupt Flag 1**

When an edge or logic change on the INT1 pin triggers an interrupt request, INTF1 becomes set (one). If the I-bit in SREG and the INT1 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. This flag is always cleared when INT1 is configured as a level interrupt.

• **Bit 6 – INTF0: External Interrupt Flag 0**

When an edge or logic change on the INT0 pin triggers an interrupt request, INTF0 becomes set (one). If the I-bit in SREG and the INT0 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. This flag is always cleared when INT0 is configured as a level interrupt.

• **Bit 5 – INTF2: External Interrupt Flag 2**

When an event on the INT2 pin triggers an interrupt request, INTF2 becomes set (one). If the I-bit in SREG and the INT2 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. Note that when entering some sleep modes with the INT2 interrupt disabled, the input buffer on this pin will be disabled. This may cause a logic change in internal signals which will set the INTF2 Flag. See "Digital Input Enable and Sleep Modes" on page 54 for more information.

## 8-bit Timer/Counter0 with PWM

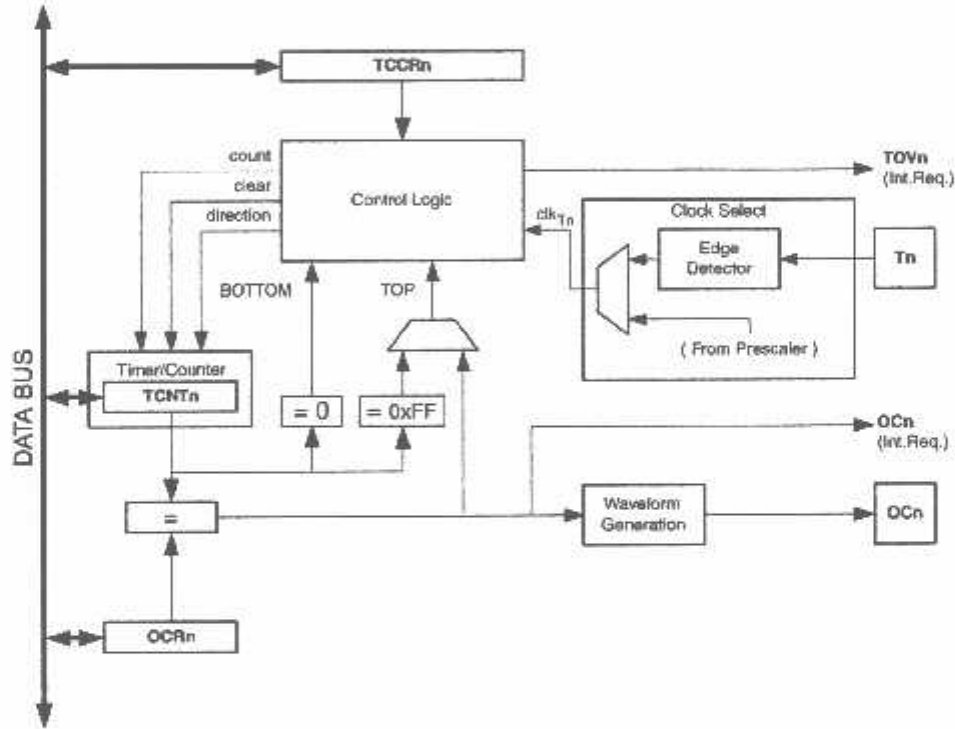
Timer/Counter0 is a general purpose, single channel, 8-bit Timer/Counter module. The main features are:

- Single Channel Counter
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- Frequency Generator
- External Event Counter
- 10-bit Clock Prescaler
- Overflow and Compare Match Interrupt Sources (TOV0 and OCF0)

### Overview

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 27. For the actual placement of I/O pins, refer to "Pinout ATmega8535" on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the "8-bit Timer/Counter Register Description" on page 82.

Figure 27. 8-bit Timer/Counter Block Diagram



### Registers

The Timer/Counter (TCNT0) and Output Compare Register (OCR0) are 8-bit registers. Interrupt request (abbreviated to Int. Req. in the figure) signals are all visible in the Timer Interrupt Flag Register (TIFR). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK). TIFR and TIMSK are not shown in the figure since these registers are shared by other timer units.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T0 pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock ( $clk_{T0}$ ).

The double buffered Output Compare Register (OCR0) is compared with the Timer/Counter value at all times. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin (OC0). See "Output Compare Unit" on page 72. for details. The Compare Match event will also set the Compare Flag (OCF0) which can be used to generate an output compare interrupt request.

## Definitions

Many register and bit references in this document are written in general form. A lower case "n" replaces the Timer/Counter number, in this case 0. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT0 for accessing Timer/Counter0 counter value and so on.

The definitions in Table 38 are also used extensively throughout the document.

**Table 38.** Definitions

<b>BOTTOM</b>	The counter reaches the <b>BOTTOM</b> when it becomes 0x00.
<b>MAX</b>	The counter reaches its <b>MAXimum</b> when it becomes 0xFF (decimal 255).
<b>TOP</b>	The counter reaches the <b>TOP</b> when it becomes equal to the highest value in the count sequence. The <b>TOP</b> value can be assigned to be the fixed value 0xFF ( <b>MAX</b> ) or the value stored in the OCR0 Register. The assignment is dependent on the mode of operation.

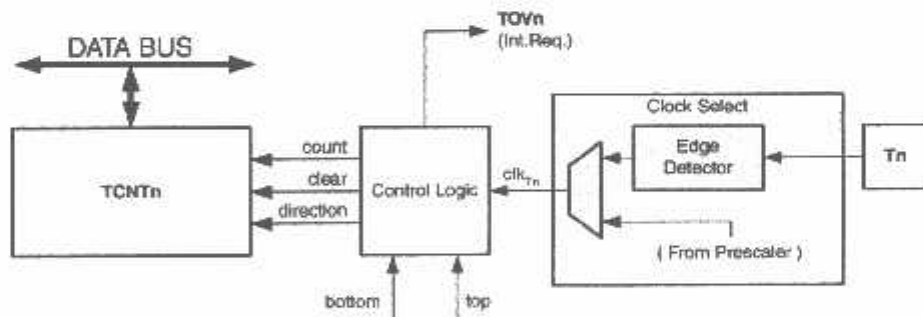
## Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the clock select (CS02:0) bits located in the Timer/Counter Control Register (TCCR0). For details on clock sources and prescaler, see "Timer/Counter0 and Timer/Counter1 Prescalers" on page 86.

## Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 28 shows a block diagram of the counter and its surroundings.

**Figure 28.** Counter Unit Block Diagram



Signal description (internal signals):

- count** Increment or decrement TCNT0 by 1.
- direction** Select between increment and decrement.
- clear** Clear TCNT0 (set all bits to zero).
- clk<sub>Tn</sub>** Timer/Counter clock, referred to as clk<sub>T0</sub> in the following.

- top**            Signalize that TCNT0 has reached maximum value.
- bottom**        Signalize that TCNT0 has reached minimum value (zero).

Depending of the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock ( $clk_{T0}$ ).  $clk_{T0}$  can be generated from an external or internal clock source, selected by the Clock Select bits (CS02:0). When no clock source is selected (CS02:0 = 0) the timer is stopped. However, the TCNT0 value can be accessed by the CPU, regardless of whether  $clk_{T0}$  is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the WGM01 and WGM00 bits located in the Timer/Counter Control Register (TCCR0). There are close connections between how the counter behaves (counts) and how waveforms are generated on the output compare output OC0. For more details about advanced counting sequences and waveform generation, see "Modes of Operation" on page 75.

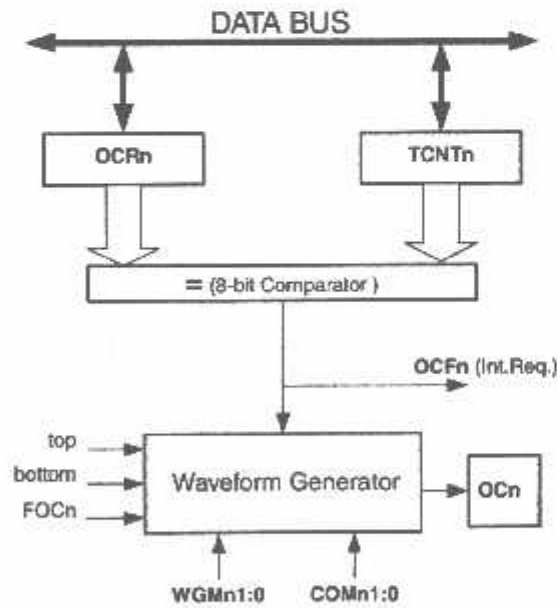
The Timer/Counter Overflow Flag (TOV0) is set according to the mode of operation selected by the WGM01:0 bits. TOV0 can be used for generating a CPU interrupt.

### Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the Output Compare Register (OCR0). Whenever TCNT0 equals OCR0, the comparator signals a match. A match will set the Output Compare Flag (OCF0) at the next timer clock cycle. If enabled (OCIE0 = 1 and Global Interrupt Flag in SREG is set), the Output Compare Flag generates an output compare interrupt. The OCF0 Flag is automatically cleared when the interrupt is executed. Alternatively, the OCF0 Flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the WGM01:0 bits and Compare Output mode (COM01:0) bits. The max and bottom signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (See "Modes of Operation" on page 75.).

Figure 29 shows a block diagram of the output compare unit.

**Figure 29.** Output Compare Unit, Block Diagram



The OCR0 Register is double buffered when using any of the Pulse Width Modulation (PWM) modes. For the normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR0 Compare Register to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR0 Register access may seem complex, but this is not the case. When the double buffering is enabled, the CPU has access to the OCR0 Buffer Register, and if double buffering is disabled the CPU will access the OCR0 directly.

#### **Force Output Compare**

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (FOC0) bit. Forcing Compare Match will not set the OCF0 Flag or reload/clear the timer, but the OC0 pin will be updated as if a real Compare Match had occurred (the COM01:0 bits settings define whether the OC0 pin is set, cleared or toggled).

#### **Compare Match Blocking by TCNT0 Write**

All CPU write operations to the TCNT0 Register will block any Compare Match that occur in the next timer clock cycle, even when the timer is stopped. This feature allows OCR0 to be initialized to the same value as TCNT0 without triggering an interrupt when the Timer/Counter clock is enabled.

#### **Configuring the Output Compare**

Since writing TCNT0 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT0 when using the output compare channel, independently of whether the Timer/Counter is running or not. If the value written to TCNT0 equals the OCR0 value, the Compare Match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT0 value equal to BOTTOM when the counter is down-counting.

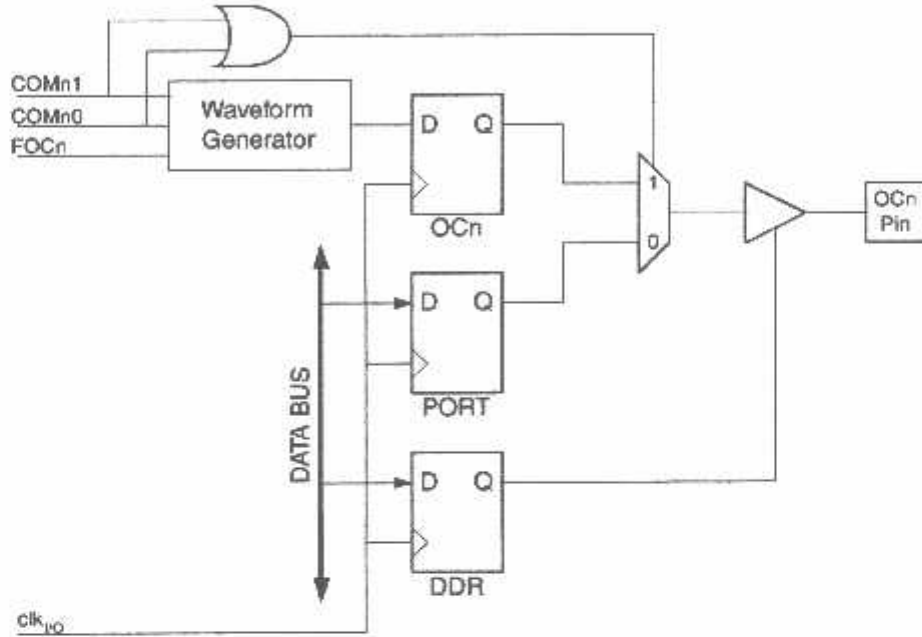
The setup of the OC0 should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC0 value is to use the force output compare (FOC0) strobe bits in normal mode. The OC0 Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COM01:0 bits are not double buffered together with the compare value. Changing the COM01:0 bits will take effect immediately.

## Compare Match Output

The Compare Output mode (COM01:0) bits have two functions. The Waveform Generator uses the COM01:0 bits for defining the Output Compare (OC0) state at the next Compare Match. Also, the COM01:0 bits control the OC0 pin output source. Figure 30 shows a simplified schematic of the logic affected by the COM01:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O Port Control Registers (DDR and PORT) that are affected by the COM01:0 bits are shown. When referring to the OC0 state, the reference is for the internal OC0 Register, not the OC0 pin. If a System Reset occur, the OC0 Register is reset to "0".

**Figure 30.** Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OC0) from the waveform generator if either of the COM01:0 bits are set. However, the OC0 pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OC0 pin (DDR\_OC0) must be set as output before the OC0 value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the output compare pin logic allows initialization of the OC0 state before the output is enabled. Note that some COM01:0 bit settings are reserved for certain modes of operation. See "8-bit Timer/Counter Register Description" on page 82.

## Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM01:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM01:0 = 0 tells the Waveform Generator that no action on the OC0 Register is to be performed on the next Compare Match. For compare output actions in the non-PWM modes refer to Table 40 on page 83. For fast PWM mode, refer to Table 41 on page 83, and for phase correct PWM refer to Table 42 on page 83.

A change of the COM01:0 bits state will have effect at the first Compare Match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC0 strobe bits.

## Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the output compare pins, is defined by the combination of the Waveform Generation mode (WGM01:0) and Compare Output mode (COM01:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM01:0 bits control whether the PWM output generated should be inverted or not (Inverted or non-inverted PWM). For non-PWM modes the COM01:0 bits control whether the output should be set, cleared, or toggled at a Compare Match (See "Compare Match Output Unit" on page 74.).

For detailed timing information refer to Figure 34, Figure 35, Figure 36, and Figure 37 in "Timer/Counter Timing Diagrams" on page 79.

## Normal Mode

The simplest mode of operation is the Normal mode (WGM01:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero. The TOV0 Flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the Timer Overflow interrupt that automatically clears the TOV0 Flag, the timer resolution can be increased by software. There are no special cases to consider in the normal mode, a new counter value can be written anytime.

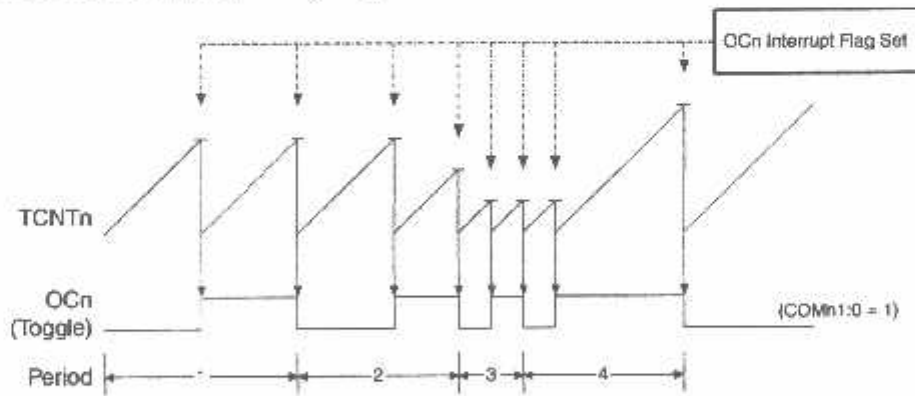
The output compare unit can be used to generate interrupts at some given time. Using the output compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

## Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM01:0 = 2), the OCR0 Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT0) matches the OCR0. The OCR0 defines the top value for the counter, hence also its resolution. This mode allows greater control of the Compare Match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 31. The counter value (TCNT0) increases until a Compare Match occurs between TCNT0 and OCR0, and then counter (TCNT0) is cleared.

Figure 31. CTC Mode, Timing Diagram



An interrupt can be generated each time the counter value reaches the TOP value by using the OCF0 Flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing TOP to a value close to BOTTOM



when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR0 is lower than the current value of TCNT0, the counter will miss the Compare Match. The counter will then have to count to its maximum value (0xFF) and wrap around starting at 0x00 before the Compare Match can occur.

For generating a waveform output in CTC mode, the OC0 output can be set to toggle its logical level on each Compare Match by setting the Compare Output mode bits to toggle mode (COM01:0 = 1). The OC0 value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of  $f_{OC0} = f_{clk_{I/O}}/2$  when OCR0 is set to zero (0x00). The waveform frequency is defined by the following equation:

$$f_{OCn} = \frac{f_{clk_{I/O}}}{2 \cdot N \cdot (1 + OCRn)}$$

The "N" variable represents the prescale factor (1, 8, 64, 256, or 1024).

As for the normal mode of operation, the TOV0 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

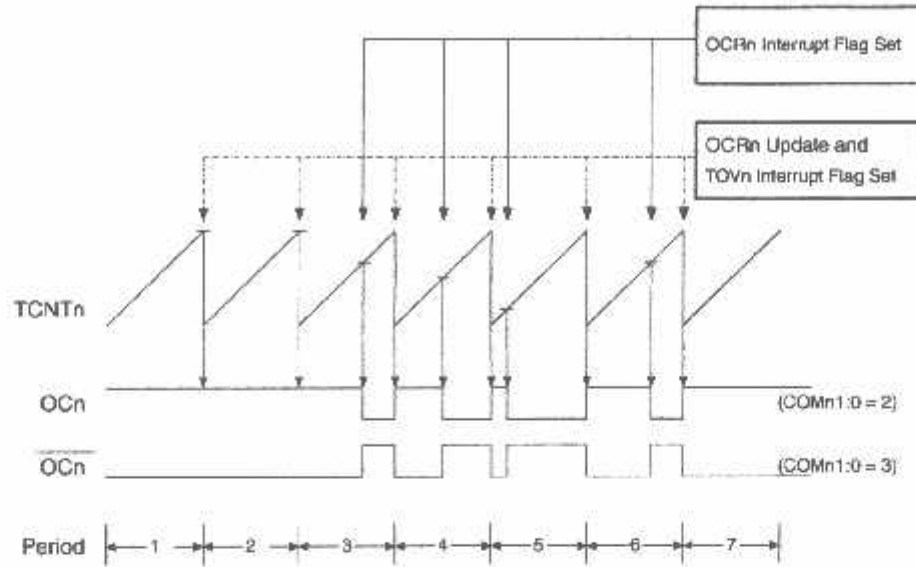
### PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM01:0 = 3) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to MAX then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC0) is cleared on the Compare Match between TCNT0 and OCR0, and set at BOTTOM. In inverting Compare Output mode, the output is set on Compare Match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the MAX value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 32. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent compare matches between OCR0 and TCNT0.



Figure 32. Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches MAX. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC0 pin. Setting the COM01:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM01:0 to three (See Table 41 on page 83). The actual OC0 value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by setting (or clearing) the OC0 Register at the Compare Match between OCR0 and TCNT0, and clearing (or setting) the OC0 Register at the timer clock cycle the counter is cleared (changes from MAX to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$$

The "N" variable represents the prescale factor (1, 8, 64, 256, or 1024).

The extreme values for the OCR0 Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCR0 is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR0 equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM01:0 bits).

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC0 to toggle its logical level on each Compare Match (COM01:0 = 1). The waveform generated will have a maximum frequency of  $f_{OC0} = f_{clk\_I/O}/2$  when OCR0 is set to zero. This feature is similar to the OC0 toggle in CTC mode, except the double buffer feature of the output compare unit is enabled in the fast PWM mode.

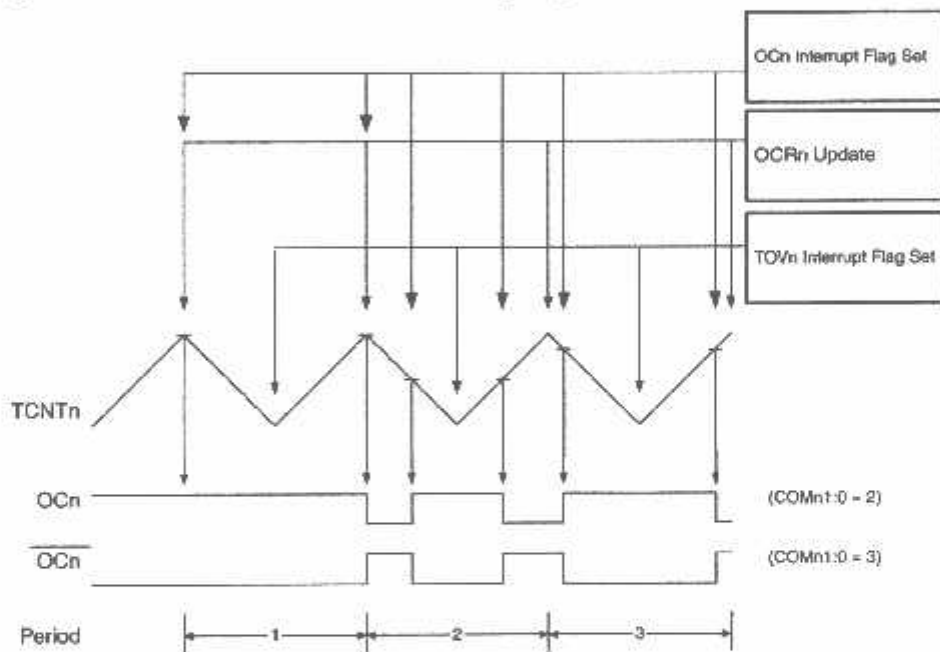


### Phase Correct PWM Mode

The phase correct PWM mode (WGM01:0 = 1) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC0) is cleared on the Compare Match between TCNT0 and OCR0 while up-counting, and set on the Compare Match while down-counting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode is fixed to eight bits. In phase correct PWM mode the counter is incremented until the counter value matches MAX. When the counter reaches MAX, it changes the count direction. The TCNT0 value will be equal to MAX for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 33. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent compare matches between OCR0 and TCNT0.

**Figure 33.** Phase Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches BOTTOM. The interrupt flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC0 pin. Setting the COM01:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM01:0 to three (See Table 42 on page 83). The actual OC0 value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC0 Register at the Compare Match between OCR0 and TCNT0 when the counter increments, and setting (or clearing) the OC0 Register at Compare Match between

OCR0 and TCNT0 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnPCPWM} = \frac{f_{clk_{I/O}}}{N \cdot 510}$$

The "N" variable represents the prescale factor (1, 8, 64, 256, or 1024).

The extreme values for the OCR0 Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR0 is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

AT the very start of period 2 in Figure 33 OCn has a transition from high to low even though there is no Compare Match. The point of this transition is to guarantee symmetry around BOTTOM. There are two cases that give a transition without Compare Match.

- OCR0 changes its value from MAX, like in Figure 33. When the OCR0 value is MAX the OCn pin value is the same as the result of a down-counting Compare Match. To ensure symmetry around BOTTOM the OCn value at MAX must correspond to the result of an up-counting Compare Match.
- The timer starts counting from a value higher than the one in OCR0, and for that reason misses the Compare Match and hence the OCn change that would have happened on the way up.

## ner/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock ( $clk_{T0}$ ) is therefore shown as a clock enable signal in the following figures. The figures include information on when interrupt flags are set. Figure 34 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

**Figure 34.** Timer/Counter Timing Diagram, no Prescaling

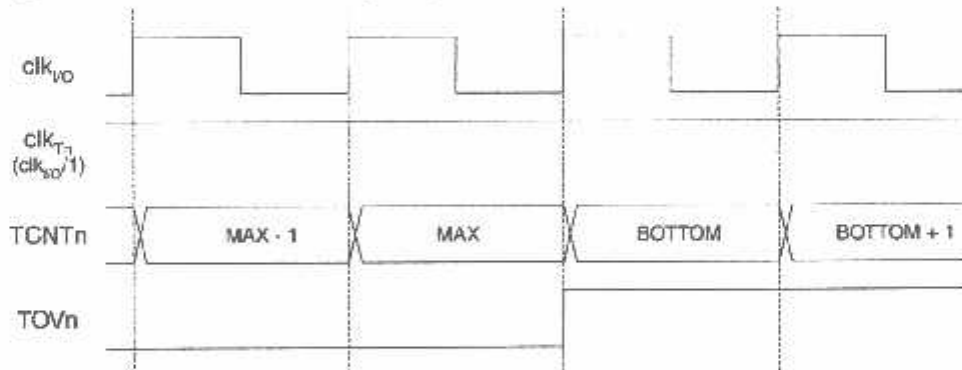


Figure 35 shows the same timing data, but with the prescaler enabled.



**Figure 35.** Timer/Counter Timing Diagram, with Prescaler ( $f_{clk_{VO}}/8$ )

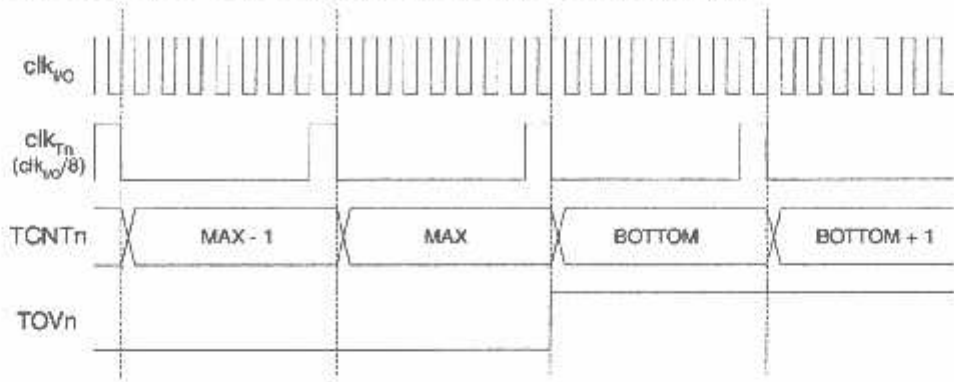


Figure 36 shows the setting of OCF0 in all modes except CTC mode.

**Figure 36.** Timer/Counter Timing Diagram, Setting of OCF0, with Prescaler ( $f_{clk_{VO}}/8$ )

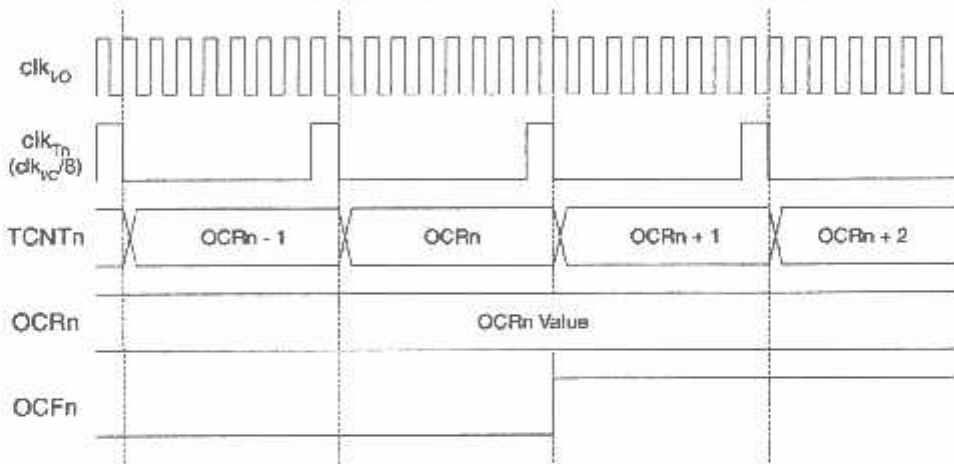
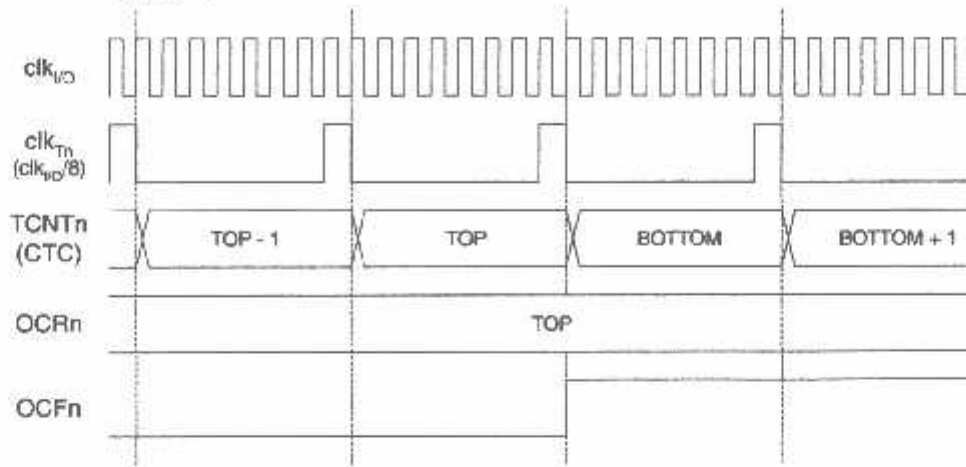


Figure 37 shows the setting of OCF0 and the clearing of TCNT0 in CTC mode.

**Figure 37.** Timer/Counter Timing Diagram, Clear Timer on Compare Match Mode, with Prescaler ( $f_{clk_{VD}}/8$ )



## Timer/Counter Register Description

### Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	TCCR0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC0: Force Output Compare**

The FOC0 bit is only active when the WGM00 bit specifies a non-PWM mode. However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0 is written when operating in PWM mode. When writing a logical one to the FOC0 bit, an immediate Compare Match is forced on the Waveform Generation unit. The OC0 output is changed according to its COM01:0 bits setting. Note that the FOC0 bit is implemented as a strobe. Therefore it is the value present in the COM01:0 bits that determines the effect of the forced compare.

A FOC0 strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR0 as TOP.

The FOC0 bit is always read as zero.

- **Bit 6, 3 – WGM01:0: Waveform Generation Mode**

These bits control the counting sequence of the counter, the source for the maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode, Clear Timer on Compare Match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes. See Table 39 and "Modes of Operation" on page 75.

**Table 39. Waveform Generation Mode Bit Description<sup>(1)</sup>**

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Note: 1. The CTC0 and PWM0 bit definition names are now obsolete. Use the WGM01:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the Timer.

- **Bit 5:4 – COM01:0: Compare Match Output Mode**

These bits control the Output Compare pin (OC0) behavior. If one or both of the COM01:0 bits are set, the OC0 output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OC0 pin must be set in order to enable the output driver.

When OC0 is connected to the pin, the function of the COM01:0 bits depends on the WGM01:0 bit setting. Table 40 shows the COM01:0 bit functionality when the WGM01:0 bits are set to a normal or CTC mode (non-PWM).

**Table 40. Compare Output Mode, non-PWM Mode**

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on Compare Match
1	0	Clear OC0 on Compare Match
1	1	Set OC0 on Compare Match

Table 41 shows the COM01:0 bit functionality when the WGM01:0 bits are set to fast PWM mode.

**Table 41. Compare Output Mode, Fast PWM Mode<sup>(1)</sup>**

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on Compare Match, set OC0 at TOP
1	1	Set OC0 on Compare Match, clear OC0 at TOP

Note: 1. A special case occurs when OCR0 equals TOP and COM01 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See "Fast PWM Mode" on page 76 for more details.

Table 42 shows the COM01:0 bit functionality when the WGM01:0 bits are set to phase correct PWM mode.

**Table 42. Compare Output Mode, Phase Correct PWM Mode<sup>(1)</sup>**

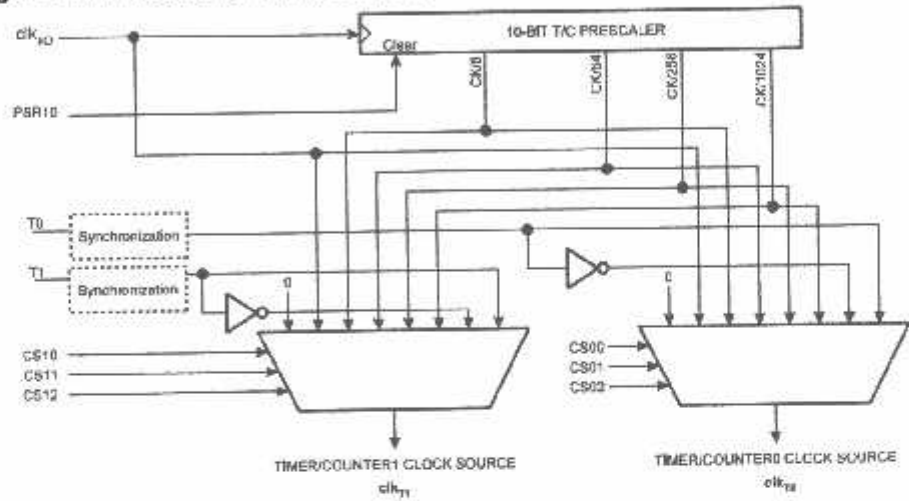
COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on Compare Match when up-counting. Set OC0 on Compare Match when down-counting.
1	1	Set OC0 on Compare Match when up-counting. Clear OC0 on Compare Match when down-counting.

Note: 1. A special case occurs when OCR0 equals TOP and COM01 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See "Phase Correct PWM Mode" on page 78 for more details.

Each half period of the external clock applied must be longer than one system clock cycle to ensure correct sampling. The external clock must be guaranteed to have less than half the system clock frequency ( $f_{ExtClk} < f_{clk\_I/O}/2$ ) given a 50/50% duty cycle. Since the edge detector uses sampling, the maximum frequency of an external clock it can detect is half the sampling frequency (Nyquist sampling theorem). However, due to variation of the system clock frequency and duty cycle caused by Oscillator source (crystal, resonator, and capacitors) tolerances, it is recommended that maximum frequency of an external clock source is less than  $f_{clk\_I/O}/2.5$ .

An external clock source can not be prescaled.

**Figure 39. Prescaler for Timer/Counter0 and Timer/Counter1<sup>(1)</sup>**



Note: 1. The synchronization logic on the input pins (T1/T0) is shown in Figure 38.

## Special Function IO Register – SFIOR

Bit	7	6	5	4	3	2	1	0	SFIO R
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### • Bit 0 – PSR10: Prescaler Reset Timer/Counter1 and Timer/Counter0

When this bit is written to one, the Timer/Counter1 and Timer/Counter0 prescaler will be reset. The bit will be cleared by hardware after the operation is performed. Writing a zero to this bit will have no effect. Note that Timer/Counter1 and Timer/Counter0 share the same prescaler and a reset of this prescaler will affect both timers. This bit will always be read as zero.



## 16-bit Timer/Counter1

The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement. The main features are:

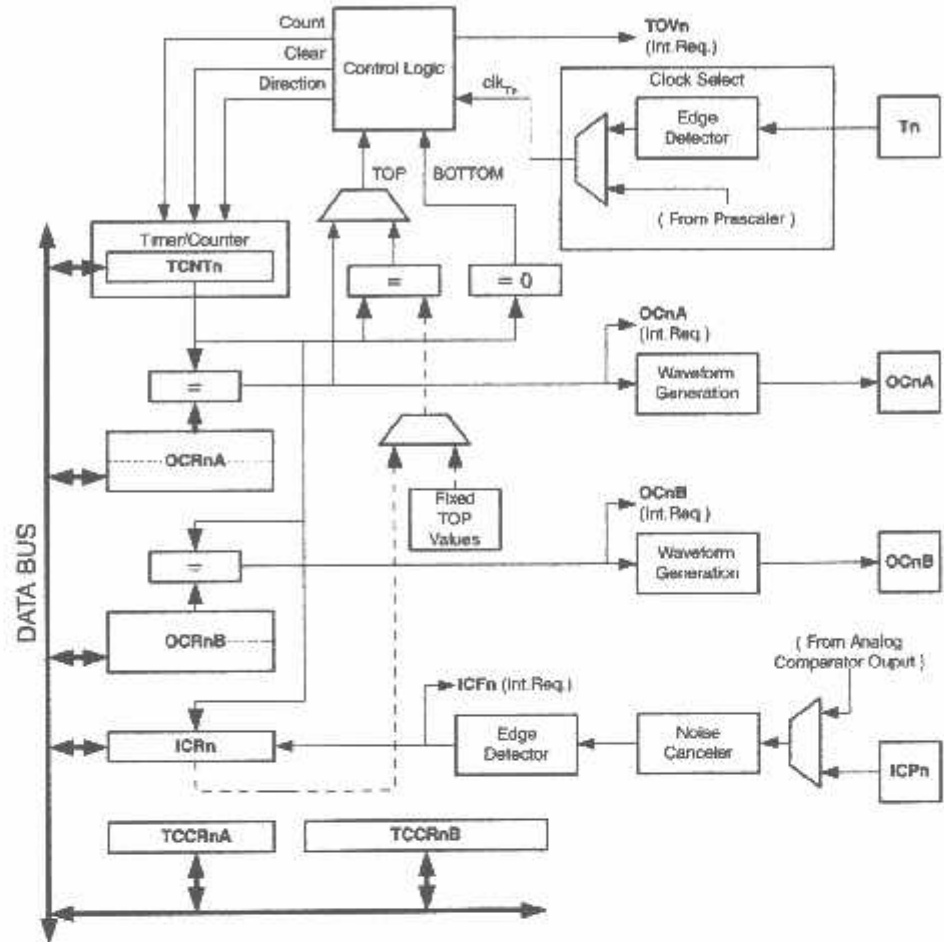
- True 16-bit Design (i.e., Allows 16-bit PWM)
- Two Independent Output Compare Units
- Double Buffered Output Compare Registers
- One Input Capture Unit
- Input Capture Noise Canceler
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- External Event Counter
- Four Independent Interrupt Sources (TOV1, OCF1A, OCF1B, and ICF1)

### Overview

Most register and bit references in this section are written in general form. A lower case "n" replaces the Timer/Counter number, and a lower case "x" replaces the Output Compare unit channel. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT1 for accessing Timer/Counter1 counter value and so on.

A simplified block diagram of the 16-bit Timer/Counter is shown in Figure 40. For the actual placement of I/O pins, refer to "Pinout ATmega8535" on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the "16-bit Timer/Counter Register Description" on page 109.

Figure 40. 16-bit Timer/Counter Block Diagram<sup>(1)</sup>



Note: 1. Refer to Figure 1 on page 2, Table 26 on page 59, and Table 32 on page 63 for Timer/Counter1 pin placement and description.

**Registers**

The *Timer/Counter (TCNT1)*, *Output Compare Registers (OCR1A/B)*, and *Input Capture Register (ICR1)* are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in the section "Accessing 16-bit Registers" on page 91. The *Timer/Counter Control Registers (TCCR1A/B)* are 8-bit registers and have no CPU access restrictions. Interrupt requests (abbreviated to Int.Req. in the figure) signals are all visible in the *Timer Interrupt Flag Register (TIFR)*. All interrupts are individually masked with the *Timer Interrupt Mask Register (TIMSK)*. TIFR and TIMSK are not shown in the figure since these registers are shared by other timer units.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T1 pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock ( $clk_{T1}$ ).

The double buffered Output Compare Registers (OCR1A/B) are compared with the Timer/Counter value at all times. The result of the compare can be used by the

Waveform Generator to generate a PWM or variable frequency output on the Output Compare Pin (OC1A/B). See "Output Compare Units" on page 97. The Compare Match event will also set the Compare Match Flag (OCF1A/B) which can be used to generate an output compare interrupt request.

The Input Capture Register can capture the Timer/Counter value at a given external (edge triggered) event on either the Input Capture Pin (ICP1) or on the Analog Comparator pins (See "Analog Comparator" on page 201.) The Input Capture unit includes a digital filtering unit (Noise Canceler) for reducing the chance of capturing noise spikes.

The TOP value, or maximum Timer/Counter value, can in some modes of operation be defined by either the OCR1A Register, the ICR1 Register, or by a set of fixed values. When using OCR1A as TOP value in a PWM mode, the OCR1A Register can not be used for generating a PWM output. However, the TOP value will, in this case, be double buffered allowing the TOP value to be changed in run time. If a fixed TOP value is required, the ICR1 Register can be used as an alternative, freeing the OCR1A to be used as PWM output.

## Definitions

The following definitions are used extensively throughout the document

**Table 44.** Definitions

BOTTOM	The counter reaches the <i>BOTTOM</i> when it becomes 0x0000.
MAX	The counter reaches its <i>MAXimum</i> when it becomes 0xFFFF (decimal 65535).
TOP	The counter reaches the <i>TOP</i> when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be one of the fixed values: 0x00FF, 0x01FF, or 0x03FF, or to the value stored in the OCR1A or ICR1 Register. The assignment is dependent of the mode of operation.

## Compatibility

The 16-bit Timer/Counter has been updated and improved from previous versions of the 16-bit AVR Timer/Counter. This 16-bit Timer/Counter is fully compatible with the earlier version regarding:

- All 16-bit Timer/Counter related I/O Register address locations, including Timer Interrupt Registers.
- Bit locations inside all 16-bit Timer/Counter Registers, including Timer Interrupt Registers.
- Interrupt Vectors.

The following control bits have changed names, but have the same functionality and register location:

- PWM10 is changed to WGM10.
- PWM11 is changed to WGM11.
- CTC1 is changed to WGM12.

The following bits are added to the 16-bit Timer/Counter Control Registers:

- FOC1A and FOC1B are added to TCCR1A.
- WGM13 is added to TCCR1B.

The 16-bit Timer/Counter has improvements that will affect the compatibility in some special cases.

## Accessing 16-bit Registers

The TCNT1, OCR1A/B, and ICR1 are 16-bit registers that can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bit access. The same temporary register is shared between all 16-bit registers within each 16-bit timer. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the temporary register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the temporary register in the same clock cycle as the low byte is read.

Not all 16-bit accesses uses the temporary register for the high byte. Reading the OCR1A/B 16-bit registers does not involve using the temporary register.

To do a 16-bit write, *the high byte must be written before the low byte*. For a 16-bit read, *the low byte must be read before the high byte*.

The following code examples show how to access the 16-bit Timer Registers assuming that no interrupts update the temporary register. The same principle can be used directly for accessing the OCR1A/B and ICR1 Registers. Note that when using "C", the compiler handles the 16-bit access.

### Assembly Code Examples<sup>(1)</sup>

```
...
; Set TCNT1 to 0x01FF
ldi r17,0x01
ldi r16,0xFF
out TCNT1H,r17
out TCNT1L,r16
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
...
```

### C Code Examples<sup>(1)</sup>

```
unsigned int i;
...
/* Set TCNT1 to 0x01FF */
TCNT1 = 0x1FF;
/* Read TCNT1 into i */
i = TCNT1;
...
```

Note: 1. See "About Code Examples" on page 6.

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

It is important to notice that accessing 16-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 16-bit register, and the interrupt code updates the temporary register by accessing the same or any other of the 16-bit Timer Registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

The following code examples show how to do an atomic read of the TCNT1 Register contents. Reading any of the OCR1A/B or ICR1 Registers can be done by using the same principle.

**Assembly Code Example<sup>(1)</sup>**

```
TIM16_ReadTCNT1:
; Save Global Interrupt Flag
in r18,SREG
; Disable interrupts
cli
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
; Restore Global Interrupt Flag
out SREG,r18
ret
```

**C Code Example<sup>(1)</sup>**

```
unsigned int TIM16_ReadTCNT1( void )
{
    unsigned char sreg;
    unsigned int i;
    /* Save Global Interrupt Flag */
    sreg = SREG;
    /* Disable interrupts */
    _CLI();
    /* Read TCNT1 into i */
    i = TCNT1;
    /* Restore Global Interrupt Flag */
    SREG = sreg;
    return i;
}
```

Note: 1. See "About Code Examples" on page 6.

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

The following code examples show how to do an atomic write of the TCNT1 Register contents. Writing any of the OCR1A/B or ICR1 Registers can be done by using the same principle.

### Assembly Code Example<sup>(1)</sup>

```
TIM16_WriteTCNT1:
; Save Global Interrupt Flag
in r18,SREG
; Disable interrupts
cli
; Set TCNT1 to r17:r16
out TCNT1H,r17
out TCNT1L,r16
; Restore Global Interrupt Flag
out SREG,r18
ret
```

### C Code Example<sup>(1)</sup>

```
void TIM16_WriteTCNT1( unsigned int i )
{
    unsigned char sreg;
    /* Save Global Interrupt Flag */
    sreg = SREG;
    /* Disable interrupts */
    _CLI();
    /* Set TCNT1 to i */
    TCNT1 = i;
    /* Restore Global Interrupt Flag */
    SREG = sreg;
}
```

Note: 1. See "About Code Examples" on page 6.

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNT1.

### Re-using the Temporary High Byte Register

If writing to more than one 16-bit register where the high byte is the same for all registers written, then the high byte only needs to be written once. However, note that the same rule of atomic operation described previously also applies in this case.

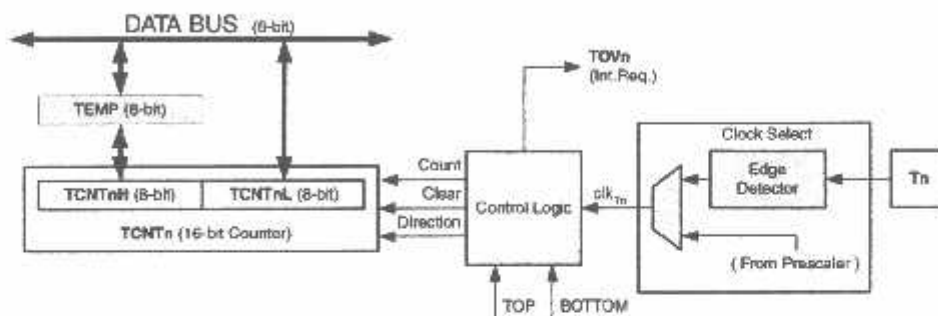
## Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the *Clock Select* (CS12:0) bits located in the *Timer/Counter Control Register B* (TCCR1B). For details on clock sources and prescaler, see "Timer/Counter0 and Timer/Counter1 Prescalers" on page 86.

## Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit bi-directional counter unit. Figure 41 shows a block diagram of the counter and its surroundings.

**Figure 41.** Counter Unit Block Diagram



Signal description (internal signals):

- Count** Increment or decrement TCNT1 by 1.
- Direction** Select between increment and decrement.
- Clear** Clear TCNT1 (set all bits to zero).
- clk<sub>T1</sub>** Timer/Counter clock.
- TOP** Signalize that TCNT1 has reached maximum value.
- BOTTOM** Signalize that TCNT1 has reached minimum value (zero).

The 16-bit counter is mapped into two 8-bit I/O memory locations: *Counter High* (TCNT1H) containing the upper eight bits of the counter, and *Counter Low* (TCNT1L) containing the lower eight bits. The TCNT1H Register can only be indirectly accessed by the CPU. When the CPU does an access to the TCNT1H I/O location, the CPU accesses the high byte temporary register (TEMP). The temporary register is updated with the TCNT1H value when the TCNT1L is read, and TCNT1H is updated with the temporary register value when TCNT1L is written. This allows the CPU to read or write the entire 16-bit counter value within one clock cycle via the 8-bit data bus. It is important to notice that there are special cases of writing to the TCNT1 Register when the counter is counting that will give unpredictable results. The special cases are described in the sections where they are of importance.

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each *Timer Clock* (clk<sub>T1</sub>). The clk<sub>T1</sub> can be generated from an external or internal clock source, selected by the *Clock Select* bits (CS12:0). When no clock source is selected (CS12:0 = 0) the timer is stopped. However, the TCNT1 value can be accessed by the CPU, independent of whether clk<sub>T1</sub> is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the *Waveform Generation mode* bits (WGM13:0) located in the *Timer/Counter Control Registers A and B* (TCCR1A and

TCCR1B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OC1x. For more details about advanced counting sequences and waveform generation, see "Modes of Operation" on page 100.

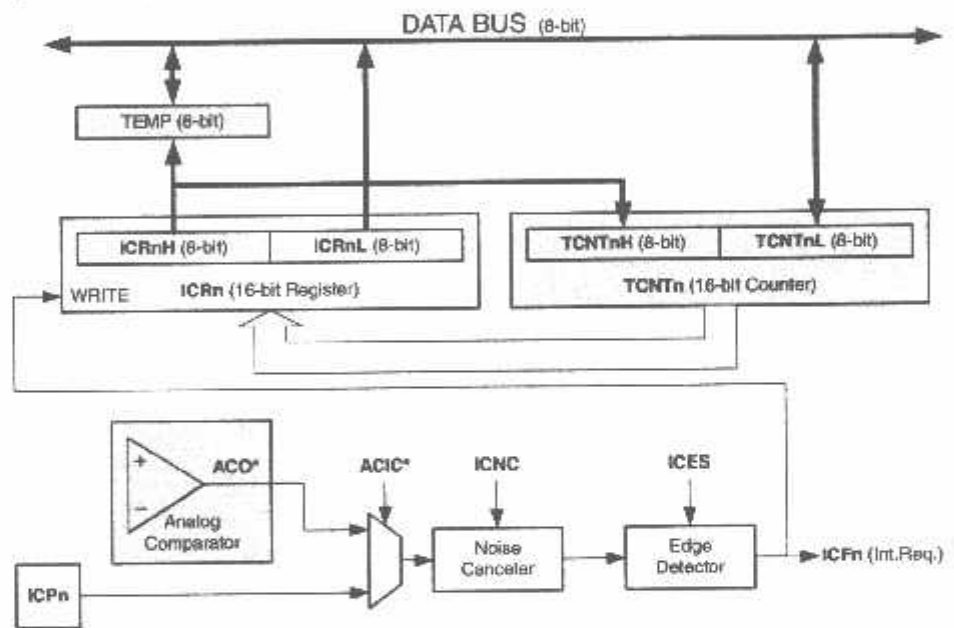
The *Timer/Counter Overflow Flag* (TOV1) is set according to the mode of operation selected by the WGM13:0 bits. TOV1 can be used for generating a CPU interrupt.

### Input Capture Unit

The Timer/Counter incorporates an Input Capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICP1 pin or alternatively, via the analog comparator unit. The time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.

The Input Capture unit is illustrated by the block diagram shown in Figure 42. The elements of the block diagram that are not directly a part of the Input Capture unit are gray shaded. The small "n" in register and bit names indicates the Timer/Counter number.

Figure 42. Input Capture Unit Block Diagram



When a change of the logic level (an event) occurs on the *Input Capture pin* (ICP1), alternatively on the *Analog Comparator output* (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNT1) is written to the *Input Capture Register* (ICR1). The *Input Capture Flag* (ICF1) is set at the same system clock as the TCNT1 value is copied into ICR1 Register. If enabled (TICIE1 = 1), the Input Capture Flag generates an Input Capture interrupt. The ICF1 Flag is automatically cleared when the Interrupt is executed. Alternatively the ICF1 Flag can be cleared by software by writing a logical one to its I/O bit location.

Reading the 16-bit value in the *Input Capture Register* (ICR1) is done by first reading the low byte (ICR1L) and then the high byte (ICR1H). When the low byte is read the high



byte is copied into the high byte temporary register (TEMP). When the CPU reads the ICR1H I/O location it will access the TEMP Register.

The ICR1 Register can only be written when using a Waveform Generation mode that utilizes the ICR1 Register for defining the counter's TOP value. In these cases the *Waveform Generation mode* (WGM13:0) bits must be set before the TOP value can be written to the ICR1 Register. When writing the ICR1 Register the high byte must be written to the ICR1H I/O location before the low byte is written to ICR1L.

For more information on how to access the 16-bit registers, refer to "Accessing 16-bit Registers" on page 91.

#### Input Capture Trigger Source

The main trigger source for the Input Capture unit is the *Input Capture pin* (ICP1). Timer/Counter1 can alternatively use the Analog Comparator output as trigger source for the Input Capture unit. The Analog Comparator is selected as trigger source by setting the *Analog Comparator Input Capture* (ACIC) bit in the *Analog Comparator Control and Status Register* (ACSR). Be aware that changing trigger source can trigger a capture. The Input Capture Flag must therefore be cleared after the change.

Both the *Input Capture pin* (ICP1) and the *Analog Comparator output* (ACO) inputs are sampled using the same technique as the T1 pin (Figure 38 on page 86). The edge detector is also identical. However, when the noise canceler is enabled, additional logic is inserted before the edge detector, which increases the delay by four system clock cycles. Note that the input of the noise canceler and edge detector is always enabled unless the Timer/Counter is set in a Waveform Generation mode that uses ICR1 to define TOP.

An Input Capture can be triggered by software by controlling the port of the ICP1 pin.

#### Noise Canceler

The noise canceler improves noise immunity by using a simple digital filtering scheme. The noise canceler input is monitored over four samples, and all four must be equal for changing the output that, in turn, is used by the edge detector.

The noise canceler is enabled by setting the *Input Capture Noise Canceler* (ICNC1) bit in *Timer/Counter Control Register B* (TCCR1B). When enabled, the noise canceler introduces additional four system clock cycles of delay from a change applied to the input, to the update of the ICR1 Register. The noise canceler uses the system clock and is therefore not affected by the prescaler.

#### Using the Input Capture Unit

The main challenge when using the Input Capture unit is to assign enough processor capacity for handling the incoming events. The time between two events is critical. If the processor has not read the captured value in the ICR1 Register before the next event occurs, the ICR1 will be overwritten with a new value. In this case the result of the capture will be incorrect.

When using the Input Capture interrupt, the ICR1 Register should be read as early in the interrupt handler routine as possible. Even though the Input Capture interrupt has relatively high priority, the maximum interrupt response time is dependent on the maximum number of clock cycles it takes to handle any of the other interrupt requests.

Using the Input Capture unit in any mode of operation when the TOP value (resolution) is actively changed during operation, is not recommended.

Measurement of an external signal's duty cycle requires that the trigger edge is changed after each capture. Changing the edge sensing must be done as early as possible after the ICR1 Register has been read. After a change of the edge, the Input Capture Flag (ICF1) must be cleared by software (writing a logical one to the I/O bit location). For

measuring frequency only, the clearing of the ICF1 Flag is not required (if an interrupt handler is used).

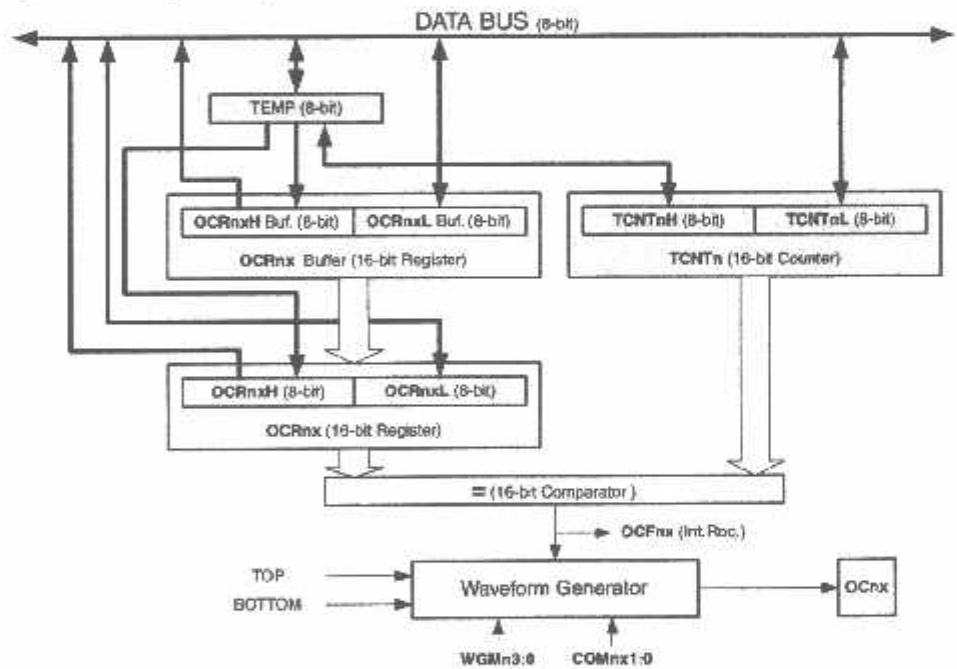
**Output Compare Units**

The 16-bit comparator continuously compares TCNT1 with the *Output Compare Register* (OCR1x). If TCNT equals OCR1x the comparator signals a match. A match will set the *Output Compare Flag* (OCF1x) at the next timer clock cycle. If enabled (OCIE1x = 1), the Output Compare Flag generates an output compare interrupt. The OCF1x Flag is automatically cleared when the interrupt is executed. Alternatively the OCF1x Flag can be cleared by software by writing a logical one to its I/O bit location. The *Waveform Generator* uses the match signal to generate an output according to operating mode set by the *Waveform Generation mode* (WGM13:0) bits and *Compare Output mode* (COM1x1:0) bits. The TOP and BOTTOM signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation. (See "Modes of Operation" on page 100.)

A special feature of output compare unit A allows it to define the Timer/Counter TOP value (i.e., counter resolution). In addition to the counter resolution, the TOP value defines the period time for waveforms generated by the Waveform Generator.

Figure 43 shows a block diagram of the output compare unit. The small "n" in the register and bit names indicates the device number (n = 1 for Timer/Counter1), and the "x" indicates output compare unit (A/B). The elements of the block diagram that are not directly a part of the output compare unit are gray shaded.

**Figure 43. Output Compare Unit, Block Diagram**



The OCR1x Register is double buffered when using any of the twelve *Pulse Width Modulation (PWM)* modes. For the Normal and *Clear Timer on Compare (CTC)* modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR1x Compare Register to either TOP or BOTTOM of the counting

sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR1x Register access may seem complex, but this is not the case. When the double buffering is enabled, the CPU has access to the OCR1x Buffer Register, and if double buffering is disabled the CPU will access the OCR1x directly. The content of the OCR1x (Buffer or Compare) Register is only changed by a write operation (the Timer/Counter does not update this register automatically as does the TCNT1- and ICR1 Register). Therefore OCR1x is not read via the high byte temporary register (TEMP). However, it is a good practice to read the low byte first as with accessing other 16-bit registers. Writing the OCR1x Registers must be done via the TEMP Register since the compare of all 16 bits is done continuously. The high byte (OCR1xH) has to be written first. When the high byte I/O location is written by the CPU, the TEMP Register will be updated by the value written. Then when the low byte (OCR1xL) is written to the lower eight bits, the high byte will be copied into the upper 8-bits of either the OCR1x Buffer or OCR1x Compare Register in the same system clock cycle.

For more information of how to access the 16-bit registers refer to "Accessing 16-bit Registers" on page 91.

#### Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the *Force Output Compare* (FOC1x) bit. Forcing Compare Match will not set the OCF1x Flag or reload/clear the timer, but the OC1x pin will be updated as if a real Compare Match had occurred (the COM11:0 bits settings define whether the OC1x pin is set, cleared or toggled).

#### Compare Match Blocking by TCNT1 Write

All CPU writes to the TCNT1 Register will block any Compare Match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR1x to be initialized to the same value as TCNT1 without triggering an interrupt when the Timer/Counter clock is enabled.

#### Using the Output Compare Unit

Since writing TCNT1 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT1 when using any of the output compare channels, independent of whether the Timer/Counter is running or not. If the value written to TCNT1 equals the OCR1x value, the Compare Match will be missed, resulting in incorrect waveform generation. Do not write the TCNT1 equal to TOP in PWM modes with variable TOP values. The Compare Match for the TOP will be ignored and the counter will continue to 0xFFFF. Similarly, do not write the TCNT1 value equal to BOTTOM when the counter is down-counting.

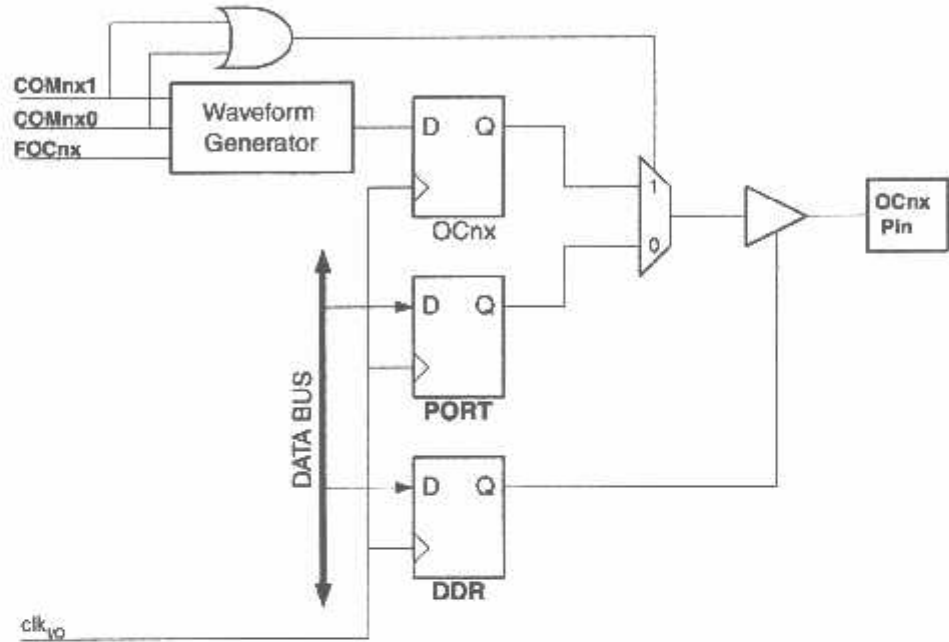
The setup of the OC1x should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC1x value is to use the Force Output Compare (FOC1x) strobe bits in Normal mode. The OC1x Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COM1x1:0 bits are not double buffered together with the compare value. Changing the COM1x1:0 bits will take effect immediately.

**Compare Match Output Unit**

The *Compare Output Mode* (COM1x1:0) bits have two functions. The waveform generator uses the COM1x1:0 bits for defining the Output Compare (OC1x) state at the next Compare Match. Secondly the COM1x1:0 bits control the OC1x pin output source. Figure 44 shows a simplified schematic of the logic affected by the COM1x1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O port Control Registers (DDR and PORT) that are affected by the COM1x1:0 bits are shown. When referring to the OC1x state, the reference is for the internal OC1x Register, not the OC1x pin. If a System Reset occurs, the OC1x Register is reset to "0".

**Figure 44.** Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OC1x) from the waveform generator if either of the COM1x1:0 bits are set. However, the OC1x pin direction (input or output) is still controlled by the *Data Direction Register* (DDR) for the port pin. The Data Direction Register bit for the OC1x pin (DDR\_OC1x) must be set as output before the OC1x value is visible on the pin. The port override function is generally independent of the Waveform Generation mode, but there are some exceptions. Refer to Table 45, Table 46 and Table 47 for details.

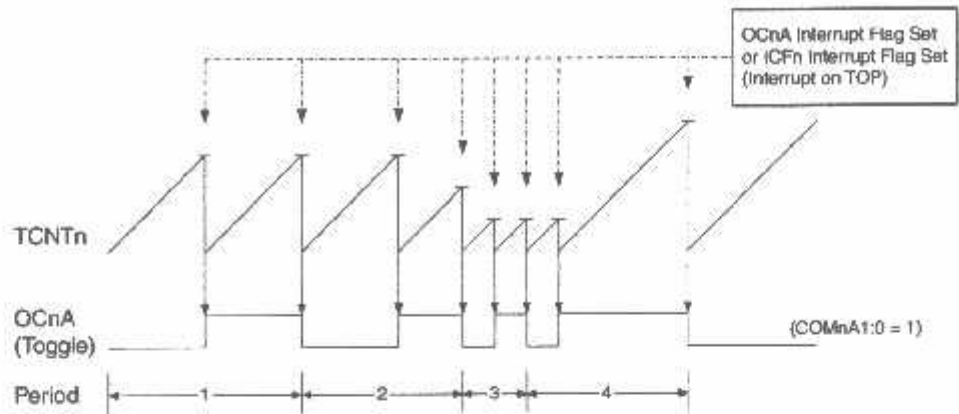
The design of the output compare pin logic allows initialization of the OC1x state before the output is enabled. Note that some COM1x1:0 bit settings are reserved for certain modes of operation. See "16-bit Timer/Counter Register Description" on page 109.

The COM1x1:0 bits have no effect on the Input Capture unit.

**Compare Output Mode and Waveform Generation**

The Waveform Generator uses the COM1x1:0 bits differently in Normal, CTC, and PWM modes. For all modes, setting the COM1x1:0 = 0 tells the Waveform Generator that no action on the OC1x Register is to be performed on the next Compare Match. For compare output actions in the non-PWM modes refer to Table 45 on page 109. For fast PWM mode refer to Table 46 on page 110, and for phase correct and phase and frequency correct PWM refer to Table 47 on page 110.

Figure 45. CTC Mode, Timing Diagram



An interrupt can be generated each time the counter value reaches the TOP value by either using the OCF1A or ICF1 Flag according to the register used to define the TOP value. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with a low or no prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR1A or ICR1 is lower than the current value of TCNT1, the counter will miss the Compare Match. The counter will then have to count to its maximum value (0xFFFF) and wrap around starting at 0x0000 before the Compare Match can occur. In many cases this feature is not desirable. An alternative will then be to use the fast PWM mode using OCR1A for defining TOP (WGM13:0 = 15) since the OCR1A then will be double buffered.

For generating a waveform output in CTC mode, the OC1A output can be set to toggle its logical level on each Compare Match by setting the compare output mode bits to toggle mode (COM1A1:0 = 1). The OC1A value will not be visible on the port pin unless the data direction for the pin is set to output (DDR\_OC1A = 1). The waveform generated will have a maximum frequency of  $f_{OC1A} = f_{clk\_I/O} / 2$  when OCR1A is set to zero (0x0000). The waveform frequency is defined by the following equation:

$$f_{OCnA} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

The N variable represents the prescaler factor (1, 8, 64, 256, or 1024).

As for the Normal mode of operation, the TOV1 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x0000.

**Fast PWM Mode**

The *fast Pulse Width Modulation* or fast PWM mode (WGM13:0 = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is set on the Compare Match between TCNT1 and OCR1x, and cleared at TOP. In inverting Compare Output mode output is cleared on Compare Match and set at TOP. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct and phase and frequency correct PWM modes that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High fre-

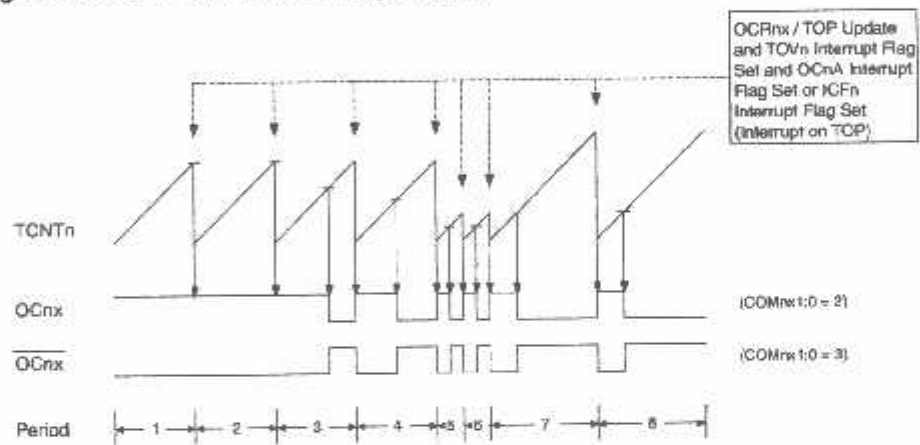
quency allows physically small sized external components (coils, capacitors), hence reducing total system cost.

The PWM resolution for fast PWM can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution can be calculated in bits by using the following equation:

$$R_{FPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM13:0 = 5, 6, or 7), the value in ICR1 (WGM13:0 = 14), or the value in OCR1A (WGM13:0 = 15). The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 46. The figure shows fast PWM when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x Interrupt Flag will be set when a Compare Match occurs.

**Figure 46.** Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set each time the counter reaches TOP. In addition the OC1A or ICF1 Flag is set at the same timer clock cycle as TOV1 is set when either OCR1A or ICR1 is used for defining the TOP value. If one of the interrupts are enabled, the interrupt handler routine can be used for updating the TOP and compare values.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the compare registers. If the TOP value is lower than any of the compare registers, a Compare Match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values the unused bits are masked to zero when any of the OCR1x Registers are written.

The procedure for updating ICR1 differs from updating OCR1A when used for defining the TOP value. The ICR1 Register is not double buffered. This means that if ICR1 is changed to a low value when the counter is running with a low or none prescaler value, there is a risk that the new ICR1 value written is lower than the current value of TCNT1. The result will then be that the counter will miss the Compare Match at the TOP value.

The counter will then have to count to the MAX value (0xFFFF) and wrap around starting at 0x0000 before the Compare Match can occur. The OCR1A Register however, is double buffered. This feature allows the OCR1A I/O location to be written anytime. When the OCR1A I/O location is written the value written will be put into the OCR1A Buffer Register. The OCR1A Compare Register will then be updated with the value in the Buffer Register at the next timer clock cycle the TCNT1 matches TOP. The update is done at the same timer clock cycle as the TCNT1 is cleared and the TOV1 Flag is set.

Using the ICR1 Register for defining TOP works well when using fixed TOP values. By using ICR1, the OCR1A Register is free to be used for generating a PWM output on OC1A. However, if the base PWM frequency is actively changed (by changing the TOP value), using the OCR1A as TOP is clearly a better choice due to its double buffer feature.

In fast PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to three (See Table 46 on page 110). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR\_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x Register at the Compare Match between OCR1x and TCNT1, and clearing (or setting) the OC1x Register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot (1 + TOP)}$$

The "N" variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCR1x is set equal to BOTTOM (0x0000) the output will be a narrow spike for each TOP+1 timer clock cycle. Setting the OCR1x equal to TOP will result in a constant high or low output (depending on the polarity of the output set by the COM1x1:0 bits).

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC1A to toggle its logical level on each Compare Match (COM1A1:0 = 1). This applies only if OCR1A is used to define the TOP value (WGM13:0 = 15). The waveform generated will have a maximum frequency of  $f_{OC1A} = f_{clk\_I/O}/2$  when OCR1A is set to zero (0x0000). This feature is similar to the OC1A toggle in CTC mode, except the double buffer feature of the output compare unit is enabled in the fast PWM mode.

## Phase Correct PWM Mode

The *phase correct Pulse Width Modulation* or phase correct PWM mode (WGM13:0 = 1, 2, 3, 10, or 11) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is, like the phase and frequency correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is cleared on the Compare Match between TCNT1 and OCR1x while up-counting and set on the Compare Match while down-counting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

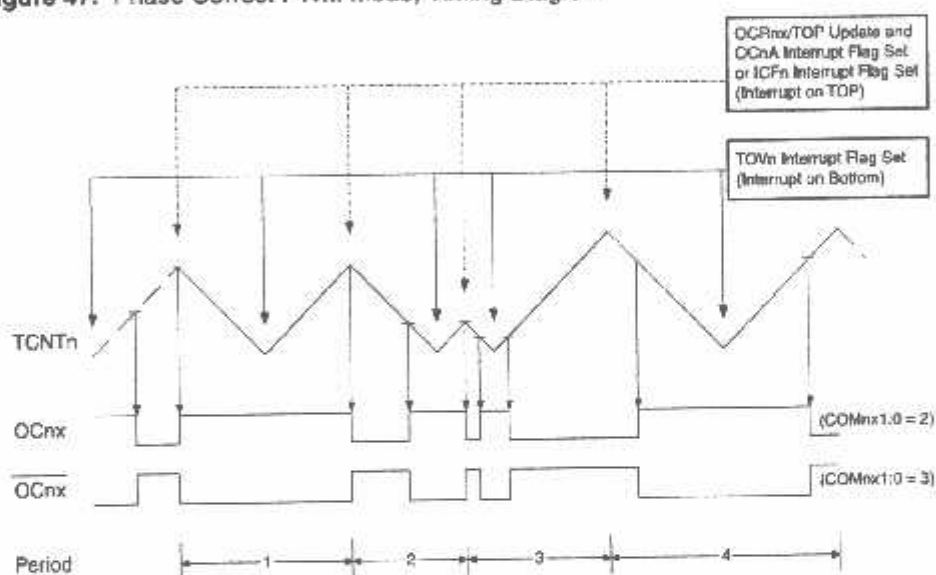
The PWM resolution for the phase correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or

OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution can be calculated in bits by using the following equation:

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase correct PWM mode, the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM13:0 = 1, 2, or 3), the value in ICR1 (WGM13:0 = 10), or the value in OCR1A (WGM13:0 = 11). The counter has then reached the TOP and changes the count direction. The TCNT1 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 47. The figure shows phase correct PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x Interrupt Flag will be set when a Compare Match occurs.

**Figure 47.** Phase Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set each time the counter reaches BOTTOM. When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 Flag is set accordingly at the same timer clock cycle as the OCR1x Registers are updated with the double buffer value (at TOP). The interrupt flags can be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value, the program must ensure that the new TOP value is higher or equal to the value of all of the compare registers. If the TOP value is lower than any of the compare registers, a Compare Match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values, the unused bits are masked to zero when any of the OCR1x Registers are written. As the third period shown in Figure 47 illustrates, changing the TOP actively while the Timer/Counter is running in the phase correct mode can result in an unsymmetrical output. The reason for this can be found in the time of update of the OCR1x Register. Since the OCR1x update occurs at TOP, the PWM period starts and ends at TOP. This implies that the length of the fall-



ing slope is determined by the previous TOP value, while the length of the rising slope is determined by the new TOP value. When these two values differ the two slopes of the period will differ in length. The difference in length gives the unsymmetrical result on the output.

It is recommended to use the phase and frequency correct mode instead of the phase correct mode when changing the TOP value while the Timer/Counter is running. When using a static TOP value, there are practically no differences between the two modes of operation.

In phase correct PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to three (See Table 47 on page 110). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR\_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x Register at the Compare Match between OCR1x and TCNT1 when the counter increments, and clearing (or setting) the OC1x Register at Compare Match between OCR1x and TCNT1 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OC1xPCPWM} = \frac{f_{clk I/O}}{2 \cdot N \cdot TOP}$$

The "N" variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR1x is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 11) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

## Phase and Frequency Correct PWM Mode

The *phase and frequency correct Pulse Width Modulation*, or phase and frequency correct PWM mode (WGM13:0 = 8 or 9) provides a high resolution phase and frequency correct PWM waveform generation option. The phase and frequency correct PWM mode is, like the phase correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is cleared on the Compare Match between TCNT1 and OCR1x while up-counting, and set on the Compare Match while down-counting. In inverting Compare Output mode, the operation is inverted. The dual-slope operation gives a lower maximum operation frequency compared to the single-slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

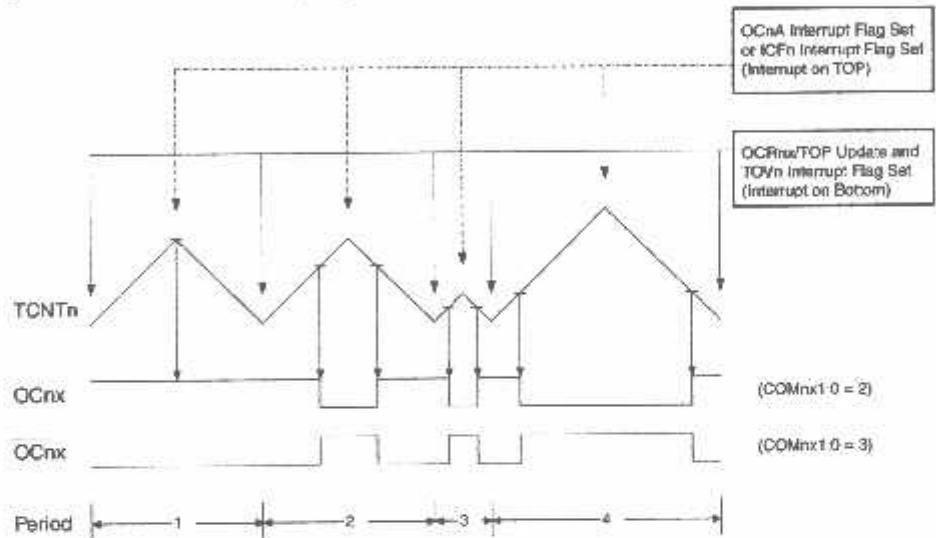
The main difference between the phase correct and the phase and frequency correct PWM mode is the time the OCR1x Register is updated by the OCR1x Buffer Register, (see Figure 47 and Figure 48).

The PWM resolution for the phase and frequency correct PWM mode can be defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated using the following equation:

$$R_{PFPCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase and frequency correct PWM mode the counter is incremented until the counter value matches either the value in ICR1 (WGM13:0 = 8), or the value in OCR1A (WGM13:0 = 9). The counter has then reached the TOP and changes the count direction. The TCNT1 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct and frequency correct PWM mode is shown in Figure 48. The figure shows phase and frequency correct PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x Interrupt Flag will be set when a Compare Match occurs.

**Figure 48.** Phase and Frequency Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV1) is set at the same timer clock cycle as the OCR1x Registers are updated with the double buffer value (at BOTTOM). When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 Flag set when TCNT1 has reached TOP. The interrupt flags can then be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the compare registers. If the TOP value is lower than any of the compare registers, a Compare Match will never occur between the TCNT1 and the OCR1x.

As Figure 48 shows the output generated is, in contrast to the phase correct mode, symmetrical in all periods. Since the OCR1x Registers are updated at BOTTOM, the length of the rising and the falling slopes will always be equal. This gives symmetrical output pulses and the frequency is, therefore, correct.

Using the ICR1 Register for defining TOP works well when using fixed TOP values. By using ICR1, the OCR1A Register is free to be used for generating a PWM output on OC1A. However, if the base PWM frequency is actively changed by changing the TOP value, using the OCR1A as TOP is clearly a better choice due to its double buffer feature.

In phase and frequency correct PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to three (see Table 47 on page 110). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR\_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x Register at the Compare Match between OCR1x and TCNT1 when the counter increments, and clearing (or setting) the OC1x Register at Compare Match between OCR1x and TCNT1 when the counter decrements. The PWM frequency for the output when using phase and frequency correct PWM can be calculated by the following equation:

$$f_{OCnFCPWM} = \frac{f_{clk_{IO}}}{2 \cdot N \cdot TOP}$$

The “N” variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x Register represents special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR1x is set equal to BOTTOM the output will be continuously low and if set equal to TOP, the output will be set to high for non-inverted PWM mode. For inverted PWM, the output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 9) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

**Timer/Counter Timing Diagrams**

The Timer/Counter is a synchronous design and the timer clock (clk<sub>T1</sub>) is therefore shown as a clock enable signal in the following figures. The figures include information on when interrupt flags are set, and when the OCR1x Register is updated with the OCR1x buffer value (only for modes utilizing double buffering). Figure 49 shows a timing diagram for the setting of OCF1x.

**Figure 49. Timer/Counter Timing Diagram, Setting of OCF1x, no Prescaling**

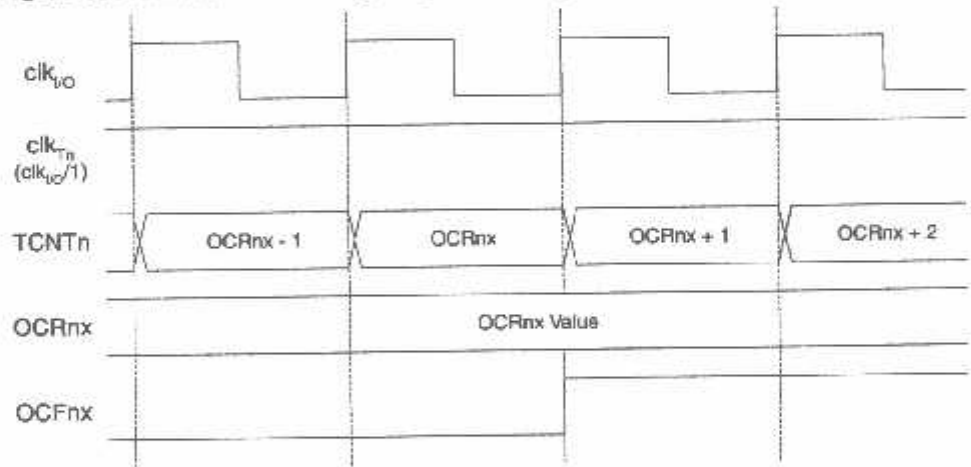


Figure 50 shows the same timing data, but with the prescaler enabled.

**Figure 50.** Timer/Counter Timing Diagram, Setting of OCF1x, with Prescaler ( $f_{clk_{IO}}/8$ )

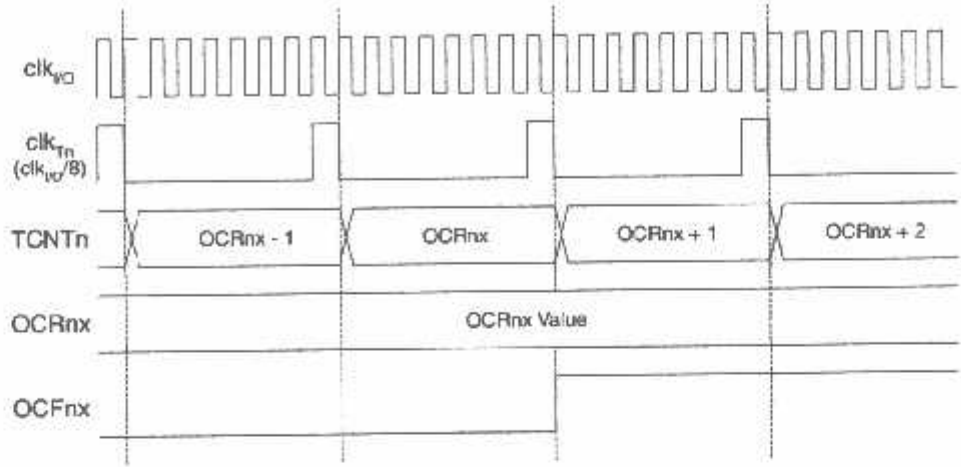


Figure 51 shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode, the OCR1x Register is updated at BOTTOM. The timing diagrams will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that set the TOV1 Flag at BOTTOM.

**Figure 51.** Timer/Counter Timing Diagram, no Prescaling

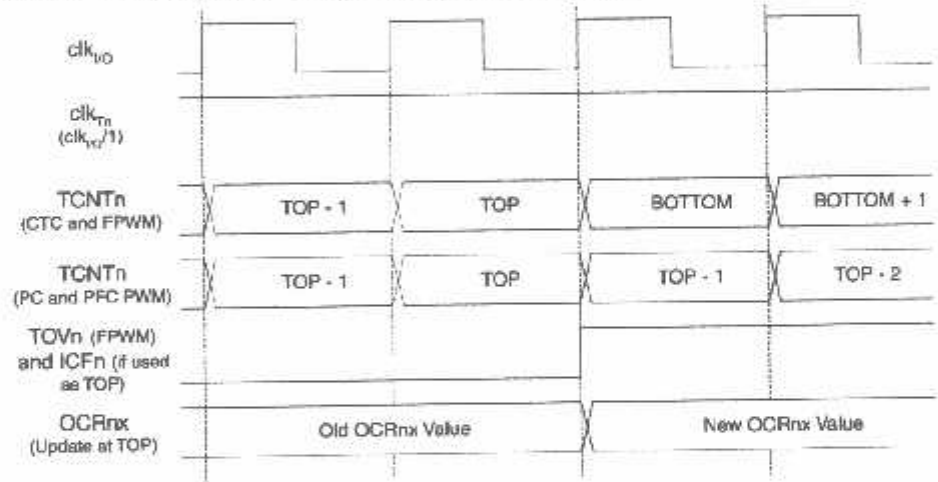
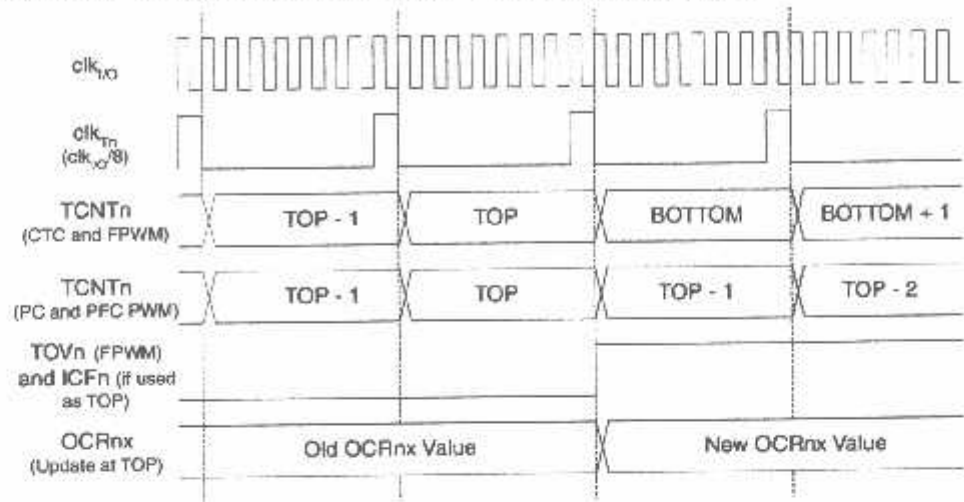


Figure 52 shows the same timing data, but with the prescaler enabled.

Figure 52. Timer/Counter Timing Diagram, with Prescaler ( $t_{clk_{IO}/8}$ )



### 16-bit Timer/Counter Register Description

#### Timer/Counter1 Control Register A – TCCR1A

Bit	7	6	5	4	3	2	1	0	TCCR1A
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value:	0	0	0	0	0	0	0	0	

- Bit 7:6 – COM1A1:0: Compare Output Mode for Channel A
- Bit 5:4 – COM1B1:0: Compare Output Mode for Channel B

The COM1A1:0 and COM1B1:0 control the Output Compare pins (OC1A and OC1B respectively) behavior. If one or both of the COM1A1:0 bits are written to one, the OC1A output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COM1B1:0 bit are written to one, the OC1B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the *Data Direction Register (DDR)* bit corresponding to the OC1A or OC1B pin must be set in order to enable the output driver.

When the OC1A or OC1B is connected to the pin, the function of the COM1x1:0 bits is dependent of the WGM13:0 bits setting. Table 45 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to a normal or a CTC mode (non-PWM).

Table 45. Compare Output Mode, non-PWM

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on Compare Match.
1	0	Clear OC1A/OC1B on Compare Match (set output to low level).
1	1	Set OC1A/OC1B on Compare Match (set output to high level).

Table 46 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the fast PWM mode.

**Table 46.** Compare Output Mode, Fast PWM<sup>(1)</sup>

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at TOP.
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at TOP.

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. In this case the Compare Match is ignored, but the set or clear is done at TOP. See "Fast PWM Mode" on page 101 for more details.

Table 47 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the phase correct or the phase and frequency correct, PWM mode.

**Table 47.** Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM<sup>(1)</sup>

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 9 or 14: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match when up-counting. Set OC1A/OC1B on Compare Match when down-counting.
1	1	Set OC1A/OC1B on Compare Match when up-counting. Clear OC1A/OC1B on Compare Match when down-counting.

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. See "Phase Correct PWM Mode" on page 103 for more details.

- **Bit 3 – FOC1A: Force Output Compare for Channel A**
- **Bit 2 – FOC1B: Force Output Compare for Channel B**

The FOC1A/FOC1B bits are only active when the WGM13:0 bits specify a non-PWM mode. However, for ensuring compatibility with future devices, these bits must be set to zero when TCCR1A is written when operating in a PWM mode. When writing a logical one to the FOC1A/FOC1B bit, an immediate Compare Match is forced on the waveform generation unit. The OC1A/OC1B output is changed according to its COM1x1:0 bits setting. Note that the FOC1A/FOC1B bits are implemented as strobes. Therefore it is the value present in the COM1x1:0 bits that determine the effect of the forced compare.

A FOC1A/FOC1B strobe will not generate any interrupt nor will it clear the timer in Clear Timer on Compare Match (CTC) mode using OCR1A as TOP.

The FOC1A/FOC1B bits are always read as zero.

- **Bit 1:0 – WGM11:0: Waveform Generation Mode**

Combined with the WGM13:2 bits found in the TCCR1B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 48. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare Match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. See "Modes of Operation" on page 100.

**Table 48. Waveform Generation Mode Bit Description<sup>(1)</sup>**

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Timer/Counter1 Control Register B – TCCR1B

Bit	7	6	5	4	3	2	1	0	TCCR1B
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 7 – ICNC1: Input Capture Noise Canceler**

Setting this bit (to one) activates the Input Capture Noise Canceler. When the Noise Canceler is activated, the input from the Input Capture Pin (ICP1) is filtered. The filter function requires four successive equal valued samples of the ICP1 pin for changing its output. The Input Capture is therefore delayed by four oscillator cycles when the noise canceler is enabled.

• **Bit 6 – ICES1: Input Capture Edge Select**

This bit selects which edge on the Input Capture Pin (ICP1) is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICES1 setting, the counter value is copied into the Input Capture Register (ICR1). The event will also set the Input Capture Flag (ICF1), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

When the ICR1 is used as TOP value (see description of the WGM13:0 bits located in the TCCR1A and the TCCR1B Register), the ICP1 is disconnected, and consequently, the Input Capture function is disabled.

• **Bit 5 – Reserved Bit**

This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCR1B is written.

• **Bit 4:3 – WGM13:2: Waveform Generation Mode**

See TCCR1A Register description.

• **Bit 2:0 – CS12:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter, see Figure 49 and Figure 50.

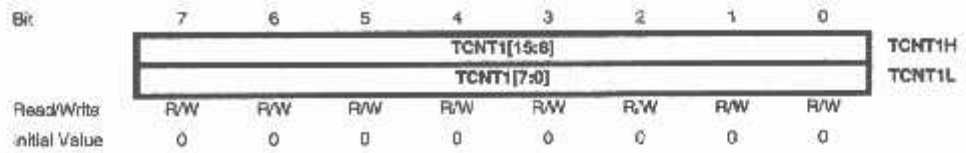
**Table 49.** Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{ICP1}/1$ (No prescaling)
0	1	0	$clk_{ICP1}/8$ (From prescaler)
0	1	1	$clk_{ICP1}/64$ (From prescaler)
1	0	0	$clk_{ICP1}/256$ (From prescaler)
1	0	1	$clk_{ICP1}/1024$ (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter1, transitions on the T1 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.



## Timer/Counter1 – TCNT1H and TCNT1L

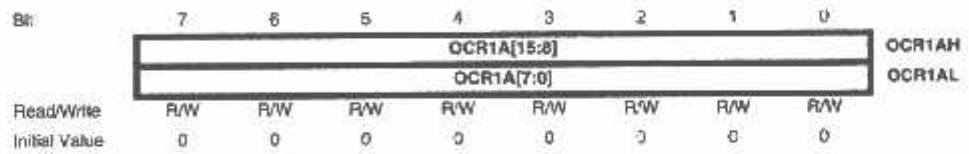


The two Timer/Counter I/O locations (TCNT1H and TCNT1L, combined TCNT1) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See "Accessing 16-bit Registers" on page 91.

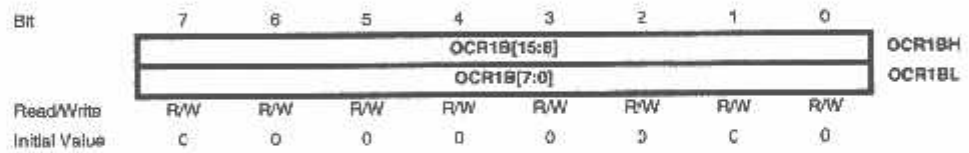
Modifying the counter (TCNT1) while the counter is running introduces a risk of missing a Compare Match between TCNT1 and one of the OCR1x Registers.

Writing to the TCNT1 Register blocks (removes) the Compare Match on the following timer clock for all compare units.

## Output Compare Register 1 A – OCR1AH and OCR1AL



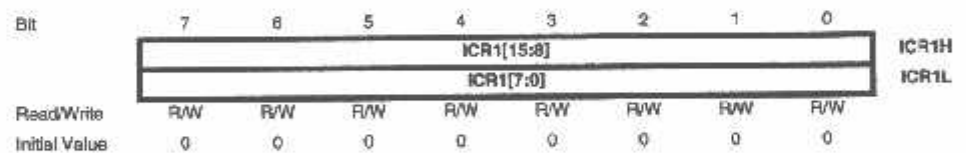
## Output Compare Register 1 B – OCR1BH and OCR1BL



The Output Compare Registers contain a 16-bit value that is continuously compared with the counter value (TCNT1). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC1x pin.

The Output Compare Registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See "Accessing 16-bit Registers" on page 91.

## Input Capture Register 1 – ICR1H and ICR1L



The Input Capture is updated with the counter (TCNT1) value each time an event occurs on the ICP1 pin (or optionally on the Analog Comparator output for Timer/Counter1). The Input Capture can be used for defining the counter TOP value.

The Input Capture Register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See "Accessing 16-bit Registers" on page 91.

## Timer/Counter Interrupt Mask Register – TIMSK<sup>(1)</sup>

Bit	7	6	5	4	3	2	1	0	TIMSK
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TCIE1	OCIE0	TOIE0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: 1. This register contains interrupt control bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.

- **Bit 5 – TICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 45) is executed when the ICF1 Flag, located in TIFR, is set.

- **Bit 4 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 output compare A match interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 45) is executed when the OCF1A Flag, located in TIFR, is set.

- **Bit 3 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable**

When this bit is written to one and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 output compare B match interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 45) is executed when the OCF1B Flag, located in TIFR, is set.

- **Bit 2 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

When this bit is written to one and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 overflow interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 45) is executed when the TOV1 Flag, located in TIFR, is set.

Timer/Counter Interrupt Flag Register – TIFR<sup>(1)</sup>

Bit	7	6	5	4	3	2	1	0	TIFR
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: 1. This register contains flag bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.

• **Bit 5 – ICF1: Timer/Counter1, Input Capture Flag**

This flag is set when a capture event occurs on the ICP1 pin. When the Input Capture Register (ICR1) is set by the WGM13:0 to be used as the TOP value, the ICF1 Flag is set when the counter reaches the TOP value.

ICF1 is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

• **Bit 4 – OCF1A: Timer/Counter1, Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register A (OCR1A).

Note that a Forced Output Compare (FOC1A) strobe will not set the OCF1A Flag.

OCF1A is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCF1A can be cleared by writing a logic one to its bit location.

• **Bit 3 – OCF1B: Timer/Counter1, Output Compare B Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register B (OCR1B).

Note that a Forced Output Compare (FOC1B) strobe will not set the OCF1B Flag.

OCF1B is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCF1B can be cleared by writing a logic one to its bit location.

• **Bit 2 – TOV1: Timer/Counter1, Overflow Flag**

The setting of this flag is dependent of the WGM13:0 bits setting. In normal and CTC modes, the TOV1 Flag is set when the timer overflows. Refer to Table 48 on page 111 for the TOV1 Flag behavior when using another WGM13:0 bit setting.

TOV1 is automatically cleared when the Timer/Counter1 Overflow Interrupt Vector is executed. Alternatively, TOV1 can be cleared by writing a logic one to its bit location.

## 8-bit Timer/Counter2 with PWM and Asynchronous Operation

Timer/Counter2 is a general purpose, single channel, 8-bit Timer/Counter module. The main features are:

- Single Channel Counter
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- Frequency Generator
- 10-bit Clock Prescaler
- Overflow and Compare Match Interrupt Sources (TOV2 and OCF2)
- Allows Clocking from External 32 kHz Watch Crystal Independent of the I/O Clock

### Overview

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 53. For the actual placement of I/O pins, refer to "Pinout ATmega8535" on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the "8-bit Timer/Counter Register Description" on page 127.

Figure 53. 8-bit Timer/Counter Block Diagram

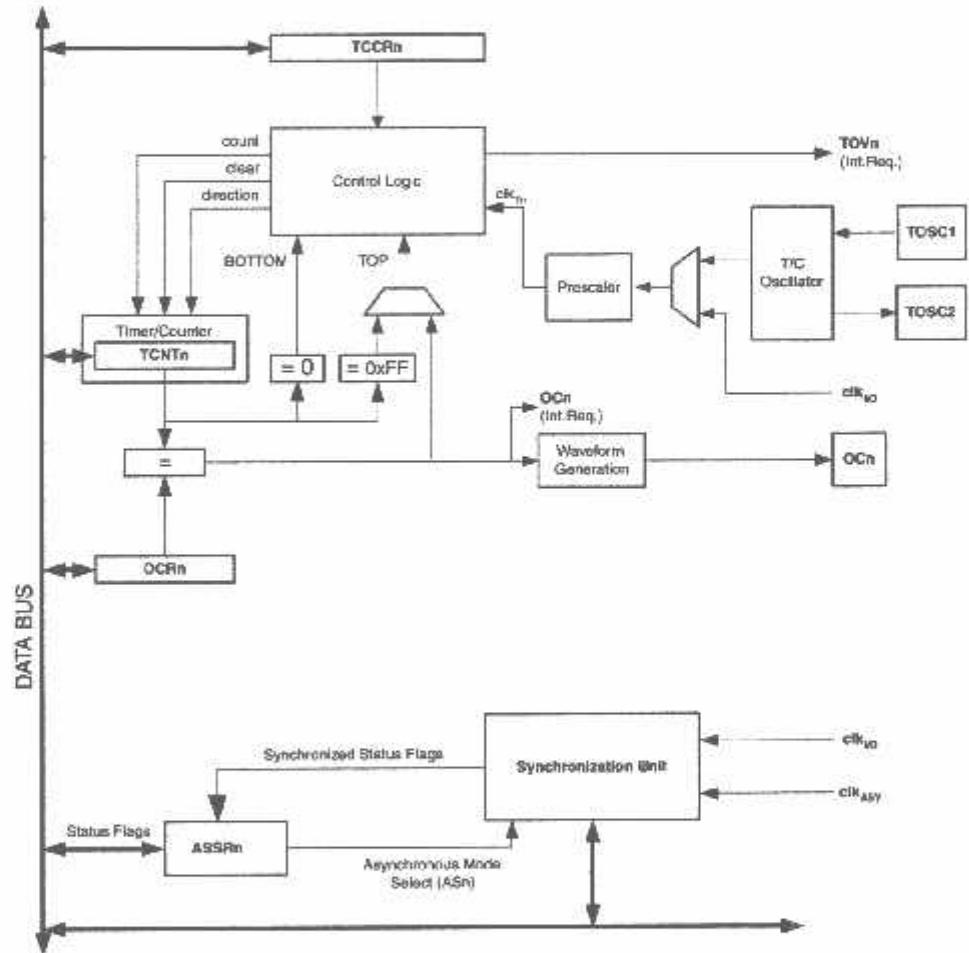
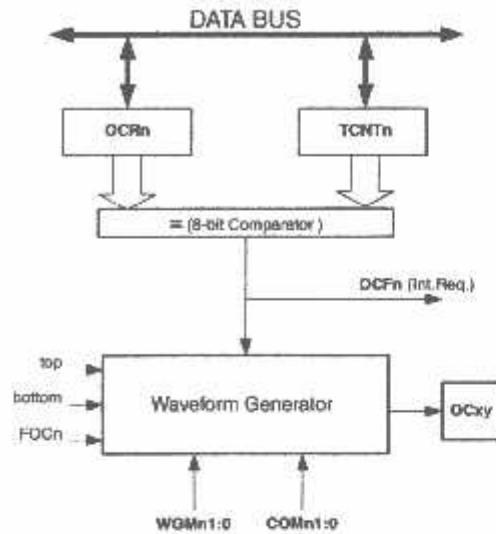


Figure 55. Output Compare Unit, Block Diagram



The OCR2 Register is double buffered when using any of the pulse width modulation (PWM) modes. For the normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR2 Compare Register to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR2 Register access may seem complex, but this is not the case. When the double buffering is enabled, the CPU has access to the OCR2 Buffer Register, and if double buffering is disabled the CPU will access the OCR2 directly.

**Force Output Compare**

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the Force Output Compare (FOC2) bit. Forcing Compare Match will not set the OCF2 Flag or reload/clear the timer, but the OC2 pin will be updated as if a real Compare Match had occurred (the COM21:0 bits settings define whether the OC2 pin is set, cleared or toggled).

**Compare Match Blocking by TCNT2 Write**

All CPU write operations to the TCNT2 Register will block any Compare Match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR2 to be initialized to the same value as TCNT2 without triggering an interrupt when the Timer/Counter clock is enabled.

**Using the Output Compare Unit**

Since writing TCNT2 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT2 when using the output compare channel, independently of whether the Timer/Counter is running or not. If the value written to TCNT2 equals the OCR2 value, the Compare Match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT2 value equal to BOTTOM when the counter is down-counting.

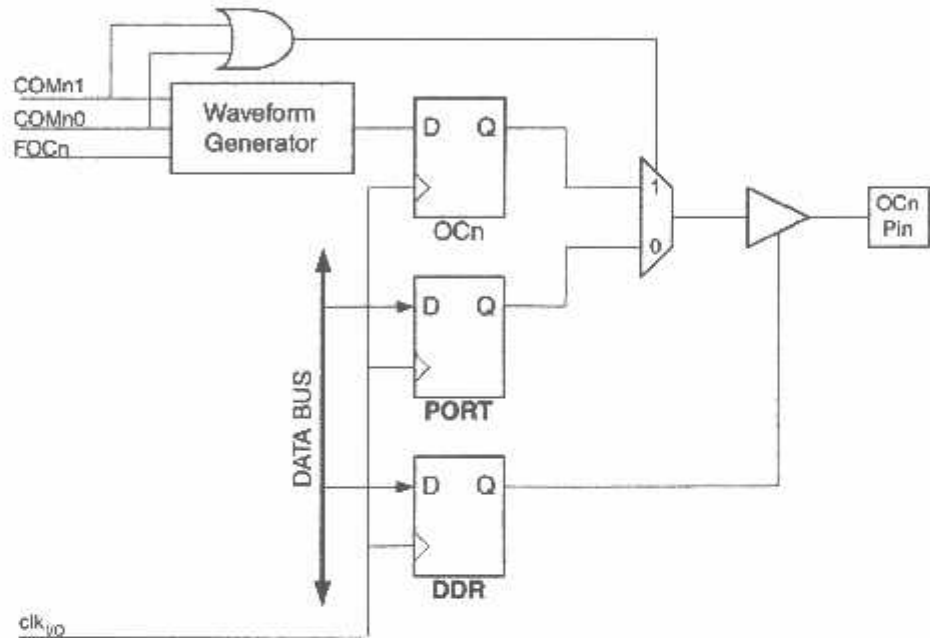
The setup of the OC2 should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC2 value is to use the force output compare (FOC2) strobe bit in Normal mode. The OC2 Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COM21:0 bits are not double buffered together with the compare value. Changing the COM21:0 bits will take effect immediately.

### Compare Match Output Unit

The Compare Output mode (COM21:0) bits have two functions. The Waveform Generator uses the COM21:0 bits for defining the Output Compare (OC2) state at the next Compare Match. Also, the COM21:0 bits control the OC2 pin output source. Figure 56 shows a simplified schematic of the logic affected by the COM21:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O port control registers (DDR and PORT) that are affected by the COM21:0 bits are shown. When referring to the OC2 state, the reference is for the internal OC2 Register, not the OC2 pin.

**Figure 56.** Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OC2) from the waveform generator if either of the COM21:0 bits are set. However, the OC2 pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OC2 pin (DDR\_OC2) must be set as output before the OC2 value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the output compare pin logic allows initialization of the OC2 state before the output is enabled. Note that some COM21:0 bit settings are reserved for certain modes of operation. See "8-bit Timer/Counter Register Description" on page 127.

### Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM21:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM21:0 = 0 tells the waveform generator that no action on the OC2 Register is to be performed on the next Compare Match. For compare output actions in the non-PWM modes refer to Table 52 on page 128. For fast PWM mode, refer to Table 53 on page 128, and for phase correct PWM refer to Table 54 on page 129.

A change of the COM21:0 bits state will take effect at the first Compare Match after the bits are written. For non-PWM modes, the action can be forced to have an immediate effect by using the FOC2 strobe bits.

## Modes of Operation

The mode of operation (i.e., the behavior of the Timer/Counter and the Output Compare pins) is defined by the combination of the Waveform Generation mode (WGM21:0) and Compare Output mode (COM21:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM21:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM21:0 bits control whether the output should be set, cleared, or toggled at a Compare Match (See "Compare Match Output Unit" on page 120.).

For detailed timing information refer to "Timer/Counter Timing Diagrams" on page 125.

### Normal Mode

The simplest mode of operation is the Normal mode (WGM21:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV2) will be set in the same timer clock cycle as the TCNT2 becomes zero. The TOV2 Flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV2 Flag, the timer resolution can be increased by software. There are no special cases to consider in the normal mode, a new counter value can be written anytime.

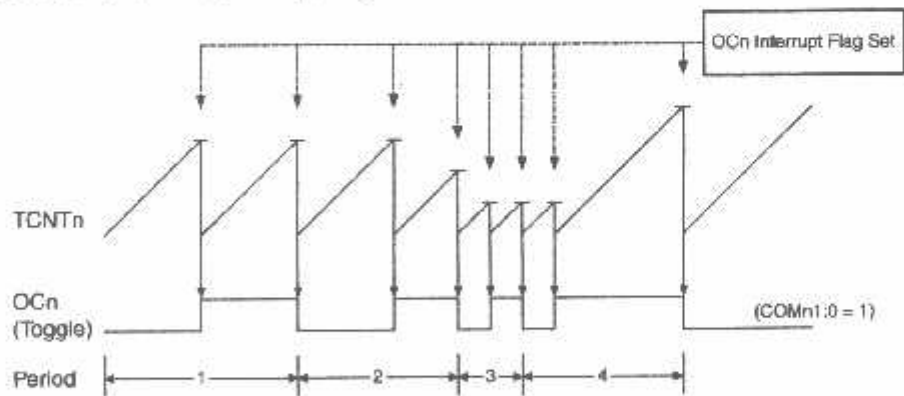
The output compare unit can be used to generate interrupts at some given time. Using the output compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

### Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM21:0 = 2), the OCR2 Register is used to manipulate the counter resolution. In CTC mode, the counter is cleared to zero when the counter value (TCNT2) matches the OCR2. The OCR2 defines the top value for the counter, hence also its resolution. This mode allows greater control of the Compare Match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 57. The counter value (TCNT2) increases until a Compare Match occurs between TCNT2 and OCR2, and then counter (TCNT2) is cleared.

**Figure 57. CTC Mode, Timing Diagram**



An interrupt can be generated each time the counter value reaches the TOP value by using the OCF2 Flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR2 is lower than the current value of TCNT2, the counter will miss the Compare Match. The counter will then have to count to its maximum value (0xFF) and wrap around starting at 0x00 before the Compare Match can occur.

For generating a waveform output in CTC mode, the OC2 output can be set to toggle its logical level on each Compare Match by setting the Compare Output mode bits to toggle mode (COM21:0 = 1). The OC2 value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of  $f_{OC2} = f_{clk\_IO}/2$  when OCR2 is set to zero (0x00). The waveform frequency is defined by the following equation:

$$f_{OCn} = \frac{f_{clk\_IO}}{2 \cdot N \cdot (1 + OCRn)}$$

The "N" variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

As for the Normal mode of operation, the TOV2 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

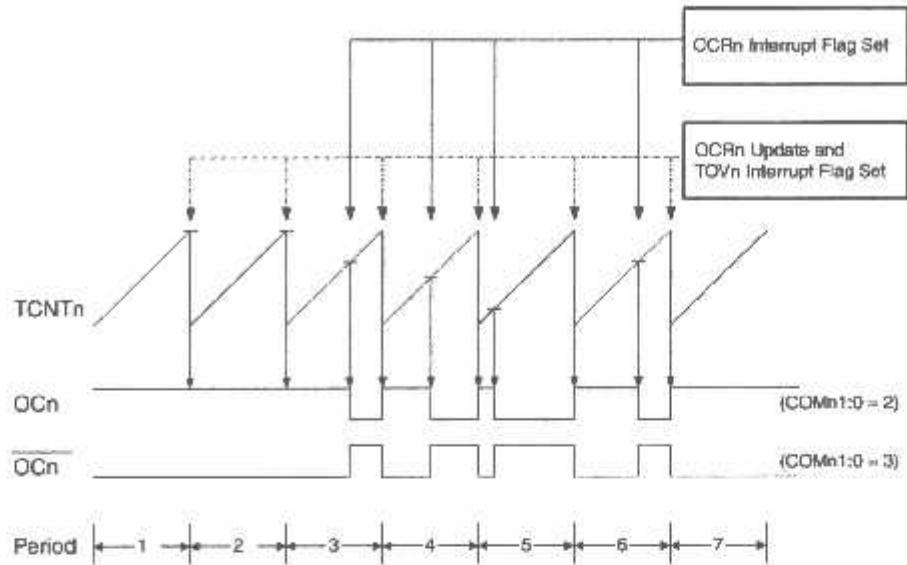
### Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM21:0 = 3) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to MAX then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC2) is cleared on the Compare Match between TCNT2 and OCR2, and set at BOTTOM. In inverting Compare Output mode, the output is set on Compare Match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that uses dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the MAX value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 58. The TCNT2 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT2 slopes represent compare matches between OCR2 and TCNT2.



Figure 58. Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV2) is set each time the counter reaches MAX. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC2 pin. Setting the COM21:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM21:0 to three (See Table 53 on page 128). The actual OC2 value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by setting (or clearing) the OC2 Register at the Compare Match between OCR2 and TCNT2, and clearing (or setting) the OC2 Register at the timer clock cycle the counter is cleared (changes from MAX to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$$

The 'N' variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

The extreme values for the OCR2 Register represent special cases when generating a PWM waveform output in the fast PWM mode. If the OCR2 is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR2 equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM21:0 bits.)

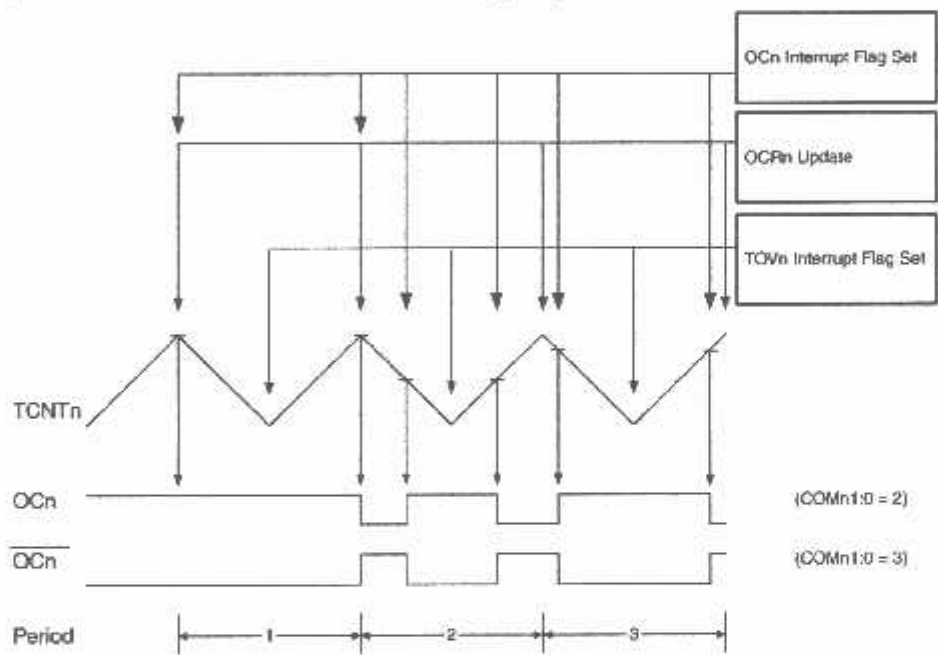
A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC2 to toggle its logical level on each Compare Match (COM21:0 = 1). The waveform generated will have a maximum frequency of  $f_{oc2} = f_{clk\_I/O}/2$  when OCR2 is set to zero. This feature is similar to the OC2 toggle in CTC mode, except the double buffer feature of the output compare unit is enabled in the fast PWM mode.

## Phase Correct PWM Mode

The Phase Correct PWM mode (WGM21:0 = 1) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC2) is cleared on the Compare Match between TCNT2 and OCR2 while up-counting, and set on the Compare Match while down-counting. In inverting output compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode is fixed to eight bits. In phase correct PWM mode the counter is incremented until the counter value matches MAX. When the counter reaches MAX, it changes the count direction. The TCNT2 value will be equal to MAX for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 59. The TCNT2 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT2 slopes represent compare matches between OCR2 and TCNT2.

**Figure 59.** Phase Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV2) is set each time the counter reaches BOTTOM. The Interrupt Flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC2 pin. Setting the COM21:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM21:0 to three (See Table 54 on page 129). The actual OC2 value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC2 Register at the Compare Match between OCR2 and TCNT2 when the counter increments, and setting (or clearing) the OC2 Register at Compare Match

between OCR2 and TCNT2 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnPCPWM} = \frac{f_{clk_{I/O}}}{N \cdot 510}$$

The ‘N’ variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

The extreme values for the OCR2 Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR2 is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

At the very start of period 2 in Figure 59 OCn has a transition from high to low even though there is no Compare Match. The point of this transition is to guarantee symmetry around BOTTOM. There are two cases that give a transition without Compare Match.

- OCR2 changes its value from MAX, like in Figure 59. When the OCR2 value is MAX the OCn pin value is the same as the result of a down-counting Compare Match. To ensure symmetry around BOTTOM the OCn value at MAX must correspond to the result of an up-counting Compare Match.
- The timer starts counting from a value higher than the one in OCR2, and for that reason misses the Compare Match and hence the OCn change that would have happened on the way up.

**Timer/Counter Timing Diagrams**

The following figures show the Timer/Counter in Synchronous mode, and the Timer Clock (clk<sub>T2</sub>) is therefore shown as a clock enable signal. In Asynchronous mode, clk<sub>I/O</sub> should be replaced by the Timer/Counter Oscillator clock. The figures include information on when interrupt flags are set. Figure 60 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

**Figure 60. Timer/Counter Timing Diagram, no Prescaling**

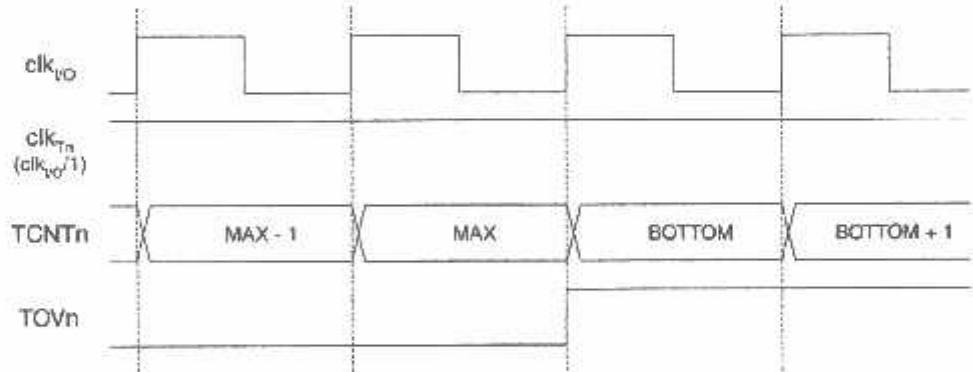


Figure 61 shows the same timing data, but with the prescaler enabled.

**Figure 61. Timer/Counter Timing Diagram, with Prescaler ( $f_{clk_{I/O}}/8$ )**

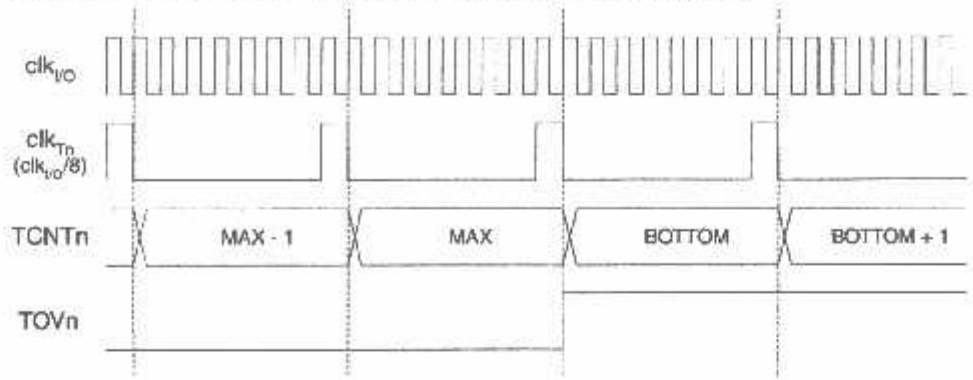


Figure 62 shows the setting of OCF2 in all modes except CTC mode.

**Figure 62. Timer/Counter Timing Diagram, Setting of OCF2, with Prescaler ( $f_{clk_{I/O}}/8$ )**

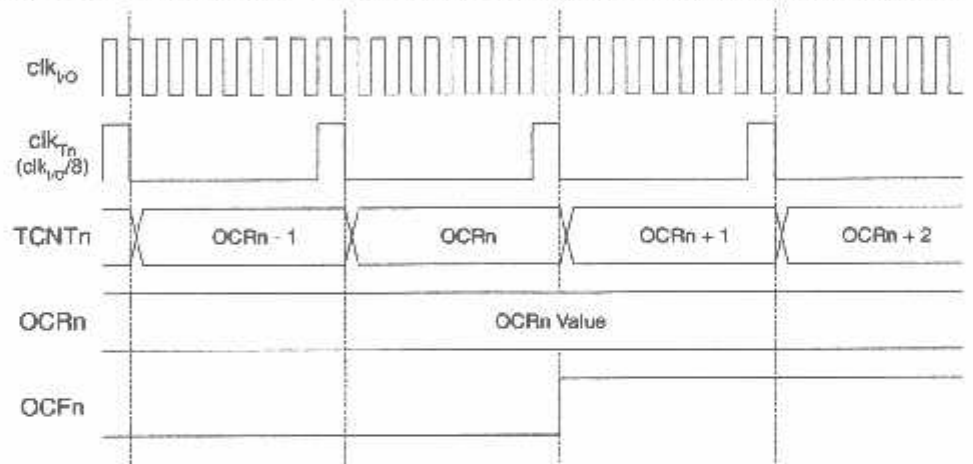
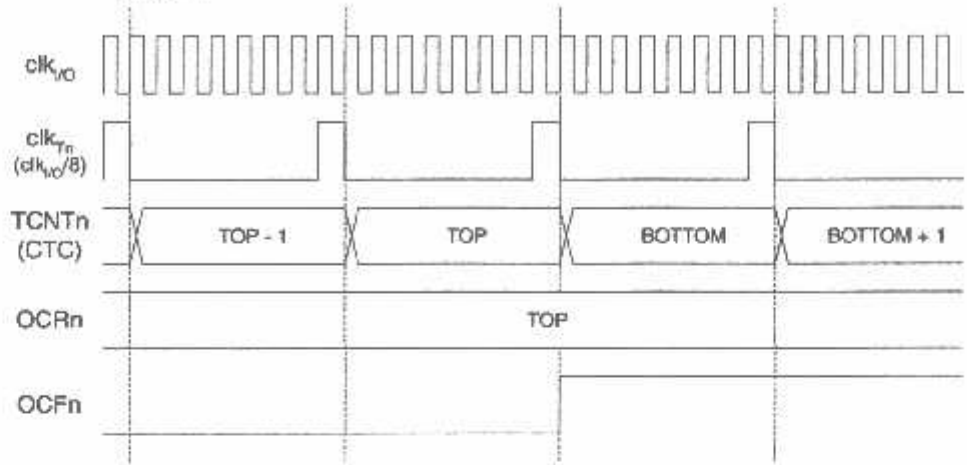


Figure 63 shows the setting of OCF2 and the clearing of TCNT2 in CTC mode.

**Figure 63.** Timer/Counter Timing Diagram, Clear Timer on Compare Match Mode, with Prescaler ( $f_{clk_{IO}}/8$ )



**8-bit Timer/Counter Register Description**

**Timer/Counter Control Register – TCCR2**

Bit	7	6	5	4	3	2	1	0	
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	TCCR2
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 7 – FOC2: Force Output Compare**

The FOC2 bit is only active when the WGM bits specify a non-PWM mode. However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR2 is written when operating in PWM mode. When writing a logical one to the FOC2 bit, an immediate Compare Match is forced on the Waveform Generation unit. The OC2 output is changed according to its COM21:0 bits setting. Note that the FOC2 bit is implemented as a strobe. Therefore it is the value present in the COM21:0 bits that determines the effect of the forced compare.

A FOC2 strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR2 as TOP.

The FOC2 bit is always read as zero.

• **Bit 6, 3 – WGM21:0: Waveform Generation Mode**

These bits control the counting sequence of the counter, the source for the maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode, Clear Timer on Compare Match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes. See Table 51 and "Modes of Operation" on page 121.

**Table 51. Waveform Generation Mode Bit Description<sup>(1)</sup>**

Mode	WGM21 (CTC2)	WGM20 (PWM2)	Timer/Counter Mode of Operation	TOP	Update of OCR2	TOV2 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR2	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Note: 1. The CTC2 and PWM2 bit definition names are now obsolete. Use the WGM21:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

▪ **Bit 5:4 – COM21:0: Compare Match Output Mode**

These bits control the Output Compare pin (OC2) behavior. If one or both of the COM21:0 bits are set, the OC2 output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to OC2 pin must be set in order to enable the output driver.

When OC2 is connected to the pin, the function of the COM21:0 bits depends on the WGM21:0 bit setting. Table 52 shows the COM21:0 bit functionality when the WGM21:0 bits are set to a Normal or CTC mode (non-PWM).

**Table 52. Compare Output Mode, non-PWM Mode**

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Toggle OC2 on Compare Match.
1	0	Clear OC2 on Compare Match.
1	1	Set OC2 on Compare Match.

Table 53 shows the COM21:0 bit functionality when the WGM21:0 bits are set to fast PWM mode.

**Table 53. Compare Output Mode, Fast PWM Mode<sup>(1)</sup>**

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on Compare Match, set OC2 at TOP.
1	1	Set OC2 on Compare Match, clear OC2 at TOP.

Note: 1. A special case occurs when OCR2 equals TOP and COM21 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See "Fast PWM Mode" on page 122 for more details.

Table 54 shows the COM21:0 bit functionality when the WGM21:0 bits are set to phase correct PWM mode.

**Table 54.** Compare Output Mode, Phase Correct PWM Mode<sup>(1)</sup>

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on Compare Match when up-counting. Set OC2 on Compare Match when down-counting.
1	1	Set OC2 on Compare Match when up-counting. Clear OC2 on Compare Match when down-counting.

Note: 1. A special case occurs when OCR2 equals TOP and COM21 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See "Phase Correct PWM Mode" on page 124 for more details.

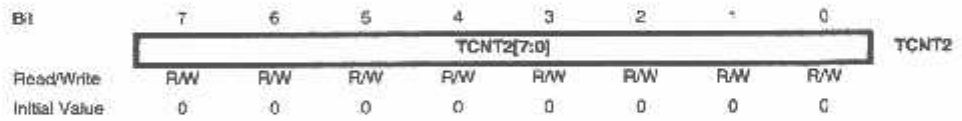
• **Bit 2:0 – CS22:0: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter, see Table 55.

**Table 55.** Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{T2S}$ (No prescaling)
0	1	0	$clk_{T2S}/8$ (From prescaler)
0	1	1	$clk_{T2S}/32$ (From prescaler)
1	0	0	$clk_{T2S}/64$ (From prescaler)
1	0	1	$clk_{T2S}/128$ (From prescaler)
1	1	0	$clk_{T2S}/256$ (From prescaler)
1	1	1	$clk_{T2S}/1024$ (From prescaler)

**Timer/Counter Register – TCNT2**



The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT2 Register blocks (removes) the Compare Match on the following timer clock. Modifying the counter (TCNT2) while the counter is running, introduces a risk of missing a Compare Match between TCNT2 and the OCR2 Register.

## Output Compare Register – OCR2

Bit	7	6	5	4	3	2	1	0	
	OCR2[7:0]								OCR2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Output Compare Register contains an 8-bit value that is continuously compared with the counter value (TCNT2). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC2 pin.

## Asynchronous Operation of the Timer/Counter

### Asynchronous Status Register – ASSR

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	AS2	TCN2UB	OCR2UB	TCR2UB	ASSR
Read/Write	R	R	R	R	R/W	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 3 – AS2: Asynchronous Timer/Counter2**

When AS2 is written to zero, Timer/Counter2 is clocked from the I/O clock,  $clk_{I/O}$ . When AS2 is written to one, Timer/Counter2 is clocked from a crystal Oscillator connected to the Timer Oscillator1 (TOSC1) pin. When the value of AS2 is changed, the contents of TCNT2, OCR2, and TCCR2 might be corrupted.

- **Bit 2 – TCN2UB: Timer/Counter2 Update Busy**

When Timer/Counter2 operates asynchronously and TCNT2 is written, this bit becomes set. When TCNT2 has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that TCNT2 is ready to be updated with a new value.

- **Bit 1 – OCR2UB: Output Compare Register 2 Update Busy**

When Timer/Counter2 operates asynchronously and OCR2 is written, this bit becomes set. When OCR2 has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that OCR2 is ready to be updated with a new value.

- **Bit 0 – TCR2UB: Timer/Counter Control Register 2 Update Busy**

When Timer/Counter2 operates asynchronously and TCCR2 is written, this bit becomes set. When TCCR2 has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that TCCR2 is ready to be updated with a new value.

If a write is performed to any of the three Timer/Counter2 Registers while its update busy flag is set, the updated value might get corrupted and cause an unintentional interrupt to occur.

The mechanisms for reading TCNT2, OCR2, and TCCR2 are different. When reading TCNT2, the actual timer value is read. When reading OCR2 or TCCR2, the value in the temporary storage register is read.



## Asynchronous Operation of Timer/Counter2

When Timer/Counter2 operates asynchronously, some considerations must be taken.

- **Warning:** When switching between asynchronous and synchronous clocking of Timer/Counter2, the timer registers TCNT2, OCR2, and TCCR2 might be corrupted. A safe procedure for switching clock source is:
  1. Disable the Timer/Counter2 interrupts by clearing OCIE2 and TOIE2.
  2. Select clock source by setting AS2 as appropriate.
  3. Write new values to TCNT2, OCR2, and TCCR2.
  4. To switch to asynchronous operation: Wait for TCN2UB, OCR2UB, and TCR2UB.
  5. Clear the Timer/Counter2 Interrupt Flags.
  6. Enable interrupts, if needed.
- The Oscillator is optimized for use with a 32.768 kHz watch crystal. Applying an external clock to the TOSC1 pin may result in incorrect Timer/Counter2 operation. The CPU main clock frequency must be more than four times the oscillator frequency.
- When writing to one of the registers TCNT2, OCR2, or TCCR2, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the temporary register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that, e.g., writing to TCNT2 does not disturb an OCR2 write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR – has been implemented.
- When entering Power-save or Extended Standby mode after having written to TCNT2, OCR2, or TCCR2, the user must wait until the written register has been updated if Timer/Counter2 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare 2 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR2 or TCNT2. If the write cycle is not finished, and the MCU enters sleep mode before the OCR2UB bit returns to zero, the device will never receive a Compare Match interrupt, and the MCU will not wake up.
- If Timer/Counter2 is used to wake the device up from Power-save or Extended Standby mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or Extended Standby mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR2, TCNT2, or OCR2.
  2. Wait until the corresponding Update Busy Flag in ASSR returns to zero.
  3. Enter Power-save or Extended Standby mode.
- When the asynchronous operation is selected, the 32.768 kHz Oscillator for Timer/Counter2 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter2 after Power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter2 Registers must be considered lost after a wake-up from Power-

down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.

- Description of wake-up from Power-save or Extended Standby mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT2 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT2 is clocked on the asynchronous TOSC clock, reading TCNT2 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock (clk<sub>I/O</sub>) again becomes active, TCNT2 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT2 is thus as follows:
  1. Write any value to either of the registers OCR2 or TCCR2.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT2.
- During asynchronous operation, the synchronization of the interrupt flags for the asynchronous timer takes three processor cycles plus one timer cycle. The timer is therefore advanced by at least one before the processor can read the timer value causing the setting of the Interrupt Flag. The output compare pin is changed on the timer clock and is not synchronized to the processor clock.

#### Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

#### • Bit 7 – OCIE2: Timer/Counter2 Output Compare Match Interrupt Enable

When the OCIE2 bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Compare Match interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter2 occurs (i.e., when the OCF2 bit is set in the Timer/Counter Interrupt Flag Register – TIFR).

#### • Bit 6 – TOIE2: Timer/Counter2 Overflow Interrupt Enable

When the TOIE2 bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter2 occurs (i.e., when the TOV2 bit is set in the Timer/Counter Interrupt Flag Register – TIFR).

## Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	TIFR
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### • Bit 7 – OCF2: Output Compare Flag 2

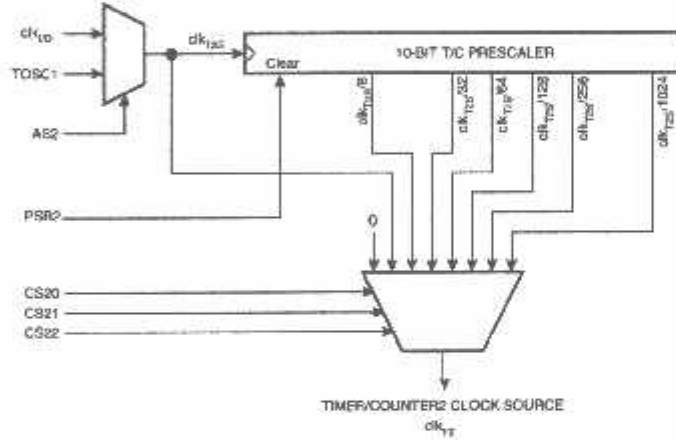
The OCF2 bit is set (one) when a Compare Match occurs between the Timer/Counter2 and the data in OCR2 – Output Compare Register 2. OCF2 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF2 is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE2 (Timer/Counter2 Compare Match interrupt enable), and OCF2 are set (one), the Timer/Counter2 Compare Match Interrupt is executed.

### • Bit 6 – TOV2: Timer/Counter2 Overflow Flag

The TOV2 bit is set (one) when an overflow occurs in Timer/Counter2. TOV2 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV2 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE2 (Timer/Counter2 Overflow Interrupt Enable), and TOV2 are set (one), the Timer/Counter2 Overflow interrupt is executed. In PWM mode, this bit is set when Timer/Counter2 changes counting direction at 0x00.

## Timer/Counter Prescaler

Figure 64. Prescaler for Timer/Counter2



The clock source for Timer/Counter2 is named  $clk_{T2S}$ .  $clk_{T2S}$  is, by default, connected to the main system I/O clock  $clk_{I/O}$ . By setting the AS2 bit in ASSR, Timer/Counter2 is asynchronously clocked from the TOSC1 pin. This enables use of Timer/Counter2 as a Real Time Counter (RTC). When AS2 is set, pins TOSC1 and TOSC2 are disconnected from Port C. A crystal can then be connected between the TOSC1 and TOSC2 pins to serve as an independent clock source for Timer/Counter2. The Oscillator is optimized for use with a 32.768 kHz crystal. Applying an external clock source to TOSC1 is not recommended.

For Timer/Counter2, the possible prescaled selections are:  $clk_{T2S}/8$ ,  $clk_{T2S}/32$ ,  $clk_{T2S}/64$ ,  $clk_{T2S}/128$ ,  $clk_{T2S}/256$ , and  $clk_{T2S}/1024$ . Additionally,  $clk_{T2S}$  as well as 0 (stop) may be selected. Setting the PSR2 bit in SFIOR resets the prescaler. This allows the user to operate with a predictable prescaler.

**Special Function IO Register – SFIOR**

Bit	7	6	5	4	3	2	1	0	SFIOR
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 1 – PSR2: Prescaler Reset Timer/Counter2**

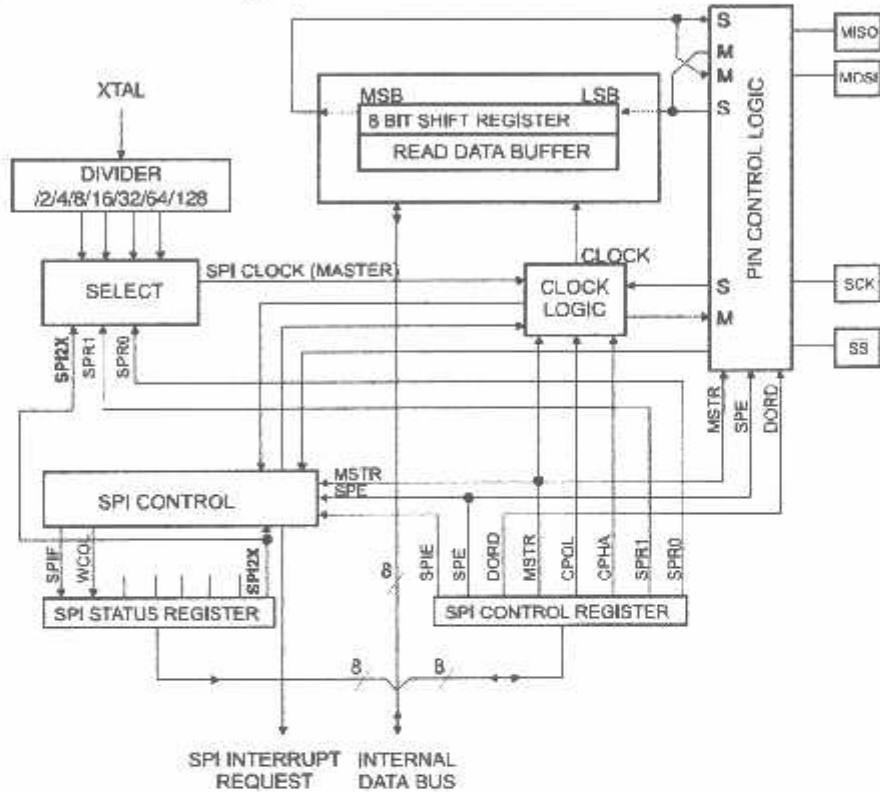
When this bit is written to one, the Timer/Counter2 prescaler will be reset. The bit will be cleared by hardware after the operation is performed. Writing a zero to this bit will have no effect. This bit will always be read as zero if Timer/Counter2 is clocked by the internal CPU clock. If this bit is written when Timer/Counter2 is operating in asynchronous mode, the bit will remain one until the prescaler has been reset.

Serial Peripheral Interface – SPI

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the ATmega8535 and peripheral devices or between several AVR devices. The ATmega8535 SPI includes the following features:

- Full Duplex, Three-wire Synchronous Data Transfer
- Master or Slave Operation
- LSB First or MSB First Data Transfer
- Seven Programmable Bit Rates
- End of Transmission Interrupt Flag
- Write Collision Flag Protection
- Wake-up from Idle Mode
- Double Speed (CK/2) Master SPI Mode

Figure 65. SPI Block Diagram<sup>(1)</sup>



Note: 1. Refer to Figure 1 on page 2, and Table 26 on page 59 for SPI pin placement.

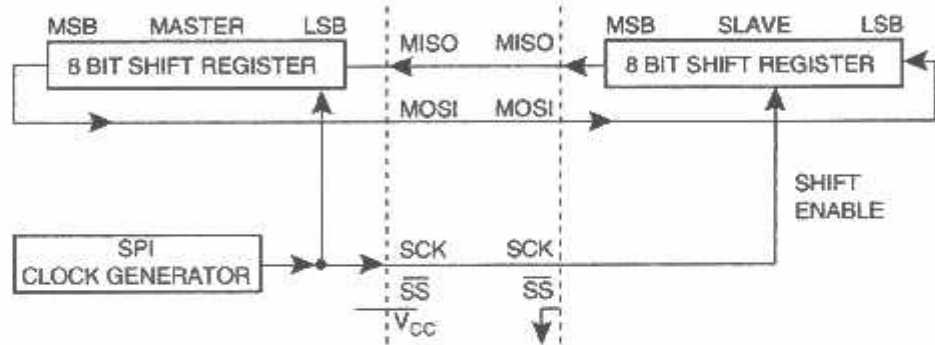
The interconnection between Master and Slave CPUs with SPI is shown in Figure 66. The system consists of two Shift Registers, and a Master clock generator. The SPI Master initiates the communication cycle when pulling low the Slave Select  $\overline{SS}$  pin of the desired Slave. Master and Slave prepare the data to be sent in their respective Shift Registers, and the Master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from Master to Slave on the Master Out – Slave In, MOSI, line, and from Slave to Master on the Master In – Slave Out, MISO, line. After each data packet, the Master will synchronize the Slave by pulling high the Slave Select,  $\overline{SS}$ , line.

When configured as a Master, the SPI interface has no automatic control of the  $\overline{SS}$  line. This must be handled by user software before communication can start. When this is

done, writing a byte to the SPI Data Register starts the SPI Clock Generator, and the hardware shifts the eight bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of Transmission Flag (SPIF). If the SPI Interrupt Enable bit (SPIE) in the SPCR Register is set, an interrupt is requested. The Master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the Slave Select,  $\overline{SS}$  line. The last incoming byte will be kept in the buffer register for later use.

When configured as a Slave, the SPI interface will remain sleeping with MISO tri-stated as long as the  $\overline{SS}$  pin is driven high. In this state, software may update the contents of the SPI Data Register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the  $\overline{SS}$  pin is driven low. As one byte has been completely shifted, the end of Transmission Flag, SPIF is set. If the SPI Interrupt Enable bit, SPIE, in the SPCR Register is set, an interrupt is requested. The Slave may continue to place new data to be sent into SPDR before reading the incoming data. The last incoming byte will be kept in the buffer register for later use.

Figure 66. SPI Master-Slave Interconnection



The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI Data Register before the entire shift cycle is completed. When receiving data, however, a received character must be read from the SPI Data Register before the next character has been completely shifted in. Otherwise, the first byte is lost.

In SPI Slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the frequency of the SPI clock should never exceed  $f_{osc}/4$ .

When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and  $\overline{SS}$  pins is overridden according to Table 56. For more details on automatic port overrides, refer to "Alternate Port Functions" on page 56.

Table 56. SPI Pin Overrides<sup>(1)</sup>

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

Note: 1. See "Alternate Functions Of Port B" on page 59 for a detailed description of how to define the direction of the user defined SPI pins.

The following code examples show how to initialize the SPI as a Master and how to perform a simple transmission. `DDR_SPI` in the examples must be replaced by the actual Data Direction Register controlling the SPI pins. `DD_MOSI`, `DD_MISO` and `DD_SCK` must be replaced by the actual data direction bits for these pins. For example, if MOSI is placed on pin PB5, replace `DD_MOSI` with `DDB5`, and `DDR_SPI` with `DDRB`.

Assembly Code Example <sup>(1)</sup>
<pre> SPI_MasterInit: ; Set MOSI and SCK output, all others input ldi r17,(1&lt;&lt;DD_MOSI) (1&lt;&lt;DD_SCK) out DDR_SPI,r17 ; Enable SPI, Master, set clock rate fck/16 ldi r17,(1&lt;&lt;SPE) (1&lt;&lt;MSTR) (1&lt;&lt;SPR0) out SPCR,r17 ret  SPI_MasterTransmit: ; Start transmission of data (r16) out SPDR,r16 Wait_Transmit: ; Wait for transmission complete sbis SPSR,SPIF rjmp Wait_Transmit ret         </pre>
C Code Example <sup>(1)</sup>
<pre> void SPI_MasterInit(void) {     /* Set MOSI and SCK output, all others input */     DDR_SPI = (1&lt;&lt;DD_MOSI) (1&lt;&lt;DD_SCK);     /* Enable SPI, Master, set clock rate fck/16 */     SPCR = (1&lt;&lt;SPE) (1&lt;&lt;MSTR) (1&lt;&lt;SPR0); }  void SPI_MasterTransmit(char cData) {     /* Start transmission */     SPDR = cData;     /* Wait for transmission complete */     while(!(SPSR &amp; (1&lt;&lt;SPIF)))         ; }         </pre>

Note: 1. See "About Code Examples" on page 6.

The following code examples show how to initialize the SPI as a Slave and how to perform a simple reception.

#### Assembly Code Example<sup>(1)</sup>

```

SPI_SlaveInit:
    ; Set MISO output, all others input
    ldi r17,(1<<DD_MISO)
    out DDR_SPI,r17
    ; Enable SPI
    ldi r17,(1<<SPE)
    out SPCR,r17
    ret

SPI_SlaveReceive:
    ; Wait for reception complete
    sbis SPSR,SPIF
    rjmp SPI_SlaveReceive
    ; Read received data and return
    in r16,SPDR
    ret
    
```

#### C Code Example<sup>(1)</sup>

```

void SPI_SlaveInit(void)
{
    /* Set MISO output, all others input */
    DDR_SPI = (1<<DD_MISO);
    /* Enable SPI */
    SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
    /* Wait for reception complete */
    while(!(SPSR & (1<<SPIF)))
        ;
    /* Return Data Register */
    return SPDR;
}
    
```

Note: 1. See "About Code Examples" on page 6.



**$\overline{SS}$  Pin Functionality**

**Slave Mode**

When the SPI is configured as a Slave, the Slave Select ( $\overline{SS}$ ) pin is always input. When  $\overline{SS}$  is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs. When  $\overline{SS}$  is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI logic will be reset once the  $\overline{SS}$  pin is driven high.

The  $\overline{SS}$  pin is useful for packet/byte synchronization to keep the Slave bit counter synchronous with the Master clock generator. When the  $\overline{SS}$  pin is driven high, the SPI Slave will immediately reset the send and receive logic, and drop any partially received data in the Shift Register.

**Master Mode**

When the SPI is configured as a Master (MSTR in SPCR is set), the user can determine the direction of the  $\overline{SS}$  pin.

If  $\overline{SS}$  is configured as an output, the pin is a general output pin which does not affect the SPI system. Typically, the pin will be driving the  $\overline{SS}$  pin of the SPI Slave.

If  $\overline{SS}$  is configured as an input, it must be held high to ensure Master SPI operation. If the  $\overline{SS}$  pin is driven low by peripheral circuitry when the SPI is configured as a Master with the  $\overline{SS}$  pin defined as an input, the SPI system interprets this as another Master selecting the SPI as a Slave and starting to send data to it. To avoid bus contention, the SPI system takes the following actions:

1. The MSTR bit in SPCR is cleared and the SPI system becomes a Slave. As a result of the SPI becoming a Slave, the MOSI and SCK pins become inputs.
2. The SPIF Flag in SPSR is set, and if the SPI interrupt is enabled, and the I-bit in SREG is set, the interrupt routine will be executed.

Thus, when interrupt-driven SPI transmission is used in Master mode, and there exists a possibility that  $\overline{SS}$  is driven low, the interrupt should always check that the MSTR bit is still set. If the MSTR bit has been cleared by a Slave Select, it must be set by the user to re-enable SPI Master mode.

**SPI Control Register – SPCR**

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 7 – SPIE: SPI Interrupt Enable**

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the If the Global Interrupt Enable bit in SREG is set.

• **Bit 6 – SPE: SPI Enable**

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

• **Bit 5 – DORD: Data Order**

When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

• **Bit 4 – MSTR: Master/Slave Select**

This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If  $\overline{SS}$  is configured as an input and is driven low while MSTR is set, MSTR will



be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.

• **Bit 3 – CPOL: Clock Polarity**

When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. Refer to Figure 67 and Figure 68 for an example. The CPOL functionality is summarized below:

**Table 57. CPOL Functionality**

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

• **Bit 2 – CPHA: Clock Phase**

The settings of the Clock Phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to Figure 67 and Figure 68 for an example. The CPOL functionality is summarized below:

**Table 58. CPHA Functionality**

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

• **Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0**

These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on the Slave. The relationship between SCK and the Oscillator Clock frequency  $f_{osc}$  is shown in the following table:

**Table 59. Relationship between SCK and the Oscillator Frequency**

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

## SPI Status Register – SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	-	-	-	-	-	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIF: SPI Interrupt Flag**

When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If  $\overline{SS}$  is an Input and is driven low when the SPI is in Master mode, this will also set the SPIF Flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the SPI Data Register (SPDR).

- **Bit 6 – WCOL: Write COLLision flag**

The WCOL bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

- **Bit 5..1 – Res: Reserved Bits**

These bits are reserved bits in the ATmega8535 and will always read as zero.

- **Bit 0 – SPI2X: Double SPI Speed Bit**

When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode (see Table 59). This means that the minimum SCK period will be two CPU clock periods. When the SPI is configured as Slave, the SPI is only guaranteed to work at  $f_{osc}/4$  or lower.

The SPI interface on the ATmega8535 is also used for program memory and EEPROM downloading or uploading. See page 249 for Serial Programming and verification.

## SPI Data Register – SPDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

The SPI Data Register is a read/write register used for data transfer between the Register File and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register receive buffer to be read.

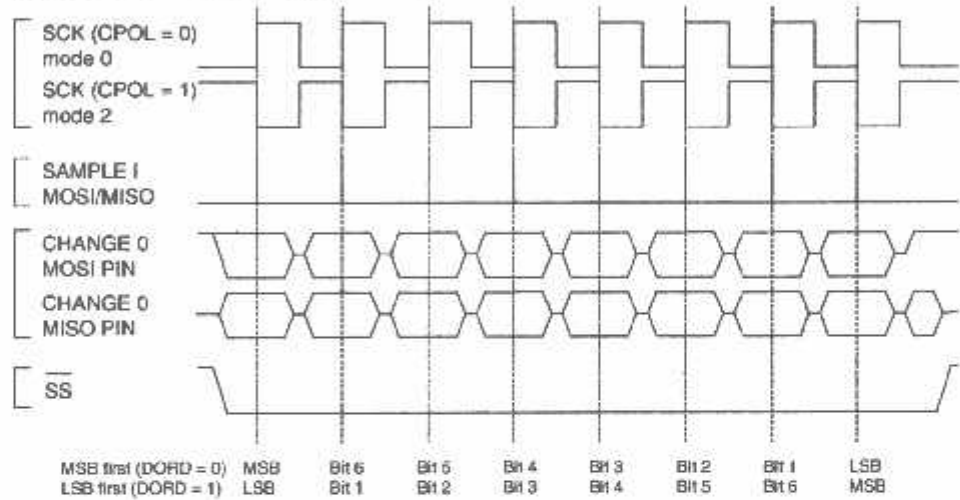
## Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 67 and Figure 68. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. This is clearly seen by summarizing Table 57 and Table 58, as done below:

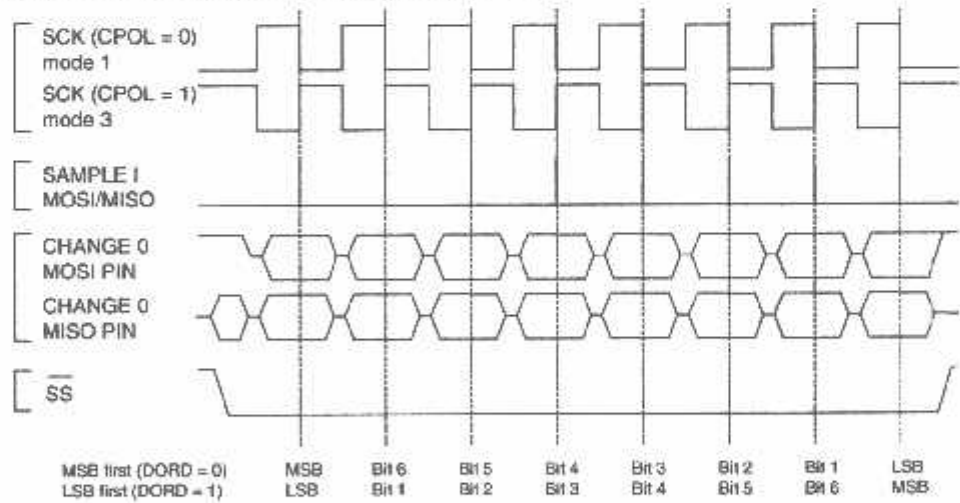
**Table 60.** CPOL Functionality

	Leading Edge	Trailing Edge	SPI Mode
CPOL=0, CPHA=0	Sample (Rising)	Setup (Falling)	0
CPOL=0, CPHA=1	Setup (Rising)	Sample (Falling)	1
CPOL=1, CPHA=0	Sample (Falling)	Setup (Rising)	2
CPOL=1, CPHA=1	Setup (Falling)	Sample (Rising)	3

**Figure 67.** SPI Transfer Format with CPHA = 0



**Figure 68.** SPI Transfer Format with CPHA = 1



**USART**

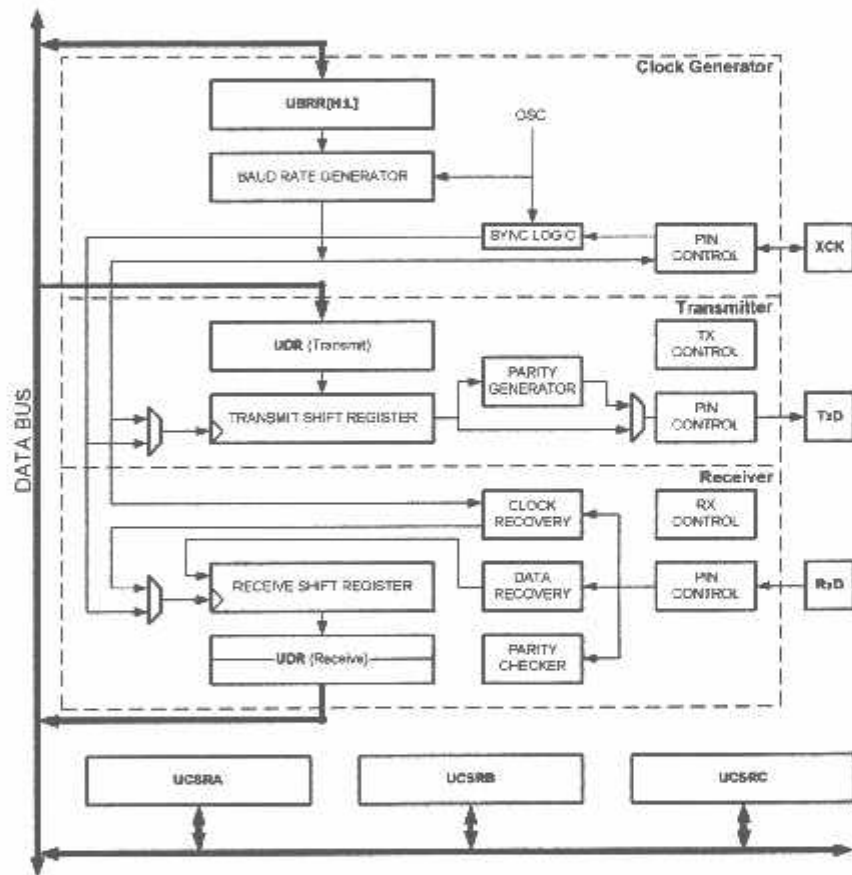
The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. The main features are:

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Master or Slave Clocked Synchronous Operation
- High Resolution Baud Rate Generator
- Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data OverRun Detection
- Framing Error Detection
- Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode

**Overview**

A simplified block diagram of the USART Transmitter is shown in Figure 69. CPU accessible I/O Registers and I/O pins are shown in bold.

**Figure 69. USART Block Diagram<sup>(1)</sup>**



Note: 1. Refer to Figure 1 on page 2, Table 34 on page 65, and Table 28 on page 61 for USART pin placement.

The dashed boxes in the block diagram separate the three main parts of the USART (listed from the top): Clock Generator, Transmitter and Receiver. Control registers are shared by all units. The clock generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCK (Transfer Clock) pin is only used by Synchronous Transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, Parity Generator and control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the Receiver includes a Parity Checker, control logic, a Shift Register and a two level receive buffer (UDR). The Receiver supports the same frame formats as the Transmitter, and can detect frame error, data overrun and parity errors.

#### AVR USART vs. AVR UART – Compatibility

The USART is fully compatible with the AVR UART regarding:

- Bit locations inside all USART Registers
- Baud Rate Generation
- Transmitter Operation
- Transmit Buffer Functionality
- Receiver Operation

However, the receive buffering has two improvements that will affect the compatibility in some special cases:

- A second buffer register has been added. The two buffer registers operate as a circular FIFO buffer. Therefore the UDR must only be read once for each incoming data! More important is the fact that the Error Flags (FE and DOR) and the ninth data bit (RXB8) are buffered with the data in the receive buffer. Therefore the status bits must always be read before the UDR Register is read. Otherwise the error status will be lost since the buffer state is lost.
- The Receiver Shift Register can now act as a third buffer level. This is done by allowing the received data to remain in the serial Shift Register (see Figure 69) if the buffer registers are full, until a new start bit is detected. The USART is therefore more resistant to Data OverRun (DOR) error conditions.

The following control bits have changed name, but have same functionality and register location:

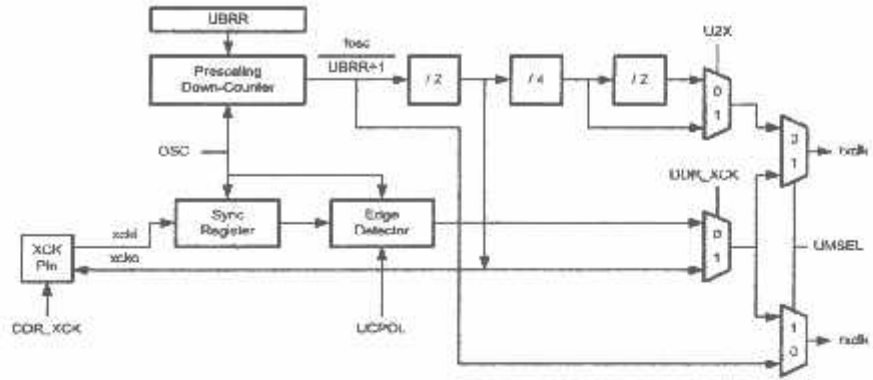
- CHR9 is changed to UCSZ2
- OR is changed to DOR

#### Clock Generation

The clock generation logic generates the base clock for the Transmitter and Receiver. The USART supports four modes of clock operation: Normal asynchronous, Double Speed asynchronous, Master synchronous and Slave synchronous mode. The UMSEL bit in USART Control and Status Register C (UCSRC) selects between asynchronous and synchronous operation. Double Speed (asynchronous mode only) is controlled by the U2X found in the UCSRA Register. When using Synchronous mode (UMSEL = 1), the Data Direction Register for the XCK pin (DDR\_XCK) controls whether the clock source is internal (Master mode) or external (Slave mode). The XCK pin is only active when using synchronous mode.

Figure 70 shows a block diagram of the clock generation logic.

Figure 70. Clock Generation Logic, Block Diagram



Signal description:

- txclk** Transmitter clock (Internal Signal).
- rxclk** Receiver base clock (Internal Signal).
- xckl** Input from XCK pin (internal Signal). Used for synchronous slave operation.
- xcko** Clock output to XCK pin (Internal Signal). Used for synchronous master operation.
- fosc** XTAL pin frequency (System Clock).

Internal Clock Generation – The Baud Rate Generator

Internal clock generation is used for the asynchronous and the synchronous master modes of operation. The description in this section refers to Figure 70.

The USART Baud Rate Register (UBRR) and the down-counter connected to it function as a programmable prescaler or baud rate generator. The down-counter, running at system clock (fosc), is loaded with the UBRR value each time the counter has counted down to zero or when the UBRR Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output ( $= fosc/(UBRR+1)$ ). The Transmitter divides the baud rate generator clock output by 2, 8, or 16 depending on mode. The baud rate generator output is used directly by the receiver's clock and data recovery units. However, the recovery units use a state machine that uses 2, 8, or 16 states depending on mode set by the state of the UMSEL, U2X and DDR\_XCK bits.

Table 61 contains equations for calculating the baud rate (in bits per second) and for calculating the UBRR value for each mode of operation using an internally generated clock source.

**Table 61. Equations for Calculating Baud Rate Register Setting**

Operating Mode	Equation for Calculating Baud Rate <sup>(1)</sup>	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{osc}}{16(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{osc}}{8(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{osc}}{2(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps).

BAUD Baud rate (in bits per second, bps)

$f_{osc}$  System Oscillator clock frequency

UBRR Contents of the UBRRH and UBRRL Registers, (0 - 4095)

Some examples of UBRR values for some system clock frequencies are found in Table 69 (see page 168).

### Double Speed Operation (U2X)

The transfer rate can be doubled by setting the U2X bit in UCSRA. Setting this bit only has effect for the asynchronous operation. Set this bit to zero when using synchronous operation.

Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. Note however that the Receiver will in this case only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used. For the Transmitter, there are no downsides.

### External Clock

External clocking is used by the synchronous slave modes of operation. The description in this section refers to Figure 70 for details.

External clock input from the XCK pin is sampled by a synchronization register to minimize the chance of meta-stability. The output from the synchronization register must then pass through an edge detector before it can be used by the Transmitter and Receiver. This process introduces a two CPU clock period delay and therefore the maximum external XCK clock frequency is limited by the following equation:

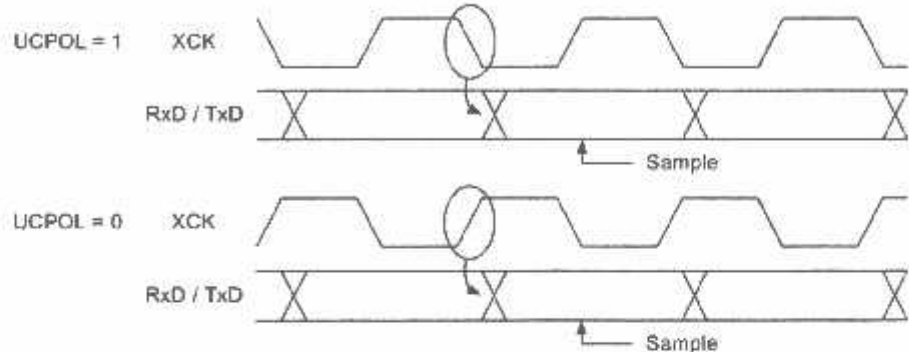
$$f_{XCK} < \frac{f_{osc}}{4}$$

Note that  $f_{osc}$  depends on the stability of the system clock source. It is therefore recommended to add some margin to avoid possible loss of data due to frequency variations.



**Synchronous Clock Operation** When Synchronous mode is used (UMSEL = 1), the XCK pin will be used as either clock input (Slave) or clock output (Master). The dependency between the clock edges and data sampling or data change is the same. The basic principle is that data input (on RxD) is sampled at the opposite XCK clock edge of the edge the data output (TxD) is changed.

**Figure 71.** Synchronous Mode XCK Timing



The UCPOL bit UCRSC selects which XCK clock edge is used for data sampling and which is used for data change. As Figure 71 shows, when UCPOL is zero the data will be changed at rising XCK edge and sampled at falling XCK edge. If UCPOL is set, the data will be changed at falling XCK edge and sampled at rising XCK edge.

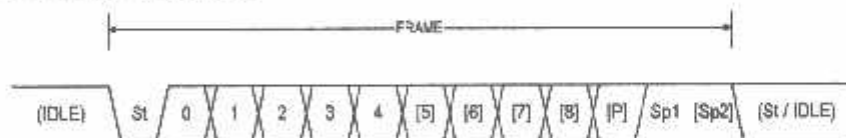
## Frame Formats

A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The USART accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. Figure 72 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

**Figure 72.** Frame Formats



- St Start bit, always low.
- (n) Data bits (0 to 8).
- P Parity bit. Can be odd or even.

- Sp Stop bit, always high.
- IDLE No transfers on the communication line (RxD or TxD). An IDLE line must be high.

The frame format used by the USART is set by the UCSZ2:0, UPM1:0 and USBS bits in UCSRB and UCSRC. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

The USART Character SiZe (UCSZ2:0) bits select the number of data bits in the frame. The USART Parity mode (UPM1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the USART Stop Bit Select (USBS) bit. The Receiver ignores the second stop bit. An FE (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

### Parity Bit Calculation

The parity bit is calculated by doing an exclusive-or of all the data bits. If odd parity is used, the result of the exclusive or is inverted. The relation between the parity bit and data bits is as follows::

$$P_{even} = d_{n-1} \oplus \dots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$

$$P_{odd} = d_{n-1} \oplus \dots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

$P_{even}$  Parity bit using even parity

$P_{odd}$  Parity bit using odd parity

$d_n$  Data bit n of the character

If used, the parity bit is located between the last data bit and first stop bit of a serial frame.

### USART Initialization

The USART has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting frame format and enabling the Transmitter or the Receiver depending on the usage. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and interrupts globally disabled) when doing the initialization.

Before doing a re-initialization with a changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The TXC Flag can be used to check that the Transmitter has completed all transfers and the RXC Flag can be used to check that there are no unread data in the receive buffer. Note that the TXC Flag must be cleared before each transmission (before UDR is written) if it is used for this purpose.

The following simple USART initialization code examples show one assembly and one C function that are equal in functionality. The examples assume asynchronous operation using polling (no interrupts enabled) and a fixed frame format. The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 registers. When the function writes to the UCSRC Register, the URSEL bit (MSB) must be set due to the sharing of I/O location by UBRRH and UCSRC.

### Assembly Code Example<sup>(1)</sup>

```

USART_Init:
    ; Set baud rate
    out  UBRRH, r17
    out  UBRRL, r16
    ; Enable Receiver and Transmitter
    ldi  r16, (1<<RXEN)|(1<<TXEN)
    out  UCSRB, r16
    ; Set frame format: 8data, 2stop bit
    ldi  r16, (1<<URSEL)|(1<<USBS)|(3<<UCSZ0)
    out  UCSRC, r16
    ret
    
```

### C Code Example<sup>(1)</sup>

```

#define FOSC 1643200 // Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1
void main( void )
{
    ...
    USART_INIT ( MYUBRR );
    ...
}

void USART_Init( unsigned int baud )
{
    /* Set baud rate */
    UBRRH = (unsigned char) (ubrr>>8);
    UBRRL = (unsigned char)ubrr;
    /* Enable receiver and transmitter */
    UCSRB = (1<<RXEN)|(1<<TXEN);
    /* Set frame format: 8data, 2stop bit */
    UCSRC = (1<<USBS)|(3<<UCSZ0);
}
    
```

Note: 1. See "About Code Examples" on page 6.

More advanced initialization routines can be made that include frame format as parameters, disable interrupts and so on. However, many applications use a fixed setting of the baud and control registers, and for these types of applications the initialization code can be placed directly in the main routine or be combined with initialization code for other I/O modules.



## Data Transmission – The USART Transmitter

The USART Transmitter is enabled by setting the *Transmit Enable* (TXEN) bit in the UCSRB Register. When the Transmitter is enabled, the normal port operation of the TxD pin is overridden by the USART and given the function as the Transmitter's serial output. The baud rate, mode of operation and frame format must be set up once before doing any transmissions. If synchronous operation is used, the clock on the XCK pin will be overridden and used as transmission clock.

### Sending Frames with 5 to 8 Data Bits

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The CPU can load the transmit buffer by writing to the UDR I/O location. The buffered data in the transmit buffer will be moved to the Shift Register when the Shift Register is ready to send a new frame. The Shift Register is loaded with new data if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the Shift Register is loaded with new data, it will transfer one complete frame at the rate given by the baud register, U2X bit or by XCK depending on mode of operation.

The following code examples show a simple USART transmit function based on polling of the *Data Register Empty* (UDRE) Flag. When using frames with less than eight bits, the most significant bits written to the UDR are ignored. The USART has to be initialized before the function can be used. For the assembly code, the data to be sent is assumed to be stored in register R16.

Assembly Code Example <sup>(1)</sup>
<pre> USART_Transmit: ; Wait for empty transmit buffer sbis UCSRA,UDRE rjmp USART_Transmit ; Put data (r16) into buffer, sends the data out UDR,r16 ret </pre>
C Code Example <sup>(1)</sup>
<pre> void USART_Transmit( unsigned char data ) { /* Wait for empty transmit buffer */ while ( !( UCSRA &amp; (1&lt;&lt;UDRE)) ) ; /* Put data into buffer, sends the data */ UDR = data; } </pre>

Note: 1. See "About Code Examples" on page 6.

The function simply waits for the transmit buffer to be empty by checking the UDRE Flag, before loading it with new data to be transmitted. If the Data Register Empty interrupt is utilized, the interrupt routine writes the data into the buffer.

## Sending Frames with 9 Data Bits

If 9-bit characters are used (UCSZ = 7), the ninth bit must be written to the TXB8 bit in UCSRB before the low byte of the character is written to UDR. The following code examples show a transmit function that handles 9-bit characters. For the assembly code, the data to be sent is assumed to be stored in registers R17:R16.

Assembly Code Example <sup>(1)</sup>
<pre> USART_Transmit: ; Wait for empty transmit buffer sbis UCSRA,UDRE rjmp USART_Transmit ; Copy 9th bit from r17 to TXB8 cbl UCSRB,TXB8 sbrc r17,0 sbi UCSRB,TXB8 ; Put LSB data (r16) into buffer, sends the data out UDR,r16 ret                     </pre>
C Code Example <sup>(1)</sup>
<pre> void USART_Transmit( unsigned int data ) {     /* Wait for empty transmit buffer */     while ( !( UCSRA &amp; (1&lt;&lt;UDRE)) )         ;      /* Copy 9th bit to TXB8 */     UCSRB &amp;= ~(1&lt;&lt;TXB8);     if ( data &amp; 0x0100 )         UCSRB  = (1&lt;&lt;TXB8);     /* Put data into buffer, sends the data */     UDR = data; }                     </pre>

Note: 1. These transmit functions are written to be general functions. They can be optimized if the contents of the UCSRB is static. (For example, only the TXB8 bit of the UCSRB Register is used after initialization.)

The ninth bit can be used for indicating an address frame when using multi processor communication mode or for other protocol handling as for example synchronization.

## Transmitter Flags and Interrupts

The USART Transmitter has two flags that indicate its state: USART Data Register Empty (UDRE) and Transmit Complete (TXC). Both flags can be used for generating interrupts.

The Data Register Empty (UDRE) Flag indicates whether the transmit buffer is ready to receive new data. This bit is set when the transmit buffer is empty, and cleared when the transmit buffer contains data to be transmitted that has not yet been moved into the Shift Register. For compatibility with future devices, always write this bit to zero when writing the UCSRA Register.

When the Data Register Empty Interrupt Enable (UDRIE) bit in UCSRB is written to one, the USART Data Register Empty interrupt will be executed as long as UDRE is set (provided that global interrupts are enabled). UDRE is cleared by writing UDR. When

interrupt-driven data transmission is used, the Data Register Empty interrupt routine must either write new data to UDR in order to clear UDRE or disable the Data Register Empty interrupt, otherwise a new interrupt will occur once the interrupt routine terminates.

The Transmit Complete (TxC) Flag bit is set to one when the entire frame in the transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer. The TxC Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TxC Flag is useful in half-duplex communication interfaces (like the RS-485 standard), where a transmitting application must enter receive mode and free the communication bus immediately after completing the transmission.

When the Transmit Complete Interrupt Enable (TxCIE) bit in UCSRB is set, the USART Transmit Complete Interrupt will be executed when the TxC Flag becomes set (provided that global interrupts are enabled). When the transmit complete interrupt is used, the interrupt handling routine does not have to clear the TxC Flag, this is done automatically when the interrupt is executed.

#### **Parity Generator**

The parity generator calculates the parity bit for the serial frame data. When parity bit is enabled (UPM1 = 1), the Transmitter Control Logic inserts the parity bit between the last data bit and the first stop bit of the frame that is sent.

#### **Disabling the Transmitter**

The disabling of the Transmitter (setting the TXEN to zero) will not become effective until ongoing and pending transmissions are completed (i.e., when the transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted). When disabled, the Transmitter will no longer override the TxD pin.

## Data Reception – The USART Receiver

The USART Receiver is enabled by writing the Receive Enable (RXEN) bit in the UCSRB Register to one. When the Receiver is enabled, the normal pin operation of the RxD pin is overridden by the USART and given the function as the Receiver's serial input. The baud rate, mode of operation and frame format must be set up once before any serial reception can be done. If synchronous operation is used, the clock on the XCK pin will be used as a transfer clock.

## Receiving Frames with 5 to 8 Data Bits

The Receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCK clock, and shifted into the Receive Shift Register until the first stop bit of a frame is received. A second stop bit will be ignored by the Receiver. When the first stop bit is received, (i.e., a complete serial frame is present in the receive Shift Register, the contents of the Shift Register will be moved into the receive buffer. The receive buffer can then be read by reading the UDR I/O location.

The following code example shows a simple USART receive function based on polling of the Receive Complete (RXC) Flag. When using frames with less than eight bits the most significant bits of the data read from the UDR will be masked to zero. The USART has to be initialized before the function can be used.

### Assembly Code Example<sup>(1)</sup>

```

USART_Receive:
    ; Wait for data to be received
    sbis UCSRA, RXC
    rjmp USART_Receive
    ; Get and return received data from buffer
    In    r16, UDR
    ret
    
```

### C Code Example<sup>(1)</sup>

```

unsigned char USART_Receive( void )
{
    /* Wait for data to be received */
    while ( !(UCSRA & (1<<RXC)) )
        ;
    /* Get and return received data from buffer */
    return UDR;
}
    
```

Note: 1. See "About Code Examples" on page 6.

The function simply waits for data to be present in the receive buffer by checking the RXC Flag, before reading the buffer and returning the value.

## Receiving Frames with 9 Data Bits

If 9-bit characters are used (UCSZ=7) the ninth bit must be read from the RXB8 bit in UCSRB **before** reading the low bits from the UDR. This rule applies to the FE, DOR, and PE Status Flags as well. Read status from UCSRA, then data from UDR. Reading the UDR I/O location will change the state of the receive buffer FIFO and consequently the TXB8, FE, DOR, and PE bits, which all are stored in the FIFO, will change.

The following code example shows a simple USART receive function that handles both 9-bit characters and the status bits.

### Assembly Code Example<sup>(1)</sup>

```

USART_Receive:
    ; Wait for data to be received
    sbis UCSRA, RXC
    rjmp USART_Receive
    ; Get status and 9th bit, then data from buffer
    in r18, UCSRA
    in r17, UCSRB
    in r16, UDR
    ; If error, return -1
    andi r18, (1<<FE)|(1<<DOR)|(1<<PE)
    breq USART_ReceiveNoError
    ldi r17, HIGH(-1)
    ldi r16, LOW(-1)
USART_ReceiveNoError:
    ; Filter the 9th bit, then return
    lsr r17
    andi r17, 0x01
    ret
    
```

### C Code Example<sup>(1)</sup>

```

unsigned int USART_Receive( void )
{
    unsigned char status, resh, resl;
    /* Wait for data to be received */
    while ( !(UCSRA & (1<<RXC)) )
        ;
    /* Get status and 9th bit, then data */
    /* from buffer */
    status = UCSRA;
    resh = UCSRB;
    resl = UDR;
    /* If error, return -1 */
    if ( status & (1<<FE)|(1<<DOR)|(1<<PE) )
        return -1;
    /* Filter the 9th bit, then return */
    resh = (resh >> 1) & 0x01;
    return ((resh << 8) | resl);
}
    
```



Note: 1. See "About Code Examples" on page 6.

The receive function example reads all the I/O Registers into the Register File before any computation is done. This gives an optimal receive buffer utilization since the buffer location read will be free to accept new data as early as possible.

### Receive Complete Flag and Interrupt

The USART Receiver has one flag that indicates the receiver state.

The Receive Complete (RXC) Flag indicates if there are unread data present in the receive buffer. This flag is one when unread data exist in the receive buffer and zero when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled (RXEN = 0), the receive buffer will be flushed and consequently the RXC bit will become zero.

When the Receive Complete Interrupt Enable (RXCIE) in UCSRB is set, the USART Receive Complete Interrupt will be executed as long as the RXC Flag is set (provided that global interrupts are enabled). When interrupt-driven data reception is used, the receive complete routine must read the received data from UDR in order to clear the RXC Flag, otherwise a new interrupt will occur once the interrupt routine terminates.

### Receiver Error Flags

The USART Receiver has three Error Flags: Frame Error (FE), Data OverRun (DOR) and Parity Error (PE). All can be accessed by reading UCSRA. Common for the error flags is that they are located in the receive buffer together with the frame for which they indicate the error status. Due to the buffering of the error flags, the UCSRA must be read before the receive buffer (UDR), since reading the UDR I/O location changes the buffer read location. Another equality for the error flags is that they can not be altered by software doing a write to the flag location. However, all flags must be set to zero when the UCSRA is written for upward compatibility of future USART implementations. None of the error flags can generate interrupts.

The Frame Error (FE) Flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The FE Flag is zero when the stop bit was correctly read (as one), and the FE Flag will be one when the stop bit was incorrect (zero). This flag can be used for detecting out-of-sync conditions, detecting break conditions and protocol handling. The FE Flag is not affected by the setting of the USBS bit in UCSRC since the Receiver ignores all, except for the first, stop bits. For compatibility with future devices, always set this bit to zero when writing to UCSRA.

The Data OverRun (DOR) Flag indicates data loss due to a Receiver Buffer full condition. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the receive Shift Register, and a new start bit is detected. If the DOR Flag is set there was one or more serial frame lost between the frame last read from UDR, and the next frame read from UDR. For compatibility with future devices, always write this bit to zero when writing to UCSRA. The DOR Flag is cleared when the frame received was successfully moved from the Shift Register to the receive buffer.

The Parity Error (PE) Flag indicates that the next frame in the receive buffer had a Parity Error when received. If parity check is not enabled the PE bit will always be read zero. For compatibility with future devices, always set this bit to zero when writing to UCSRA. For more details see "Parity Bit Calculation" on page 148 and "Parity Checker" on page 155.

### Parity Checker

The Parity Checker is active when the high USART Parity mode (UPM1) bit is set. The type of Parity Check to be performed (odd or even) is selected by the UPM0 bit. When enabled, the Parity Checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit from the serial frame. The result of the check is

stored in the receive buffer together with the received data and stop bits. The Parity Error (PE) Flag can then be read by software to check if the frame had a Parity Error.

The PE bit is set if the next character that can be read from the receive buffer had a parity error when received and the parity checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read.

#### Disabling the Receiver

In contrast to the Transmitter, disabling of the Receiver will be immediate. Data from ongoing receptions will therefore be lost. When disabled (i.e., the RXEN is set to zero) the Receiver will no longer override the normal function of the RxD port pin. The receiver buffer FIFO will be flushed when the Receiver is disabled. Remaining data in the buffer will be lost.

#### Flushing the Receive Buffer

The receiver buffer FIFO will be flushed when the Receiver is disabled, i.e., the buffer will be emptied of its contents. Unread data will be lost. If the buffer has to be flushed during normal operation, due to for instance an error condition, read the UDR I/O location until the RXC Flag is cleared. The following code example shows how to flush the receive buffer.

<p>Assembly Code Example<sup>(1)</sup></p> <pre> USART_Flush:     sbis UCSRA, RXC     ret     in    r16, UDR     rjmp USART_Flush                 </pre>
<p>C Code Example<sup>(1)</sup></p> <pre> void USART_Flush( void ) {     unsigned char dummy;     while ( UCSRA &amp; (1&lt;&lt;RXC) ) dummy = UDR; }                 </pre>

Note: 1. See "About Code Examples" on page 6.

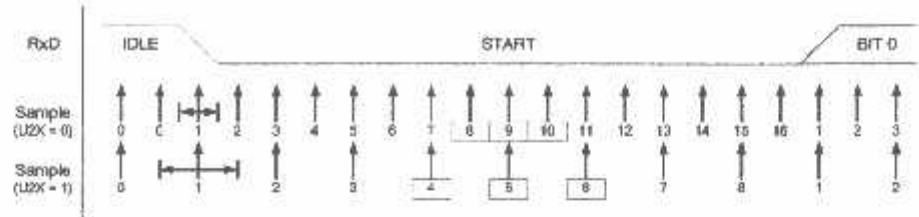
#### Asynchronous Data Reception

The USART includes a clock recovery and a data recovery unit for handling asynchronous data reception. The clock recovery logic is used for synchronizing the internally generated baud rate clock to the incoming asynchronous serial frames at the RxD pin. The data recovery logic samples and low pass filters each incoming bit, thereby improving the noise immunity of the Receiver. The asynchronous reception operational range depends on the accuracy of the internal baud rate clock, the rate of the incoming frames, and the frame size in number of bits.

#### Asynchronous Clock Recovery

The clock recovery logic synchronizes internal clock to the incoming serial frames. Figure 73 illustrates the sampling process of the start bit of an incoming frame. The sample rate is 16 times the baud rate for normal mode, and eight times the baud rate for double speed mode. The horizontal arrows illustrate the synchronization variation due to the sampling process. Note the larger time variation when using the Double Speed mode (U2X = 1) of operation. Samples denoted zero are samples done when the RxD line is idle (i.e., no communication activity).

Figure 73. Start Bit Sampling

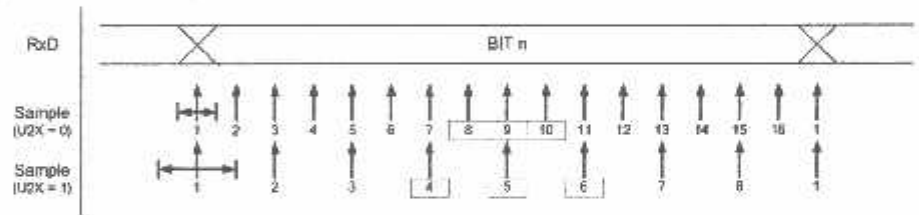


When the clock recovery logic detects a high (idle) to low (start) transition on the RxD line, the start bit detection sequence is initiated. Let sample 1 denote the first zero-sample as shown in the figure. The clock recovery logic then uses samples 8, 9, and 10 for normal mode, and samples 4, 5, and 6 for Double Speed mode (indicated with sample numbers inside boxes on the figure), to decide if a valid start bit is received. If two or more of these three samples have logical high levels (the majority wins), the start bit is rejected as a noise spike and the Receiver starts looking for the next high to low-transition. If however, a valid start bit is detected, the clock recovery logic is synchronized and the data recovery can begin. The synchronization process is repeated for each start bit.

**Asynchronous Data Recovery**

When the receiver clock is synchronized to the start bit, the data recovery can begin. The data recovery unit uses a state machine that has 16 states for each bit in normal mode and eight states for each bit in Double Speed mode. Figure 74 shows the sampling of the data bits and the parity bit. Each of the samples is given a number that is equal to the state of the recovery unit.

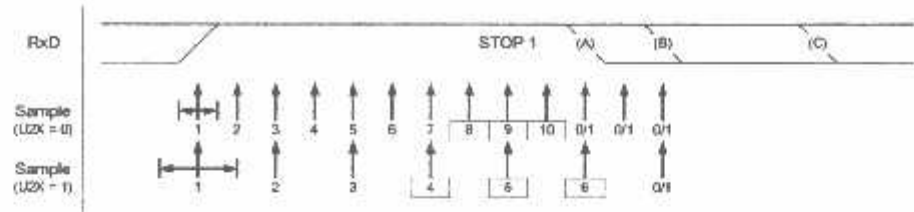
Figure 74. Sampling of Data and Parity Bit



The decision of the logic level of the received bit is taken by doing a majority voting of the logic value to the three samples in the center of the received bit. The center samples are emphasized on the figure by having the sample number inside boxes. The majority voting process is done as follows: If two or all three samples have high levels, the received bit is registered to be a logic 1. If two or all three samples have low levels, the received bit is registered to be a logic 0. This majority voting process acts as a low pass filter for the incoming signal on the RxD pin. The recovery process is then repeated until a complete frame is received. Including the first stop bit. Note that the Receiver only uses the first stop bit of a frame.

Figure 75 shows the sampling of the stop bit and the earliest possible beginning of the start bit of the next frame.

**Figure 75. Stop Bit Sampling and Next Start Bit Sampling**



The same majority voting is done to the stop bit as done for the other bits in the frame. If the stop bit is registered to have a logic 0 value, the Frame Error (FE) Flag will be set.

A new high to low transition indicating the start bit of a new frame can come right after the last of the bits used for majority voting. For Normal Speed mode, the first low level sample can be at point marked (A) in Figure 75. For Double Speed mode the first low level must be delayed to (B). (C) marks a stop bit of full length. The early start bit detection influences the operational range of the Receiver.

### Asynchronous Operational Range

The operational range of the Receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If the Transmitter is sending frames at too fast or too slow bit rates, or the internally generated baud rate of the Receiver does not have a similar (see Table 62) base frequency, the Receiver will not be able to synchronize the frames to the start bit.

The following equations can be used to calculate the ratio of the incoming data rate and internal receiver baud rate.

$$R_{slow} = \frac{(D+1)S}{S-1+D \cdot S + S_F} \quad R_{fast} = \frac{(D+2)S}{(D+1)S + S_M}$$

- D Sum of character size and parity size (D = 5 to 10 bit).
- S Samples per bit. S = 16 for Normal Speed mode and S = 8 for Double Speed mode.
- S<sub>F</sub> First sample number used for majority voting. S<sub>F</sub> = 8 for Normal Speed and S<sub>F</sub> = 4 for Double Speed mode.
- S<sub>M</sub> Middle sample number used for majority voting. S<sub>M</sub> = 9 for Normal Speed and S<sub>M</sub> = 5 for Double Speed mode.
- R<sub>slow</sub> is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate. R<sub>fast</sub> is the ratio of the fastest incoming data rate that can be accepted in relation to the Receiver baud rate.

Table 62 and Table 63 list the maximum receiver baud rate error that can be tolerated. Note that normal speed mode has higher toleration of baud rate variations.

**Table 62.** Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (U2X = 0)

D # (Data+Parity Bit)	R <sub>slow</sub> (%)	R <sub>fast</sub> (%)	Max Total Error (%)	Recommended Max Receiver Error (%)
5	93.20	106.67	+6.67/-6.8	± 3.0
6	94.12	105.79	+5.79/-5.88	± 2.5
7	94.81	105.11	+5.11/-5.19	± 2.0
8	95.36	104.58	+4.58/-4.54	± 2.0
9	95.81	104.14	+4.14/-4.19	± 1.5
10	96.17	103.78	+3.78/-3.83	± 1.5

**Table 63.** Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (U2X = 1)

D # (Data+Parity Bit)	R <sub>slow</sub> (%)	R <sub>fast</sub> (%)	Max Total Error (%)	Recommended Max Receiver Error (%)
5	94.12	105.66	+5.66/-5.88	± 2.5
6	94.92	104.92	+4.92/-5.08	± 2.0
7	95.52	104.35	+4.35/-4.48	± 1.5
8	96.00	103.90	+3.90/-4.00	± 1.5
9	96.39	103.53	+3.53/-3.61	± 1.5
10	96.70	103.23	+3.23/-3.30	± 1.0

The recommendations of the maximum receiver baud rate error was made under the assumption that the Receiver and Transmitter equally divides the maximum total error.

There are two possible sources for the receivers baud rate error. The receiver's system clock (XTAL) will always have some minor instability over the supply voltage range and the temperature range. When using a crystal to generate the system clock, this is rarely a problem, but for a resonator the system clock may differ more than 2% depending of the resonators tolerance. The second source for the error is more controllable. The baud rate generator can not always do an exact division of the system frequency to get the baud rate wanted. In this case an UBRR value that gives an acceptable low error can be used if possible.

## Multi-processor Communication Mode

Setting the Multi-processor Communication Mode (MPCM) bit in UCSRA enables a filtering function of incoming frames received by the USART Receiver. Frames that do not contain address information will be ignored and not put into the receive buffer. This effectively reduces the number of incoming frames that has to be handled by the CPU, in a system with multiple MCUs that communicate via the same serial bus. The Transmitter is unaffected by the MPCM setting, but has to be used differently when it is a part of a system utilizing the Multi-processor Communication Mode.

If the Receiver is set up to receive frames that contain five to eight data bits, then the first stop bit indicates if the frame contains data or address information. If the Receiver is set up for frames with nine data bits, then the ninth bit (RXB8) is used for identifying address and data frames. When the frame type bit (the first stop or the ninth bit) is one, the frame contains an address. When the frame type bit is zero the frame is a data frame.

The Multi-processor Communication Mode enables several slave MCUs to receive data from a Master MCU. This is done by first decoding an address frame to find out which MCU has been addressed. If a particular Slave MCU has been addressed, it will receive the following data frames as normal, while the other Slave MCUs will ignore the received frames until another address frame is received.

### Using MPCM

For an MCU to act as a Master MCU, it can use a 9-bit character frame format (UCSZ = 7). The ninth bit (TXB8) must be set when an address frame (TXB8 = 1) or cleared when a data frame (TXB8 = 0) is being transmitted. The Slave MCUs must, in this case, be set to use a 9-bit character frame format.

The following procedure should be used to exchange data in Multi-processor Communication Mode:

1. All Slave MCUs are in Multi-processor Communication Mode (MPCM in UCSRA is set).
2. The Master MCU sends an address frame, and all slaves receive and read this frame. In the Slave MCUs, the RXC Flag in UCSRA will be set as normal.
3. Each Slave MCU reads the UDR Register and determines if it has been selected. If so, it clears the MPCM bit in UCSRA, otherwise it waits for the next address byte and keeps the MPCM setting.
4. The addressed MCU will receive all data frames until a new address frame is received. The other Slave MCUs, which still have the MPCM bit set, will ignore the data frames.
5. When the last data frame is received by the addressed MCU, the addressed MCU sets the MPCM bit and waits for a new address frame from Master. The process then repeats from 2.

Using any of the 5- to 8-bit character frame formats is possible, but impractical since the Receiver must change between using  $n$  and  $n+1$  character frame formats. This makes full-duplex operation difficult since the Transmitter and Receiver uses the same character size setting. If 5- to 8-bit character frames are used, the Transmitter must be set to use two stop bit (USBS = 1) since the first stop bit is used for indicating the frame type.

Do not use Read-Modify-Write instructions (SBI and CBI) to set or clear the MPCM bit. The MPCM bit shares the same I/O location as the TXC Flag and this might accidentally be cleared when using SBI or CBI instructions.

## Read Access

Doing a read access to the UBRRH or the UCSRC Register is a more complex operation. However, in most applications, it is rarely necessary to read any of these registers.

The read access is controlled by a timed sequence. Reading the I/O location once returns the UBRRH Register contents. If the register location was read in previous system clock cycle, reading the register in the current clock cycle will return the UCSRC contents. Note that the timed sequence for reading the UCSRC is an atomic operation. Interrupts must therefore be controlled (e.g., by disabling interrupts globally) during the read operation.

The following code example shows how to read the UCSRC Register contents.

Assembly Code Example <sup>(1)</sup>
<pre> USART_ReadUCSRC: ; Read UCSRC in r16,UBRRH in r16,UCSRC ret </pre>
C Code Example <sup>(1)</sup>
<pre> unsigned char USART_ReadUCSRC( void ) {     unsigned char ucsrc;     /* Read UCSRC */     ucsrc = UBRRH;     ucsrc = UCSRC;     return ucsrc; } </pre>

Note: 1. See "About Code Examples" on page 6.

The assembly code example returns the UCSRC value in r16.

Reading the UBRRH contents is not an atomic operation and therefore it can be read as an ordinary register, as long as the previous instruction did not access the register location.

## USART Register Description

### USART I/O Data Register – JDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR Register location. Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDRE Flag in the UCSRA Register is set. Data written to UDR when the UDRE Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD pin.

The receive buffer consists of a two-level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use Read-Modify-Write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

## USART Control and Status Register A – UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – RXC: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero. The RXC Flag can be used to generate a Receive Complete interrupt (see description of the RXCIE bit).

- **Bit 6 – TXC: USART Transmit Complete**

This flag bit is set when the entire frame in the transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDR). The TXC Flag bit is automatically cleared when a Transmit Complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC Flag can generate a Transmit Complete interrupt (see description of the TXCIE bit).

- **Bit 5 – UDRE: USART Data Register Empty**

The UDRE Flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE Flag can generate a Data Register Empty interrupt (see description of the UDRIE bit).

UDRE is set after a reset to indicate that the Transmitter is ready.

- **Bit 4 – FE: Frame Error**

This bit is set if the next character in the receive buffer had a Frame Error when received. For example, when the first stop bit of the next character in the receive buffer is zero. This bit is valid until the receive buffer (UDR) is read. The FE bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRA.

- **Bit 3 – DOR: Data OverRun**

This bit is set if a data overrun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.

- **Bit 2 – PE: Parity Error**

This bit is set if the next character in the receive buffer had a Parity Error when received and the parity checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.



- **Bit 1 – U2X: Double the USART Transmission Speed**

This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.

Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

- **Bit 0 – MPCM: Multi-processor Communication Mode**

This bit enables the Multi-processor Communication mode. When the MPCM bit is written to one, all the incoming frames received by the USART Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the MPCM setting. For more detailed information see "Multi-processor Communication Mode" on page 160.

**USART Control and Status Register B – UCSRB**

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – RXCIE: RX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the RXC Flag. A USART Receive Complete interrupt will be generated only if the RXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set.

- **Bit 6 – TXCIE: TX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the TXC Flag. A USART Transmit Complete interrupt will be generated only if the TXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC bit in UCSRA is set.

- **Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable**

Writing this bit to one enables interrupt on the UDRE Flag. A Data Register Empty interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE bit in UCSRA is set.

- **Bit 4 – RXEN: Receiver Enable**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FE, DOR, and PE Flags.

- **Bit 3 – TXEN: Transmitter Enable**

Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. The disabling of the Transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed, (i.e., when the transmit Shift Register and transmit Buffer Register do not contain data to be transmitted). When disabled, the Transmitter will no longer override the TxD port.

- **Bit 2 – UCSZ2: Character Size**

The UCSZ2 bits combined with the UCSZ1:0 bit in UCSRC sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

- **Bit 1 – RXB8: Receive Data Bit 8**

RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDR.

- **Bit 0 – TXB8: Transmit Data Bit 8**

TXB8 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDR.

**USART Control and Status Register C – UCSRC<sup>(1)</sup>**

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

Note: 1. The UCSRC Register shares the same I/O location as the UBRRH Register. See the "Accessing UBRRH/UCSRC Registers" on page 161 section which describes how to access this register.

• **Bit 7 – URSEL: Register Select**

This bit selects between accessing the UCSRC or the UBRRH Register. It is read as one when reading UCSRC. The URSEL must be one when writing the UCSRC.

• **Bit 6 – UMSEL: USART Mode Select**

This bit selects between asynchronous and synchronous mode of operation.

**Table 64. UMSEL Bit Settings**

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

- **Bit 5:4 – UPM1:0: Parity Mode**

These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPM0 setting. If a mismatch is detected, the PE Flag in UCSRA will be set.

**Table 65. UPM Bits Settings**

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- **Bit 3 – USBS: Stop Bit Select**

This bit selects the number of stop bits to be inserted by the Transmitter. The Receiver ignores this setting.

**Table 66. USBS Bit Settings**

USBS	Stop Bit(s)
0	1-bit
1	2-bit

- **Bit 2:1 – UCSZ1:0: Character Size**

The UCSZ1:0 bits combined with the UCSZ2 bit in UCSRB sets the number of data bits (character size) in a frame the Receiver and Transmitter use.

**Table 67. UCSZ Bits Settings**

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- **Bit 0 – UCPOL: Clock Polarity**

This bit is used for Synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOL bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK).

**Table 68. UCPOL Bit Settings**

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

**USART Baud Rate Registers – UBRRL and UBRRH<sup>(1)</sup>**

Bit	15	14	13	12	11	10	9	8	
	URSEL	-	-	-	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Note: 1. The UBRRH Register shares the same I/O location as the UCSRC Register. See the "Accessing UBRRH/UCSRC Registers" on page 161 section which describes how to access this register.

• **Bit 15 – URSEL: Register Select**

This bit selects between accessing the UBRRH or the UCSRC Register. It is read as zero when reading UBRRH. The URSEL must be zero when writing the UBRRH.

• **Bit 14:12 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRH is written.

• **Bit 11:0 – UBRR11:0: USART Baud Rate Register**

This is a 12-bit register which contains the USART Baud Rate. The UBRRH contains the four most significant bits, and the UBRRL contains the eight least significant bits of the USART baud rate. Ongoing transmissions by the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing UBRRL will trigger an immediate update of the baud rate prescaler.

## Examples of Baud Rate Setting

For standard crystal and resonator frequencies, the most commonly used baud rates for asynchronous operation can be generated by using the UBRR settings in Table 69. UBRR values which yield an actual baud rate differing less than 0.5% from the target baud rate, are bold in the table. Higher error ratings are acceptable, but the Receiver will have less noise resistance when the error ratings are high, especially for large serial frames (see "Asynchronous Operational Range" on page 158). The error values are calculated using the following equation:

$$\text{Error}[\%] = \left( \frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1 \right) \cdot 100\%$$

**Table 69.** Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	$f_{\text{osc}} = 1.0000 \text{ MHz}$				$f_{\text{osc}} = 1.8432 \text{ MHz}$				$f_{\text{osc}} = 2.0000 \text{ MHz}$			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	-	-	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	-	-	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	-	-	-	-	-	-	0	0.0%	-	-	-	-
250k	-	-	-	-	-	-	-	-	-	-	0	0.0%
Max <sup>(1)</sup>	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

1. UBRR = 0, Error = 0.0%

Table 70. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	$f_{osc} = 3.6864 \text{ MHz}$				$f_{osc} = 4.0000 \text{ MHz}$				$f_{osc} = 7.3728 \text{ MHz}$			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	-	-	0	-7.8%	-	-	0	0.0%	0	-7.8%	1	-7.8%
1M	-	-	-	-	-	-	-	-	-	-	0	-7.8%
Max <sup>(1)</sup>	230.4 kbps		460.8 kbps		250k bps		0.5 Mbps		460.8 kbps		921.6 kbps	

1. UBRR = 0, Error = 0.0%

**Table 71. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)**

Baud Rate (bps)	$f_{osc} = 8.0000$ MHz				$f_{osc} = 11.0592$ MHz				$f_{osc} = 14.7456$ MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	-	-	2	-7.8%	1	-7.8%	3	-7.8%
1M	-	-	0	0.0%	-	-	-	-	0	-7.8%	1	-7.8%
Max <sup>(1)</sup>	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, Error = 0.0%

**Table 72. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)**

Baud Rate (bps)	$f_{osc} = 16.0000 \text{ MHz}$				$f_{osc} = 16.4320 \text{ MHz}$				$f_{osc} = 20.0000 \text{ MHz}$			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	-	-	4	-7.8%	-	-	4	0.0%
1M	0	0.0%	1	0.0%	-	-	-	-	-	-	-	-
Max <sup>(1)</sup>	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

1. UBRR = 0, Error = 0.0%



## Two-wire Serial Interface

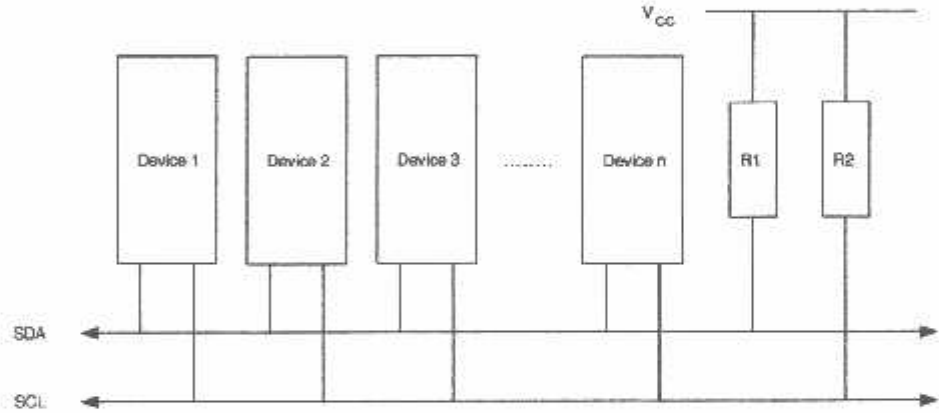
### Features

- Simple yet Powerful and Flexible Communication Interface, only Two Bus Lines Needed
- Both Master and Slave Operation Supported
- Device can Operate as Transmitter or Receiver
- 7-bit Address Space Allows up to 128 Different Slave Addresses
- Multi-master Arbitration Support
- Up to 400 kHz Data Transfer Speed
- Slew-rate Limited Output Drivers
- Noise Suppression Circuitry Rejects Spikes on Bus Lines
- Fully Programmable Slave Address with General Call Support
- Address Recognition Causes Wake-up when AVR is in Sleep Mode

### Two-wire Serial Interface Bus Definition

The Two-wire Serial Interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 different devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA). The only external hardware needed to implement the bus is a single pull-up resistor for each of the TWI bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

Figure 76. TWI Bus Interconnection



### TWI Terminology

The following definitions are frequently encountered in this section.

Table 73. TWI Terminology

Term	Description
Master	The device that initiates and terminates a transmission. The Master also generates the SCL clock.
Slave	The device addressed by a Master.
Transmitter	The device placing data on the bus.
Receiver	The device reading data from the bus.

## Electrical Interconnection

As depicted in Figure 76, both bus lines are connected to the positive supply voltage through pull-up resistors. The bus drivers of all TWI-compliant devices are open-drain or open-collector. This implements a wired-AND function which is essential to the operation of the interface. A low level on a TWI bus line is generated when one or more TWI devices output a zero. A high level is output when all TWI devices tri-state their outputs, allowing the pull-up resistors to pull the line high. Note that all AVR devices connected to the TWI bus must be powered in order to allow any bus operation.

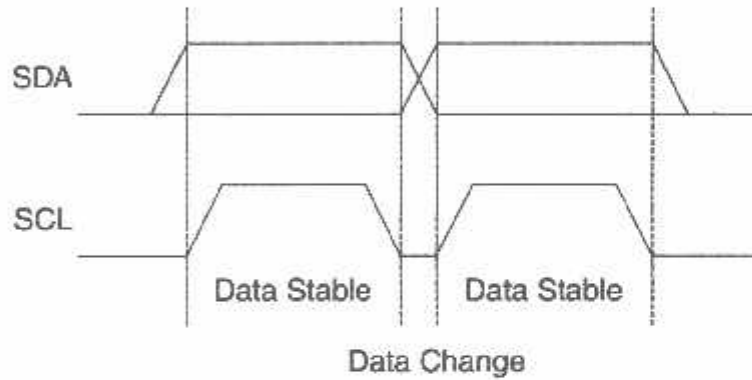
The number of devices that can be connected to the bus is only limited by the bus capacitance limit of 400 pF and the 7-bit slave address space. A detailed specification of the electrical characteristics of the TWI is given in "Two-wire Serial Interface Characteristics" on page 257. Two different sets of specifications are presented there, one relevant for bus speeds below 100 kHz, and one valid for bus speeds up to 400 kHz.

## Data Transfer and Frame Format

### Transferring Bits

Each data bit transferred on the TWI bus is accompanied by a pulse on the clock line. The level of the data line must be stable when the clock line is high. The only exception to this rule is for generating start and stop conditions.

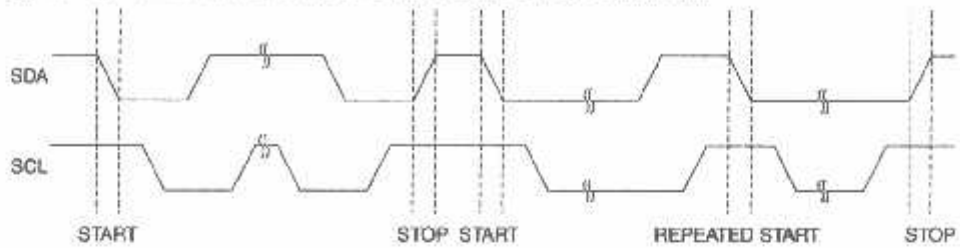
Figure 77. Data Validity



### START and STOP Conditions

The Master initiates and terminates a data transmission. The transmission is initiated when the Master issues a START condition on the bus, and it is terminated when the Master issues a STOP condition. Between a START and a STOP condition, the bus is considered busy, and no other Master should try to seize control of the bus. A special case occurs when a new START condition is issued between a START and STOP condition. This is referred to as a REPEATED START condition, and is used when the Master wishes to initiate a new transfer without relinquishing control of the bus. After a REPEATED START, the bus is considered busy until the next STOP. This is identical to the START behavior, and therefore START is used to describe both START and REPEATED START for the remainder of this datasheet, unless otherwise noted. As depicted below, START and STOP conditions are signalled by changing the level of the SDA line when the SCL line is high.

**Figure 78. START, REPEATED START, and STOP Conditions**



**Address Packet Format**

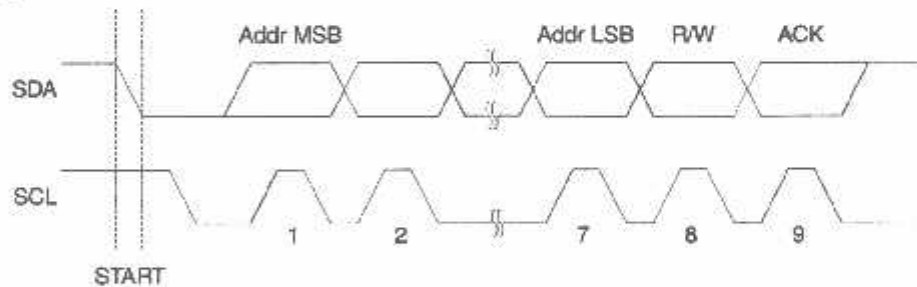
All address packets transmitted on the TWI bus are nine bits long, consisting of seven address bits, one READ/WRITE control bit and an acknowledge bit. If the READ/WRITE bit is set, a read operation is to be performed, otherwise a write operation should be performed. When a Slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle. If the addressed slave is busy, or for some other reason can not service the master's request, the SDA line should be left high in the ACK clock cycle. The Master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission. An address packet consisting of a slave address and a READ or a WRITE bit is called SLA+R or SLA+W, respectively.

The MSB of the address byte is transmitted first. Slave addresses can freely be allocated by the designer, but the address 0000 000 is reserved for a general call.

When a general call is issued, all slaves should respond by pulling the SDA line low in the ACK cycle. A general call is used when a Master wishes to transmit the same message to several slaves in the system. When the general call address followed by a Write bit is transmitted on the bus, all slaves set up to acknowledge the general call will pull the SDA line low in the ack cycle. The following data packets will then be received by all the slaves that acknowledged the general call. Note that transmitting the general call address followed by a Read bit is meaningless, as this would cause contention if several slaves started transmitting different data.

All addresses of the format 1111 xxx should be reserved for future purposes.

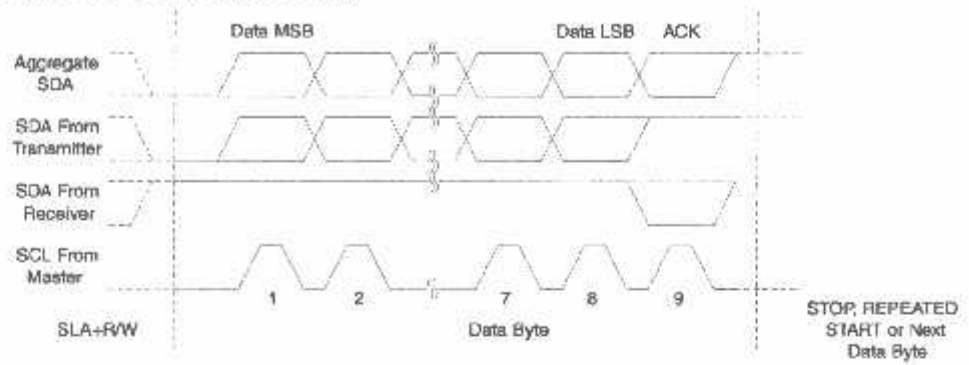
**Figure 79. Address Packet Format**



**Data Packet Format**

All data packets transmitted on the TWI bus are nine bits long, consisting of one data byte and an acknowledge bit. During a data transfer, the Master generates the clock and the START and STOP conditions, while the Receiver is responsible for acknowledging the reception. An Acknowledge (ACK) is signalled by the Receiver pulling the SDA line low during the ninth SCL cycle. If the Receiver leaves the SDA line high, a NACK is signalled. When the Receiver has received the last byte, or for some reason cannot receive any more bytes, it should inform the Transmitter by sending a NACK after the final byte. The MSB of the data byte is transmitted first.

Figure 80. Data Packet Format

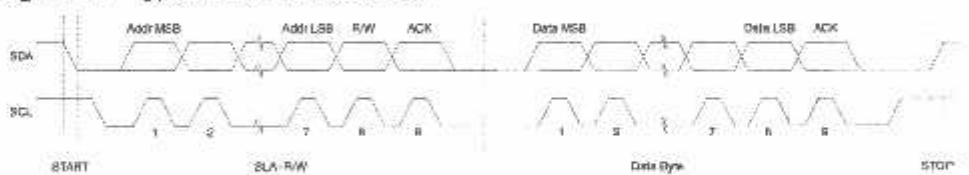


**Combining Address and Data Packets into a Transmission**

A transmission basically consists of a START condition, a SLA+R/W, one or more data packets and a STOP condition. An empty message, consisting of a START followed by a STOP condition is illegal. Note that the wired-ANDing of the SCL line can be used to implement handshaking between the Master and the Slave. The Slave can extend the SCL low period by pulling the SCL line low. This is useful if the clock speed set up by the Master is too fast for the Slave or the Slave needs extra time for processing between the data transmissions. The Slave extending the SCL low period will not affect the SCL high period, which is determined by the Master. As a consequence, the Slave can reduce the TWI data transfer speed by prolonging the SCL duty cycle.

Figure 81 shows a typical data transmission. Note that several data bytes can be transmitted between the SLA+R/W and the STOP condition, depending on the software protocol implemented by the application software.

Figure 81. Typical Data Transmission



**Multi-master Bus Systems, Arbitration and Synchronization**

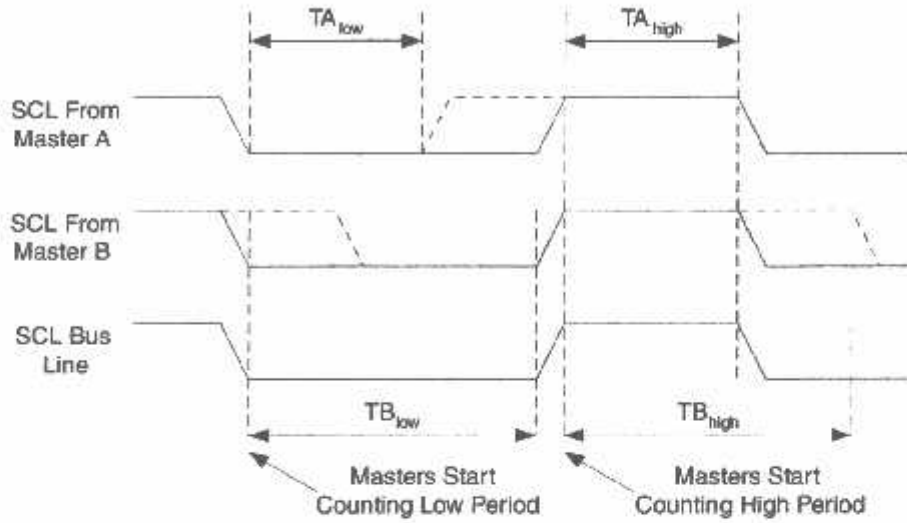
The TWI protocol allows bus systems with several masters. Special concerns have been taken in order to ensure that transmissions will proceed as normal, even if two or more masters initiate a transmission at the same time. Two problems arise in multi-master systems:

- An algorithm must be implemented allowing only one of the Masters to complete the transmission. All other masters should cease transmission when they discover that they have lost the selection process. This selection process is called arbitration. When a contending Master discovers that it has lost the arbitration process, it should immediately switch to Slave mode to check whether it is being addressed by the winning Master. The fact that multiple masters have started transmission at the same time should not be detectable to the slaves, i.e., the data being transferred on the bus must not be corrupted.
- Different Masters may use different SCL frequencies. A scheme must be devised to synchronize the serial clocks from all Masters, in order to let the transmission proceed in a lockstep fashion. This will facilitate the arbitration process.

The wired-ANDing of the bus lines is used to solve both these problems. The serial clocks from all Masters will be wired-ANDed, yielding a combined clock with a high

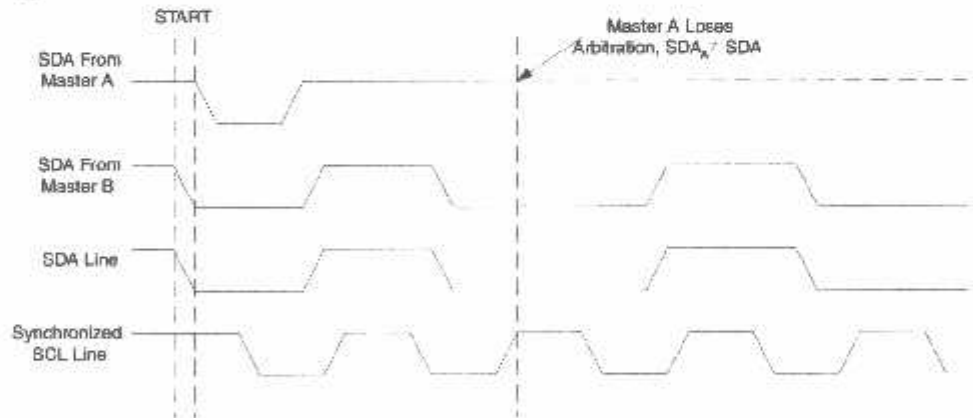
period equal to the one from the Master with the shortest high period. The low period of the combined clock is equal to the low period of the Master with the longest low period. Note that all masters listen to the SCL line, effectively starting to count their SCL high and low time-out periods when the combined SCL line goes high or low, respectively.

**Figure 82. SCL Synchronization Between Multiple Masters**



Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data. If the value read from the SDA line does not match the value the Master had output, it has lost the arbitration. Note that a Master can only lose arbitration when it outputs a high SDA value while another Master outputs a low value. The losing master should immediately go to Slave mode, checking if it is being addressed by the winning master. The SDA line should be left high, but losing masters are allowed to generate a clock signal until the end of the current data or address packet. Arbitration will continue until only one master remains, and this may take many bits. If several masters are trying to address the same Slave, arbitration will continue into the data packet.

**Figure 83. Arbitration Between Two Masters**



Note that arbitration is not allowed between:

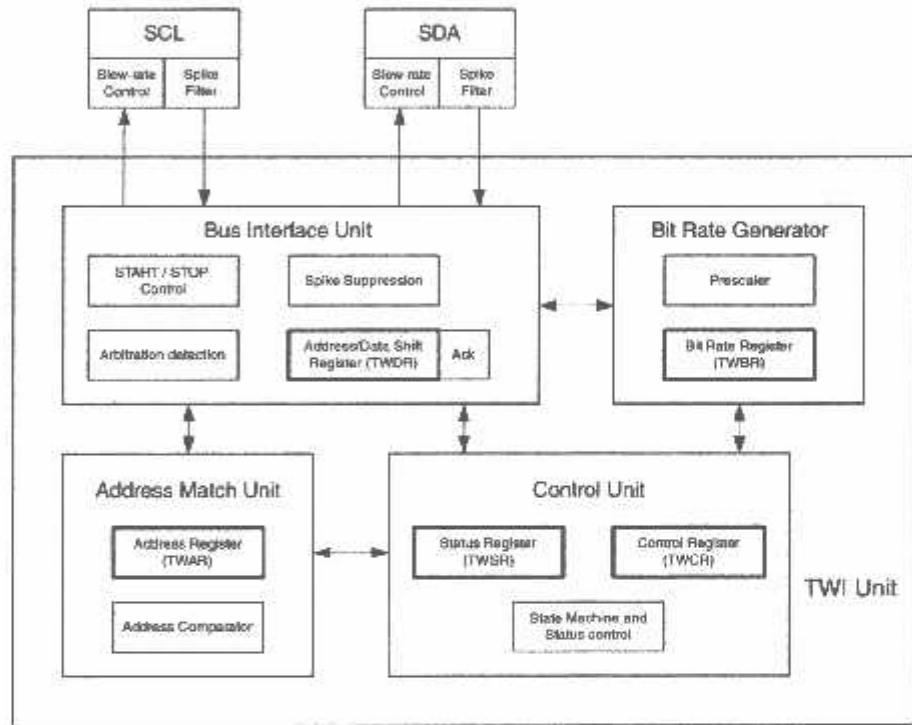
- A REPEATED START condition and a data bit
- A STOP condition and a data bit
- A REPEATED START and a STOP condition

It is the user software's responsibility to ensure that these illegal arbitration conditions never occur. This implies that in multi-master systems, all data transfers must use the same composition of SLA+R/W and data packets. In other words: All transmissions must contain the same number of data packets, otherwise the result of the arbitration is undefined.

**Overview of the TWI Module**

The TWI module is comprised of several submodules, as shown in Figure 84. All registers drawn in a thick line are accessible through the AVR data bus.

**Figure 84. Overview of the TWI Module**



**SCL and SDA Pins**

These pins interface the AVR TWI with the rest of the MCU system. The output drivers contain a slow-rate limiter in order to conform to the TWI specification. The input stages contain a spike suppression unit removing spikes shorter than 50 ns. Note that the internal pull-ups in the AVR pads can be enabled by setting the PORT bits corresponding to the SCL and SDA pins, as explained in the I/O Port section. The internal pull-ups can in some systems eliminate the need for external ones.

### Bit Rate Generator Unit

This unit controls the period of SCL when operating in a Master mode. The SCL period is controlled by settings in the TWI Bit Rate Register (TWBR) and the Prescaler bits in the TWI Status Register (TWSR). Slave operation does not depend on Bit Rate or Prescaler settings, but the CPU clock frequency in the Slave must be at least 16 times higher than the SCL frequency. Note that slaves may prolong the SCL low period, thereby reducing the average TWI bus clock period. The SCL frequency is generated according to the following equation:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{\text{TWPS}}}$$

- TWBR = Value of the TWI Bit Rate Register
- TWPS = Value of the prescaler bits in the TWI Status Register

**Note:** TWBR should be 10 or higher if the TWI operates in Master mode. If TWBR is lower than 10, the Master may produce an incorrect output on SDA and SCL for the remainder of the byte. The problem occurs when operating the TWI in Master mode, sending Start + SLA + R/W to a Slave (a Slave does not need to be connected to the bus for the condition to happen).

### Bus Interface Unit

This unit contains the Data and Address Shift Register (TWDR), a START/STOP Controller and Arbitration detection hardware. The TWDR contains the address or data bytes to be transmitted, or the address or data bytes received. In addition to the 8-bit TWDR, the Bus Interface Unit also contains a register containing the (N)ACK bit to be transmitted or received. This (N)ACK Register is not directly accessible by the application software. However, when receiving, it can be set or cleared by manipulating the TWI Control Register (TWCR). When in Transmitter mode, the value of the received (N)ACK bit can be determined by the value in the TWSR.

The START/STOP Controller is responsible for generation and detection of START, REPEATED START, and STOP conditions. The START/STOP controller is able to detect START and STOP conditions even when the AVR MCU is in one of the sleep modes, enabling the MCU to wake up if addressed by a Master.

If the TWI has initiated a transmission as Master, the Arbitration Detection hardware continuously monitors the transmission trying to determine if arbitration is in process. If the TWI has lost an arbitration, the Control Unit is informed. Correct action can then be taken and appropriate status codes generated.

### Address Match Unit

The Address Match unit checks if received address bytes match the 7-bit address in the TWI Address Register (TWAR). If the TWI General Call Recognition Enable (TWGCE) bit in the TWAR is written to one, all incoming address bits will also be compared against the General Call address. Upon an address match, the Control Unit is informed, allowing correct action to be taken. The TWI may or may not acknowledge its address, depending on settings in the TWCR. The Address Match unit is able to compare addresses even when the AVR MCU is in sleep mode, enabling the MCU to wake up if addressed by a Master.

### Control Unit

The Control unit monitors the TWI bus and generates responses corresponding to settings in the TWI Control Register (TWCR). When an event requiring the attention of the application occurs on the TWI bus, the TWI Interrupt Flag (TWINT) is asserted. In the next clock cycle, the TWI Status Register (TWSR) is updated with a status code identifying the event. The TWSR only contains relevant status information when the TWI Interrupt Flag is asserted. At all other times, the TWSR contains a special status code indicating that no relevant status information is available. As long as the TWINT Flag is

set, the SCL line is held low. This allows the application software to complete its tasks before allowing the TWI transmission to continue.

The TWINT Flag is set in the following situations:

- After the TWI has transmitted a START/REPEATED START condition.
- After the TWI has transmitted SLA+R/W.
- After the TWI has transmitted an address byte.
- After the TWI has lost arbitration.
- After the TWI has been addressed by own Slave address or general call.
- After the TWI has received a data byte.
- After a STOP or REPEATED START has been received while still addressed as a Slave.
- When a bus error has occurred due to an illegal START or STOP condition.

## TWI Register Description

### TWI Bit Rate Register – TWBR

Bit	7	6	5	4	3	2	1	0	
	<b>TWBR7</b>	<b>TWBR6</b>	<b>TWBR5</b>	<b>TWBR4</b>	<b>TWBR3</b>	<b>TWBR2</b>	<b>TWBR1</b>	<b>TWBR0</b>	<b>TWBR</b>
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

#### • Bits 7..0 – TWI Bit Rate Register

TWBR selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the Master modes. See “Bit Rate Generator Unit” on page 178 for calculating bit rates.

### TWI Control Register – TWCR

Bit	7	6	5	4	3	2	1	0	
	<b>TWINT</b>	<b>TWEA</b>	<b>TWSTA</b>	<b>TWSTO</b>	<b>TWWC</b>	<b>TWEN</b>	–	<b>TWIE</b>	<b>TWCR</b>
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The TWCR is used to control the operation of the TWI. It is used to enable the TWI, to initiate a master access by applying a START condition to the bus, to generate a receiver acknowledge, to generate a stop condition, and to control halting of the bus while the data to be written to the bus are written to the TWDR. It also indicates a write collision if data is attempted written to TWDR while the register is inaccessible.

#### • Bit 7 – TWINT: TWI Interrupt Flag

This bit is set by hardware when the TWI has finished its current job and expects application software response. If the I-bit in SREG and TWIE in TWCR are set, the MCU will jump to the TWI Interrupt Vector. While the TWINT Flag is set, the SCL low period is stretched. The TWINT Flag must be cleared by software by writing a logic one to it. Note that this flag is not automatically cleared by hardware when executing the interrupt routine. Also note that clearing this flag starts the operation of the TWI, so all accesses to the TWI Address Register (TWAR), TWI Status Register (TWSR), and TWI Data Register (TWDR) must be complete before clearing this flag.

#### • Bit 6 – TWEA: TWI Enable Acknowledge Bit

The TWEA bit controls the generation of the acknowledge pulse. If the TWEA bit is written to one, the ACK pulse is generated on the TWI bus if the following conditions are met:



1. The device's own slave address has been received.
2. A general call has been received, while the TWGCE bit in the TWAR is set.
3. A data byte has been received in Master Receiver or Slave Receiver mode.

By writing the TWEA bit to zero, the device can be virtually disconnected from the Two-wire Serial Bus temporarily. Address recognition can then be resumed by writing the TWEA bit to one again.

- **Bit 5 – TWSTA: TWI START Condition Bit**

The application writes the TWSTA bit to one when it desires to become a Master on the Two-wire Serial Bus. The TWI hardware checks if the bus is available, and generates a START condition on the bus if it is free. However, if the bus is not free, the TWI waits until a STOP condition is detected, and then generates a new START condition to claim the bus Master status. TWSTA must be cleared by software when the START condition has been transmitted.

- **Bit 4 – TWSTO: TWI STOP Condition Bit**

Writing the TWSTO bit to one in Master mode will generate a STOP condition on the Two-wire Serial Bus. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically. In Slave mode, setting the TWSTO bit can be used to recover from an error condition. This will not generate a STOP condition, but the TWI returns to a well-defined unaddressed Slave mode and releases the SCL and SDA lines to a high impedance state.

- **Bit 3 – TWWC: TWI Write Collision Flag**

The TWWC bit is set when attempting to write to the TWI Data Register – TWDR when TWINT is low. This flag is cleared by writing the TWDR Register when TWINT is high.

- **Bit 2 – TWEN: TWI Enable Bit**

The TWEN bit enables TWI operation and activates the TWI interface. When TWEN is written to one, the TWI takes control over the I/O pins connected to the SCL and SDA pins, enabling the slew-rate limiters and spike filters. If this bit is written to zero, the TWI is switched off and all TWI transmissions are terminated, regardless of any ongoing operation.

- **Bit 1 – Res: Reserved Bit**

This bit is a reserved bit and will always read as zero.

- **Bit 0 – TWIE: TWI Interrupt Enable**

When this bit is written to one, and the I-bit in SREG is set, the TWI interrupt request will be activated for as long as the TWINT flag is high.

## TWI Status Register – TWSR

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

- **Bits 7..3 – TWS: TWI Status**

These five bits reflect the status of the TWI logic and the Two-wire Serial Bus. The different status codes are described later in this section. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the prescaler bits to zero when checking the status bits. This makes status checking independent of prescaler setting. This approach is used in this datasheet, unless otherwise noted.

- **Bit 2 – Res: Reserved bit**

This bit is reserved and will always read as zero.

- **Bits 1..0 – TWPS: TWI Prescaler Bits**

These bits can be read and written, and control the bit rate prescaler.

**Table 74. TWI Bit Rate Prescaler**

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

To calculate bit rates, see “Bit Rate Generator Unit” on page 178. The value of TWPS1..0 is used in the equation.

## TWI Data Register – TWDR

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

In Transmit mode, TWDR contains the next byte to be transmitted. In Receive mode, the TWDR contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when the TWI Interrupt Flag (TWINT) is set by hardware. Note that the Data Register cannot be initialized by the user before the first interrupt occurs. The data in TWDR remains stable as long as TWINT is set. While data is shifted out, data on the bus is simultaneously shifted in. TWDR always contains the last byte present on the bus, except after a wake-up from a sleep mode by the TWI interrupt. In this case, the contents of TWDR is undefined. In the case of a lost bus arbitration, no data is lost in the transition from Master to Slave. Handling of the ACK bit is controlled automatically by the TWI logic, the CPU cannot access the ACK bit directly.

- **Bits 7..0 – TWD: TWI Data Register**

These eight bits constitute the next data byte to be transmitted, or the latest data byte received on the Two-wire Serial Bus.

## TWI (Slave) Address Register – TWAR

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR



Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	1	1	1	1	1	1	1	0

The TWAR should be loaded with the 7-bit slave address (in the seven most significant bits of TWAR) to which the TWI will respond when programmed as a Slave Transmitter or Receiver, and not needed in the Master modes. In multimaster systems, TWAR must be set in masters which can be addressed as slaves by other masters.

The LSB of TWAR is used to enable recognition of the general call address (0x00). There is an associated address comparator that looks for the slave address (or general call address if enabled) in the received serial address. If a match is found, an interrupt request is generated.

- **Bits 7..1 – TWA: TWI (Slave) Address Register**

These seven bits constitute the slave address of the TWI unit.

- **Bit 0 – TWGCE: TWI General Call Recognition Enable Bit**

If set, this bit enables the recognition of a General Call given over the Two-wire Serial Bus.

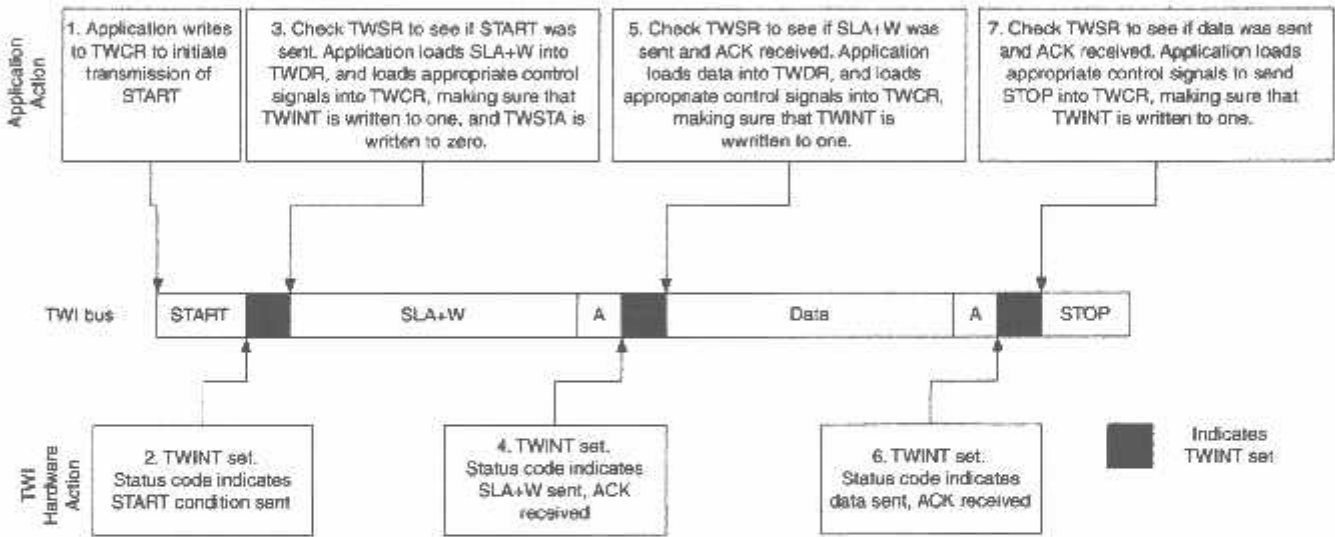
Using the TWI

The AVR TWI is byte-oriented and interrupt based. Interrupts are issued after all bus events, like reception of a byte or transmission of a START condition. Because the TWI is interrupt-based, the application software is free to carry on other operations during a TWI byte transfer. Note that the TWI Interrupt Enable (TWIE) bit in TWCR together with the Global Interrupt Enable bit in SREG allow the application to decide whether or not assertion of the TWINT Flag should generate an interrupt request. If the TWIE bit is cleared, the application must poll the TWINT Flag in order to detect actions on the TWI bus.

When the TWINT Flag is asserted, the TWI has finished an operation and awaits application response. In this case, the TWI Status Register (TWSR) contains a value indicating the current state of the TWI bus. The application software can then decide how the TWI should behave in the next TWI bus cycle by manipulating the TWCR and TWDR Registers.

Figure 85 is a simple example of how the application can interface to the TWI hardware. In this example, a Master wishes to transmit a single data byte to a Slave. This description is quite abstract, a more detailed explanation follows later in this section. A simple code example implementing the desired behavior is also presented.

Figure 85. Interfacing the Application to the TWI in a Typical Transmission



1. The first step in a TWI transmission is to transmit a START condition. This is done by writing a specific value into TWCR, instructing the TWI hardware to transmit a START condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the START condition.
2. When the START condition has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the START condition has successfully been sent.
3. The application software should now examine the value of TWSR, to make sure that the START condition was successfully transmitted. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must

load SLA+W into TWDR. Remember that TWDR is used both for address and data. After TWDR has been loaded with the desired SLA+W, a specific value must be written to TWCR, instructing the TWI hardware to transmit the SLA+W present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the address packet.

4. When the address packet has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the address packet has successfully been sent. The status code will also reflect whether a Slave acknowledged the packet or not.
5. The application software should now examine the value of TWSR, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load a data packet into TWDR. Subsequently, a specific value must be written to TWCR, instructing the TWI hardware to transmit the data packet present in TWDR. The value which to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the data packet.
6. When the data packet has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the data packet has successfully been sent. The status code will also reflect whether a Slave acknowledged the packet or not.
7. The application software should now examine the value of TWSR, to make sure that the data packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must write a specific value to TWCR, instructing the TWI hardware to transmit a STOP condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the STOP condition. Note that TWINT is NOT set after a STOP condition has been sent.

Even though this example is simple, it shows the principles involved in all TWI transmissions. These can be summarized as follows:

- When the TWI has finished an operation and expects application response, the TWINT Flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT Flag is set, the user must update all TWI Registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI Register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be set. Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

In the following an assembly and C implementation of the example is given. Note that the code below assumes that several definitions have been made, for example by using include-files.

	Assembly Code Example	C Example	Comments
1	<pre>ldi r16, (1&lt;&lt;TWINT) (1&lt;&lt;TWSTA)  (1&lt;&lt;TWEN) out TWCR, r16</pre>	<pre>TWCR = (1&lt;&lt;TWINT) (1&lt;&lt;TWSTA)  (1&lt;&lt;TWEN)</pre>	Send START condition.
2	<pre>wait1: in r16, TWCR sbrs r16, TWINT rjmp wait1</pre>	<pre>while (!(TWCR &amp; (1&lt;&lt;TWINT))) ;</pre>	Wait for TWINT Flag set. This indicates that the START condition has been transmitted.
3	<pre>in r16, TWSR andi r16, 0xF8 cpi r16, START brne ERROR</pre>	<pre>if ((TWSR &amp; 0xF8) != START) ERROR();</pre>	Check value of TWI Status Register. Mask prescaler bits. If status different from START go to ERROR.
	<pre>ldi r16, SLA_W out TWDR, r16 ldi r16, (1&lt;&lt;TWINT)   (1&lt;&lt;TWEN) out TWCR, r16</pre>	<pre>TWDR = SLA_W; TWCR = (1&lt;&lt;TWINT)   (1&lt;&lt;TWEN);</pre>	Load SLA_W into TWDR Register. Clear TWINT bit in TWCR to start transmission of address.
4	<pre>wait2: in r16, TWCR sbrs r16, TWINT rjmp wait2</pre>	<pre>while (!(TWCR &amp; (1&lt;&lt;TWINT))) ;</pre>	Wait for TWINT Flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received.
5	<pre>in r16, TWSR andi r16, 0xF8 cpi r16, MT_SLA_ACK brne ERROR</pre>	<pre>if ((TWSR &amp; 0xF8) != MT_SLA_ACK) ERROR();</pre>	Check value of TWI Status Register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR.
	<pre>ldi r16, DATA out TWDR, r16 ldi r16, (1&lt;&lt;TWINT)   (1&lt;&lt;TWEN) out TWCR, r16</pre>	<pre>TWDR = DATA; TWCR = (1&lt;&lt;TWINT)   (1&lt;&lt;TWEN);</pre>	Load DATA into TWDR Register. Clear TWINT bit in TWCR to start transmission of address.
3	<pre>wait3: in r16, TWCR sbrs r16, TWINT rjmp wait3</pre>	<pre>while (!(TWCR &amp; (1&lt;&lt;TWINT))) ;</pre>	Wait for TWINT Flag set. This indicates that the DATA has been transmitted, and ACK/NACK has been received.
7	<pre>in r16, TWSR andi r16, 0xF8 cpi r16, MT_DATA_ACK brne ERROR</pre>	<pre>if ((TWSR &amp; 0xF8) != MT_DATA_ACK) ERROR();</pre>	Check value of TWI Status Register. Mask prescaler bits. If status different from MT_DATA_ACK go to ERROR.
	<pre>ldi r16, (1&lt;&lt;TWINT) (1&lt;&lt;TWEN)  (1&lt;&lt;TWSTO) out TWCR, r16</pre>	<pre>TWCR = (1&lt;&lt;TWINT) (1&lt;&lt;TWEN)  (1&lt;&lt;TWSTO);</pre>	Transmit STOP condition.

## Transmission Modes

The TWI can operate in one of four major modes. These are named Master Transmitter (MT), Master Receiver (MR), Slave Transmitter (ST), and Slave Receiver (SR). Several of these modes can be used in the same application. As an example, the TWI can use MT mode to write data into a TWI EEPROM, MR mode to read the data back from the EEPROM. If other masters are present in the system, some of these might transmit data to the TWI, and then SR mode would be used. It is the application software that decides which modes are legal.

The following sections describe each of these modes. Possible status codes are described along with figures detailing data transmission in each of the modes. These figures contain the following abbreviations:

S:	START condition
Rs:	REPEATED START condition
R:	Read bit (high level at SDA)
W:	Write bit (low level at SDA)
A:	Acknowledge bit (low level at SDA)
$\bar{A}$ :	Not acknowledge bit (high level at SDA)
Data:	8-bit data byte
P:	STOP condition
SLA:	Slave Address

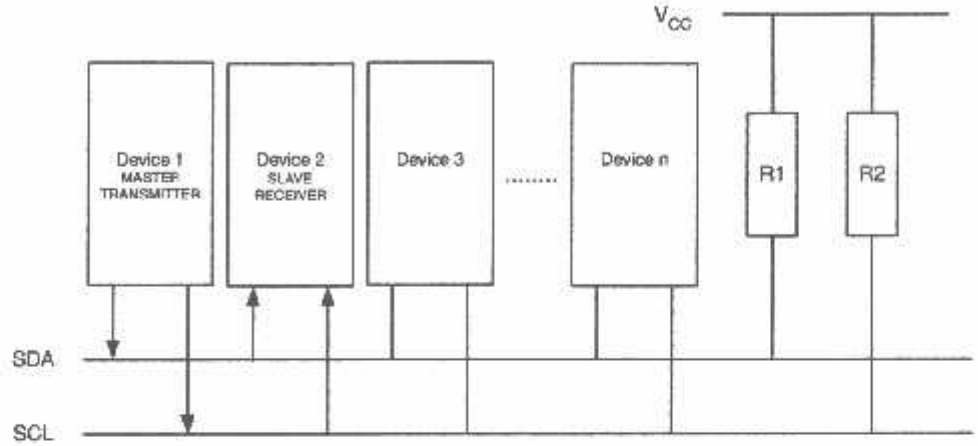
In Figure 87 to Figure 93, circles are used to indicate that the TWINT Flag is set. The numbers in the circles show the status code held in TWSR, with the prescaler bits masked to zero. At these points, actions must be taken by the application to continue or complete the TWI transfer. The TWI transfer is suspended until the TWINT Flag is cleared by software.

When the TWINT Flag is set, the status code in TWSR is used to determine the appropriate software action. For each status code, the required software action and details of the following serial transfer are given in Table 75 to Table 78. Note that the prescaler bits are masked to zero in these tables.

## Master Transmitter Mode

In the Master Transmitter mode, a number of data bytes are transmitted to a Slave Receiver (see Figure 86). In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether Master Transmitter or Master Receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 86.** Data Transfer in Master Transmitter Mode



A START condition is sent by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Value	1	X	1	0	X	1	0	X

TWEN must be set to enable the Two-wire Serial Interface, TWSTA must be written to one to transmit a START condition and TWINT must be written to one to clear the TWINT Flag. The TWI will then test the Two-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT Flag is set by hardware, and the status code in TWSR will be 0x08 (see Table 75). In order to enter MT mode, SLA+W must be transmitted. This is done by writing SLA+W to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Value	1	X	0	0	X	1	0	X

When SLA+W have been transmitted and an acknowledgement bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in Master mode are 0x18, 0x20, or 0x38. The appropriate action to be taken for each of these status codes is detailed in Table 75.

When SLA+W has been successfully transmitted, a data packet should be transmitted. This is done by writing the data byte to TWDR. TWDR must only be written when TWINT is high. If not, the access will be discarded, and the Write Collision bit (TWWC) will be set in the TWCR Register. After updating TWDR, the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Value	1	X	0	0	X	1	0	X



This scheme is repeated until the last byte has been sent and the transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Value	1	X	0	1	X	1	0	X

A REPEATED START condition is generated by writing the following value to TWCR:

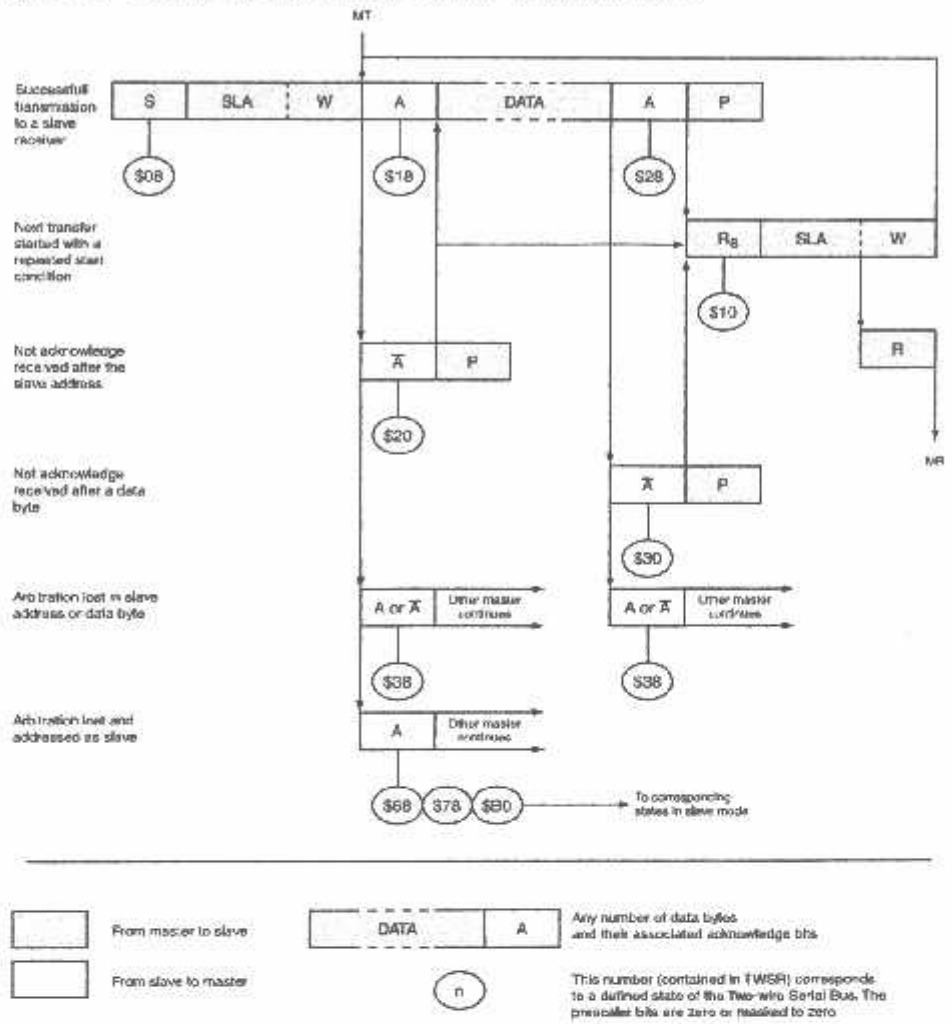
TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Value	1	X	1	0	X	1	0	X

After a repeated START condition (state 0x10) the Two-wire Serial Interface can access the same Slave again, or a new Slave without transmitting a STOP condition. Repeated START enables the Master to switch between Slaves, Master Transmitter mode and Master Receiver mode without losing control of the bus..

**Table 75. Status Codes for Master Transmitter Mode**

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+W	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
0x10	A repeated START condition has been transmitted	Load SLA+W or	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received SLA+R will be transmitted; Logic will switch to Master Receiver mode
		Load SLA+R	0	0	1	X	
0x18	SLA+W has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received Repeated START will be transmitted STOP condition will be transmitted and TWSTO Flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
		No TWDR action or	1	0	1	X	
		No TWDR action or	0	1	1	X	
0x20	SLA+W has been transmitted; NOT ACK has been received	No TWDR action	1	1	1	X	
		Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received Repeated START will be transmitted STOP condition will be transmitted and TWSTO Flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
		No TWDR action or	1	0	1	X	
No TWDR action or	0	1	1	X			
0x28	Data byte has been transmitted; ACK has been received	No TWDR action	1	1	1	X	
		Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received Repeated START will be transmitted STOP condition will be transmitted and TWSTO Flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
		No TWDR action or	1	0	1	X	
No TWDR action or	0	1	1	X			
0x30	Data byte has been transmitted; NOT ACK has been received	No TWDR action	1	1	1	X	
		Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received Repeated START will be transmitted STOP condition will be transmitted and TWSTO Flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
		No TWDR action or	1	0	1	X	
No TWDR action or	0	1	1	X			
0x38	Arbitration lost in SLA+W or data bytes	No TWDR action	0	0	1	X	Two-wire Serial Bus will be released and not addressed Slave mode entered A START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	X	

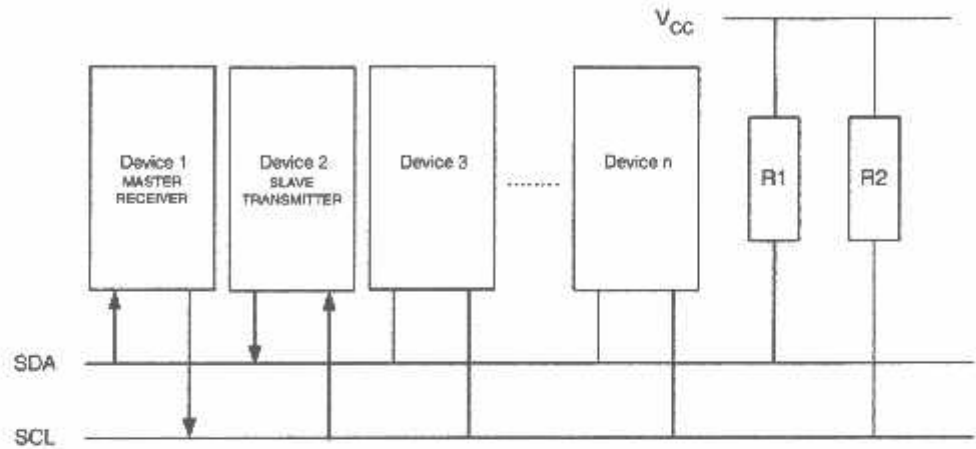
Figure 87. Formats and States in the Master Transmitter Mode



## Master Receiver Mode

In the Master Receiver mode, a number of data bytes are received from a Slave Transmitter (see Figure 88). In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether Master Transmitter or Master Receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered. If SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 88.** Data Transfer in Master Receiver Mode



A START condition is sent by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Value	1	X	1	0	X	1	0	X

TWEN must be written to one to enable the Two-wire Serial Interface, TWSTA must be written to one to transmit a START condition and TWINT must be set to clear the TWINT Flag. The TWI will then test the Two-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT Flag is set by hardware, and the status code in TWSR will be 0x08 (see Table 75). In order to enter MR mode, SLA+R must be transmitted. This is done by writing SLA+R to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Value	1	X	0	0	X	1	0	X

When SLA+R have been transmitted and an acknowledgement bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in Master mode are 0x38, 0x40, or 0x48. The appropriate action to be taken for each of these status codes is detailed in Table 76. Received data can be read from the TWDR Register when the TWINT Flag is set high by hardware. This scheme is repeated until the last byte has been received. After the last byte has been received, the MR should inform the ST by sending a NACK after the last received data byte. The transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Value	1	X	0	1	X	1	0	X

A REPEATED START condition is generated by writing the following value to TWCR:

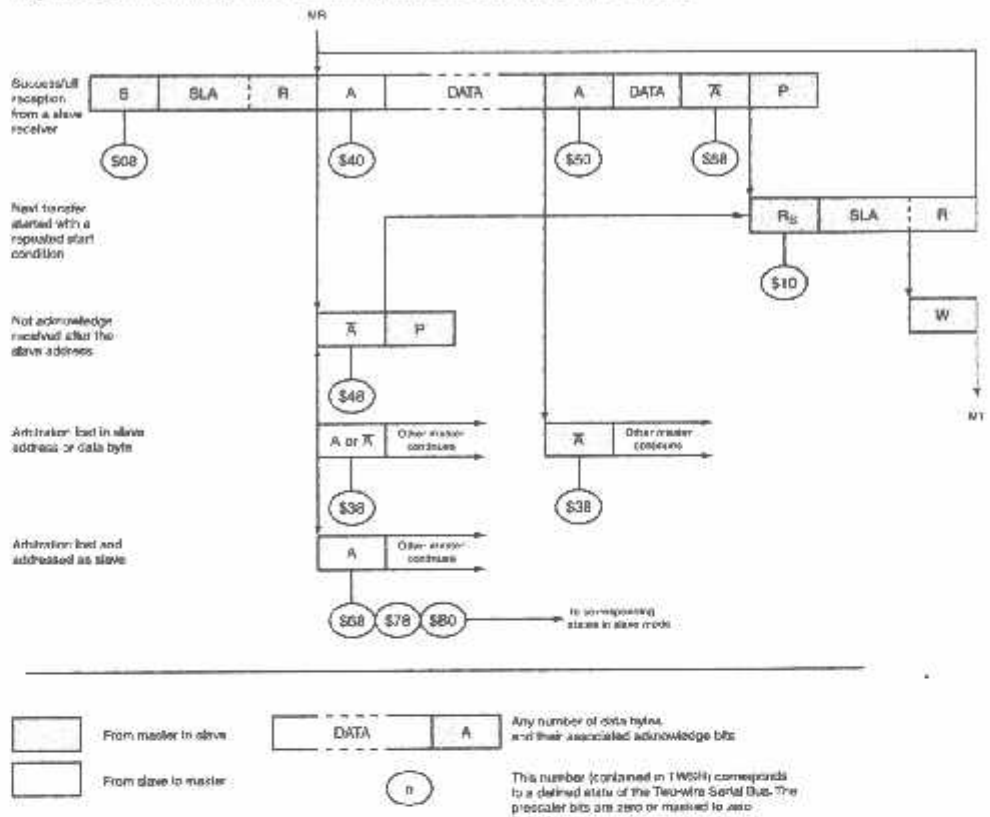
TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Value	1	X	1	0	X	1	0	X

After a repeated START condition (state 0x10) the Two-wire Serial Interface can access the same Slave again, or a new Slave without transmitting a STOP condition. Repeated START enables the Master to switch between Slaves, Master Transmitter mode and Master Receiver mode without losing control over the bus.

**Table 76. Status Codes for Master Receiver Mode**

Status Code (TWSR) Register Bits 0-7	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+R	0	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
0x10	A repeated START condition has been transmitted	Load SLA+R or	0	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received SLA+W will be transmitted Logic will switch to Master Transmitter mode
		Load SLA+W	0	0	1	X	
0x38	Arbitration lost in SLA+R or NOT ACK bit	No TWDR action or	0	0	1	X	Two-wire Serial Bus will be released and not addressed Slave mode will be entered A START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	X	
0x40	SLA+R has been transmitted; ACK has been received	No TWDR action or	0	0	1	0	Data byte will be received and NOT ACK will be returned Data byte will be received and ACK will be returned
		No TWDR action	0	0	1	1	
0x48	SLA+R has been transmitted; NOT ACK has been received	No TWDR action or	1	0	1	X	Repeated START will be transmitted STOP condition will be transmitted and TWSTO Flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
		No TWDR action or	0	1	1	X	
		No TWDR action	1	1	1	X	
0x50	Data byte has been received; ACK has been returned	Read data byte or	0	0	1	0	Data byte will be received and NOT ACK will be returned Data byte will be received and ACK will be returned
		Read data byte	0	0	1	1	
0x58	Data byte has been received; NOT ACK has been returned	Read data byte or	1	0	1	X	Repeated START will be transmitted STOP condition will be transmitted and TWSTO flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
		Read data byte or	0	1	1	X	
		Read data byte	1	1	1	X	

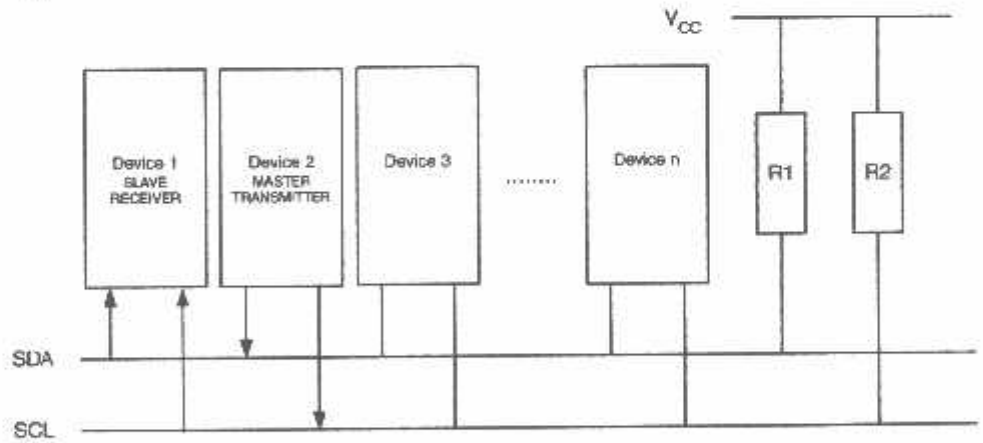
**Figure 89. Formats and States in the Master Receiver Mode**



**Slave Receiver Mode**

In the Slave Receiver mode, a number of data bytes are received from a Master Transmitter (see Figure 90). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 90. Data Transfer in Slave Receiver Mode**



To initiate the Slave Receiver mode, TWAR and TWCR must be initialized as follows:

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Value	Device's Own Slave Address							

The upper seven bits are the address to which the Two-wire Serial Interface will respond when addressed by a Master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Value	0	1	0	0	0	1	0	X

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "0" (write), the TWI will operate in SR mode, otherwise ST mode is entered. After its own slave address and the write bit have been received, the TWINT Flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 77. The Slave Receiver mode may also be entered if arbitration is lost while the TWI is in the Master mode (see states 0x68 and 0x78).

If the TWEA bit is reset during a transfer, the TWI will return a "Not Acknowledge" ("1") to SDA after the next received data byte. This can be used to indicate that the Slave is not able to receive any more bytes. While TWEA is zero, the TWI does not acknowledge its own slave address. However, the Two-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the Two-wire Serial Bus.

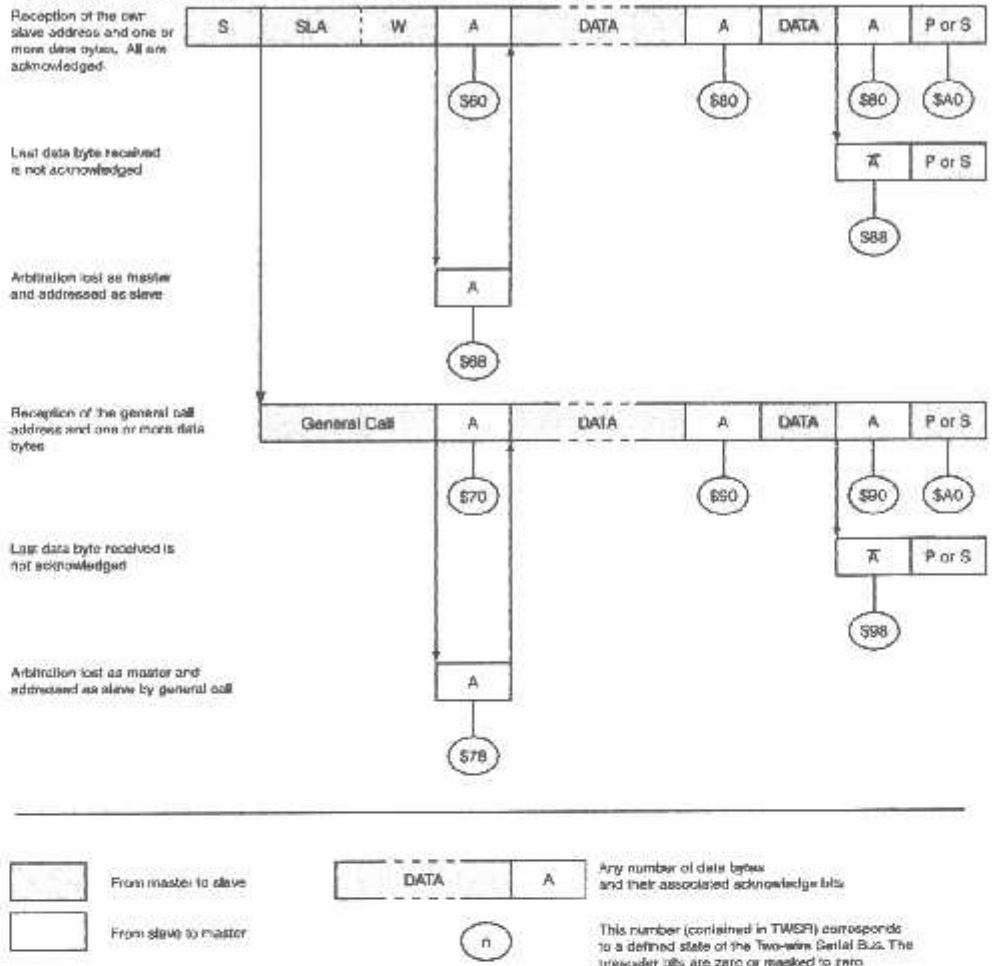
In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the Two-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock low during the wake up and until the TWINT Flag is cleared (by writing it to one). Further data reception will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the Two-wire Serial Interface Data Register – TWDR – does not reflect the last byte present on the bus when waking up from these sleep modes.

**Table 77. Status Codes for Slave Receiver Mode**

Status Code (TWSR) (rescaler Bits 7:0)	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/From TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x60	Own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x68	Arbitration lost in SLA+R/W as Master; own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x70	General call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x78	Arbitration lost in SLA+R/W as Master; General call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x80	Previously addressed with own SLA+W; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned
0x88	Previously addressed with own SLA+W; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized;
		Read data byte or	1	0	1	0	GCA will be recognized if TWGCE = "1"; Switched to the not addressed Slave mode; no recognition of own SLA or GCA;
		Read data byte	1	0	1	1	a START condition will be transmitted when the bus becomes free
0x90	Previously addressed with general call; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned
0x98	Previously addressed with general call; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized;
		Read data byte or	1	0	1	0	GCA will be recognized if TWGCE = "1"; Switched to the not addressed Slave mode; no recognition of own SLA or GCA;
		Read data byte	1	0	1	1	a START condition will be transmitted when the bus becomes free
0xA0	A STOP condition or repeated START condition has been received while still addressed as Slave	No action	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
			0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized;
			1	0	1	0	GCA will be recognized if TWGCE = "1"; Switched to the not addressed Slave mode; no recognition of own SLA or GCA;
			1	0	1	1	a START condition will be transmitted when the bus becomes free

**Figure 91. Formats and States in the Slave Receiver Mode**

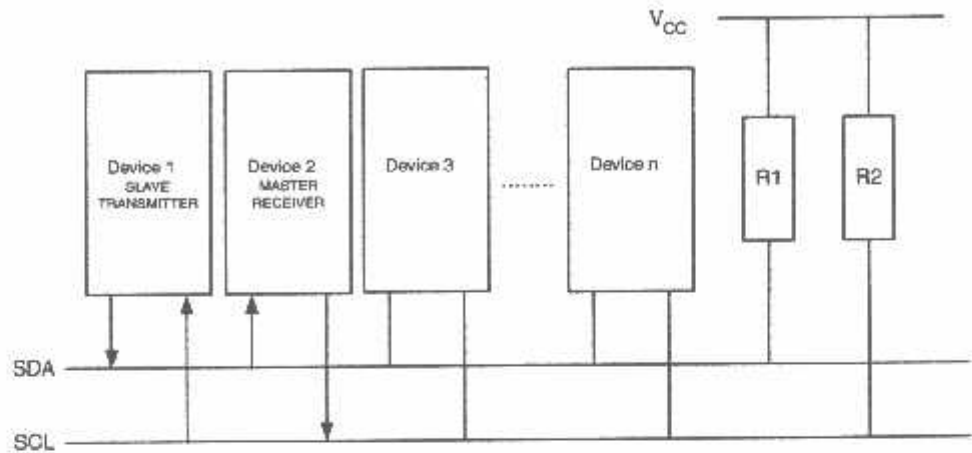




## Slave Transmitter Mode

In the Slave Transmitter mode, a number of data bytes are transmitted to a Master Receiver (see Figure 92). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

**Figure 92.** Data Transfer in Slave Transmitter Mode



To initiate the Slave Transmitter mode, TWAR and TWCR must be initialized as follows:

TWAR	TWA8	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Value	Device's Own Slave Address							

The upper seven bits are the address to which the Two-wire Serial Interface will respond when addressed by a Master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
Value	0	1	0	0	0	1	0	X

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "1" (read), the TWI will operate in ST mode, otherwise SR mode is entered. After its own slave address and the write bit have been received, the TWINT Flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 78. The Slave Transmitter mode may also be entered if arbitration is lost while the TWI is in the Master mode (see state 0xB0).

If the TWEA bit is written to zero during a transfer, the TWI will transmit the last byte of the transfer. State 0xC0 or state 0xC8 will be entered, depending on whether the Master Receiver transmits a NACK or ACK after the final byte. The TWI is switched to the not addressed Slave mode, and will ignore the Master if it continues the transfer. Thus the Master Receiver receives all "1" as serial data. State 0xCB is entered if the Master demands additional data bytes (by transmitting ACK), even though the Slave has transmitted the last byte (TWEA zero and expecting NACK from the Master).

While TWEA is zero, the TWI does not respond to its own slave address. However, the Two-wire Serial Bus is still monitored and address recognition may resume at any time.

by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the Two-wire Serial Bus.

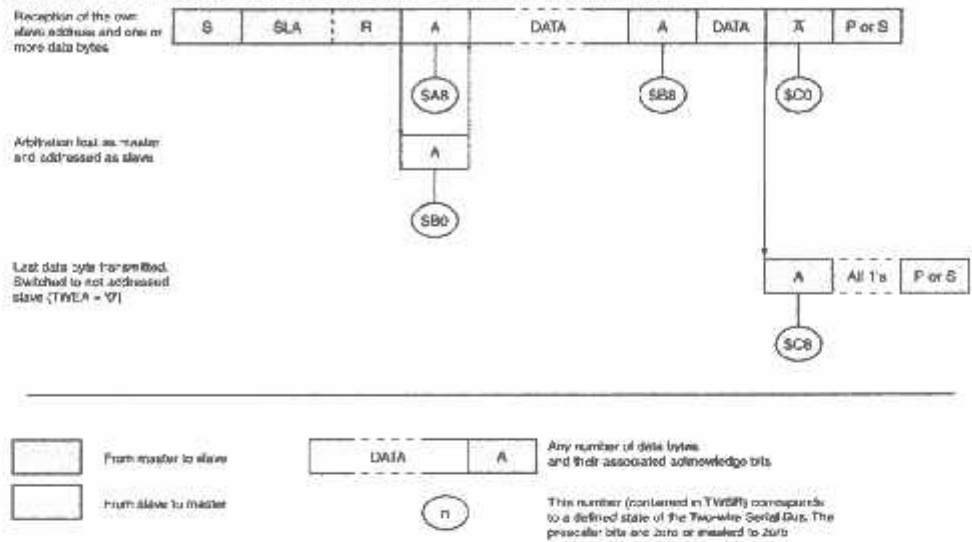
In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the Two-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock will low during the wake up and until the TWINT Flag is cleared (by writing it to one). Further data transmission will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the Two-wire Serial Interface Data Register – TWDR – does not reflect the last byte present on the bus when waking up from these sleep modes.

Table 78. Status Codes for Slave Transmitter Mode

Status Code (WSR) (rescaler Bits re 0)	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
xA8	Own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received Data byte will be transmitted and ACK should be received
		Load data byte	X	0	1	1	
xB0	Arbitration lost in SLA+R/W as Master; own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received Data byte will be transmitted and ACK should be received
		Load data byte	X	0	1	1	
xB8	Data byte in TWDR has been transmitted; ACK has been received	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received Data byte will be transmitted and ACK should be received
		Load data byte	X	0	1	1	
xC0	Data byte in TWDR has been transmitted; NOT ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1" Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"; a START condition will be transmitted when the bus becomes free
		No TWDR action or	0	0	1	1	
		No TWDR action or	1	0	1	0	
		No TWDR action	1	0	1	1	
xC8	Last data byte in TWDR has been transmitted (TWEA = "0"); ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1" Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = "1"; a START condition will be transmitted when the bus becomes free
		No TWDR action or	0	0	1	1	
		No TWDR action or	1	0	1	0	
		No TWDR action	1	0	1	1	

**Figure 93. Formats and States in the Slave Transmitter Mode**



**Miscellaneous States**

There are two status codes that do not correspond to a defined TWI state, see Table 79.

Status 0xF8 indicates that no relevant information is available because the TWINT Flag is not set. This occurs between other states, and when the TWI is not involved in a serial transfer.

Status 0x00 indicates that a bus error has occurred during a Two-wire Serial Bus transfer. A bus error occurs when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. When a bus error occurs, TWINT is set. To recover from a bus error, the TWSTO Flag must set and TWINT must be cleared by writing a logic one to it. This causes the TWI to enter the not addressed Slave mode and to clear the TWSTO Flag (no other bits in TWCR are affected). The SDA and SCL lines are released, and no STOP condition is transmitted.

**Table 79. Miscellaneous States**

Status Code (TWSP) / Rescale: Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/From TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0xF8	No relevant state information available; TWINT = '0'	No TWDR action	No TWCR Action				Wait or proceed current transfer
0x00	Bus error due to an illegal START or STOP condition	No TWDR action	0	1	1	X	Only the internal hardware is affected, no STOP condition is sent on the bus. In all cases, the bus is released and TWSTO is cleared.

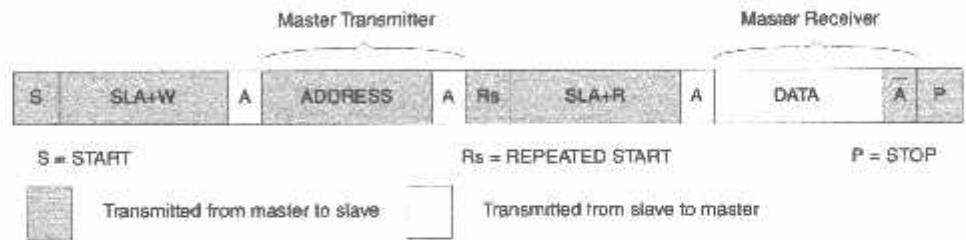
Combining Several TWI Modes

In some cases, several TWI modes must be combined in order to complete the desired action. Consider for example reading data from a serial EEPROM. Typically, such a transfer involves the following steps:

1. The transfer must be initiated.
2. The EEPROM must be instructed what location should be read.
3. The reading must be performed.
4. The transfer must be finished.

Note that data is transmitted both from Master to Slave and vice versa. The Master must instruct the Slave what location it wants to read, requiring the use of the MT mode. Subsequently, data must be read from the Slave, implying the use of the MR mode. Thus, the transfer direction must be changed. The Master must keep control of the bus during all these steps, and the steps should be carried out as an atomical operation. If this principle is violated in a multimaster system, another Master can alter the data pointer in the EEPROM between steps 2 and 3, and the Master will read the wrong data location. Such a change in transfer direction is accomplished by transmitting a REPEATED START between the transmission of the address byte and reception of the data. After a REPEATED START, the Master keeps ownership of the bus. The following figure shows the flow in this transfer.

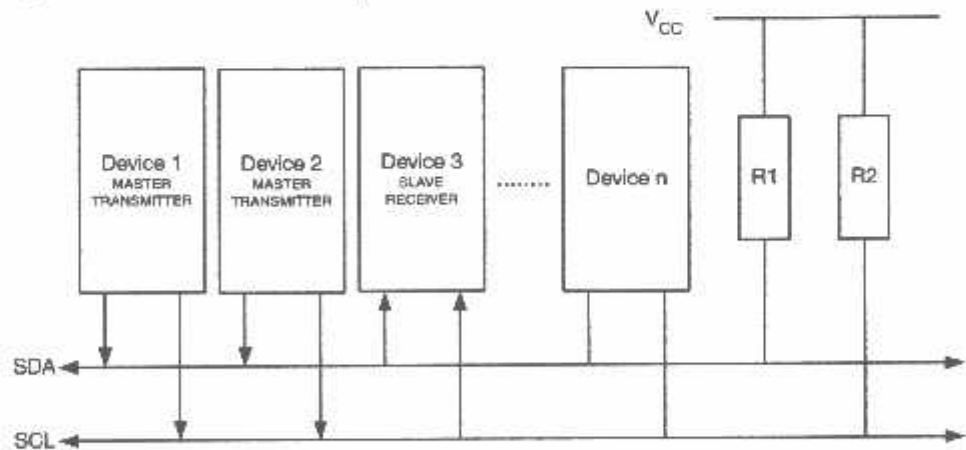
Figure 94. Combining Several TWI Modes to Access a Serial EEPROM



Multi-master Systems and Arbitration

If Multiple Masters are connected to the same bus, transmissions may be initiated simultaneously by one or more of them. The TWI standard ensures that such situations are handled in such a way that one of the masters will be allowed to proceed with the transfer, and that no data will be lost in the process. An example of an arbitration situation is depicted below, where two masters are trying to transmit data to a Slave Receiver.

Figure 95. An Arbitration Example

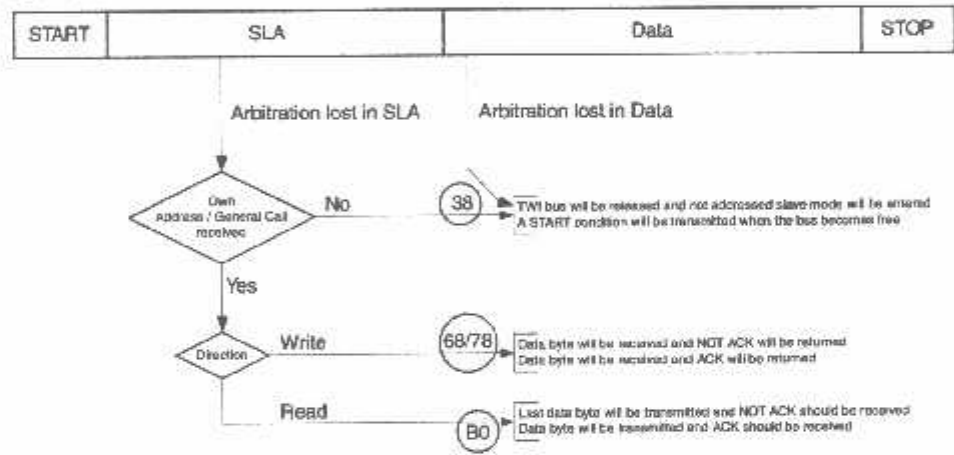


Several different scenarios may arise during arbitration, as described below:

- Two or more masters are performing identical communication with the same Slave. In this case, neither the Slave nor any of the masters will know about the bus contention.
- Two or more masters are accessing the same Slave with different data or direction bit. In this case, arbitration will occur, either in the READ/WRITE bit or in the data bits. The masters trying to output a one on SDA while another Master outputs a zero will lose the arbitration. Losing masters will switch to not addressed Slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.
- Two or more masters are accessing different slaves. In this case, arbitration will occur in the SLA bits. Masters trying to output a one on SDA while another Master outputs a zero will lose the arbitration. Masters losing arbitration in SLA will switch to Slave mode to check if they are being addressed by the winning Master. If addressed, they will switch to SR or ST mode, depending on the value of the READ/WRITE bit. If they are not being addressed, they will switch to not addressed Slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.

This is summarized in Figure 96. Possible status values are given in circles.

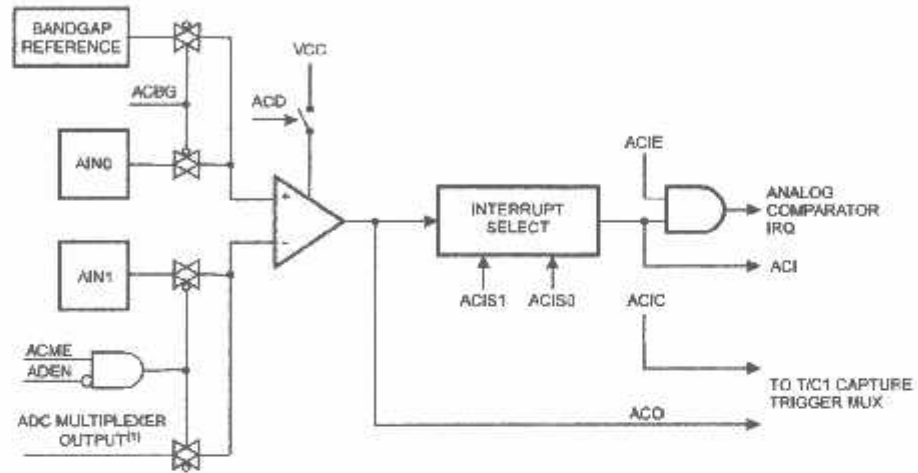
**Figure 96.** Possible Status Codes Caused by Arbitration



Analog Comparator

The Analog Comparator compares the input values on the positive pin AIN0 and negative pin AIN1. When the voltage on the positive pin AIN0 is higher than the voltage on the negative pin AIN1, the Analog Comparator Output, ACO, is set. The comparator's output can be set to trigger the Timer/Counter1 Input Capture function. In addition, the comparator can trigger a separate interrupt, exclusive to the Analog Comparator. The user can select Interrupt triggering on comparator output rise, fall or toggle. A block diagram of the comparator and its surrounding logic is shown in Figure 97.

Figure 97. Analog Comparator Block Diagram<sup>(2)</sup>



- Notes: 1. See Table 81 on page 203.  
 2. Refer to Figure 1 on page 2 and Table 26 on page 59 for Analog Comparator pin placement.

Special Function IO Register – SFIO

Bit	7	6	5	4	3	2	1	0	SFIO
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 3 – ACME: Analog Comparator Multiplexer Enable

When this bit is written logic one and the ADC is switched off (ADEN in ADCSRA is zero), the ADC multiplexer selects the negative input to the Analog Comparator. When this bit is written logic zero, AIN1 is applied to the negative input of the Analog Comparator. For a detailed description of this bit, see "Analog Comparator Multiplexed Input" on page 203.

Analog Comparator Control and Status Register – ACSR

Bit	7	6	5	4	3	2	1	0	ACSR
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

• Bit 7 – ACD: Analog Comparator Disable

When this bit is written logic one, the power to the Analog Comparator is switched off. This bit can be set at any time to turn off the Analog Comparator. This will reduce power consumption in active and Idle mode. When changing the ACD bit, the Analog Compar-

ator Interrupt must be disabled by clearing the ACIE bit in ACSR. Otherwise an interrupt can occur when the bit is changed.

- **Bit 6 – ACBG: Analog Comparator Bandgap Select**

When this bit is set, a fixed bandgap reference voltage replaces the positive input to the Analog Comparator. When this bit is cleared, AIN0 is applied to the positive input of the Analog Comparator. See "Internal Voltage Reference" on page 40.

- **Bit 5 – ACO: Analog Comparator Output**

The output of the Analog Comparator is synchronized and then directly connected to ACO. The synchronization introduces a delay of 1 - 2 clock cycles.

- **Bit 4 – ACI: Analog Comparator Interrupt Flag**

This bit is set by hardware when a comparator output event triggers the interrupt mode defined by ACIS1 and ACIS0. The Analog Comparator Interrupt routine is executed if the ACIE bit is set and the I-bit in SREG is set. ACI is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ACI is cleared by writing a logic one to the flag.

- **Bit 3 – ACIE: Analog Comparator Interrupt Enable**

When the ACIE bit is written logic one and the I-bit in the Status Register is set, the Analog Comparator Interrupt is activated. When written logic zero, the interrupt is disabled.

- **Bit 2 – ACIC: Analog Comparator Input Capture Enable**

When written logic one, this bit enables the Input Capture function in Timer/Counter1 to be triggered by the Analog Comparator. The comparator output is in this case directly connected to the Input Capture front-end logic, making the comparator utilize the noise canceler and edge select features of the Timer/Counter1 Input Capture interrupt. When written logic zero, no connection between the analog comparator and the Input Capture function exists. To make the comparator trigger the Timer/Counter1 Input Capture interrupt, the TICIE1 bit in the Timer Interrupt Mask Register (TIMSK) must be set.

- **Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select**

These bits determine which comparator events that trigger the Analog Comparator Interrupt. The different settings are shown in Table 80.

**Table 80. ACIS1/ACIS0 Settings**

ACIS1	ACIS0	Interrupt Mode
0	0	Comparator Interrupt on Output Toggle
0	1	Reserved
1	0	Comparator Interrupt on Falling Output Edge
1	1	Comparator Interrupt on Rising Output Edge

When changing the ACIS1/ACIS0 bits, the Analog Comparator Interrupt must be disabled by clearing its Interrupt Enable bit in the ACSR Register. Otherwise an interrupt can occur when the bits are changed.

**Analog Comparator Multiplexed Input**

It is possible to select any of the ADC7..0 pins to replace the negative input to the Analog Comparator. The ADC multiplexer is used to select this input, and consequently, the ADC must be switched off to utilize this feature. If the Analog Comparator Multiplexer Enable bit (ACME in SFIOR) is set and the ADC is switched off (ADEN in ADCSRA is zero), MUX2..0 in ADMUX select the input pin to replace the negative input to the Analog Comparator, as shown in Table 81. If ACME is cleared or ADEN is set, AIN1 is applied to the negative input to the Analog Comparator.

**Table 81. Analog Comparator Multiplexed Input**

ACME	ADEN	MUX2..0	Analog Comparator Negative Input
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7



## Analog-to-Digital Converter

### Features

- 10-bit Resolution
- 0.5 LSB Integral Non-linearity
- $\pm 2$  LSB Absolute Accuracy
- 65 - 260  $\mu$ s Conversion Time
- Up to 15 kSPS at Maximum Resolution
- 8 Multiplexed Single Ended Input Channels
- 7 Differential Input Channels
- 2 Differential Input Channels with Optional Gain of 10x and 200x<sup>(1)</sup>
- Optional Left Adjustment for ADC Result Readout
- 0 -  $V_{CC}$  ADC Input Voltage Range
- Selectable 2.56V ADC Reference Voltage
- Free Running or Single Conversion Mode
- ADC Start Conversion by Auto Triggering on Interrupt Sources
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

Note: 1. The differential input channel are not tested for devices in PDIP and PLCC Package. This feature is only guaranteed to work for devices in TQFP and QFN/MLF Packages.

The ATmega8535 features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows eight single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND).

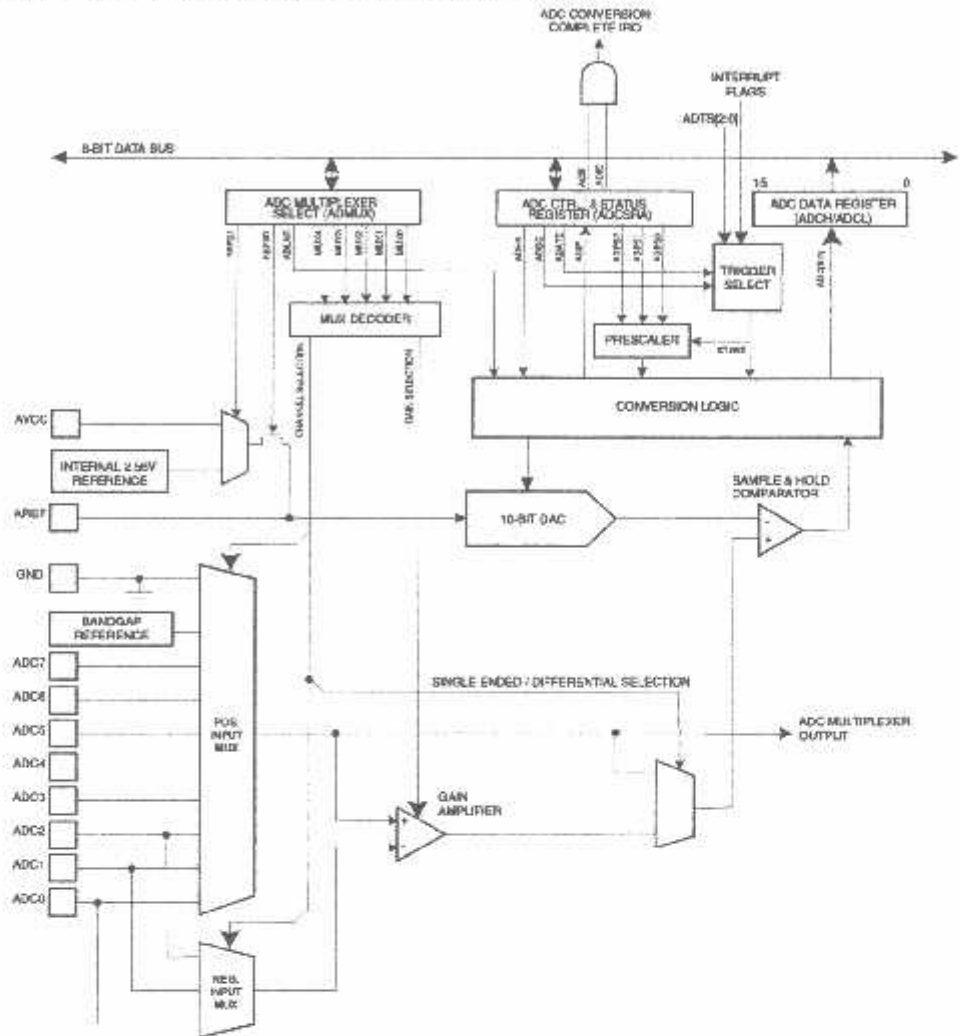
The device also supports 16 differential voltage input combinations. Two of the differential inputs (ADC1, ADC0 and ADC3, ADC2) are equipped with a programmable gain stage, providing amplification steps of 0 dB (1x), 20 dB (10x), or 46 dB (200x) on the differential input voltage before the A/D conversion. Seven differential analog input channels share a common negative terminal (ADC1), while any other ADC input can be selected as the positive input terminal. If 1x or 10x gain is used, 8-bit resolution can be expected. If 200x gain is used, 7-bit resolution can be expected.

The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 98.

The ADC has a separate analog supply voltage pin, AVCC. AVCC must not differ more than  $\pm 0.3$ V from  $V_{CC}$ . See the paragraph "ADC Noise Canceler" on page 212 on how to connect this pin.

Internal reference voltages of nominally 2.56V or AVCC are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.

Figure 98. Analog-to-Digital Converter Block Schematic



Operation

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AVCC or an internal 2.56V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

The analog input channel and differential gain are selected by writing to the MUX bits in ADMUX. Any of the ADC input pins, as well as GND and a fixed bandgap voltage reference, can be selected as single ended inputs to the ADC. A selection of ADC input pins can be selected as positive and negative inputs to the differential gain amplifier.

If differential channels are selected, the differential gain stage amplifies the voltage difference between the selected input channel pair by the selected gain factor. This amplified value then becomes the analog input to the ADC. If single ended channels are used, the gain amplifier is bypassed altogether.

The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering power saving sleep modes.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX.

If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the data registers belongs to the same conversion. Once ADCL is read, ADC access to data registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.

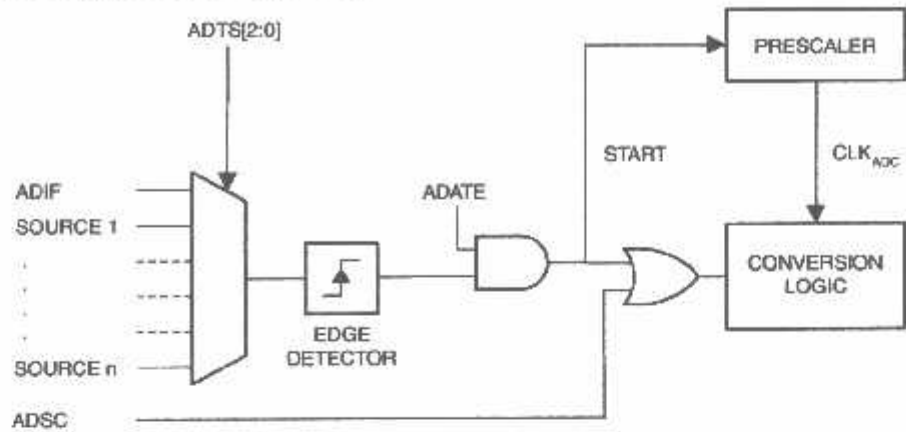
The ADC has its own interrupt which can be triggered when a conversion completes. When ADC access to the data registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

## Starting a Conversion

A single conversion is started by writing a logical one to the ADC Start Conversion bit, ADSC. This bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

Alternatively, a conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit, ADATE in ADCSRA. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in SFIOR (See description of the ADTS bits for a list of the trigger sources). When a positive edge occurs on the selected trigger signal, the ADC prescaler is reset and a conversion is started. This provides a method of starting conversions at fixed intervals. If the trigger signal still is set when the conversion completes, a new conversion will not be started. If another positive edge occurs on the trigger signal during conversion, the edge will be ignored. Note that an interrupt flag will be set even if the specific interrupt is disabled or the global interrupt enable bit in SREG is cleared. A conversion can thus be triggered without causing an interrupt. However, the interrupt flag must be cleared in order to trigger a new conversion at the next interrupt event.

Figure 99. ADC Auto Trigger Logic

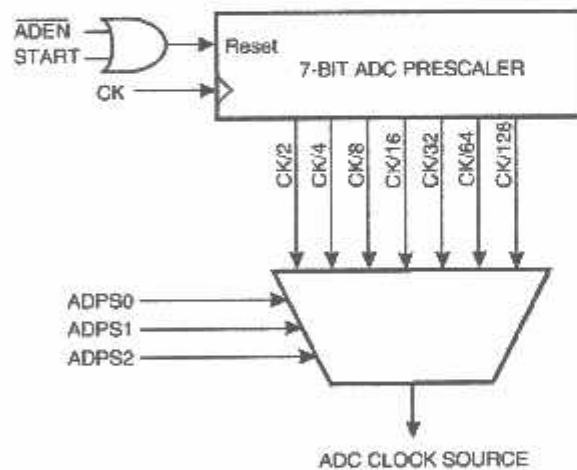


Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in Free Running mode, constantly sampling and updating the ADC Data Register. The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA. In this mode the ADC will perform successive conversions independently of whether the ADC Interrupt Flag, ADIF is cleared or not.

If Auto Triggering is enabled, single conversions can be started by writing ADSC in ADCSRA to one. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as one during a conversion, independently of how the conversion was started.

rescaling and  
onversion Timing

Figure 100. ADC Prescaler



By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate.

The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100 kHz. The prescaling is set by the ADPS bits in ADCSRA. The prescaler starts counting from the moment the ADC is switched on by setting the ADEN bit in ADCSRA. The prescaler keeps running for as long as the ADEN bit is set, and is continuously reset when ADEN is low.

When initiating a single ended conversion by setting the ADSC bit in ADCSRA, the conversion starts at the following rising edge of the ADC clock cycle. See "Differential Gain Channels" on page 210 for details on differential conversion timing.

A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry.

The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of an first conversion. When a conversion is complete, the result is written to the ADC Data Registers, and ADIF is set. In Single Conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

When Auto Triggering is used, the prescaler is reset when the trigger event occurs. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place two ADC clock cycles after the rising edge on the trigger source signal. Three additional CPU clock cycles are used for synchronization logic.

In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADSC remains high. For a summary of conversion times, see Table 82.

**Figure 101. ADC Timing Diagram, First Conversion (Single Conversion Mode)**

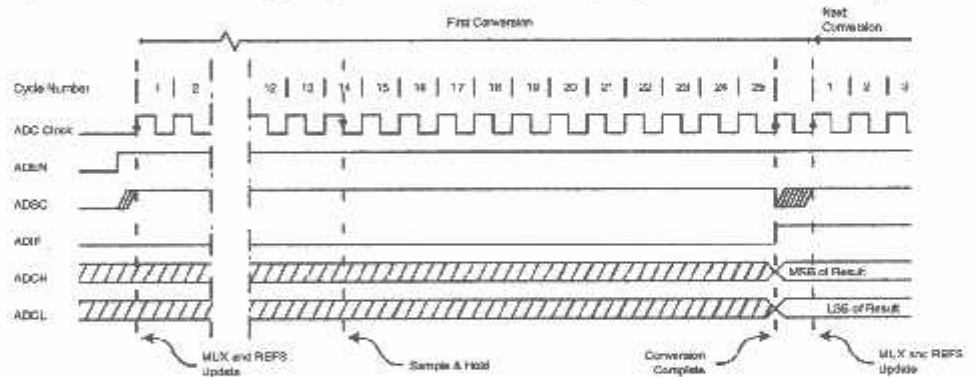


Figure 102. ADC Timing Diagram, Single Conversion

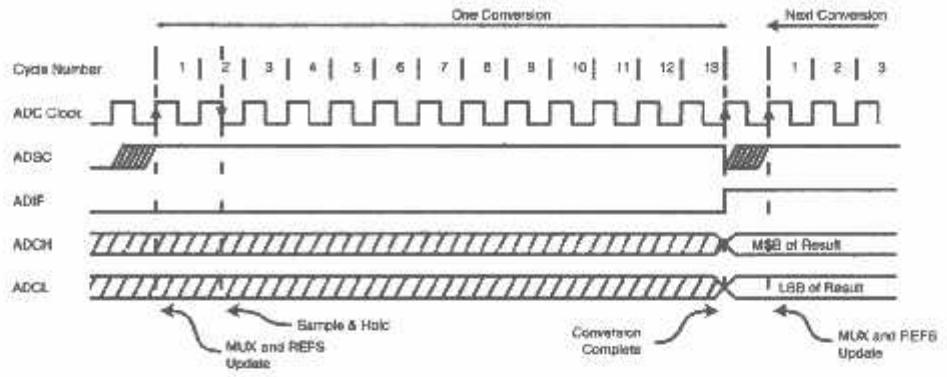


Figure 103. ADC Timing Diagram, Auto Triggered Conversion

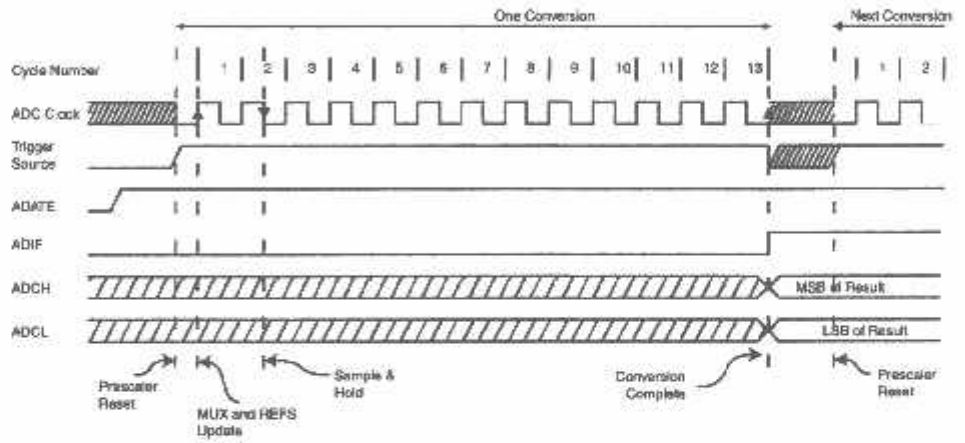


Figure 104. ADC Timing Diagram, Free Running Conversion

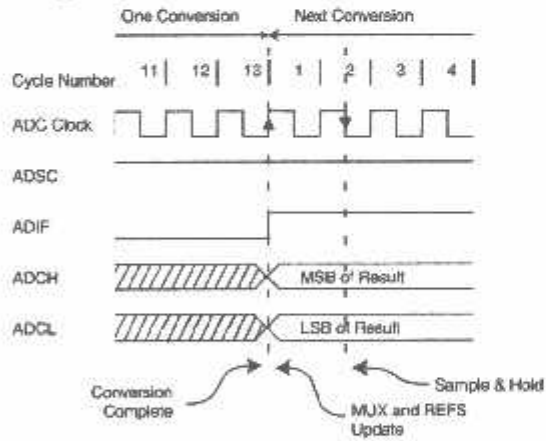




Table 82. ADC Conversion Time

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	14.5	25
Normal conversions, single ended	1.5	13
Auto Triggered conversions	2	13.5
Normal conversions, differential	1.5/2.5 <sup>(1)</sup>	13/14 <sup>(1)</sup>

Note: 1. Depending on the state of  $CK_{ADC2}$ .

#### Differential Gain Channels

When using differential gain channels, certain aspects of the conversion need to be taken into consideration.

Differential conversions are synchronized to the internal clock  $CK_{ADC2}$  equal to half the ADC clock. This synchronization is done automatically by the ADC interface in such a way that the sample-and-hold occurs at a specific phase of  $CK_{ADC2}$ . A conversion initiated by the user (i.e., all single conversions, and the first free running conversion) when  $CK_{ADC2}$  is low will take the same amount of time as a single ended conversion (13 ADC clock cycles from the next prescaled clock cycle). A conversion initiated by the user when  $CK_{ADC2}$  is high will take 14 ADC clock cycles due to the synchronization mechanism. In free running mode, a new conversion is initiated immediately after the previous conversion completes, and since  $CK_{ADC2}$  is high at this time, all automatically started (i.e., all but the first) free running conversions will take 14 ADC clock cycles.

The gain stage is optimized for a bandwidth of 4 kHz at all gain settings. Higher frequencies may be subjected to non-linear amplification. An external low-pass filter should be used if the input signal contains higher frequency components than the gain stage bandwidth. Note that the ADC clock frequency is independent of the gain stage bandwidth limitation. For example, the ADC clock period may be 6  $\mu$ s, allowing a channel to be sampled at 12 kSPS, regardless of the bandwidth of this channel.

If differential gain channels are used and conversions are started by Auto Triggering, the ADC must be switched off between conversions. When Auto Triggering is used, the ADC prescaler is reset before the conversion is started. Since the gain stage is dependent of a stable ADC clock prior to the conversion, this conversion will not be valid. By disabling and then re-enabling the ADC between each conversion (writing ADEN in ADCSRA to "0" then to "1"), only extended conversions are performed. The result from the extended conversions will be valid. See "Prescaling and Conversion Timing" on page 207 for timing details.

#### Changing Channel or Reference Selection

The MUXn and REFS1:0 bits in the ADMUX Register are single buffered through a temporary register to which the CPU has random access. This ensures that the channels and reference selection only takes place at a safe point during the conversion. The channel and reference selection is continuously updated until a conversion is started. Once the conversion starts, the channel and reference selection is locked to ensure a sufficient sampling time for the ADC. Continuous updating resumes in the last ADC clock cycle before the conversion completes (ADIF in ADCSRA is set). Note that the conversion starts on the following rising ADC clock edge after ADSC is written. The user is thus advised not to write new channel or reference selection values to ADMUX until one ADC clock cycle after ADSC is written.

If Auto Triggering is used, the exact time of the triggering event can be indeterminable. Special care must be taken when updating the ADMUX Register, in order to control which conversion will be affected by the new settings.

If both ADATE and ADEN is written to one, an interrupt event can occur at any time. If the ADMUX Register is changed in this period, the user cannot tell if the next conversion is based on the old or the new settings. ADMUX can be safely updated in the following ways:

1. When ADATE or ADEN is cleared.
2. During conversion, minimum one ADC clock cycle after the trigger event.
3. After a conversion, before the interrupt flag used as trigger source is cleared.

When updating ADMUX in one of these conditions, the new settings will affect the next ADC conversion.

Special care should be taken when changing differential channels. Once a differential channel has been selected, the gain stage may take as much as 125  $\mu$ s to stabilize to the new value. Thus conversions should not be started within the first 125  $\mu$ s after selecting a new differential channel. Alternatively, conversion results obtained within this period should be discarded.

The same settling time should be observed for the first differential conversion after changing ADC reference (by changing the REFS1:0 bits in ADMUX).

### ADC Input Channels

When changing channel selections, the user should observe the following guidelines to ensure that the correct channel is selected:

In Single Conversion mode, always select the channel before starting the conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the conversion to complete before changing the channel selection.

In Free Running mode, always select the channel before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the first conversion to complete, and then change the channel selection. Since the next conversion has already started automatically, the next result will reflect the previous channel selection. Subsequent conversions will reflect the new channel selection.

When switching to a differential gain channel, the first conversion result may have a poor accuracy due to the required settling time for the automatic offset cancellation circuitry. The user should preferably disregard the first conversion result.

### ADC Voltage Reference

The reference voltage for the ADC ( $V_{REF}$ ) indicates the conversion range for the ADC. Single ended channels that exceed  $V_{REF}$  will result in codes close to 0x3FF.  $V_{REF}$  can be selected as either AVCC, internal 2.56V reference, or external AREF pin.

AVCC is connected to the ADC through a passive switch. The internal 2.56V reference is generated from the internal bandgap reference ( $V_{BG}$ ) through an internal amplifier. In either case, the external AREF pin is directly connected to the ADC, and the reference voltage can be made more immune to noise by connecting a capacitor between the AREF pin and ground.  $V_{REF}$  can also be measured at the AREF pin with a high impedant voltmeter. Note that  $V_{REF}$  is a high impedant source, and only a capacitive load should be connected in a system.

If the user has a fixed voltage source connected to the AREF pin, the user may not use the other reference voltage options in the application, as they will be shorted to the external voltage. If no external voltage is applied to the AREF pin, the user may switch between AVCC and 2.56V as reference selection. The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result.







If differential channels are used, the selected reference should not be closer to AVCC than indicated in Table 114 on page 261 and Table 115 on page 262.

## DC Noise Canceler

The ADC features a noise canceler that enables conversion during sleep mode to reduce noise induced from the CPU core and other I/O peripherals. The noise canceler can be used with ADC Noise Reduction and Idle mode. To make use of this feature, the following procedure should be used:

1. Make sure that the ADC is enabled and is not busy converting. Single Conversion mode must be selected and the ADC conversion complete interrupt must be enabled.
2. Enter ADC Noise Reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.
3. If no other interrupts occur before the ADC conversion completes, the ADC interrupt will wake up the CPU and execute the ADC Conversion Complete interrupt routine. If another interrupt wakes up the CPU before the ADC conversion is complete, that interrupt will be executed, and an ADC Conversion Complete interrupt request will be generated when the ADC conversion completes. The CPU will remain in active mode until a new sleep command is executed.

Note that the ADC will not be automatically turned off when entering other sleep modes than Idle mode and ADC Noise Reduction mode. The user is advised to write zero to ADEN before entering such sleep modes to avoid excessive power consumption. If the ADC is enabled in such sleep modes and the user wants to perform differential conversions, the user is advised to switch the ADC off and on after waking up from sleep to prompt an extended conversion to get a valid result.

## Analog Input Circuitry

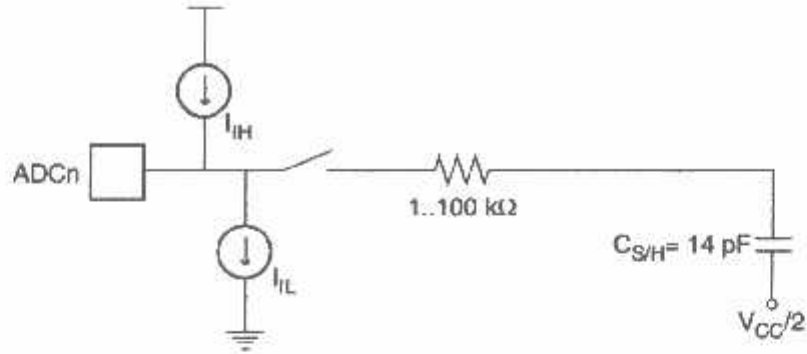
The Analog Input Circuitry for single ended channels is illustrated in Figure 105. An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin, regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10 k $\Omega$  or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, with can vary widely. The user is recommended to only use low impedant sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.

If differential gain channels are used, the input circuitry looks somewhat different, although source impedances of a few hundred k $\Omega$  or less is recommended.

Signal components higher than the Nyquist frequency ( $f_{ADC}/2$ ) should not be present for either kind of channels, to avoid distortion from unpredictable signal convolution. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.

Figure 106. Analog Input Circuitry

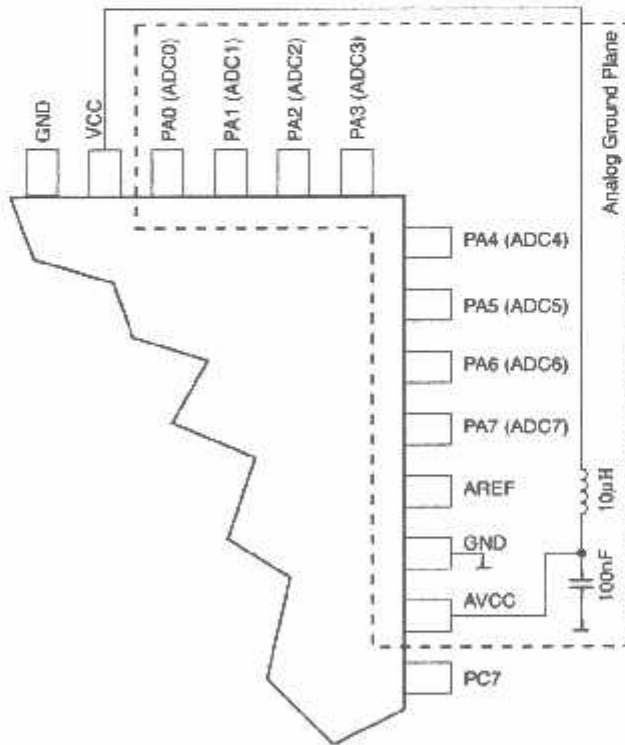


Analog Noise Cancelling Techniques

Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. If conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

1. Keep analog signal paths as short as possible. Make sure analog tracks run over the analog ground plane, and keep them well away from high-speed switching digital tracks.
2. The AVCC pin on the device should be connected to the digital V<sub>CC</sub> supply voltage via an LC network as shown in Figure 106.
3. Use the ADC noise canceler function to reduce induced noise from the CPU.
4. If any ADC port pins are used as digital outputs, it is essential that these do not switch while a conversion is in progress.

Figure 106. ADC Power Connections



## Offset Compensation Schemes

The gain stage has a built-in offset cancellation circuitry that nulls the offset of differential measurements as much as possible. The remaining offset in the analog path can be measured directly by selecting the same channel for both differential inputs. This offset residue can be then subtracted in software from the measurement results. Using this kind of software based offset correction, offset on any channel can be reduced below one LSB.

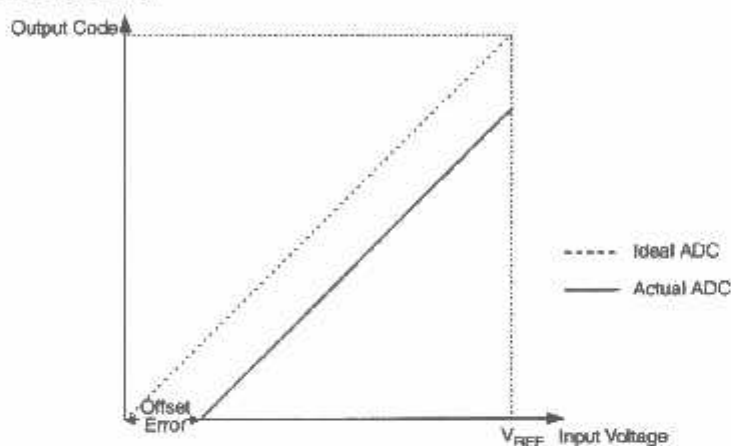
## DC Accuracy Definitions

An  $n$ -bit single-ended ADC converts a voltage linearly between GND and  $V_{REF}$  in  $2^n$  steps (LSBs). The lowest code is read as 0, and the highest code is read as  $2^n - 1$ .

Several parameters describe the deviation from the ideal behavior:

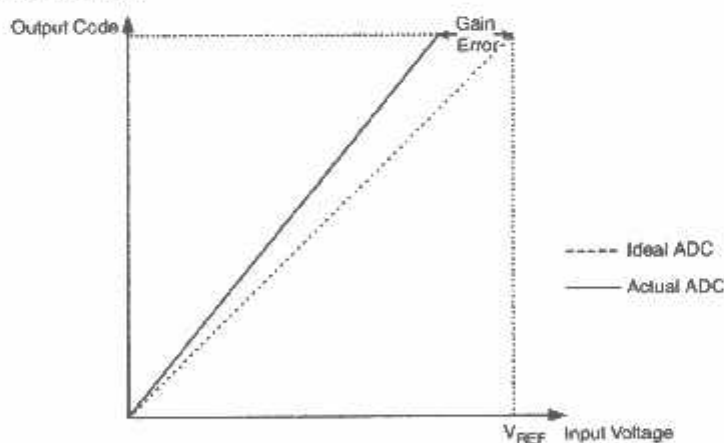
- **Offset:** The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

**Figure 107. Offset Error**



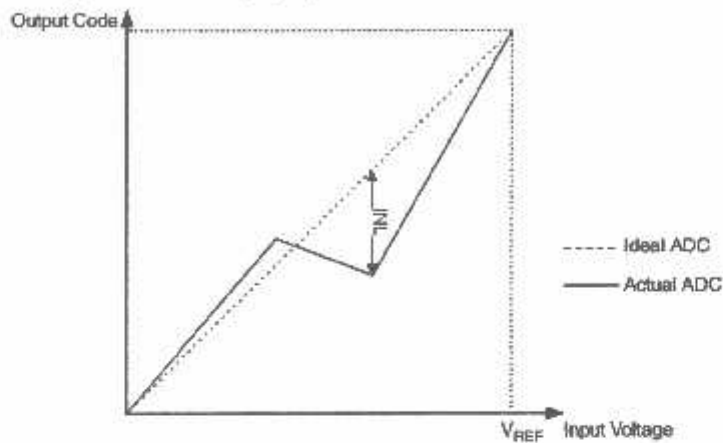
- **Gain Error:** After adjusting for offset, the Gain Error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB.

**Figure 108. Gain Error**



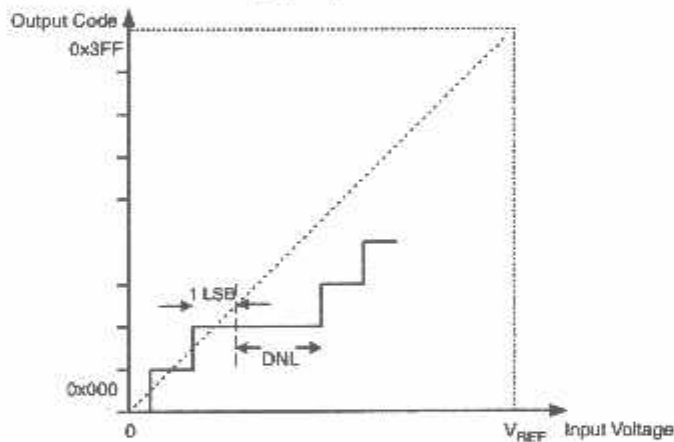
- **Integral Non-linearity (INL):** After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

**Figure 109.** Integral Non-linearity (INL)



- **Differential Non-linearity (DNL):** The maximum deviation of the actual code width (the interval between two adjacent transitions) from the ideal code width (1 LSB). Ideal value: 0 LSB.

**Figure 110.** Differential Non-linearity (DNL)



- **Quantization Error:** Due to the quantization of the input voltage into a finite number of codes, a range of input voltages (1 LSB wide) will code to the same value. Always  $\pm 0.5$  LSB.
- **Absolute Accuracy:** The maximum deviation of an actual (unadjusted) transition compared to an ideal transition for any code. This is the compound effect of Offset, Gain Error, Differential Error, Non-linearity, and Quantization Error. Ideal value:  $\pm 0.5$  LSB.

## DC Conversion Result

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

where  $V_{IN}$  is the voltage on the selected input pin and  $V_{REF}$  the selected voltage reference (see Table 84 on page 217 and Table 85 on page 218). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB.

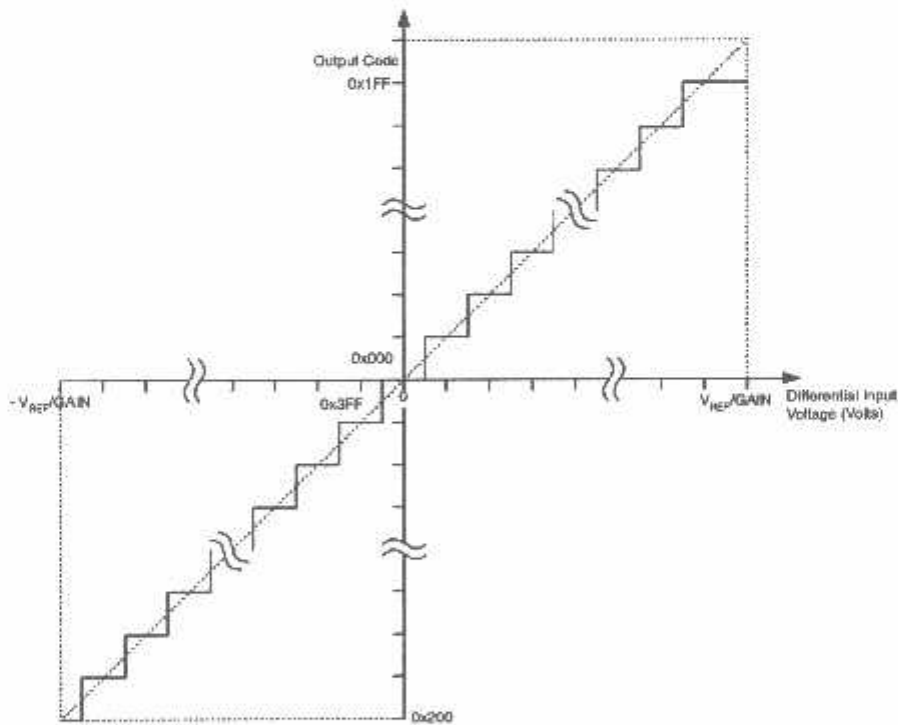
If differential channels are used, the result is

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

where  $V_{POS}$  is the voltage on the positive input pin,  $V_{NEG}$  the voltage on the negative input pin, GAIN the selected gain factor, and  $V_{REF}$  the selected voltage reference. The result is presented in two's complement form, from 0x200 (-512d) through 0x1FF (+511d). Note that if the user wants to perform a quick polarity check of the results, it is sufficient to read the MSB of the result (ADC9 in ADCH). If the bit is one, the result is negative, and if the bit is zero, the result is positive. Figure 111 shows the decoding of the differential input range.

Table 83 shows the resulting output codes if the differential input channel pair (ADCn - ADCm) is selected with a gain of GAIN and a reference voltage of  $V_{REF}$ .

**Figure 111. Differential Measurement Range**



**Table 83.** Correlation Between Input Voltage and Output Codes

$V_{ADCn}$	Read Code	Corresponding Decimal Value
$V_{ADCm} + V_{REF}/GAIN$	0x1FF	511
$V_{ADCm} + (511/512) V_{REF}/GAIN$	0x1FF	511
$V_{ADCm} + (510/512) V_{REF}/GAIN$	0x1FE	510
...	...	...
$V_{ADCm} + (1/512) V_{REF}/GAIN$	0x001	1
$V_{ADCm}$	0x000	0
$V_{ADCm} - (1/512) V_{REF}/GAIN$	0x3FF	-1
...	...	...
$V_{ADCm} - (511/512) V_{REF}/GAIN$	0x201	-511
$V_{ADCm} - V_{REF}/GAIN$	0x200	-512

Example:

ADMUX = 0xED (ADC3 - ADC2, 10x gain, 2.56V reference, left adjusted result)

Voltage on ADC3 is 300 mV, voltage on ADC2 is 500 mV.

$ADCR = 512 * 10 * (300 - 500)/2560 = -400 = 0x270$

ADCL will thus read 0x00, and ADCH will read 0x9C. Writing zero to ADLAR right adjusts the result: ADCL = 0x70, ADCH = 0x02.

**ADC Multiplexer Selection Register – ADMUX**

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**• Bit 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 84. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

**Table 84.** Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

**• Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see "The ADC Data Register – ADCL and ADCH" on page 220.



• **Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the ADC. These bits also select the gain for the differential channels. See Table 85 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

**Table 85.** Input Channel and Gain Selections

MUX4..0	Single Ended Input	Pos Differential Input	Neg Differential Input	Gain	
00000	ADC0	N/A			
00001	ADC1				
00010	ADC2				
00011	ADC3				
00100	ADC4				
00101	ADC5				
00110	ADC6				
00111	ADC7				
01000	N/A	ADC0	ADC0	10x	
01001		ADC1	ADC0	10x	
01010		ADC0	ADC0	200x	
01011		ADC1	ADC0	200x	
01100		ADC2	ADC2	10x	
01101		ADC3	ADC2	10x	
01110		ADC2	ADC2	200x	
01111		ADC3	ADC2	200x	
10000		N/A	ADC0	ADC1	1x
10001			ADC1	ADC1	1x
10010			ADC2	ADC1	1x
10011			ADC3	ADC1	1x
10100			ADC4	ADC1	1x
10101			ADC5	ADC1	1x
10110			ADC6	ADC1	1x
10111			ADC7	ADC1	1x
11000			ADC0	ADC2	1x
11001			ADC1	ADC2	1x
11010			ADC2	ADC2	1x
11011			ADC3	ADC2	1x
11100			ADC4	ADC2	1x

**Table 85.** Input Channel and Gain Selections (Continued)

MUX4.0	Single Ended Input	Pos Differential Input	Neg Differential Input	Gain
11101		ADC5	ADC2	1x
11110	1.22V ( $V_{BG}$ )	N/A		
11111	0V (GND)			

**ADC Control and Status Register A – ADCSRA**

Bit	7	6	5	4	3	2	1	0	ADCSRA
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

• **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

• **Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in SFIOR.

• **Bit 4 – ADIF: ADC Interrupt Flag**

This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

• **Bit 3 – ADIE: ADC Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

• **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**

These bits determine the division factor between the XTAL frequency and the input clock to the ADC.



**Table 86. ADC Prescaler Selections**

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

**ADC Data Register – ADCL and ADCH**

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	ADC9	ADC8									ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0									ADCL
Read/Write	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2									ADCH
	ADC1	ADC0	–	–	–	–	–	–									ADCL
Read/Write	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

When an ADC conversion is complete, the result is found in these two registers. If differential channels are used, the result is presented in two's complement form.

When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH.

The ADLAR bit in ADMUX, and the MUXn bits in ADMUX affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

• **ADC9:0: ADC Conversion Result**

These bits represent the result from the conversion, as detailed in "ADC Conversion Result" on page 216.

## Special Function IO Register – SFOR

Bit	7	6	5	4	3	2	1	0	SFOR
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### • Bit 7:5 – ADTS2:0: ADC Auto Trigger Source

If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS2:0 settings will have no effect. A conversion will be triggered by the rising edge of the selected interrupt flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to Free Running mode (ADTS[2:0]=0) will not cause a trigger event, even if the ADC Interrupt Flag is set.

**Table 87. ADC Auto Trigger Source Selections**

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

### • Bit 4 – RES: Reserved Bit

This bit is reserved bit in the ATmega8535, and will always read as zero.

## Boot Loader Support Read-While-Write Self-Programming

The Boot Loader Support provides a real Read-While-Write Self-Programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the Program memory. The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the Boot Loader memory is configurable with fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

## Boot Loader Features

- Read-While-Write Self-Programming
- Flexible Boot Memory Size
- High Security (Separate Boot Lock Bits for a Flexible Protection)
- Separate Fuse to Select Reset Vector
- Optimized Page<sup>(1)</sup> Size
- Code Efficient Algorithm
- Efficient Read-Modify-Write Support

Note: 1. A page is a section in the Flash consisting of several bytes (see Table 104 on page 239) used during programming. The page organization does not affect normal operation.

## Application and Boot Loader Flash Sections

The Flash memory is organized in two main sections, the Application section and the Boot Loader section (see Figure 113). The size of the different sections is configured by the BOOTSZ Fuses as shown in Table 93 on page 233 and Figure 113. These two sections can have different level of protection since they have different sets of Lock bits.

## Application Section

The Application section is the section of the Flash that is used for storing the application code. The protection level for the Application section can be selected by the Application Boot Lock bits (Boot Lock bits 0), see Table 89 on page 225. The Application section can never store any Boot Loader code since the SPM instruction is disabled when executed from the Application section.

## BLS – Boot Loader Section

While the Application section is used for storing the application code, the The Boot Loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire Flash, including the BLS itself. The protection level for the Boot Loader section can be selected by the Boot Loader Lock bits (Boot Lock bits 1), see Table 90 on page 225.

## Read-While-Write and No Read-While-Write Flash Sections

Whether the CPU supports Read-While-Write or if the CPU is halted during a Boot Loader software update is dependent on which address that is being programmed. In addition to the two sections that are configurable by the BOOTSZ Fuses as described above, the Flash is also divided into two fixed sections, the Read-While-Write (RWW) section and the No Read-While-Write (NRWW) section. The limit between the RWW and NRWW sections is given in Table 94 on page 233 and Figure 113 on page 224. The main difference between the two sections is:

- When erasing or writing a page located inside the RWW section, the NRWW section can be read during the operation.
- When erasing or writing a page located inside the NRWW section, the CPU is halted during the entire operation.

Note that the user software can never read any code that is located inside the RWW section during a Boot Loader software operation. The syntax "Read-While-Write Section" refers to which section that is being programmed (erased or written), not which section that actually is being read during a Boot Loader software update.

## RWW – Read-While-Write Section

If a Boot Loader software update is programming a page inside the RWW section, it is possible to read code from the Flash, but only code that is located in the NRWW section. During an on-going programming, the software must ensure that the RWW section never is being read. If the user software is trying to read code that is located inside the RWW section (i.e., by a rcall/rjmp/lpm or an interrupt) during programming, the software might end up in an unknown state. To avoid this, the interrupts should either be disabled or moved to the Boot Loader section. The Boot Loader section is always located in the NRWW section. The RWW Section Busy bit (RWWSB) in the Store Program Memory Control Register (SPMCR) will be read as logical one as long as the RWW section is blocked for reading. After a programming is completed, the RWWSB must be cleared by software before reading code located in the RWW section. See "Store Program Memory Control Register – SPMCR" on page 226, for details on how to clear RWWSB.

## NRWW – No Read-While-Write Section

The code located in the NRWW section can be read when the Boot Loader software is updating a page in the RWW section. When the Boot Loader code updates the NRWW section, the CPU is halted during the entire page erase or page write operation.

**Table 88.** Read-While-Write Features

Which Section does the Z-pointer Address During the Programming?	Which Section can be Read During Programming?	Is the CPU Halted?	Read-While-Write Supported?
RWW section	NRWW section	No	Yes
NRWW section	None	Yes	No

**Figure 112.** Read-While-Write vs. No Read-While-Write

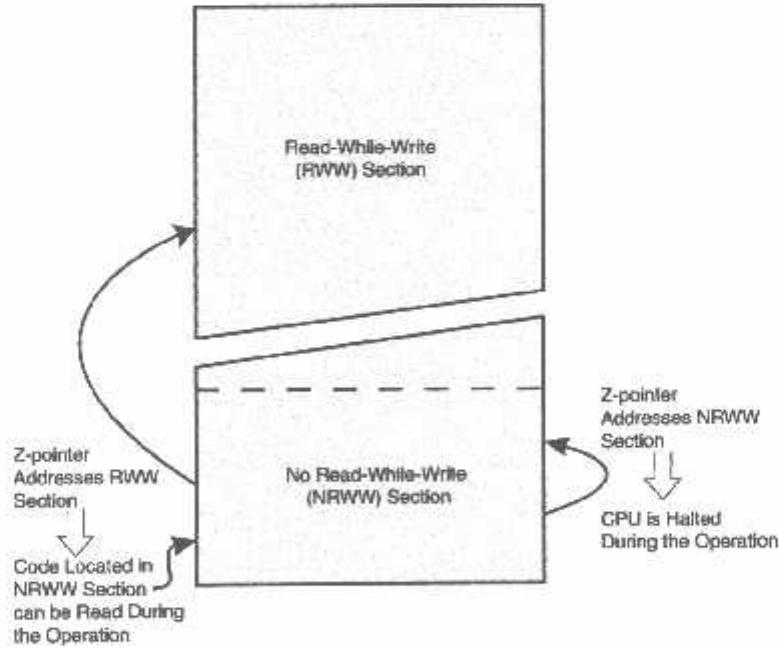
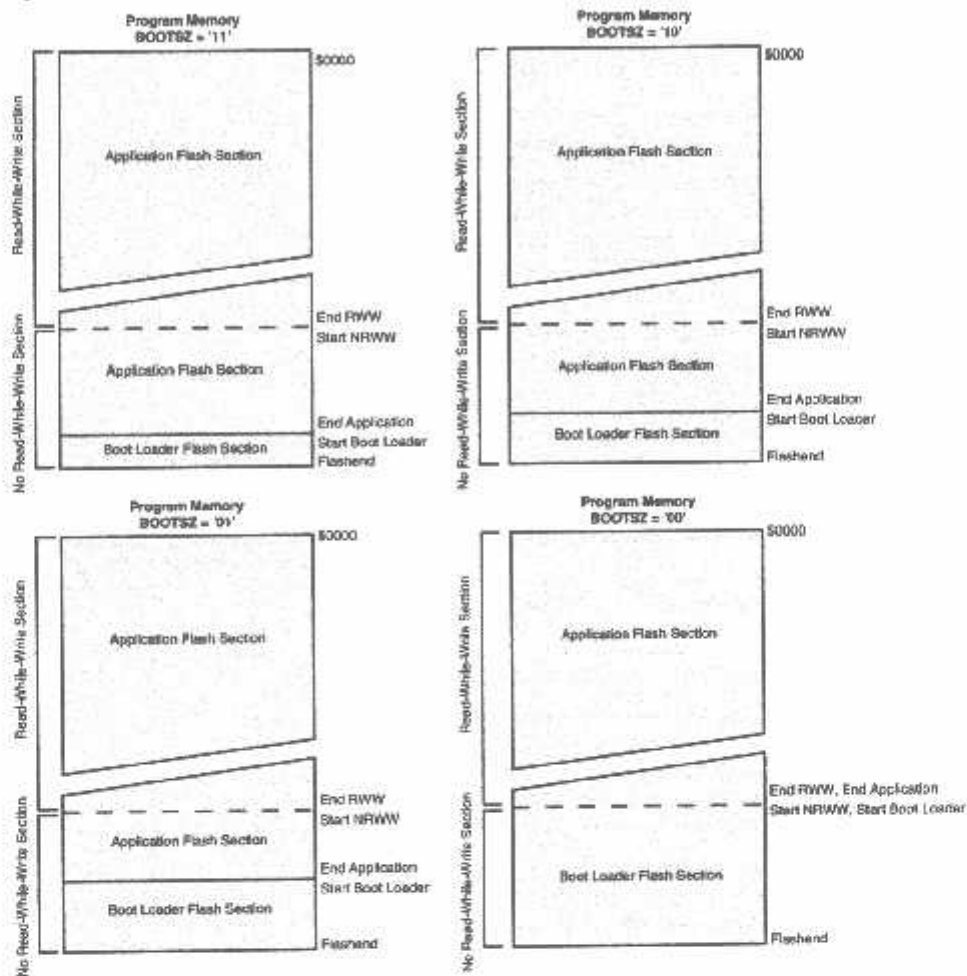


Figure 113. Memory sections<sup>(1)</sup>



Note: 1. The parameters in the figure above are given in Table 93 on page 233.

## Boot Loader Lock Bits

If no Boot Loader capability is needed, the entire Flash is available for application code. The Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

The user can select:

- To protect the entire Flash from a software update by the MCU.
- To protect only the Boot Loader Flash section from a software update by the MCU.
- To protect only the Application Flash section from a software update by the MCU.
- Allow software update in the entire Flash.

See Table 89 and Table 90 for further details. The Boot Lock bits can be set in software and in Serial or Parallel Programming mode, but they can be cleared by a Chip Erase command only. The general Write Lock (Lock Bit mode 2) does not control the programming of the Flash memory by SPM instruction. Similarly, the general Read/Write Lock (Lock Bit mode 1) does not control reading nor writing by LPM/SPM, if it is attempted.

The Store Program Memory Control Register contains the control bits needed to control the Boot Loader operations.

Bit	7	6	5	4	3	2	1	0	
	SPMIE	RWWSB	-	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCR
Read/Write	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 7 – SPMIE: SPM Interrupt Enable**

When the SPMIE bit is written to one, and the I-bit in the Status Register is set (one), the SPM ready interrupt will be enabled. The SPM ready Interrupt will be executed as long as the SPMEN bit in the SPMCR Register is cleared.

• **Bit 6 – RWWSB: Read-While-Write Section Busy**

When a Self-Programming (Page Erase or Page Write) operation to the RWW section is initiated, the RWWSB will be set (one) by hardware. When the RWWSB bit is set, the RWW section cannot be accessed. The RWWSB bit will be cleared if the RWWSRE bit is written to one after a Self-Programming operation is completed. Alternatively the RWWSB bit will automatically be cleared if a page load operation is initiated.

• **Bit 5 – Res: Reserved Bit**

This bit is a reserved bit in the ATmega8535 and always read as zero.

• **Bit 4 – RWWSRE: Read-While-Write Section Read Enable**

When programming (page erase or page write) to the RWW section, the RWW section is blocked for reading (the RWWSB will be set by hardware). To re-enable the RWW section, the user software must wait until the programming is completed (SPMEN will be cleared). Then, if the RWWSRE bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles re-enables the RWW section. The RWW section cannot be re-enabled while the Flash is busy with a page erase or a page write (SPMEN is set). If the RWWSRE bit is written while the Flash is being loaded, the Flash load operation will abort and the data loaded will be lost.

• **Bit 3 – BLBSET: Boot Lock Bit Set**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles sets Boot Lock bits, according to the data in R0. The data in R1 and the address in the Z-pointer are ignored. The BLBSET bit will automatically be cleared upon completion of the Lock bit set, or if no SPM instruction is executed within four clock cycles.

An LPM instruction within three cycles after BLBSET and SPMEN are set in the SPMCR Register, will read either the Lock bits or the Fuse bits (depending on Z0 in the Z-pointer) into the destination register. See "Reading the Fuse and Lock Bits from Software" on page 230 for details.

• **Bit 2 – PGWRT: Page Write**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles executes page write, with the data stored in the temporary buffer. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGWRT bit will auto-clear upon completion of a page write, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire page write operation if the NRWW section is addressed.

• **Bit 1 – PGERS: Page Erase**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles executes page erase. The page address is taken from the high part of

the Z-pointer. The data in R1 and R0 are ignored. The PGERS bit will auto-clear upon completion of a page erase, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire page write operation if the NRWW section is addressed.

• **Bit 0 – SPMEN: Store Program Memory Enable**

This bit enables the SPM instruction for the next four clock cycles. If written to one together with either RWWSRE, BLBSET, PGWRT or PGERS, the following SPM instruction will have a special meaning, see description above. If only SPMEN is written, the following SPM instruction will store the value in R1:R0 in the temporary page buffer addressed by the Z-pointer. The LSB of the Z-pointer is ignored. The SPMEN bit will auto-clear upon completion of an SPM instruction, or if no SPM instruction is executed within four clock cycles. During page erase and page write, the SPMEN bit remains high until the operation is completed.

Writing any other combination than "10001", "01001", "00101", "00011" or "00001" in the lower five bits will have no effect.

## Addressing the Flash During Self-programming

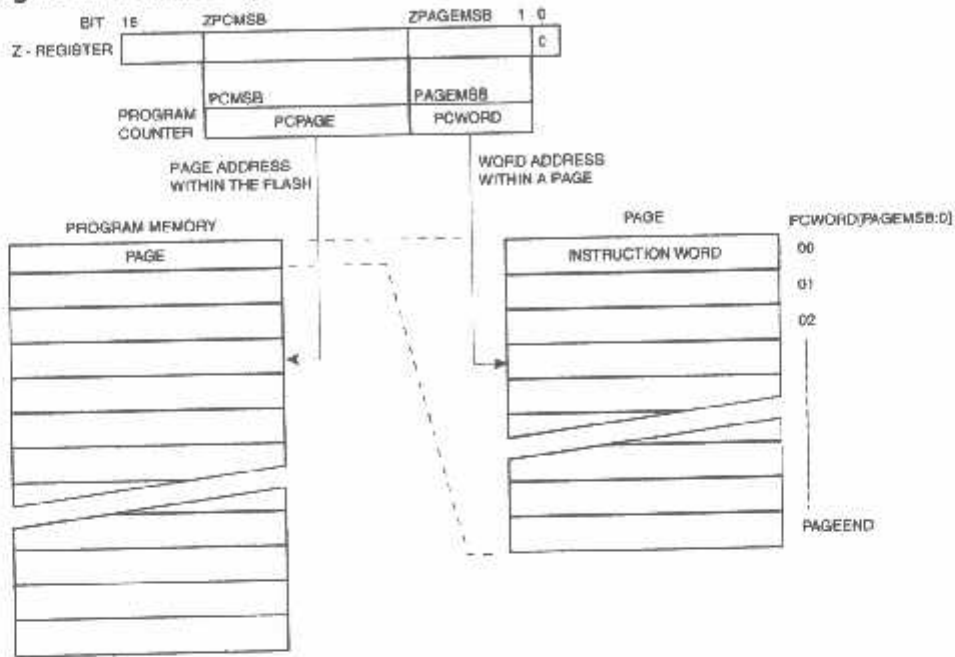
The Z-pointer is used to address the SPM commands.

Bit	15	14	13	12	11	10	9	8
ZH (R31)	Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8
ZL (R30)	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0
	7	6	5	4	3	2	1	0

Since the Flash is organized in pages (see Table 104 on page 239), the Program Counter can be treated as having two different sections. One section, consisting of the least significant bits, is addressing the words within a page, while the most significant bits are addressing the pages. This is shown in Figure 114. Note that the Page Erase and Page Write operations are addressed independently. Therefore it is of major importance that the Boot Loader software addresses the same page in both the Page Erase and Page Write operation. Once a programming operation is initiated, the address is latched and the Z-pointer can be used for other operations.

The only SPM operation that does not use the Z-pointer is Setting the Boot Loader Lock bits. The content of the Z-pointer is ignored and will have no effect on the operation. The LPM instruction does also use the Z-pointer to store the address. Since this instruction addresses the Flash byte by byte, also the LSB (bit Z0) of the Z-pointer is used.

**Figure 114. Addressing the Flash during SPM<sup>(1)</sup>**



- Notes:
1. The different variables used+ in Figure 114 are listed in Table 95 on page 234.
  2. PCPAGE and PCWORD are listed in Table 104 on page 239.

## Self-Programming the Flash

The program memory is updated in a page by page fashion. Before programming a page with the data stored in the temporary page buffer, the page must be erased. The temporary page buffer is filled one word at a time using SPM and the buffer can be filled either before the Page Erase command or between a Page Erase and a Page Write operation:

Alternative 1, fill the buffer before a Page Erase:

- Fill temporary page buffer
- Perform a Page Erase
- Perform a Page Write

Alternative 2, fill the buffer after Page Erase:

- Perform a Page Erase
- Fill temporary page buffer
- Perform a Page Write

If only a part of the page needs to be changed, the rest of the page must be stored (for example in the temporary page buffer) before the erase, and then be rewritten. When using alternative 1, the Boot Loader provides an effective Read-Modify-Write feature which allows the user software to first read the page, do the necessary changes, and then write back the modified data. If alternative 2 is used, it is not possible to read the old data while loading since the page is already erased. The temporary page buffer can be accessed in a random sequence. It is essential that the page address used in both the page erase and page write operation is addressing the same page. See "Simple Assembly Code Example for a Boot Loader" on page 231 for an assembly code example.





### Setting the Boot Loader Lock bits by SPM

To set the Boot Loader Lock bits, write the desired data to R0, write "X0001001" to SPMCR and execute SPM within four clock cycles after writing SPMCR. The only accessible Lock bits are the Boot Lock bits that may prevent the Application and Boot Loader section from any software update by the MCU.

Bit	7	6	5	4	3	2	1	0
R0	1	1	BLB12	BLB11	BLB02	BLB01	1	1

See Table 89 and Table 90 for how the different settings of the Boot Loader bits affect the Flash access.

If bits 5..2 in R0 are cleared (zero), the corresponding Boot Lock bit will be programmed if an SPM instruction is executed within four cycles after BLBSET and SPEN are set in SPMCR. The Z-pointer is don't care during this operation, but for future compatibility it is recommended to load the Z-pointer with 0x0001 (same as used for reading the Lock bits). For future compatibility it is also recommended to set bits 7, 6, 1, and 0 in R0 to "1" when writing the Lock bits. When programming the Lock bits the entire Flash can be read during the operation.

### PROM Write Prevents Writing to SPMCR

Note that an EEPROM write operation will block all software programming to Flash. Reading the Fuses and Lock bits from software will also be prevented during the EEPROM write operation. It is recommended that the user checks the status bit (EWE) in the EECR Register and verifies that the bit is cleared before writing to the SPMCR Register.

### Reading the Fuse and Lock bits from Software

It is possible to read both the Fuse and Lock bits from software. To read the Lock bits, load the Z-pointer with 0x0001 and set the BLBSET and SPEN bits in SPMCR. When an LPM instruction is executed within three CPU cycles after the BLBSET and SPEN bits are set in SPMCR, the value of the Lock bits will be loaded in the destination register. The BLBSET and SPEN bits will auto-clear upon completion of reading the Lock bits or if no LPM instruction is executed within three CPU cycles or no SPM instruction is executed within four CPU cycles. When BLBSET and SPEN are cleared, LPM will work as described in the Instruction set Manual.

Bit	7	6	5	4	3	2	1	0
Rd	-	-	BLB12	BLB11	BLB02	BLB01	LB2	LB1

The algorithm for reading the Fuse Low bits is similar to the one described above for reading the Lock bits. To read the Fuse Low bits, load the Z-pointer with 0x0000 and set the BLBSET and SPEN bits in SPMCR. When an LPM instruction is executed within three cycles after the BLBSET and SPEN bits are set in the SPMCR, the value of the Fuse Low bits (FLB) will be loaded in the destination register as shown below. Refer to Table 99 on page 237 for a detailed description and mapping of the Fuse Low bits.

Bit	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

Similarly, when reading the Fuse High bits, load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SPEN bits are set in the SPMCR, the value of the Fuse High bits (FHB) will be loaded in the destination register as shown below. Refer to Table 98 on page 236 for detailed description and mapping of the Fuse High bits.

Bit	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

Fuse and Lock bits that are programmed, will be read as zero. Fuse and Lock bits that are unprogrammed, will be read as one.

## Preventing Flash Corruption

During periods of low  $V_{CC}$ , the Flash program can be corrupted because the supply voltage is too low for the CPU and the Flash to operate properly. These issues are the same as for board level systems using the Flash, and the same design solutions should be applied.

A Flash program corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the Flash requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage for executing instructions is too low.

Flash corruption can easily be avoided by following these design recommendations (one is sufficient):

1. If there is no need for a Boot Loader update in the system, program the Boot Loader Lock bits to prevent any Boot Loader software updates.
2. Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD) if the operating voltage matches the detection level. If not, an external low  $V_{CC}$  Reset Protection circuit can be used. If a Reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.
3. Keep the AVR core in Power-down Sleep mode during periods of low  $V_{CC}$ . This will prevent the CPU from attempting to decode and execute instructions, effectively protecting the SPMCR Register and thus the Flash from unintentional writes.

## Programming Time for Flash when Using SPM

The calibrated RC Oscillator is used to time Flash accesses. Table 92 shows the typical programming time for Flash accesses from the CPU.

**Table 92. SPM Programming Time**

Symbol	Min Programming Time	Max Programming Time
Flash write (Page Erase, Page Write, and write Lock bits by SPM)	3.7 ms	4.5 ms

## Sample Assembly Code Example for a Boot Loader

```

; -the routine writes one page of data from RAM to Flash
; the first data location in RAM is pointed to by the Y pointer
; the first data location in Flash is pointed to by the Z pointer
; -error handling is not included
; -the routine must be placed inside the boot space
; (at least the Do_spm sub routine). Only code inside NRWW section can
; be read during Self-Programming (page erase and page write).
; -registers used: r0, r1, temp1 (r16), temp2 (r17), looplo (r24),
; loophi (r25), spmcrval (r20)
; storing and restoring of registers is not included in the routine
; register usage can be optimized at the expense of code size
; -It is assumed that either the interrupt table is moved to the Boot
; loader section or that the interrupts are disabled.
.equ PAGESIZEB = PAGESIZE*2 ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
Write_page:
; page erase
ldi spmcrval, (1<<PGERS) | (1<<SPMEN)
rcall Do_spm
    
```



```

; re-enable the RWW section
ldi  spmcrval, (1<<RWWSRE) | (1<<SPMEN)
rcall Do_spm

; transfer data from RAM to Flash page buffer
ldi  looplo, low(PAGESIZEB)      ;init loop variable
ldi  loophi, high(PAGESIZEB)    ;not required for PAGESIZEB<=256
Wrlong:
ld   r0, Y+
ld   r1, Y+
ldi  spmcrval, (1<<SPMEN)
rcall Do_spm
adiw ZH:ZL, 2
sbiw loophi:looplo, 2          ;use subi for PAGESIZEB<=256
brne Wrlong

; execute page write
subi ZL, low(PAGESIZEB)        ;restore pointer
sbc  ZH, high(PAGESIZEB)      ;not required for PAGESIZEB<=256
ldi  spmcrval, (1<<PGWRT) | (1<<SPMEN)
rcall Do_spm

; re-enable the RWW section
ldi  spmcrval, (1<<RWWSRE) | (1<<SPMEN)
rcall Do_spm

; read back and check, optional
ldi  looplo, low(PAGESIZEB)    ;init loop variable
ldi  loophi, high(PAGESIZEB)   ;not required for PAGESIZEB<=256
subi YL, low(PAGESIZEB)        ;restore pointer
sbc  YH, high(PAGESIZEB)
Rdloop:
lpm  r0, Z+
ld   r1, Y+
cpse r0, r1
rjmp Error
sbiw loophi:looplo, 1          ;use subi for PAGESIZEB<=256
brne Rdloop

; return to RWW section
; verify that RWW section is safe to read
Return:
in   temp1, SPMCR
sbrs temp1, RWWSB              ; If RWWSB is set, the RWW section is not ready yet
ret
; re-enable the RWW section
ldi  spmcrval, (1<<RWWSRE) | (1<<SPMEN)
rcall Do_spm
rjmp Return

Do_spm:
; check for previous SPM complete
Wait_spm:
in   temp1, SPMCR
sbrs temp1, SPEN
rjmp Wait_spm
; input: spmcrval determines SPM action
; disable interrupts if enabled, store status

```

```

in    temp2, SREG
cli
; check that no EEPROM write access is present
Wait_ee:
sbic  EECR, EEWE
rjmp  Wait_ee
; SPM timed sequence
out   SPMCR, spmcrval
spm
; restore SREG (to enable interrupts if originally enabled)
out   SREG, temp2
ret

```

## ATmega8535 Boot Loader Parameters

In Table 93 through Table 95, the parameters used in the description of the self programming are given.

**Table 93.** Boot Size Configuration<sup>(1)</sup>

BOOTS Z1	BOOTS Z0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application Section	Boot Reset Address (Start Boot Loader Section)
1	1	128 words	4	0x000 - 0xF7F	0xF80 - 0xFFF	0xF7F	0xF80
1	0	256 words	8	0x000 - 0xEFF	0xF00 - 0xFFF	0xEFF	0xF00
0	1	512 words	16	0x000 - 0xDFF	0xE00 - 0xFFF	0xDFF	0xE00
0	0	1024 words	32	0x000 - 0xBFF	0xC00 - 0xFFF	0xBFF	0xC00

Note: 1. The different BOOTSZ Fuse configurations are shown in Figure 113

**Table 94.** Read-While-Write Limit<sup>(1)</sup>

Section	Pages	Address
Read-While-Write section (RWW)	96	0x000 - 0xBFF
No Read-While-Write section (NRWW)	32	0xC00 - 0xFFF

Note: 1. For details about these two section, see "NRWW – No Read-While-Write Section" on page 223 and "RWW – Read-While-Write Section" on page 223



**Table 95.** Explanation of Different Variables used in Figure 114 and the Mapping to the Z-pointer<sup>(1)</sup>

Variable		Corresponding Z-value	Description
PCMSB	11		Most significant bit in the Program Counter. (The Program Counter is 12 bits PC[11:0])
PAGEMSB	4		Most significant bit which is used to address the words within one page (64 words in a page requires five bits PC [4:0]).
ZPCMSB		Z12	Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1.
ZPAGEMSB		Z5	Bit in Z-register that is mapped to PAGEMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1.
PCPAGE	PC[11:5]	Z12:Z6	Program Counter page address: Page select, for Page Erase and Page Write
PCWORD	PC[4:0]	Z5:Z1	Program Counter word address: Word select, for filling temporary buffer (must be zero during page write operation)

Note: 1. Z15:Z13: always ignored  
 Z0: should be zero for all SPM commands, byte select for the LPM instruction.  
 See "Addressing the Flash during Self-Programming" on page 227 for details about the use of Z-pointer during Self-Programming.

## Memory Programming

### Program And Data Memory Lock Bits

The ATmega8535 provides six Lock bits which can be left unprogrammed ("1") or can be programmed ("0") to obtain the additional features listed in Table 97. The Lock bits can only be erased to "1" with the Chip Erase command.

**Table 96. Lock Bit Byte<sup>(1)</sup>**

Lock Bit Byte	Bit No	Description	Default Value
	7	-	1 (unprogrammed)
	6	-	1 (unprogrammed)
BLB12	5	Boot Lock bit	1 (unprogrammed)
BLB11	4	Boot Lock bit	1 (unprogrammed)
BLB02	3	Boot Lock bit	1 (unprogrammed)
BLB01	2	Boot Lock bit	1 (unprogrammed)
LB2	1	Lock bit	1 (unprogrammed)
LB1	0	Lock bit	1 (unprogrammed)

Note: 1. "1" means unprogrammed, "0" means programmed

**Table 97. Lock Bit Protection Modes<sup>(2)</sup>**

Memory Lock Bits			Protection Type
LB Mode	LB2	LB1	
1	1	1	No memory lock features enabled.
2	1	0	Further programming of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode. <sup>(1)</sup>
3	0	0	Further programming and verification of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode. <sup>(1)</sup>
BLB0 Mode	BLB02	BLB01	
1	1	1	No restrictions for SPM or LPM accessing the Application section.
2	1	0	SPM is not allowed to write to the Application section.
3	0	0	SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If interrupt vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
4	0	1	LPM executing from the Boot Loader section is not allowed to read from the Application section. If interrupt vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
BLB1 Mode	BLB12	BLB11	



**Table 97. Lock Bit Protection Modes<sup>(2)</sup> (Continued)**

Memory Lock Bits			Protection Type
1	1	1	No restrictions for SPM or LPM accessing the Boot Loader section.
2	1	0	SPM is not allowed to write to the Boot Loader section.
3	0	0	SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If interrupt vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.
4	0	1	LPM executing from the Application section is not allowed to read from the Boot Loader section. If interrupt vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.

- Notes: 1. Program the Fuse bits before programming the Lock bits.  
 2. "1" means unprogrammed, "0" means programmed.

**Fuse Bits**

The ATmega8535 has two Fuse bytes. Table 98 and Table 99 describe briefly the functionality of all the fuses and how they are mapped into the fuse bytes. Note that the fuses are read as logical zero, "0", if they are programmed.

**Table 98. Fuse High Byte**

Fuse High Byte	Bit No	Description	Default Value
S8535C	7	Select AT90S8535 compatibility mode	1 (unprogrammed)
WDTON	6	WDT always on	1 (unprogrammed, WDT enabled by WDTCR)
SPIEN <sup>(1)</sup>	5	Enable Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
CKOPT <sup>(2)</sup>	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 93 for details)	0 (programmed) <sup>(3)</sup>
BOOTSZ0	1	Select Boot Size (see Table 93 for details)	0 (programmed) <sup>(3)</sup>
BOTRST	0	Select Reset Vector	1 (unprogrammed)

- Notes: 1. The SPIEN Fuse is not accessible in Serial Programming mode.  
 2. The CKOPT Fuse functionality depends on the setting of the CKSEL bits. See "Clock Sources" on page 24, for details.  
 3. The default value of BOOTSZ1..0 results in maximum Boot Size. See Table 93 on page 233.

**Table 99. Fuse Low Byte**

Fuse Low Byte	Bit no	Description	Default Value
BODLEVEL	7	Brown out detector trigger level	1 (unprogrammed)
BODEN	6	Brown out detector enable	1 (unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed) <sup>(1)</sup>
SUT0	4	Select start-up time	0 (programmed) <sup>(1)</sup>
CKSEL3	3	Select Clock source	0 (programmed) <sup>(2)</sup>
CKSEL2	2	Select Clock source	0 (programmed) <sup>(2)</sup>
CKSEL1	1	Select Clock source	0 (programmed) <sup>(2)</sup>
CKSEL0	0	Select Clock source	1 (unprogrammed) <sup>(2)</sup>

Notes: 1. The default value of SUT1..0 results in maximum start-up time. See Table 10 on page 29 for details.  
 2. The default setting of CKSEL3..0 results in internal RC Oscillator @ 1 MHz. See Table 2 on page 24 for details.

The status of the Fuse bits is not affected by Chip Erase. Note that the Fuse bits are locked if Lock bit 1 (LB1) is programmed. Program the Fuse bits before programming the Lock bits.

The Fuse values are latched when the device enters Programming mode and changes of the Fuse values will have no effect until the part leaves Programming mode. This does not apply to the EESAVE fuse which will take effect once it is programmed. The fuses are also latched on Power-up in Normal mode.

All Atmel microcontrollers have a three-byte signature code which identifies the device. This code can be read in both Serial and Parallel mode, also when the device is locked. The three bytes reside in a separate address space.

For the ATmega8535 the signature bytes are:

- 0x000: 0x1E (indicates manufactured by Atmel)
- 0x001: 0x93 (indicates 8 KB Flash memory)
- 0x002: 0x08 (indicates ATmega8535 device when 0x001 is 0x93)

The ATmega8535 stores four different calibration values for the internal RC Oscillator. These bytes reside in the signature row high byte of the addresses 0x000, 0x0001, 0x0002, and 0x0003 for 1, 2, 4, and 8 MHz respectively. During Reset, the 1 MHz value is automatically loaded into the OSCCAL Register. If other frequencies are used, the calibration value has to be loaded manually, see "Oscillator Calibration Register – OSCCAL" on page 29 for details.

**Latching of Fuses**

**Signature Bytes**

**Calibration Byte**



## Parallel Programming Parameters, Pin Mapping, and Commands

### Signal Names

This section describes how to parallel program and verify Flash Program memory, EEPROM Data memory, Memory Lock bits, and Fuse bits in the ATmega8535. Pulses are assumed to be at least 250 ns unless otherwise noted.

In this section, some pins of the ATmega8535 are referenced by signal names describing their functionality during parallel programming, see Figure 115 and Table 100. Pins not described in the following table are referenced by pin names.

The XA1/XA0 pins determine the action executed when the XTAL1 pin is given a positive pulse. The bit coding is shown in Table 102.

When pulsing  $\overline{WR}$  or  $\overline{OE}$ , the command loaded determines the action executed. The different Commands are shown in Table 103.

Figure 115. Parallel Programming

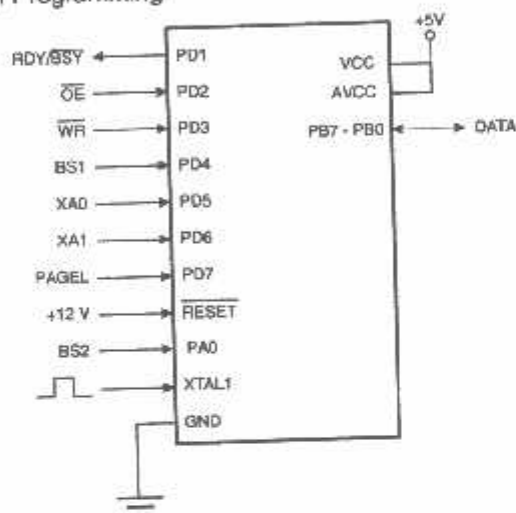


Table 100. Pin Name Mapping

Signal Name in Programming Mode	Pin Name	I/O	Function
RDY/BSY	PD1	O	0: Device is busy programming, 1: Device is ready for new command
$\overline{OE}$	PD2	I	Output Enable (Active low)
$\overline{WR}$	PD3	I	Write Pulse (Active low)
BS1	PD4	I	Byte Select 1 ("0" selects low byte, "1" selects high byte)
XA0	PD5	I	XTAL Action Bit 0
XA1	PD6	I	XTAL Action Bit 1
PAGEL	PD7	I	Program Memory and EEPROM data Page Load
BS2	PA0	I	Byte Select 2 ("0" selects low byte, "1" selects 2'nd high byte)
DATA	PB7 - 0	I/O	Bi-directional Data bus (Output when $\overline{OE}$ is low)

## Parallel Programming

### Enter Programming Mode

The following algorithm puts the device in Parallel Programming mode:

1. Apply 4.5 - 5.5V between  $V_{CC}$  and GND, and wait at least 100  $\mu$ s.
2. Set  $\overline{RESET}$  to "0" and toggle XTAL1 at least six times.
3. Set the Prog\_enable pins listed in Table 101 on page 239 to "0000" and wait at least 100 ns.
4. Apply 11.5 - 12.5V to  $\overline{RESET}$ . Any activity on Prog\_enable pins within 100 ns after +12V has been applied to  $\overline{RESET}$ , will cause the device to fail entering Programming mode.

Note, if External Crystal or External RC configuration is selected, it may not be possible to apply qualified XTAL1 pulses. In such cases, the following algorithm should be followed:

1. Set Prog\_enable pins listed in Table 101 on page 239 to "0000".
2. Apply 4.5 - 5.5V between  $V_{CC}$  and GND simultaneously as 11.5 - 12.5V is applied to  $\overline{RESET}$ .
3. Wait 100 ns.
4. Re-program the fuses to ensure that External Clock is selected as clock source (CKSEL3:0 = 0b0000) If Lock bits are programmed, a Chip Erase command must be executed before changing the fuses.
5. Exit Programming mode by power the device down or by bringing  $\overline{RESET}$  pin to 0b0.
6. Entering Programming mode with the original algorithm, as described above.

### Considerations for Efficient Programming

The loaded command and address are retained in the device during programming. For efficient programming, the following should be considered.

- The command needs only be loaded once when writing or reading multiple memory locations.
- Skip writing the data value 0xFF, that is the contents of the entire EEPROM (unless the EESAVE fuse is programmed) and Flash after a Chip Erase.
- Address high byte needs only be loaded before programming or reading a new 256 word window in Flash or 256 byte EEPROM. This consideration also applies to Signature bytes reading.

### Chip Erase

The Chip Erase will erase the Flash and EEPROM<sup>(1)</sup> memories plus Lock bits. The Lock bits are not reset until the program memory has been completely erased. The Fuse bits are not changed. A Chip Erase must be performed before the Flash and/or the EEPROM is reprogrammed.

Note: 1. The EEPROM memory is preserved during Chip Erase if the EESAVE Fuse is programmed.

Load Command "Chip Erase"

1. Set XA1, XA0 to "10". This enables command loading.
2. Set BS1 to "0".
3. Set DATA to "1000 0000". This is the command for Chip Erase.
4. Give XTAL1 a positive pulse. This loads the command.
5. Give  $\overline{WR}$  a negative pulse. This starts the Chip Erase.  $RDY/\overline{BSY}$  goes low.
6. Wait until  $RDY/\overline{BSY}$  goes high before loading a new command.

The Flash is organized in pages, see Table 104 on page 239. When programming the Flash, the program data is latched into a page buffer. This allows one page of program data to be programmed simultaneously. The following procedure describes how to program the entire Flash memory:

### A. Load Command "Write Flash"

1. Set XA1, XA0 to "10". This enables command loading.
2. Set BS1 to "0".
3. Set DATA to "0001 0000". This is the command for Write Flash.
4. Give XTAL1 a positive pulse. This loads the command.

### B. Load Address Low byte

1. Set XA1, XA0 to "00". This enables address loading.
2. Set BS1 to "0". This selects low address.
3. Set DATA = Address low byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address low byte.

### C. Load Data Low Byte

1. Set XA1, XA0 to "01". This enables data loading.
2. Set DATA = Data low byte (0x00 - 0xFF).
3. Give XTAL1 a positive pulse. This loads the data byte.

### D. Load Data High Byte

1. Set BS1 to "1". This selects high data byte.
2. Set XA1, XA0 to "01". This enables data loading.
3. Set DATA = Data high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the data byte.

### E. Latch Data

1. Set BS1 to "1". This selects high data byte.
2. Give PAGES a positive pulse. This latches the data bytes. (See Figure 117 for signal waveforms.)

F. Repeat B through E until the entire buffer is filled or until all data within the page is loaded.

While the lower bits in the address are mapped to words within the page, the higher bits address the pages within the FLASH. This is illustrated in Figure 116 on page 242. Note that if less than eight bits are required to address words in the page (pagesize < 256), the most significant bit(s) in the address low byte are used to address the page when performing a page write.

### G. Load Address High byte

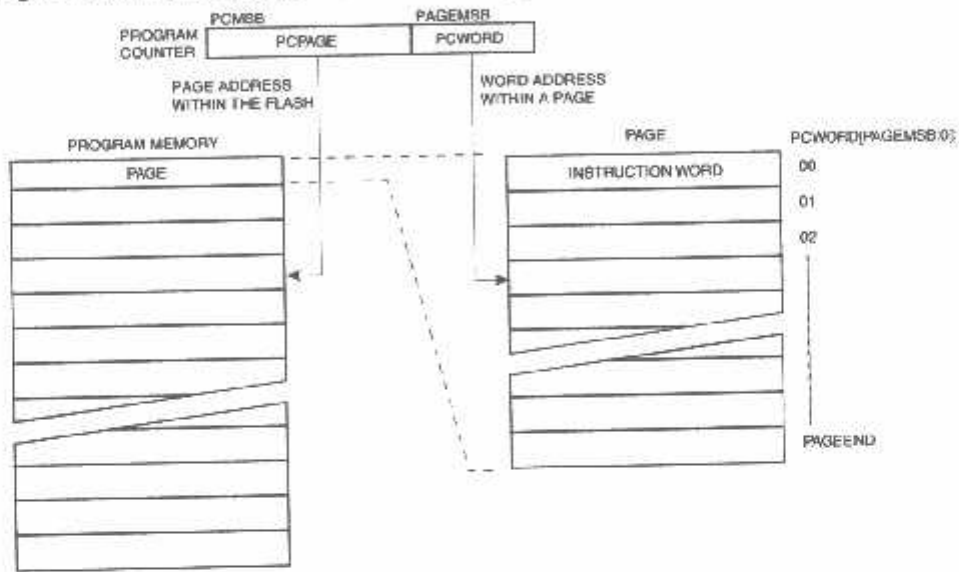
1. Set XA1, XA0 to "00". This enables address loading.
2. Set BS1 to "1". This selects high address.
3. Set DATA = Address high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address high byte.

### H. Program Page

1. Set BS1 to "0".
2. Give  $\overline{WR}$  a negative pulse. This starts programming of the entire page of data. RDY/BSY goes low.

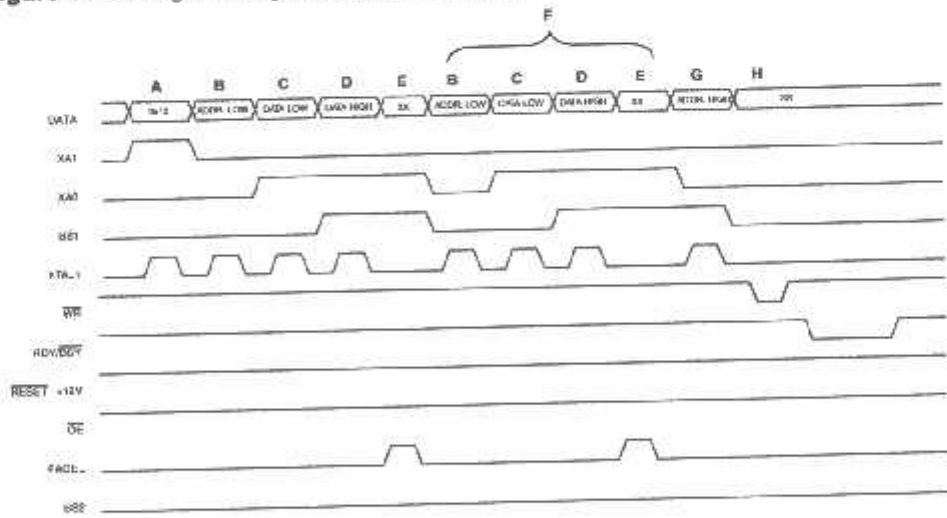
3. Wait until RDY/BSY goes high. (See Figure 117 for signal waveforms)
- I. Repeat B through H until the entire Flash is programmed or until all data has been programmed.
- J. End Page Programming
  1. Set XA1, XA0 to "10". This enables command loading.
  2. Set DATA to "0000 0000". This is the command for No Operation.
  3. Give XTAL1 a positive pulse. This loads the command, and the internal write signals are reset.

Figure 116. Addressing the Flash which is Organized in Pages<sup>(1)</sup>



Note: 1. PCPAGE and PCWORD are listed in Table 104 on page 239.

Figure 117. Programming the Flash Waveforms<sup>(1)</sup>



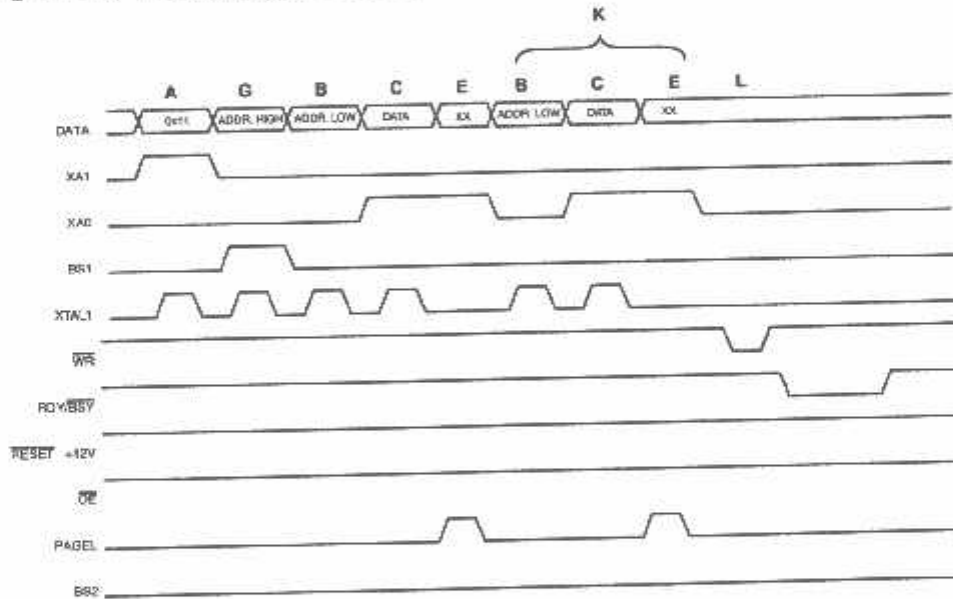
Note: 1. "XX" is don't care. The letters refer to the programming description above.

## Programming the EEPROM

The EEPROM is organized in pages, see Table 105 on page 239. When programming the EEPROM, the program data is latched into a page buffer. This allows one page of data to be programmed simultaneously. The programming algorithm for the EEPROM data memory is as follows (refer to "Programming the Flash" on page 241 for details on Command, Address and Data loading):

1. A: Load Command "0001 0001".
  2. G: Load Address High Byte (0x00 - 0xFF).
  3. B: Load Address Low Byte (0x00 - 0xFF).
  4. C: Load Data (0x00 - 0xFF).
  5. E: Latch data (give  $\overline{\text{PAGEL}}$  a positive pulse).
- K: Repeat 3 through 5 until the entire buffer is filled.
- L: Program EEPROM page.
1. Set BS1 to "0".
  2. Give  $\overline{\text{WR}}$  a negative pulse. This starts programming of the EEPROM page.  $\text{RDY}/\overline{\text{BSY}}$  goes low.
  3. Wait until  $\text{RDY}/\overline{\text{BSY}}$  goes high before programming the next page. (See Figure 118 for signal waveforms.)

**Figure 118.** Programming the EEPROM Waveforms



## Reading the Flash

The algorithm for reading the Flash memory is as follows (refer to "Programming the Flash" on page 241 for details on Command and Address loading):

1. A: Load Command "0000 0010".
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. Set  $\overline{\text{OE}}$  to "0", and BS1 to "0". The Flash word low byte can now be read at DATA.
5. Set BS1 to "1". The Flash word high byte can now be read at DATA.
6. Set  $\overline{\text{OE}}$  to "1".

### Reading the EEPROM

The algorithm for reading the EEPROM memory is as follows (refer to "Programming the Flash" on page 241 for details on Command and Address loading):

1. A: Load Command "0000 0011".
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. Set  $\overline{OE}$  to "0", and BS1 to "0". The EEPROM Data byte can now be read at DATA.
5. Set  $\overline{OE}$  to "1".

### Programming the Fuse Low bits

The algorithm for programming the Fuse Low bits is as follows (refer to "Programming the Flash" on page 241 for details on Command and Data loading):

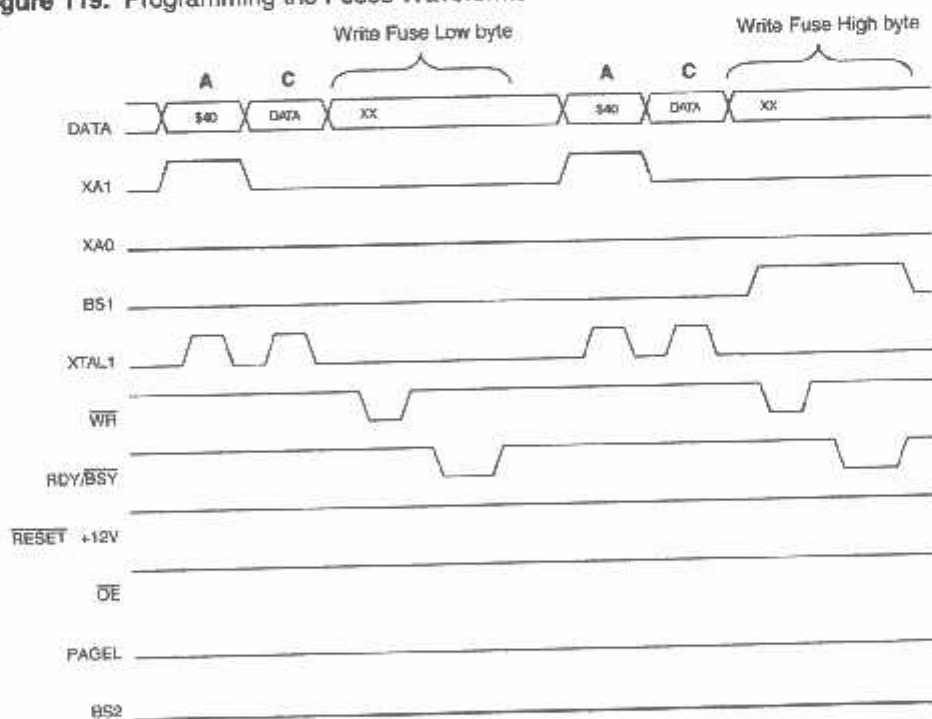
1. A: Load Command "0100 0000".
2. C: Load Data Low Byte. Bit n = "0" programs and bit n = "1" erases the Fuse bit.
3. Set BS1 to "0" and BS2 to "0". This selects low data byte.
4. Give  $\overline{WR}$  a negative pulse and wait for RDY/ $\overline{BSY}$  to go high.

### Programming the Fuse High bits

The algorithm for programming the Fuse high bits is as follows (refer to "Programming the Flash" on page 241 for details on Command and Data loading):

1. A: Load Command "0100 0000".
2. C: Load Data Low Byte. Bit n = "0" programs and bit n = "1" erases the Fuse bit.
3. Set BS1 to "1" and BS2 to "0". This selects high data byte.
4. Give  $\overline{WR}$  a negative pulse and wait for RDY/ $\overline{BSY}$  to go high.
5. Set BS1 to "0". This selects low data byte.

**Figure 119. Programming the Fuses Waveforms**



## Programming the Lock Bits

The algorithm for programming the Lock bits is as follows (refer to "Programming the Flash" on page 241 for details on Command and Data loading):

1. A: Load Command "0010 0000".
2. C: Load Data Low Byte. Bit n = "0" programs the Lock bit.
3. Give  $\overline{WR}$  a negative pulse and wait for RDY/ $\overline{BSY}$  to go high.

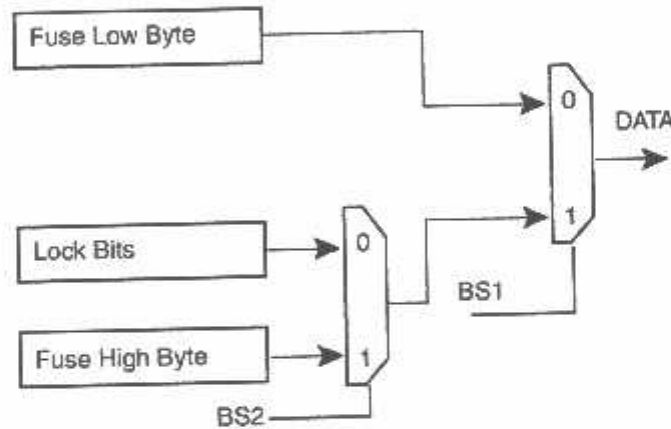
The Lock bits can only be cleared by executing Chip Erase.

## Reading the Fuse and Lock Bits

The algorithm for reading the Fuse and Lock bits is as follows (refer to "Programming the Flash" on page 241 for details on Command loading):

1. A: Load Command "0000 0100".
2. Set  $\overline{OE}$  to "0", BS2 to "0", and BS1 to "0". The status of the Fuse Low bits can now be read at DATA ("0" means programmed).
3. Set  $\overline{OE}$  to "0", BS2 to "1", and BS1 to "1". The status of the Fuse High bits can now be read at DATA ("0" means programmed).
4. Set  $\overline{OE}$  to "0", BS2 to "0", and BS1 to "1". The status of the Lock bits can now be read at DATA ("0" means programmed).
5. Set  $\overline{OE}$  to "1".

**Figure 120.** Mapping Between BS1, BS2 and the Fuse- and Lock Bits During Read



## Reading the Signature Bytes

The algorithm for reading the Signature bytes is as follows (refer to "Programming the Flash" on page 241 for details on Command and Address loading):

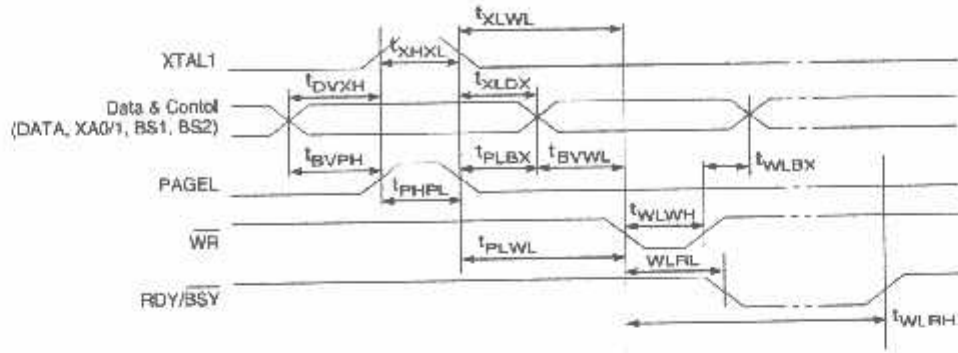
1. A: Load Command "0000 1000".
2. B: Load Address Low Byte (0x00 - 0x02).
3. Set  $\overline{OE}$  to "0", and BS to "0". The selected Signature byte can now be read at DATA.
4. Set  $\overline{OE}$  to "1".

## Reading the Calibration Byte

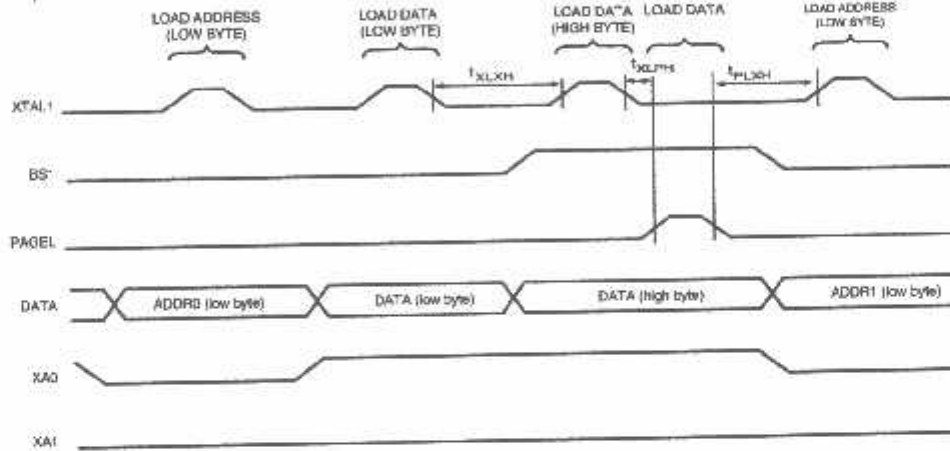
The algorithm for reading the Calibration byte is as follows (refer to "Programming the Flash" on page 241 for details on Command and Address loading):

1. A: Load Command "0000 1000".
2. B: Load Address Low Byte, 0x00.
3. Set  $\overline{OE}$  to "0", and BS1 to "1". The Calibration byte can now be read at DATA.
4. Set  $\overline{OE}$  to "1".

**Figure 121.** Parallel Programming Timing, Including some General Timing Requirements



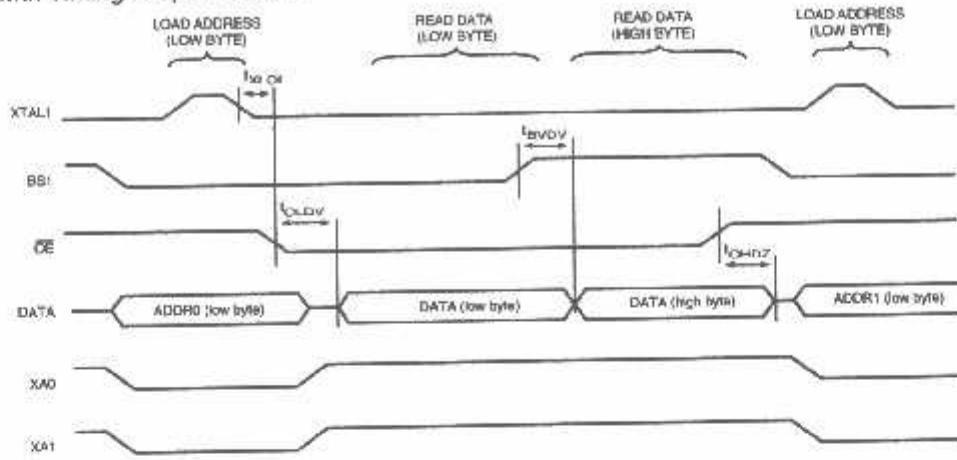
**Figure 122.** Parallel Programming Timing, Loading Sequence with Timing Requirements<sup>(1)</sup>



Note: 1. The timing requirements shown in Figure 121 (i.e.  $t_{DVXH}$ ,  $t_{XHXL}$ , and  $t_{XLDX}$ ) also apply to loading operation.



**Figure 123. Parallel Programming Timing, Reading Sequence (within the same Page) with Timing Requirements<sup>(1)</sup>**



Note: 1. The timing requirements shown in Figure 121 (i.e.  $t_{DVXH}$ ,  $t_{XHXL}$ , and  $t_{XLDX}$ ) also apply to reading operation.

**Table 106. Parallel Programming Characteristics,  $V_{CC} = 5V \pm 10\%$** 

Symbol	Parameter	Min	Typ	Max	Units
$V_{PP}$	Programming Enable Voltage	11.5		12.5	V
$I_{PP}$	Programming Enable Current			250	$\mu A$
$t_{DVXH}$	Data and Control Valid before XTAL1 High	67			ns
$t_{XLXH}$	XTAL1 Low to XTAL1 High	200			ns
$t_{XHXL}$	XTAL1 Pulse Width High	150			ns
$t_{XLDX}$	Data and Control Hold after XTAL1 Low	67			ns
$t_{XLWL}$	XTAL1 Low to $\overline{WR}$ Low	0			ns
$t_{XLPH}$	XTAL1 Low to PAGED High	0			ns
$t_{PLXH}$	PAGED low to XTAL1 High	150			ns
$t_{BVPH}$	BS1 Valid before PAGED High	67			ns
$t_{PHPL}$	PAGED Pulse Width High	150			ns
$t_{PLBX}$	BS1 Hold after PAGED Low	67			ns
$t_{WL BX}$	BS2/1 Hold after $\overline{WR}$ Low	67			ns
$t_{PLWL}$	PAGED Low to $\overline{WR}$ Low	67			ns
$t_{BVWL}$	BS1 Valid to $\overline{WR}$ Low	67			ns
$t_{WLWH}$	$\overline{WR}$ Pulse Width Low	150			ns
$t_{WLRL}$	$\overline{WR}$ Low to RDY/BSY Low	0		1	$\mu s$
$t_{WLRH}$	$\overline{WR}$ Low to RDY/BSY High <sup>(1)</sup>	3.7		4.5	ms
$t_{WLRH\_CE}$	$\overline{WR}$ Low to RDY/BSY High for Chip Erase <sup>(2)</sup>	7.5		9	ms
$t_{XLLOL}$	XTAL1 Low to $\overline{OE}$ Low	0			ns
$t_{BV DV}$	BS1 Valid to DATA Valid	0		250	ns
$t_{OLDV}$	$\overline{OE}$ Low to DATA Valid			250	ns
$t_{OHDZ}$	$\overline{OE}$ High to DATA Tri-stated			250	ns

- Notes: 1.  $t_{WLRH}$  is valid for the Write Flash, Write EEPROM, Write Fuse bits and Write Lock bits commands.  
 2.  $t_{WLRH\_CE}$  is valid for the Chip Erase command.

When writing serial data to the ATmega8535, data is clocked on the rising edge of SCK.

When reading data from the ATmega8535, data is clocked on the falling edge of SCK. See Figure 125 for timing details.

To program and verify the ATmega8535 in the Serial Programming mode, the following sequence is recommended (See four byte instruction formats in Table 109):

1. Power-up sequence:  
Apply power between  $V_{CC}$  and GND while  $\overline{RESET}$  and SCK are set to "0". In some systems, the programmer can not guarantee that SCK is held low during Power-up. In this case,  $\overline{RESET}$  must be given a positive pulse of at least two CPU clock cycles duration after SCK has been set to "0".
2. Wait for at least 20 ms and enable serial programming by sending the Programming Enable serial instruction to pin MOSI.
3. The serial programming instructions will not work if the communication is out of synchronization. When in synchronization the second byte (0x53), will echo back when issuing the third byte of the Programming Enable instruction. Whether the echo is correct or not, all four bytes of the instruction must be transmitted. If the 0x53 did not echo back, give  $\overline{RESET}$  a positive pulse and issue a new Programming Enable command.
4. The Flash is programmed one page at a time. The page size is found in Table 104 on page 239. The memory page is loaded one byte at a time by supplying the 6 LSB of the address and data together with the Load Program Memory Page instruction. To ensure correct loading of the page, the data low byte must be loaded before data high byte is applied for a given address. The Program Memory Page is stored by loading the Write Program Memory Page instruction with the 8 MSB of the address. If polling is not used, the user must wait at least  $t_{WD\_FLASH}$  before issuing the next page. (See Table 108.) Accessing the serial programming interface before the Flash write operation completes can result in incorrect programming.
5. The EEPROM array is programmed one byte at a time by supplying the address and data together with the appropriate Write instruction. An EEPROM memory location is first automatically erased before new data is written. If polling is not used, the user must wait at least  $t_{WD\_EEPROM}$  before issuing the next byte. (See Table 108). In a chip erased device, no 0xFFs in the data file(s) need to be programmed.
6. Any memory location can be verified by using the Read instruction which returns the content at the selected address at serial output MISO.
7. At the end of the programming session,  $\overline{RESET}$  can be set high to commence normal operation.
8. Power-off sequence (if needed):  
Set  $\overline{RESET}$  to "1".  
Turn  $V_{CC}$  power off.

**Data Polling Flash**

When a page is being programmed into the Flash, reading an address location within the page being programmed will give the value 0xFF. At the time the device is ready for a new page, the programmed value will read correctly. This is used to determine when the next page can be written. Note that the entire page is written simultaneously and any address within the page can be used for polling. Data polling of the Flash will not work for the value 0xFF, so when programming this value, the user will have to wait for at least  $t_{WD\_FLASH}$  before programming the next page. As a chip erased device contains 0xFF in all locations, programming of addresses that are meant to contain 0xFF, can be skipped. See Table 108 for  $t_{WD\_FLASH}$  value.

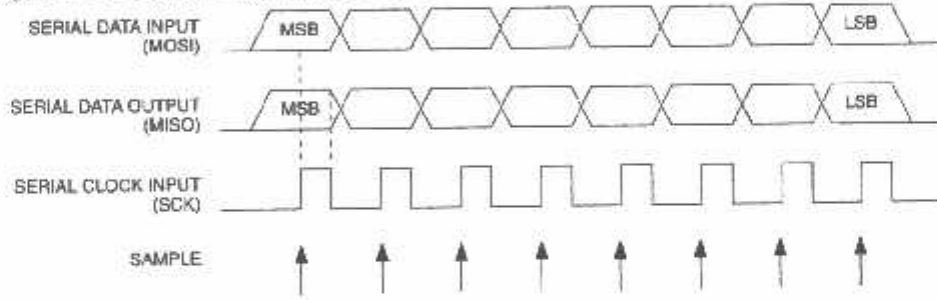
## Data Polling EEPROM

When a new byte has been written and is being programmed into EEPROM, reading the address location being programmed will give the value 0xFF. At the time the device is ready for a new byte, the programmed value will read correctly. This is used to determine when the next byte can be written. This will not work for the value 0xFF, but the user should have the following in mind: As a chip erased device contains 0xFF in all locations, programming of addresses that are meant to contain 0xFF, can be skipped. This does not apply if the EEPROM is reprogrammed without chip erasing the device. In this case, data polling cannot be used for the value 0xFF, and the user will have to wait at least  $t_{WD\_EEPROM}$  before programming the next byte. See Table 108 for  $t_{WD\_EEPROM}$  value.

**Table 108.** Minimum Wait Delay Before Writing the Next Flash or EEPROM Location

Symbol	Minimum Wait Delay
$t_{WD\_FLASH}$	4.5 ms
$t_{WD\_EEPROM}$	9.0 ms
$t_{WD\_ERASE}$	9.0 ms
$t_{WD\_FUSE}$	4.5 ms

**Figure 125.** Serial Programming Waveforms



**Table 109. Serial Programming Instruction Set**

 = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, **x** = don't care

Instruction	Instruction Format				Operation
	Byte 1	Byte 2	Byte 3	Byte 4	
Programming Enable	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Enable Serial Programming after RESET goes low.
Chip Erase	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Chip Erase EEPROM and Flash.
Read Program Memory	0010 H000	0000 aaaa	bbbb bbbb	oooo oooo	Read H (high or low) data o from Program memory at word address a:b.
Load Program Memory Page	0100 H000	0000 xxxx	xxxb bbbb	iiii iiiii	Write H (high or low) data i to Program Memory page at word address b. Data low byte must be loaded before Data high byte is applied within the same address.
Write Program Memory Page	0100 1100	0000 aaaa	bbbx xxxx	xxxx xxxx	Write Program Memory Page at address a:b.
Read EEPROM Memory	1010 0000	00xx xxxa	bbbb bbbb	oooo oooo	Read data o from EEPROM memory at address a:b.
Write EEPROM Memory	1100 0000	00xx xxxa	bbbb bbbb	iiii iiiii	Write data i to EEPROM memory at address a:b.
Read Lock Bits	0101 1000	0000 0000	xxxx xxxx	xxoo oooo	Read Lock bits. "0" = programmed, "1" = unprogrammed. See Table 96 on page 235 for details.
Write Lock Bits	1010 1100	111x xxxx	xxxx xxxx	11ii iiiii	Write Lock bits. Set bits = "0" to program Lock bits. See Table 96 on page 235 for details.
Read Signature Byte	0011 0000	00xx xxxx	xxxx xxbb	oooo oooo	Read Signature Byte o at address b.
Write Fuse Bits	1010 1100	1010 0000	xxxx xxxx	iiii iiiii	Set bits = "0" to program, "1" to unprogram. See Table 99 on page 237 for details.
Write Fuse High Bits	1010 1100	1010 1000	xxxx xxxx	iiii iiiii	Set bits = "0" to program, "1" to unprogram. See Table 98 on page 236 for details.
Read Fuse Bits	0101 0000	0000 0000	xxxx xxxx	oooo oooo	Read Fuse bits. "0" = programmed, "1" = unprogrammed. See Table 99 on page 237 for details.
Read Fuse High Bits	0101 1000	0000 1000	xxxx xxxx	oooo oooo	Read Fuse high bits. "0" = programmed, "1" = unprogrammed. See Table 98 on page 236 for details.
Read Calibration Byte	0011 1000	00xx xxxx	0000 00bb	oooo oooo	Read Calibration Byte

**Serial Programming characteristics**

For characteristics of the SPI module, see "SPI Timing Characteristics" on page 259.

## Electrical Characteristics

### Absolute Maximum Ratings\*

Operating Temperature.....	-55°C to +125°C
Storage Temperature.....	-65°C to +150°C
Voltage on any Pin except <b>RESET</b> with respect to Ground.....	-0.5V to $V_{CC}+0.5V$
Voltage on <b>RESET</b> with respect to Ground.....	-0.5V to +13.0V
Maximum Operating Voltage.....	6.0V
DC Current per I/O Pin.....	40.0 mA
DC Current $V_{CC}$ and GND Pins.....	200.0 PDIP og 400 mA TQFP/MLF/PLCCmA

\*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

### DC Characteristics

$T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ,  $V_{CC} = 2.7V$  to  $5.5V$  (unless otherwise noted)

Symbol	Parameter	Condition	Min	Typ	Max	Units
$V_{IL}$	Input Low Voltage except XTAL1 and RESET pins	$V_{CC}=2.7V - 5.5V$	-0.5		$0.2 V_{CC}^{(1)}$	V
$V_{IH}$	Input High Voltage except XTAL1 and RESET pins	$V_{CC}=2.7V - 5.5V$	$0.6 V_{CC}^{(2)}$		$V_{CC} + 0.5$	V
$V_{IL1}$	Input Low Voltage XTAL1 pin	$V_{CC}=2.7V - 5.5V$	-0.5		$0.1 V_{CC}^{(1)}$	V
$V_{IH1}$	Input High Voltage XTAL1 pin	$V_{CC}=2.7V - 5.5V$	$0.8 V_{CC}^{(2)}$		$V_{CC} + 0.5$	V
$V_{IL2}$	Input Low Voltage RESET pin	$V_{CC}=2.7V - 5.5V$	-0.5		$0.2 V_{CC}$	V
$V_{IH2}$	Input High Voltage RESET pin	$V_{CC}=2.7V - 5.5V$	$0.9 V_{CC}^{(2)}$		$V_{CC} + 0.5$	V
$V_{OL}$	Output Low Voltage <sup>(3)</sup> (Ports A,B,C,D)	$I_{OL} = 20 \text{ mA}, V_{CC} = 5V$			0.7	V
		$I_{OL} = 10 \text{ mA}, V_{CC} = 3V$			0.5	V
$V_{OH}$	Output High Voltage <sup>(4)</sup> (Ports A,B,C,D)	$I_{OH} = -20 \text{ mA}, V_{CC} = 5V$	4.2			V
		$I_{OH} = -10 \text{ mA}, V_{CC} = 3V$	2.2			V
$I_{IL}$	Input Leakage Current I/O Pin	$V_{CC} = 5.5V$ , pin low (absolute value)			1	$\mu\text{A}$
$I_{IH}$	Input Leakage Current I/O Pin	$V_{CC} = 5.5V$ , pin high (absolute value)			1	$\mu\text{A}$
$R_{RST}$	Reset Pull-up Resistor		30		60	$k\Omega$
$R_{pu}$	I/O Pin Pull-up Resistor		20		50	$k\Omega$

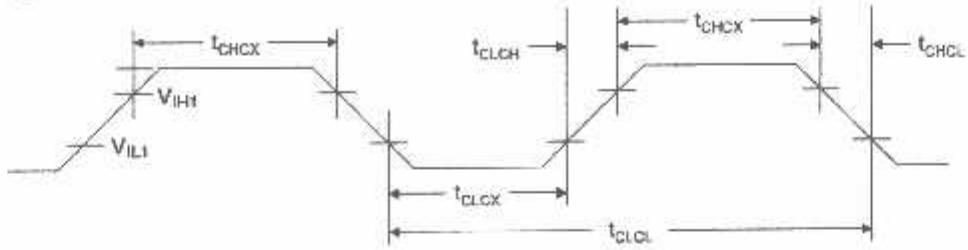
$T_A = -40^{\circ}\text{C}$  to  $85^{\circ}\text{C}$ ,  $V_{CC} = 2.7\text{V}$  to  $5.5\text{V}$  (unless otherwise noted) (Continued)

Symbol	Parameter	Condition	Min	Typ	Max	Units	
$I_{CC}$	Power Supply Current	Active 4 MHz, $V_{CC} = 3\text{V}$ (ATmega8535L)		4		mA	
		Active 8 MHz, $V_{CC} = 5\text{V}$ (ATmega8535)		14		mA	
		Idle 4 MHz, $V_{CC} = 3\text{V}$ (ATmega8535L)		3		mA	
		Idle 8 MHz, $V_{CC} = 5\text{V}$ (ATmega8535)		10		mA	
	Power-down mode <sup>(5)</sup>	WDT enabled, $V_{CC} = 3\text{V}$			< 10		$\mu\text{A}$
		WDT disabled, $V_{CC} = 3\text{V}$			< 3		$\mu\text{A}$
$V_{ACIO}$	Analog Comparator Input Offset Voltage	$V_{CC} = 5\text{V}$ $V_{in} = V_{CC}/2$			40	mV	
$I_{ACLK}$	Analog Comparator Input Leakage Current	$V_{CC} = 5\text{V}$ $V_{in} = V_{CC}/2$	-50		50	nA	
$t_{ACPD}$	Analog Comparator Propagation Delay	$V_{CC} = 2.7\text{V}$ $V_{CC} = 4.0\text{V}$		750 500		ns	

- Notes:
- "Max" means the highest value where the pin is guaranteed to be read as low.
  - "Min" means the lowest value where the pin is guaranteed to be read as high.
  - Although each I/O port can sink more than the test conditions (20mA at  $V_{CC} = 5\text{V}$ , 10mA at  $V_{CC} = 3\text{V}$ ) under steady state conditions (non-transient), the following must be observed:  
 PDIP Package:  
 1) The sum of all IOL, for all ports, should not exceed 200 mA.  
 2) The sum of all IOL, for port A0 - A7, should not exceed 100 mA.  
 3) The sum of all IOL, for ports B0 - B7, C0 - C7, D0 - D7 and XTAL2, should not exceed 100 mA.  
 PLCC/MLF/TQFP Package:  
 1) The sum of all IOL, for all ports, should not exceed 400 mA.  
 2) The sum of all IOL, for ports A0 - A7, should not exceed 100 mA.  
 3) The sum of all IOL, for ports B0 - B3, should not exceed 100 mA.  
 4) The sum of all IOL, for ports B4 - B7, should not exceed 100 mA.  
 5) The sum of all IOL, for ports C0 - C3, should not exceed 100 mA.  
 6) The sum of all IOL, for ports C4 - C7, should not exceed 100 mA.  
 7) The sum of all IOL, for ports D0 - D3 and XTAL2, should not exceed 100 mA.  
 8) The sum of all IOL, for ports D4 - D7, should not exceed 100 mA.  
 If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
  - Although each I/O port can source more than the test conditions (20mA at  $V_{CC} = 5\text{V}$ , 10mA at  $V_{CC} = 3\text{V}$ ) under steady state conditions (non-transient), the following must be observed:  
 PDIP Package:  
 1) The sum of all IOH, for all ports, should not exceed 200 mA.  
 2) The sum of all IOH, for port A0 - A7, should not exceed 100 mA.  
 3) The sum of all IOH, for ports B0 - B7, C0 - C7, D0 - D7 and XTAL2, should not exceed 100 mA.  
 PLCC/MLF/TQFP Package:  
 1) The sum of all IOH, for all ports, should not exceed 400 mA.  
 2) The sum of all IOH, for ports A0 - A7, should not exceed 100 mA.  
 3) The sum of all IOH, for ports B0 - B3, should not exceed 100 mA.  
 4) The sum of all IOH, for ports B4 - B7, should not exceed 100 mA.  
 5) The sum of all IOH, for ports C0 - C3, should not exceed 100 mA.  
 6) The sum of all IOH, for ports C4 - C7, should not exceed 100 mA.  
 7) The sum of all IOH, for ports D0 - D3 and XTAL2, should not exceed 100 mA.  
 8) The sum of all IOH, for ports D4 - D7, should not exceed 100 mA.

## External Clock Drive Waveforms

Figure 126. External Clock Drive Waveforms



## External Clock Drive

Table 110. External Clock Drive

Symbol	Parameter	$V_{CC} = 2.7V \text{ to } 5.5V$		$V_{CC} = 4.5V \text{ to } 5.5V$		Units
		Min	Max	Min	Max	
$1/t_{CLCL}$	Oscillator Frequency	0	8	0	16	MHz
$t_{CLCL}$	Clock Period	125		62.5		ns
$t_{CHCX}$	High Time	50		25		ns
$t_{CLCX}$	Low Time	50		25		ns
$t_{CLCH}$	Rise Time		1.6		0.5	$\mu s$
$t_{CHCL}$	Fall Time		1.6		0.5	$\mu s$
$\Delta t_{CLCL}$	Change in period from one clock cycle to the next		2		2	%

Table 111. External RC Oscillator, Typical Frequencies

R [k $\Omega$ ] <sup>(1)</sup>	C [pF]	f <sup>(2)</sup>
100	47	87 kHz
33	22	650 kHz
10	22	2.0 MHz

- Notes:
1. R should be in the range 3 k $\Omega$  - 100 k $\Omega$ , and C should be at least 20 pF. The C values given in the table includes pin capacitance. This will vary with package type.
  2. The frequency will vary with package type and board layout.



## Two-wire Serial Interface Characteristics

Table 112 describes the requirements for devices connected to the Two-wire Serial Bus. The ATmega8535 Two-wire Serial Interface meets or exceeds these requirements under the noted conditions.

Timing symbols refer to Figure 127.

Table 112. Two-wire Serial Bus Requirements

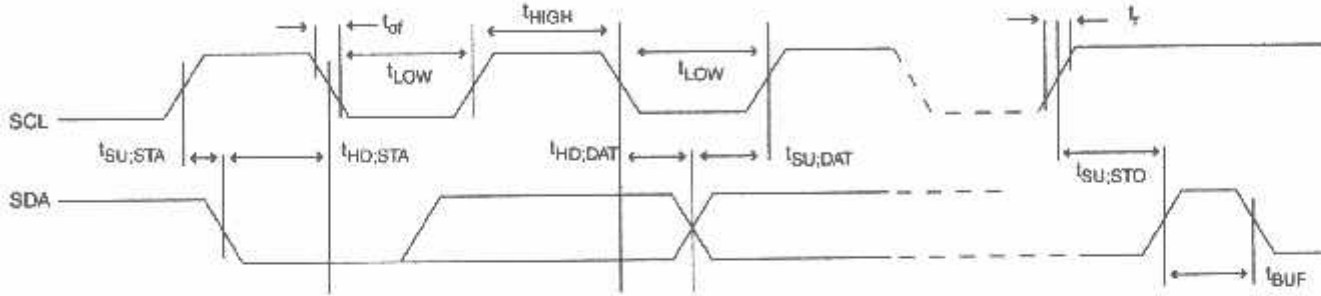
Symbol	Parameter	Condition	Min	Max	Units
$V_{IL}$	Input Low Voltage		-0.5	$0.3 V_{CC}$	V
$V_{IH}$	Input High Voltage		$0.7 V_{CC}$	$V_{CC} + 0.5$	V
$V_{HYST}^{(1)}$	Hysteresis of Schmitt Trigger Inputs		$0.05 V_{CC}^{(2)}$	-	V
$V_{OL}^{(4)}$	Output Low Voltage	3 mA sink current	0	0.4	V
$t_r^{(1)}$	Rise Time for both SDA and SCL		$20 + 0.1C_b^{(3)(2)}$	300	ns
$t_f^{(1)}$	Output Fall Time from $V_{IHmin}$ to $V_{ILmax}$	$10 \text{ pF} < C_b < 400 \text{ pF}^{(3)}$	$20 + 0.1C_b^{(3)(2)}$	250	ns
$SP^{(1)}$	Spikes Suppressed by Input Filter		0	$50^{(2)}$	ns
$I_i$	Input Current each I/O Pin	$0.1V_{CC} < V_i < 0.9V_{CC}$	-10	10	$\mu\text{A}$
$C_i^{(4)}$	Capacitance for each I/O Pin		-	10	pF
$f_{SCL}$	SCL Clock Frequency	$f_{CK}^{(4)} > \max(16f_{SCL}, 250\text{kHz})^{(5)}$	0	400	kHz
$R_p$	Value of Pull-up resistor	$f_{SCL} \leq 100 \text{ kHz}$	$\frac{V_{CC} - 0.4V}{3\text{mA}}$	$\frac{1000\text{ns}}{C_b}$	$\Omega$
		$f_{SCL} > 100 \text{ kHz}$	$\frac{V_{CC} - 0.4V}{3\text{mA}}$	$\frac{300\text{ns}}{C_b}$	$\Omega$
HD:STA	Hold Time (Repeated) START Condition	$f_{SCL} \leq 100 \text{ kHz}$	4.0	-	$\mu\text{s}$
		$f_{SCL} > 100 \text{ kHz}$	0.6	-	$\mu\text{s}$
LOW	Low Period of the SCL Clock	$f_{SCL} \leq 100 \text{ kHz}^{(6)}$	4.7	-	$\mu\text{s}$
		$f_{SCL} > 100 \text{ kHz}^{(7)}$	1.3	-	$\mu\text{s}$
HIGH	High Period of the SCL clock	$f_{SCL} \leq 100 \text{ kHz}$	4.0	-	$\mu\text{s}$
		$f_{SCL} > 100 \text{ kHz}$	0.6	-	$\mu\text{s}$
SU:STA	Set-up Time for a Repeated START Condition	$f_{SCL} \leq 100 \text{ kHz}$	4.7	-	$\mu\text{s}$
		$f_{SCL} > 100 \text{ kHz}$	0.6	-	$\mu\text{s}$
HD:DAT	Data hold Time	$f_{SCL} \leq 100 \text{ kHz}$	0	3.45	$\mu\text{s}$
		$f_{SCL} > 100 \text{ kHz}$	0	0.9	$\mu\text{s}$
SU:DAT	Data Setup Time	$f_{SCL} \leq 100 \text{ kHz}$	250	-	ns
		$f_{SCL} > 100 \text{ kHz}$	100	-	ns
SU:STO	Setup Time for STOP Condition	$f_{SCL} \leq 100 \text{ kHz}$	4.0	-	$\mu\text{s}$
		$f_{SCL} > 100 \text{ kHz}$	0.6	-	$\mu\text{s}$
3UF	Bus Free Time between a STOP and START Condition	$f_{SCL} \leq 100 \text{ kHz}$	4.7	-	$\mu\text{s}$
		$f_{SCL} > 100 \text{ kHz}$	1.3	-	$\mu\text{s}$

- Notes:
- In ATmega8535, this parameter is characterized and not 100% tested.
  - Required only for  $f_{SCL} > 100 \text{ kHz}$ .
  - $C_b$  = capacitance of one bus line in pF.



4.  $f_{CK}$  = CPU clock frequency.
5. This requirement applies to all ATmega8535 Two-wire Serial Interface operation. Other devices connected to the Two-wire Serial Bus need only obey the general  $f_{SCL}$  requirement.
6. The actual low period generated by the ATmega8535 Two-wire Serial Interface is  $(1/f_{SCL} - 2/f_{CK})$ , thus  $f_{CK}$  must be greater than 6 MHz for the low time requirement to be strictly met at  $f_{SCL} = 100$  kHz.
7. The actual low period generated by the ATmega8535 Two-wire Serial Interface is  $(1/f_{SCL} - 2/f_{CK})$ , thus the low time requirement will not be strictly met for  $f_{SCL} > 308$  kHz when  $f_{CK} = 8$  MHz. Still, ATmega8535 devices connected to the bus may communicate at full speed (400 kHz) with other ATmega8535 devices, as well as any other device with a proper  $t_{LOW}$  acceptance margin.

Figure 127. Two-wire Serial Bus Timing



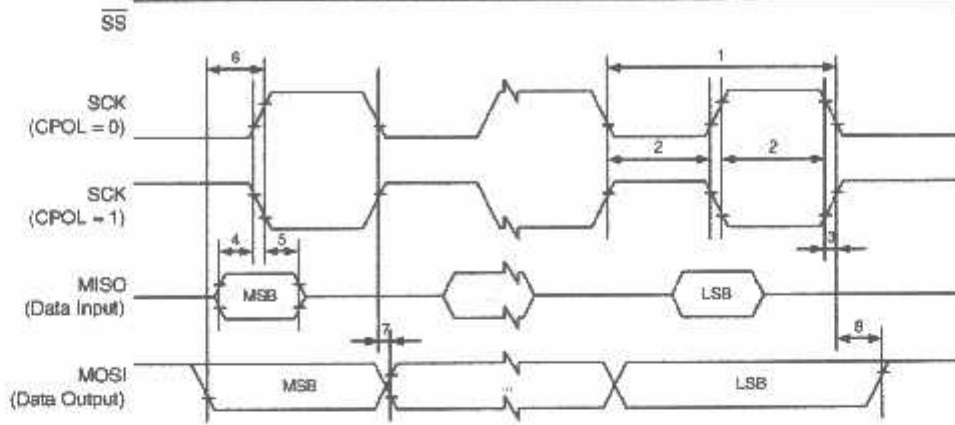
## PI Timing characteristics

See Figure 128 and Figure 129 for details.

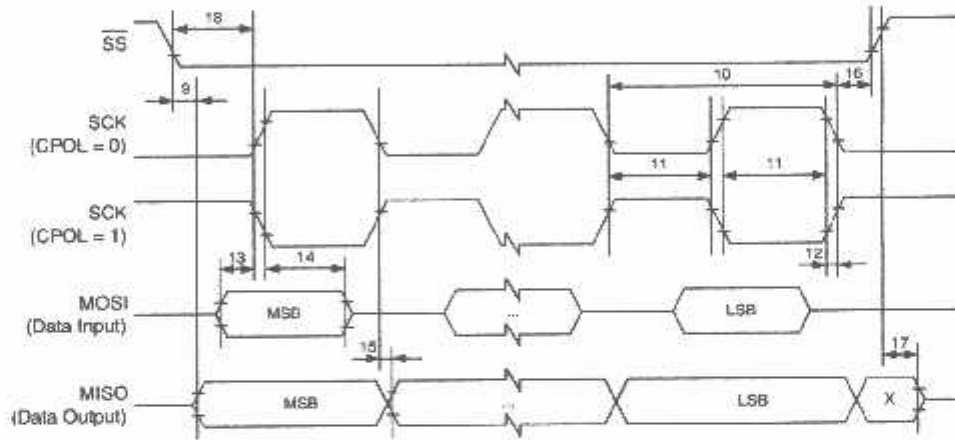
Table 113. SPI Timing Parameters

	Description	Mode	Min	Typ	Max	
1	SCK period	Master		See Table 59		ns
2	SCK high/low	Master		50% duty cycle		
3	Rise/Fall time	Master		3.6		
4	Setup	Master		10		
5	Hold	Master		10		
6	Out to SCK	Master		$5 \cdot t_{\text{SCK}}$		
7	SCK to out	Master		10		
8	SCK to out high	Master		10		
9	SS low to out	Slave		15		
10	SCK period	Slave	$4 \cdot t_{\text{ck}}$			
11	SCK high/low	Slave	$2 \cdot t_{\text{ck}}$			
12	Rise/Fall time	Slave			1.6	ns
13	Setup	Slave	10			ns
14	Hold	Slave	10			
15	SCK to out	Slave		15		
16	SCK to $\overline{\text{SS}}$ high	Slave	20			
17	$\overline{\text{SS}}$ high to tri-state	Slave		10		
18	SS low to SCK	Slave	$2 \cdot t_{\text{ck}}$			

**Figure 128. SPI Interface Timing Requirements (Master Mode)**



**Figure 129. SPI Interface Timing Requirements (Slave Mode)**



## DC Characteristics – Preliminary Data

Table 114. ADC Characteristics, Single Ended Channels

Symbol	Parameter	Condition	Min <sup>(1)</sup>	Typ <sup>(1)</sup>	Max <sup>(1)</sup>	Units
	Resolution	Single Ended Conversion		10		Bits
	Absolute Accuracy (Including INL, DNL, Quantization Error, Gain and Offset Error)	Single Ended Conversion $V_{REF} = 4V, V_{CC} = 4V$ ADC clock = 200 kHz		1.5		LSB
		Single Ended Conversion $V_{REF} = 4V, V_{CC} = 4V$ ADC clock = 1 MHz		3		LSB
		Single Ended Conversion $V_{REF} = 4V, V_{CC} = 4V$ ADC clock = 200 kHz Noise Reduction mode		1.5		LSB
		Single Ended Conversion $V_{REF} = 4V, V_{CC} = 4V$ ADC clock = 1 MHz Noise Reduction mode		3		LSB
	Integral Non-Linearity (INL)	Single Ended Conversion $V_{REF} = 4V, V_{CC} = 4V$ ADC clock = 200 kHz		0.75		LSB
	Differential Non-Linearity (DNL)	Single Ended Conversion $V_{REF} = 4V, V_{CC} = 4V$ ADC clock = 200 kHz		0.25		LSB
	Gain Error	Single Ended Conversion $V_{REF} = 4V, V_{CC} = 4V$ ADC clock = 200 kHz		0.75		LSB
	Offset error	Single Ended Conversion $V_{REF} = 4V, V_{CC} = 4V$ ADC clock = 200 kHz		0.75		LSB
	Clock Frequency		50		1000	kHz
	Conversion Time		13		260	µs
AVCC	Analog Supply Voltage		$V_{CC} - 0.3^{(2)}$		$V_{CC} + 0.3^{(3)}$	V
V <sub>REF</sub>	Reference Voltage		2.0		AVCC	V
V <sub>IN</sub>	Input Voltage		GND		V <sub>REF</sub>	V
	ADC Conversion Output		0		1023	LSB
	Input Bandwidth			38.5		kHz
V <sub>INT</sub>	Internal Voltage Reference		2.3	2.58	2.7	V
R <sub>REF</sub>	Reference Input Resistance			32		kΩ
R <sub>AIN</sub>	Analog Input Resistance			100		MΩ

- Notes:
1. Values are guidelines only.
  2. Minimum for AV<sub>CC</sub> is 2.7V.
  3. Maximum for AV<sub>CC</sub> is 5.5V



**Table 115. ADC Characteristics, Differential Channels**

Symbol	Parameter	Condition	Min <sup>(1)</sup>	Typ <sup>(1)</sup>	Max <sup>(1)</sup>	Units
	Resolution	Gain = 1x			10	Bits
		Gain = 10x			10	Bits
		Gain = 200x			10	Bits
	Absolute Accuracy	Gain = 1x $V_{REF} = 4V, V_{CC} = 5V$ ADC clock = 50 - 200 kHz		18		LSB
		Gain = 10x $V_{REF} = 4V, V_{CC} = 5V$ ADC clock = 50 - 200 kHz		18		LSB
		Gain = 200x $V_{REF} = 4V, V_{CC} = 5V$ ADC clock = 50 - 200 kHz		6		LSB
	Integral Non-Linearity (INL) (Accuracy after Calibration for Offset and Gain Error)	Gain = 1x $V_{REF} = 4V, V_{CC} = 5V$ ADC clock = 50 - 200 kHz		0.75		LSB
		Gain = 10x $V_{REF} = 4V, V_{CC} = 5V$ ADC clock = 50 - 200 kHz		0.75		LSB
		Gain = 200x $V_{REF} = 4V, V_{CC} = 5V$ ADC clock = 50 - 200 kHz		3.5		LSB
	Gain Error	Gain = 1x		1.7		%
		Gain = 10x		1.6		%
		Gain = 200x		0.3		%
	Offset Error	Gain = 1x $V_{REF} = 4V, V_{CC} = 5V$ ADC clock = 50 - 200 kHz		2		LSB
		Gain = 10x $V_{REF} = 4V, V_{CC} = 5V$ ADC clock = 50 - 200 kHz		2.5		LSB
		Gain = 200x $V_{REF} = 4V, V_{CC} = 5V$ ADC clock = 50 - 200 kHz		3.5		LSB
	Clock Frequency		50		200	kHz
	Conversion Time		65		260	$\mu s$
AVCC	Analog Supply Voltage		$V_{CC} - 0.3^{(2)}$		$V_{CC} + 0.3^{(3)}$	V
$V_{REF}$	Reference Voltage		2.0		AVCC - 0.5	V
$V_{IN}$	Input Voltage		GND		$V_{CC}$	V
$V_{DIFF}$	Input Differential Voltage		$-V_{REF}/Gain$		$V_{REF}/Gain$	V
	ADC Conversion Output		-511		511	LSB
	Input Bandwidth			4		kHz

**Table 115.** ADC Characteristics, Differential Channels (Continued)

Symbol	Parameter	Condition	Min <sup>(1)</sup>	Typ <sup>(1)</sup>	Max <sup>(1)</sup>	Units
V <sub>INT</sub>	Internal Voltage Reference		2.3	2.56	2.7	V
R <sub>REF</sub>	Reference Input Resistance			32		kΩ
R <sub>AIN</sub>	Analog Input Resistance			100		MΩ

- Notes:
1. Values are guidelines only.
  2. Minimum for AVCC is 2.7V.
  3. Maximum for AVCC is 5.5V.

## ATmega8535 Typical Characteristics – Preliminary Data

The following charts show typical behavior. These figures are not tested during manufacturing. All current consumption measurements are performed with all I/O pins configured as inputs and with internal pull-ups enabled. A sine wave generator with rail-to-rail output is used as clock source.

The power consumption in Power-down mode is independent of clock selection.

The current consumption is a function of several factors such as: Operating voltage, operating frequency, loading of I/O pins, switching rate of I/O pins, code executed and ambient temperature. The dominating factors are operating voltage and frequency.

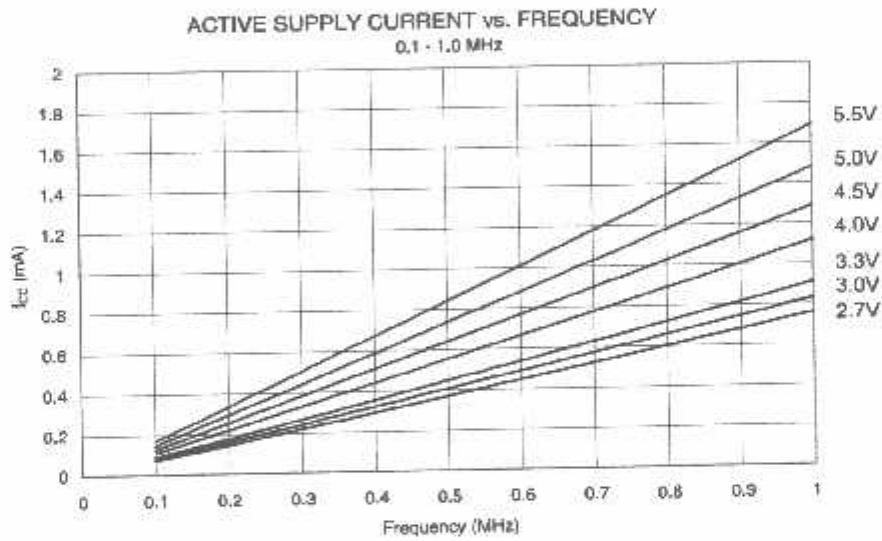
The current drawn from capacitive loaded pins may be estimated (for one pin) as  $C_L \cdot V_{CC} \cdot f$  where  $C_L$  = load capacitance,  $V_{CC}$  = operating voltage and  $f$  = average switching frequency of I/O pin.

The parts are characterized at frequencies higher than test limits. Parts are not guaranteed to function properly at frequencies higher than the ordering code indicates.

The difference between current consumption in Power-down mode with Watchdog Timer enabled and Power-down mode with Watchdog Timer disabled represents the differential current drawn by the Watchdog Timer.

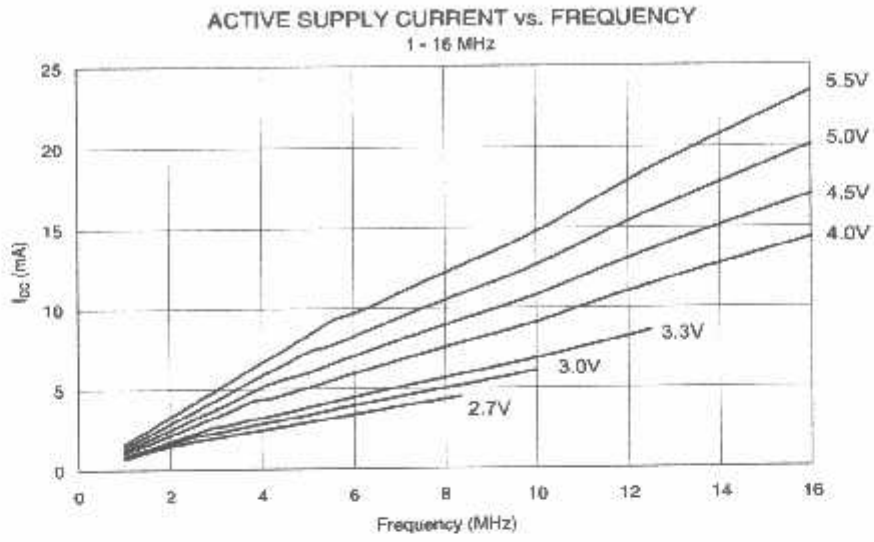
### Active Supply Current

Figure 130. Active Supply Current vs. Frequency (0.1 - 1.0 MHz)

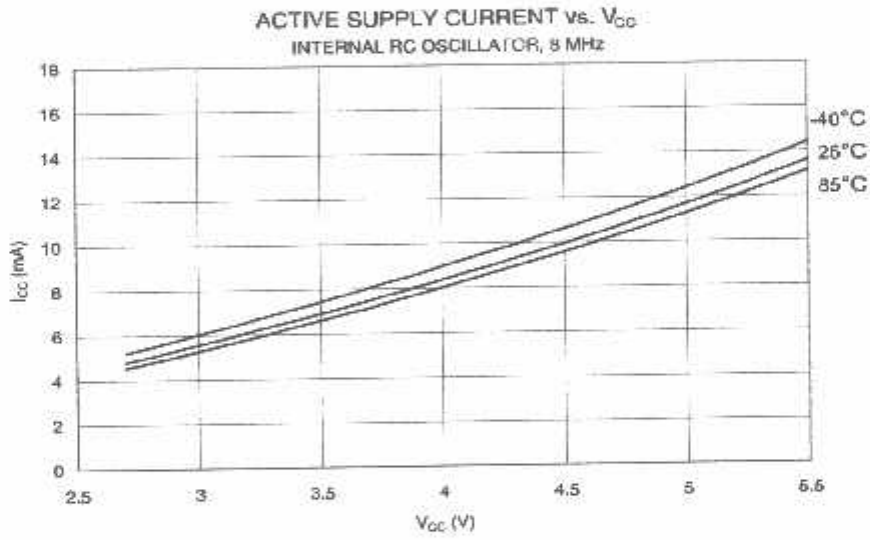




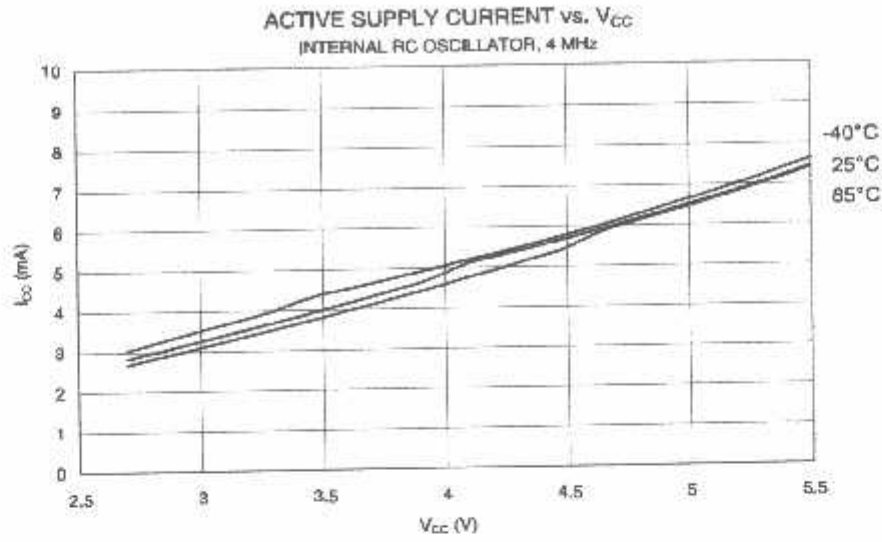
**Figure 131.** Active Supply Current vs. Frequency (1 - 16 MHz)



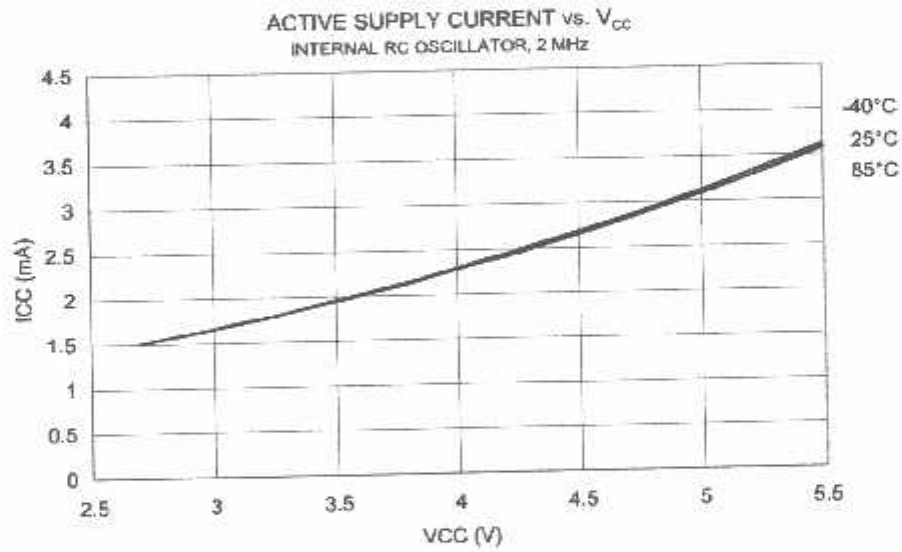
**Figure 132.** Active Supply Current vs.  $V_{CC}$  (Internal RC Oscillator, 8 MHz)



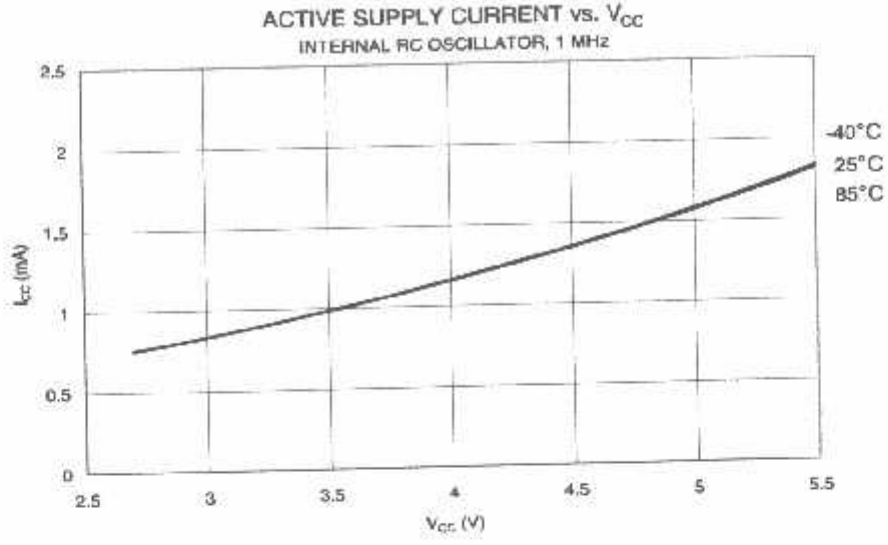
**Figure 133.** Active Supply Current vs.  $V_{CC}$  (Internal RC Oscillator, 4 MHz)



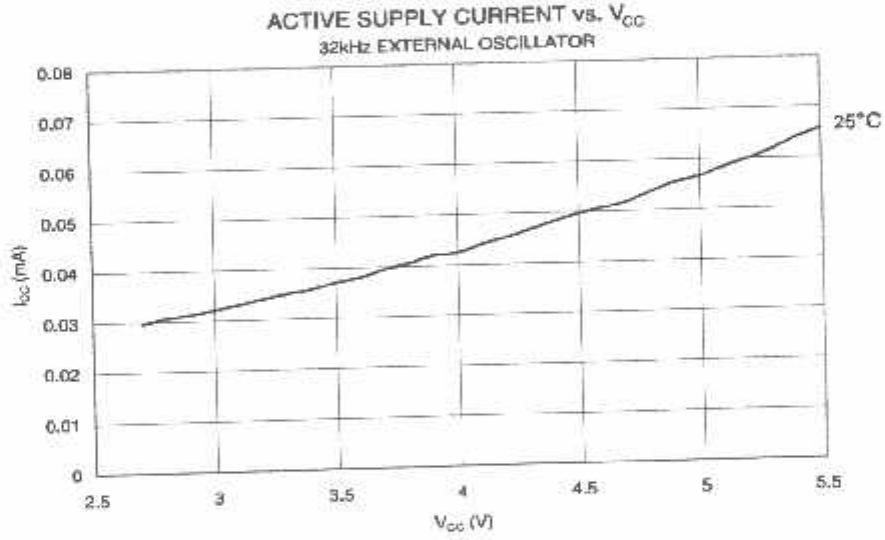
**Figure 134.** Active Supply Current vs.  $V_{CC}$  (Internal RC Oscillator, 2 MHz)



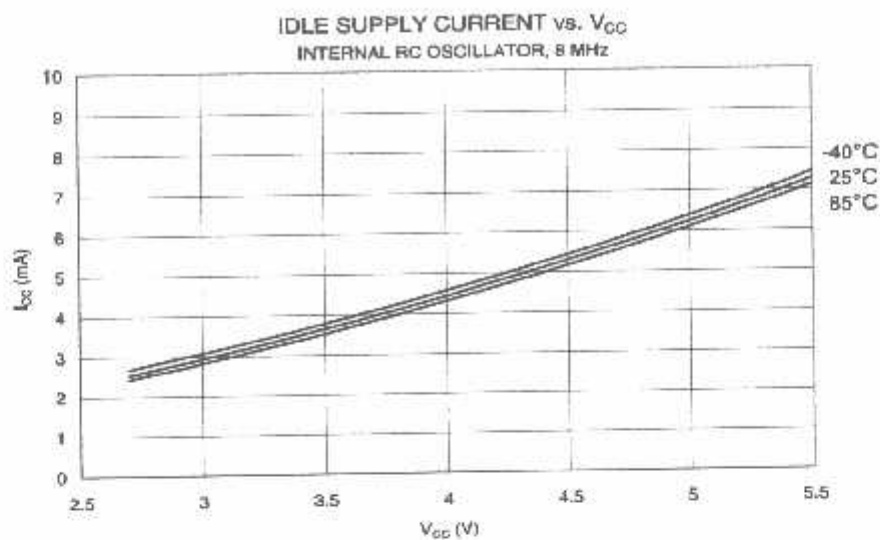
**Figure 135.** Active Supply Current vs.  $V_{CC}$  (Internal RC Oscillator, 1 MHz)



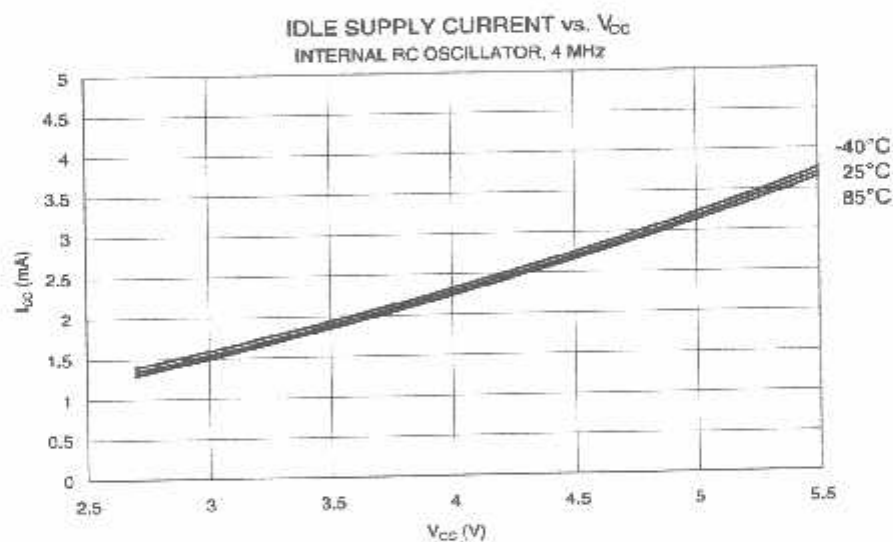
**Figure 136.** Active Supply Current vs.  $V_{CC}$  (32 kHz External Oscillator)



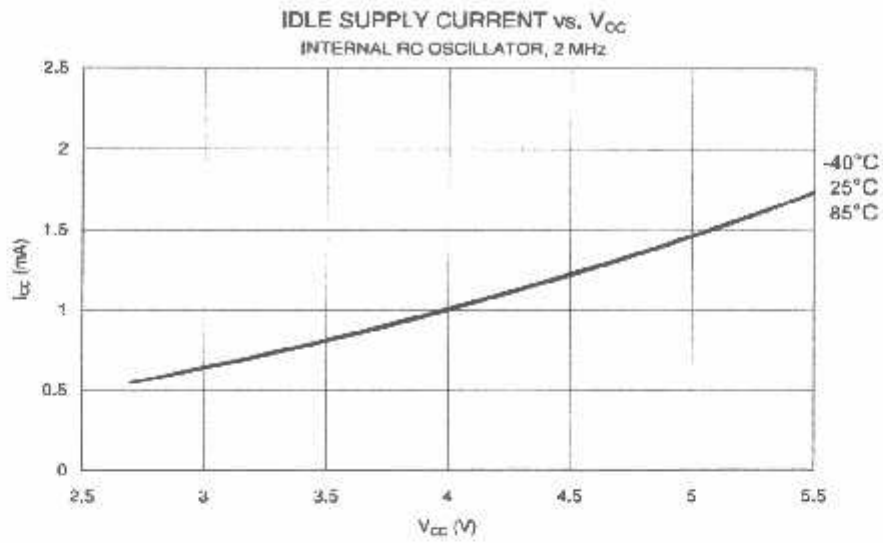
**Figure 139.** Idle Supply Current vs.  $V_{CC}$  (Internal RC Oscillator, 8 MHz)



**Figure 140.** Idle Supply Current vs.  $V_{CC}$  (Internal RC Oscillator, 4 MHz)



**Figure 141.** Idle Supply Current vs.  $V_{CC}$  (Internal RC Oscillator, 2 MHz)



**Figure 142.** Idle Supply Current vs.  $V_{CC}$  (Internal RC Oscillator, 1 MHz)

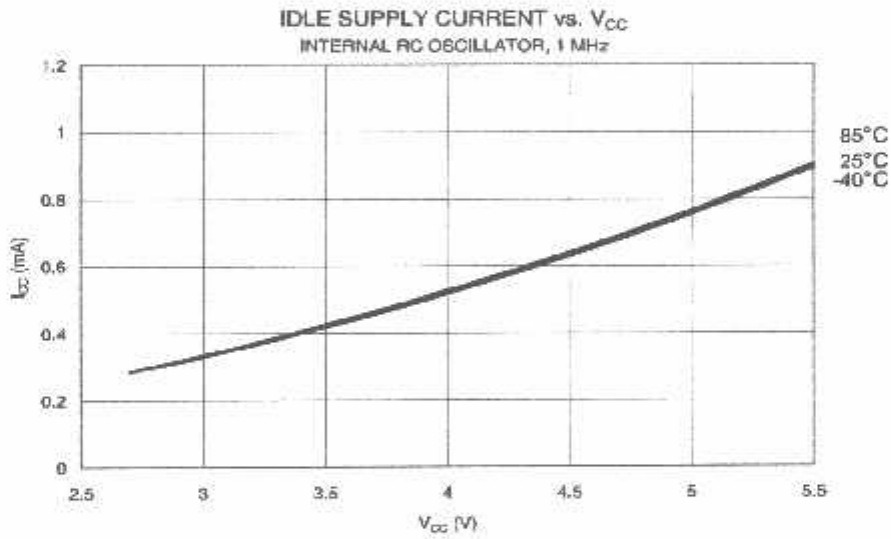
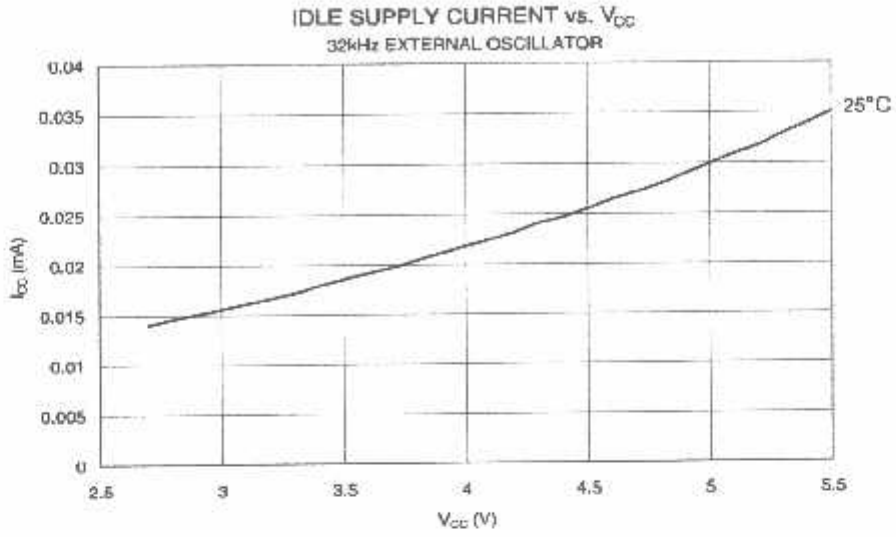
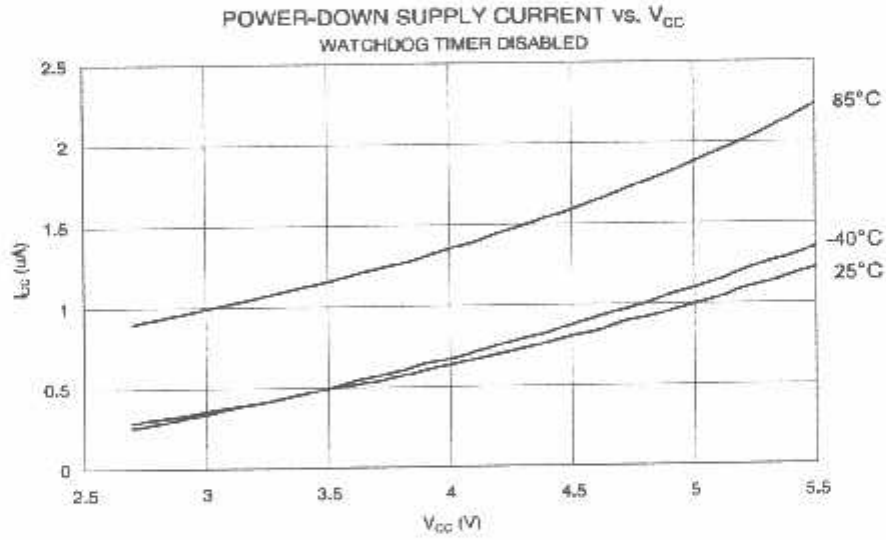


Figure 143. Idle Supply Current vs.  $V_{CC}$  (32 kHz External Oscillator)

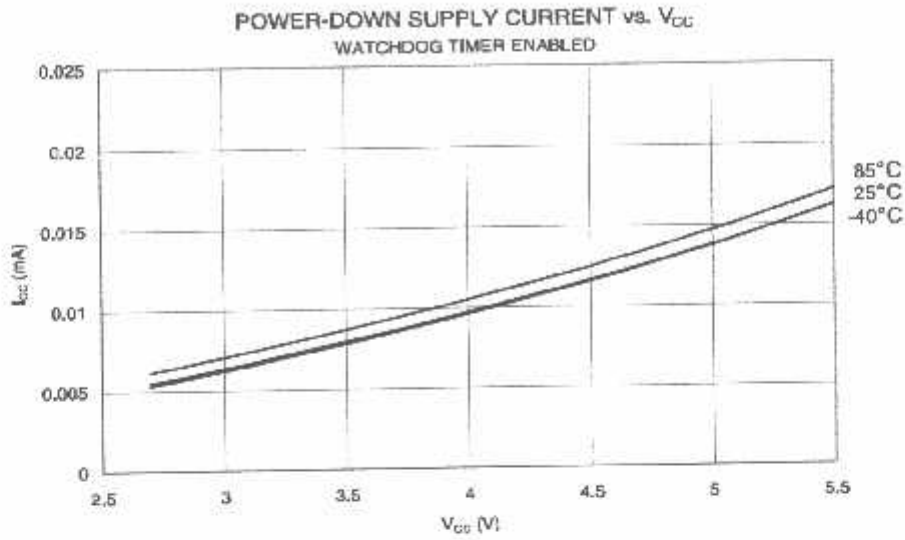


Power-Down Supply Current

Figure 144. Power-Down Supply Current vs.  $V_{CC}$  (Watchdog Timer Disabled)

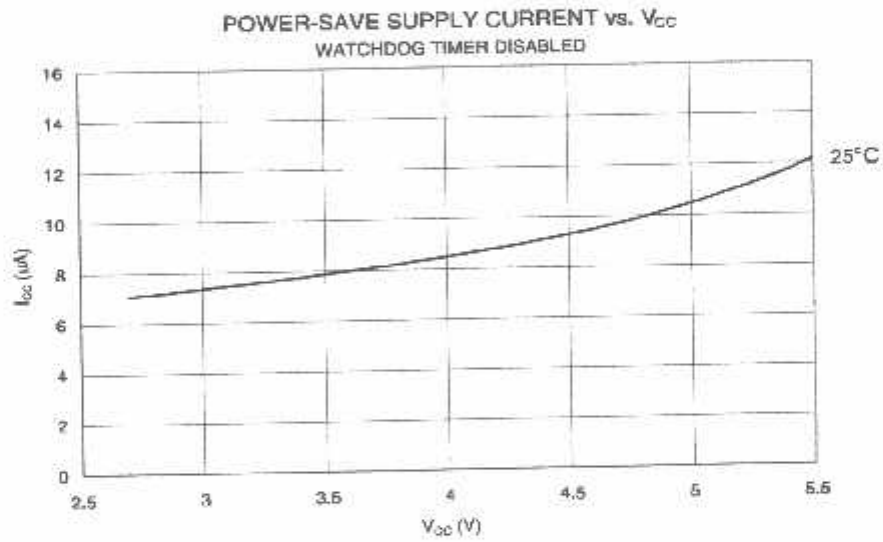


**Figure 145. Power-Down Supply Current vs.  $V_{CC}$  (Watchdog Timer Enabled)**



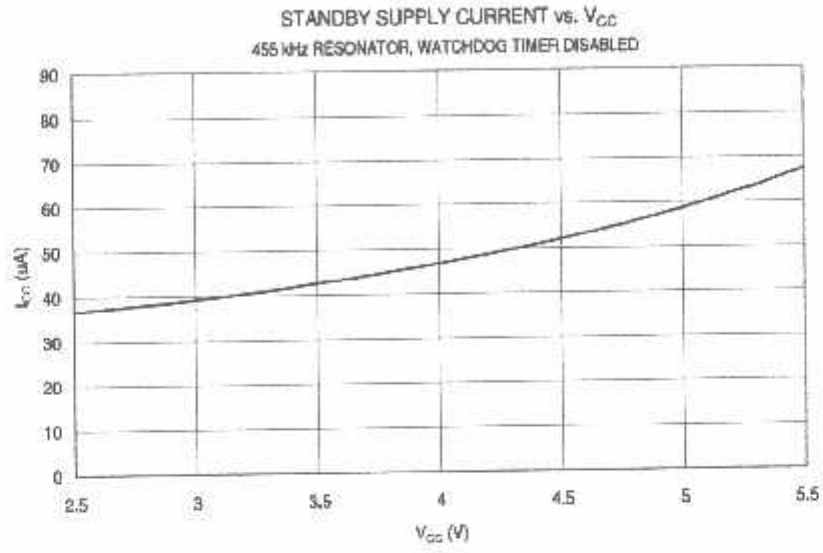
**Power-Save Supply Current**

**Figure 146. Power-Save Supply Current vs.  $V_{CC}$  (Watchdog Timer Disabled)**

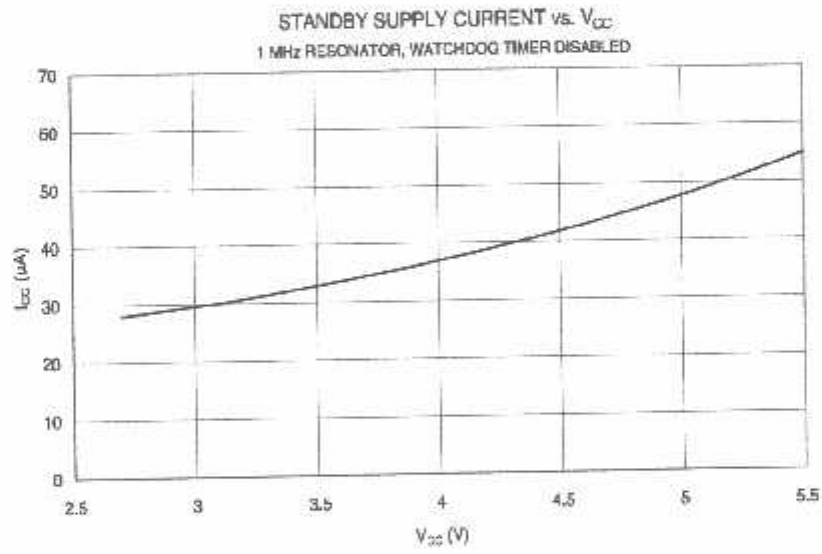


## Standby Supply Current

**Figure 147.** Standby Supply Current vs.  $V_{CC}$  (455 kHz Resonator, Watchdog Timer Disabled)

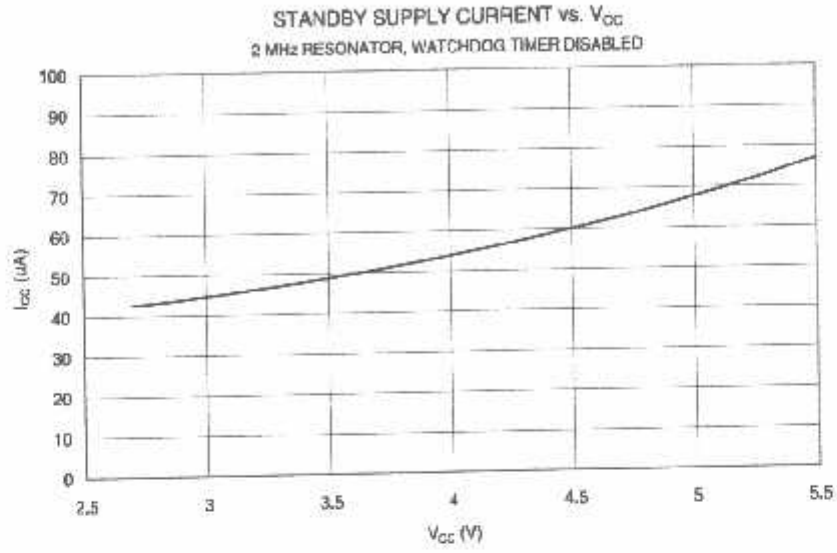


**Figure 148.** Standby Supply Current vs.  $V_{CC}$  (1 MHz Resonator, Watchdog Timer Disabled)

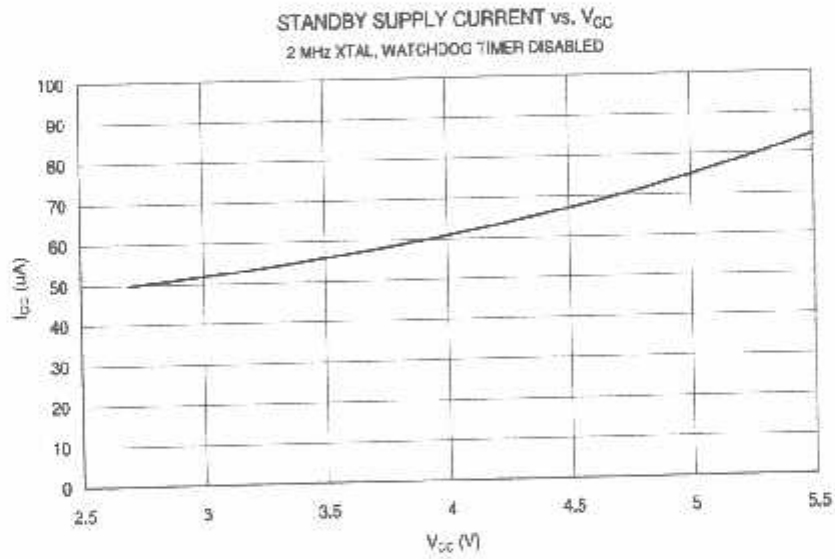




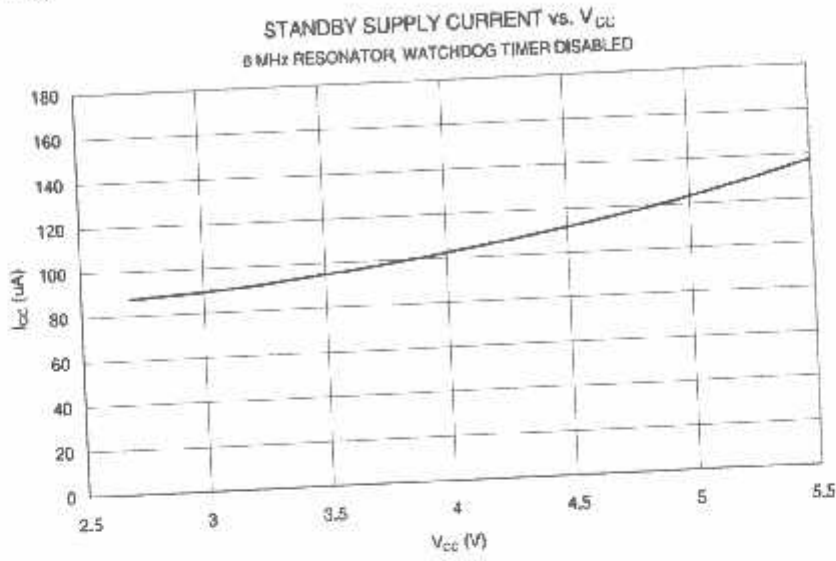
**Figure 149.** Standby Supply Current vs.  $V_{CC}$  (2 MHz Resonator, Watchdog Timer Disabled)



**Figure 150.** Standby Supply Current vs.  $V_{CC}$  (2 MHz Xtal, Watchdog Timer Disabled)



**Figure 153.** Standby Supply Current vs.  $V_{CC}$  (6 MHz Resonator, Watchdog Timer Disabled)



**Figure 154.** Standby Supply Current vs.  $V_{CC}$  (6 MHz Xtal, Watchdog Timer Disabled)

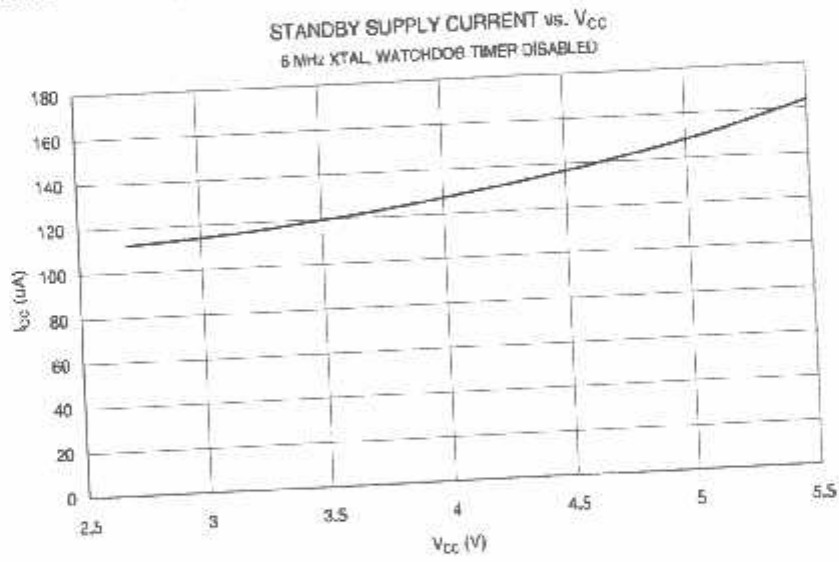


Figure 155. I/O Pin Pull-up Resistor Current vs. Input Voltage ( $V_{CC} = 5V$ )

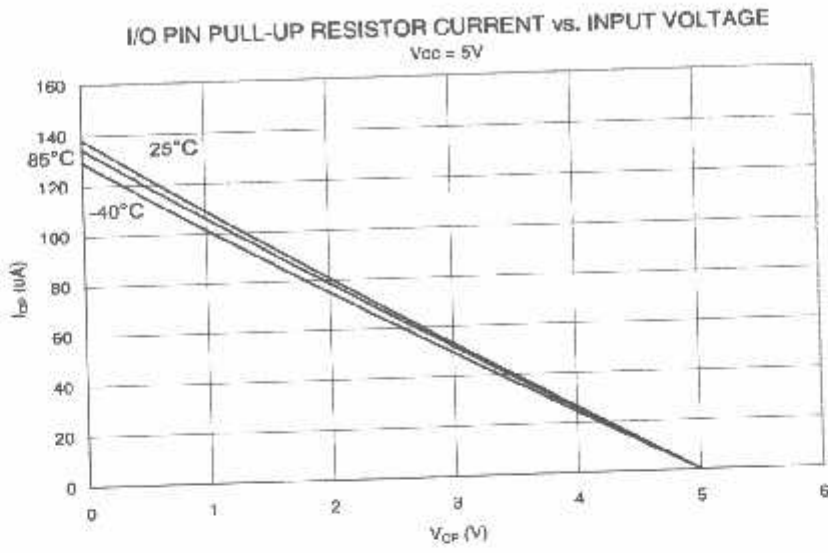
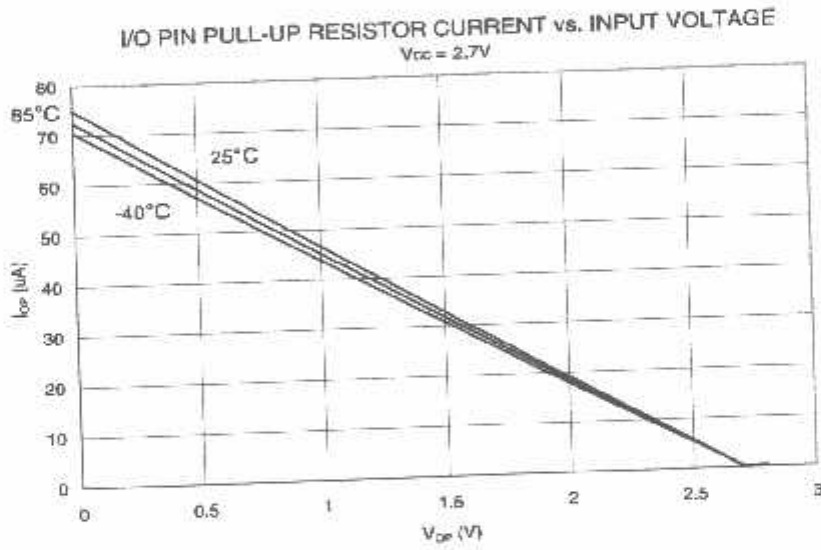
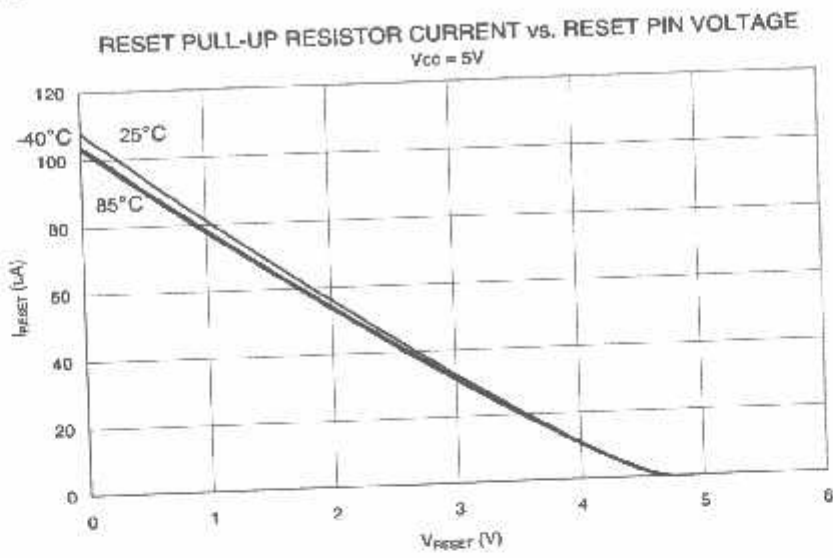


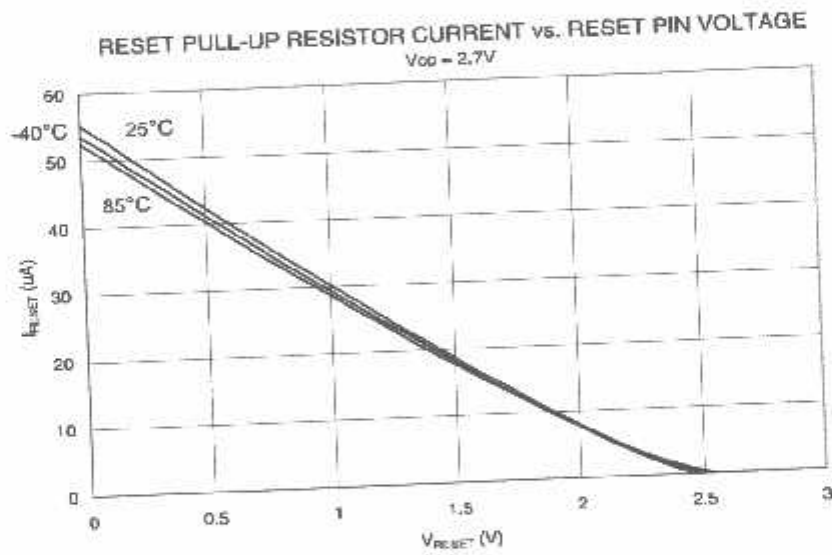
Figure 156. I/O Pin Pull-up Resistor Current vs. Input Voltage ( $V_{CC} = 2.7V$ )



**Figure 157.** Reset Pull-up Resistor Current vs. Reset Pin Voltage ( $V_{CC} = 5V$ )

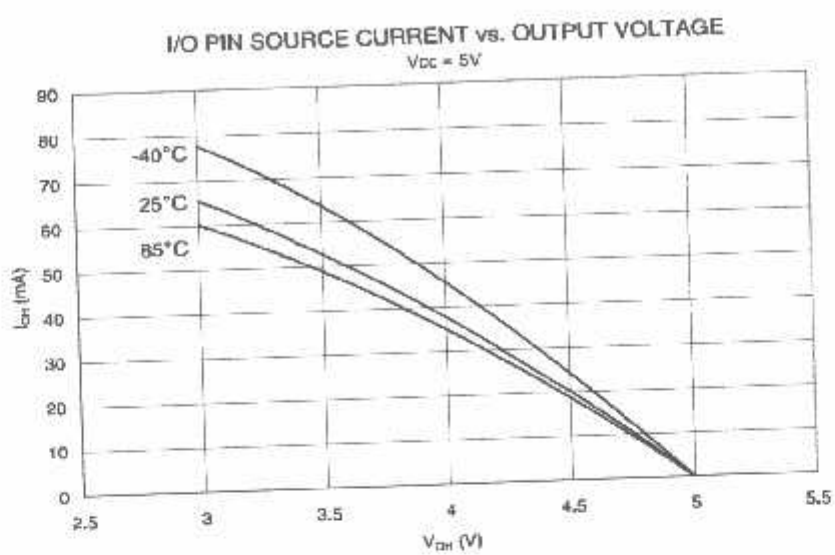


**Figure 158.** Reset Pull-up Resistor Current vs. Reset Pin Voltage ( $V_{CC} = 2.7V$ )

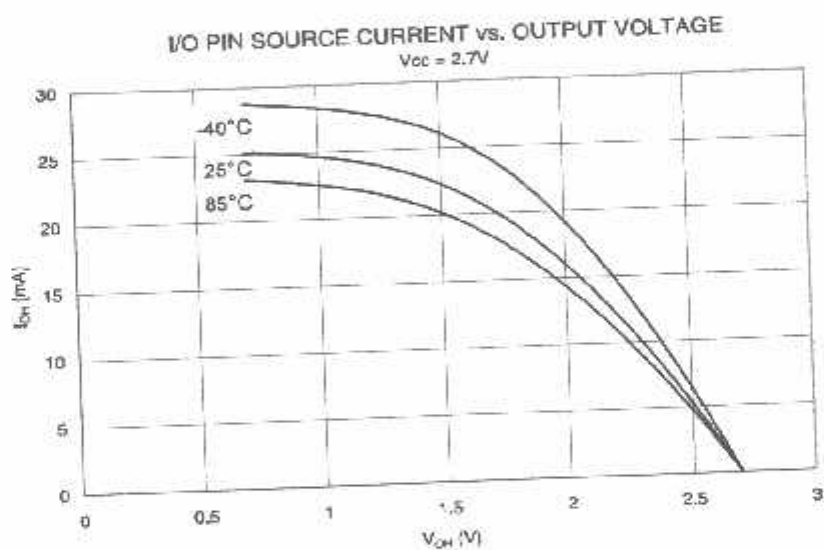


## Driver Strength

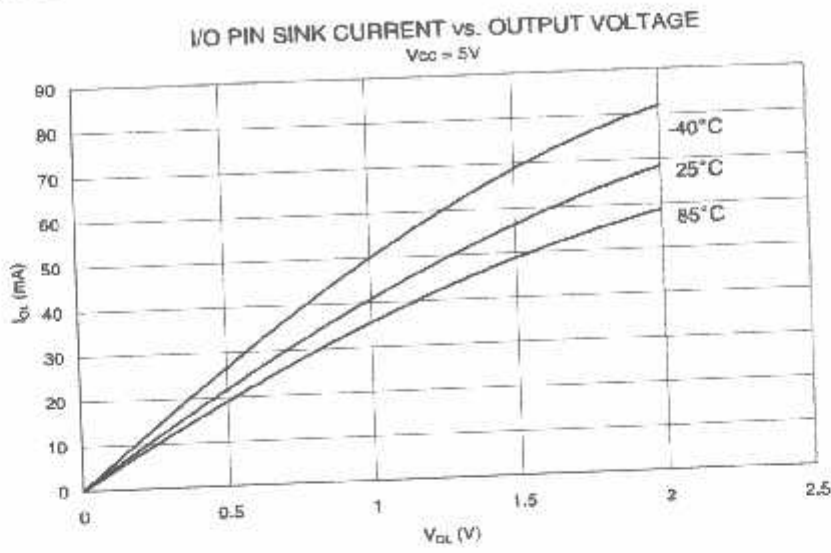
**Figure 159.** I/O Pin Source Current vs. Output Voltage ( $V_{CC} = 5V$ )



**Figure 160.** I/O Pin Source Current vs. Output Voltage ( $V_{CC} = 2.7V$ )



**Figure 161.** I/O Pin Sink Current vs. Output Voltage ( $V_{CC} = 5V$ )



**Figure 162.** I/O Pin Sink Current vs. Output Voltage ( $V_{CC} = 2.7V$ )

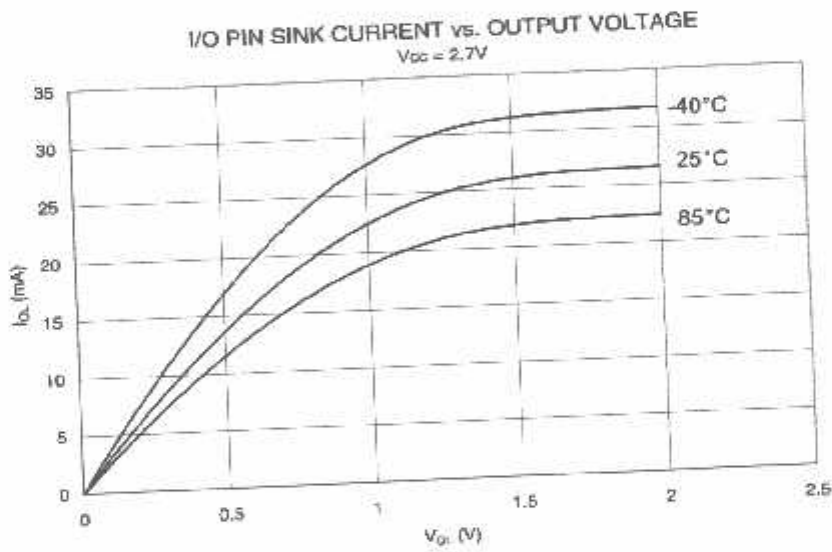


Figure 163. I/O Pin Input Threshold Voltage vs.  $V_{CC}$  ( $V_{IH}$ , I/O Pin Read As '1')

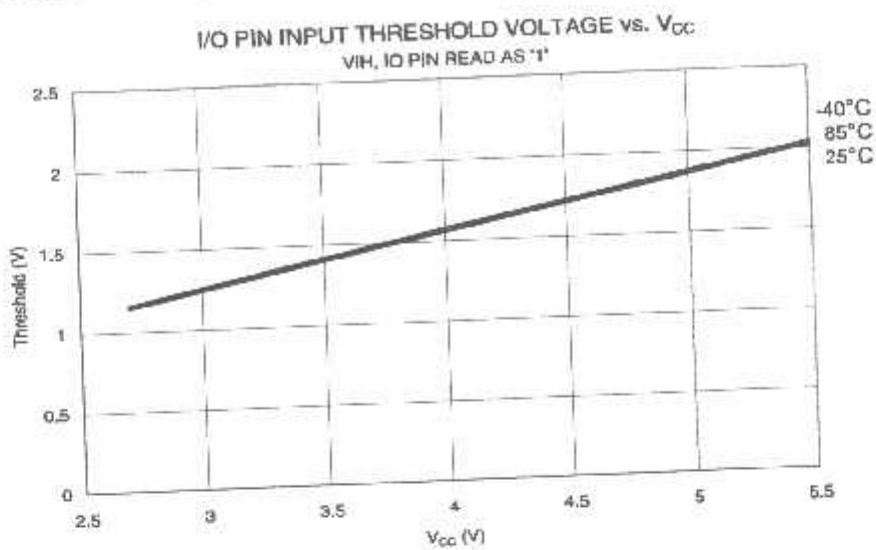


Figure 164. I/O Pin Input Threshold Voltage vs.  $V_{CC}$  ( $V_{IL}$ , I/O Pin Read As '0')

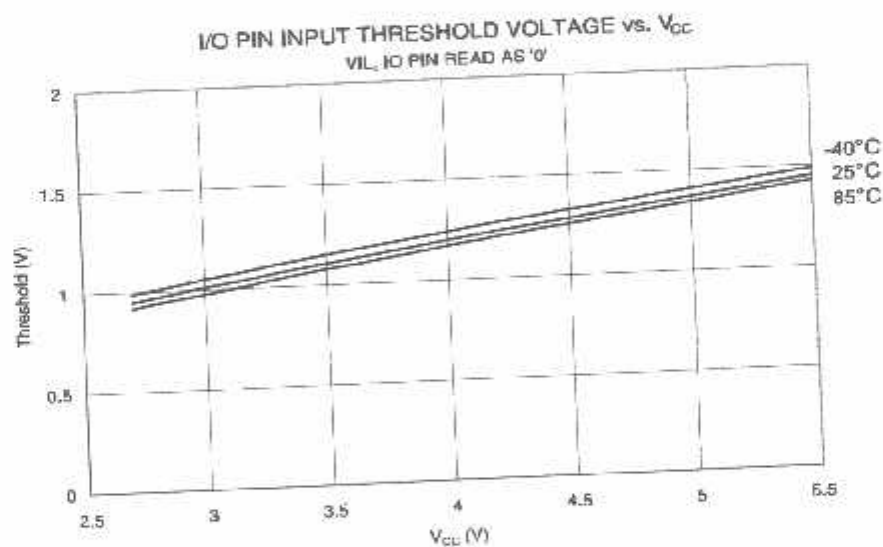


Figure 165. I/O Pin Input Hysteresis vs.  $V_{CC}$

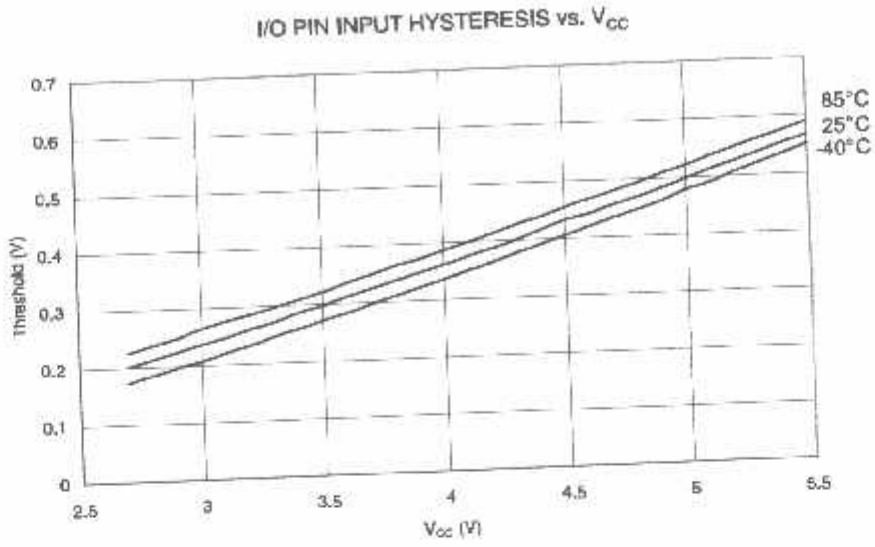


Figure 166. Reset Input Threshold Voltage vs.  $V_{CC}$  ( $V_{IH}$ , Reset Pin Read As '1')

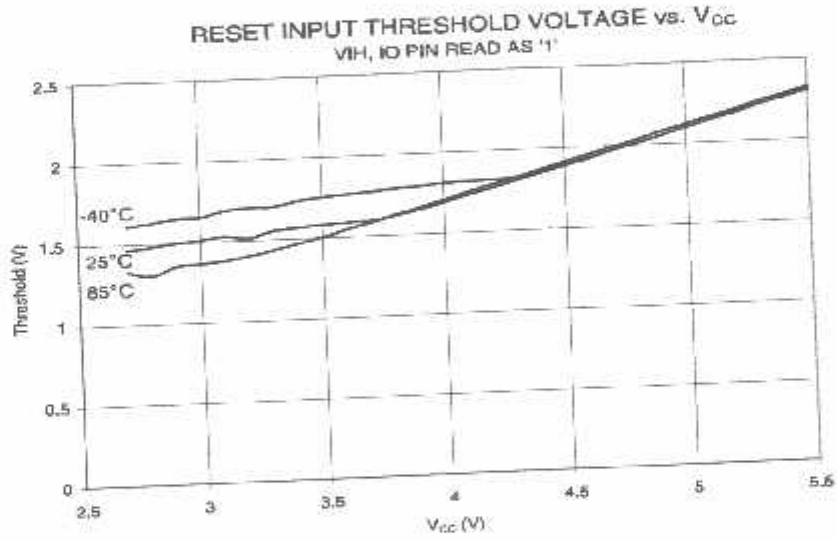




Figure 167. Reset Input Threshold Voltage vs.  $V_{CC}$  ( $V_{IL}$ , Reset Pin Read As '0')

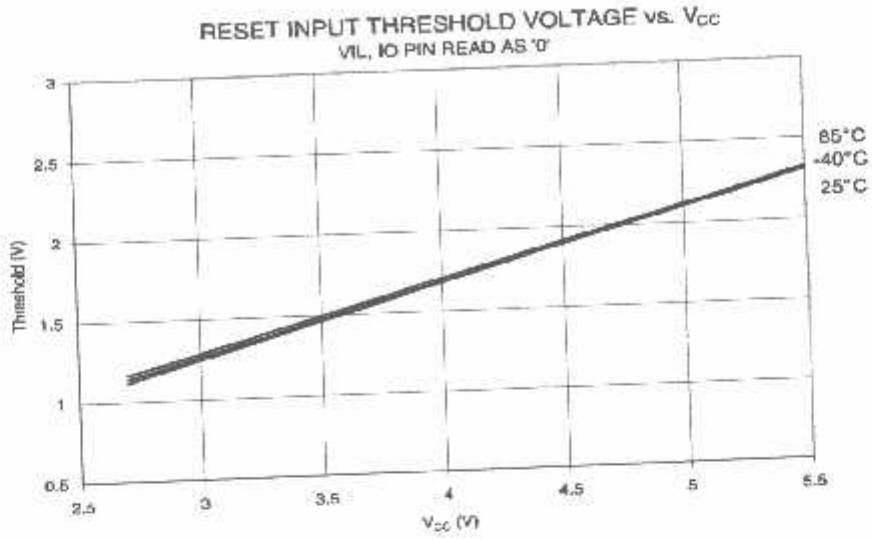


Figure 168. Reset Input Pin Hysteresis vs.  $V_{CC}$

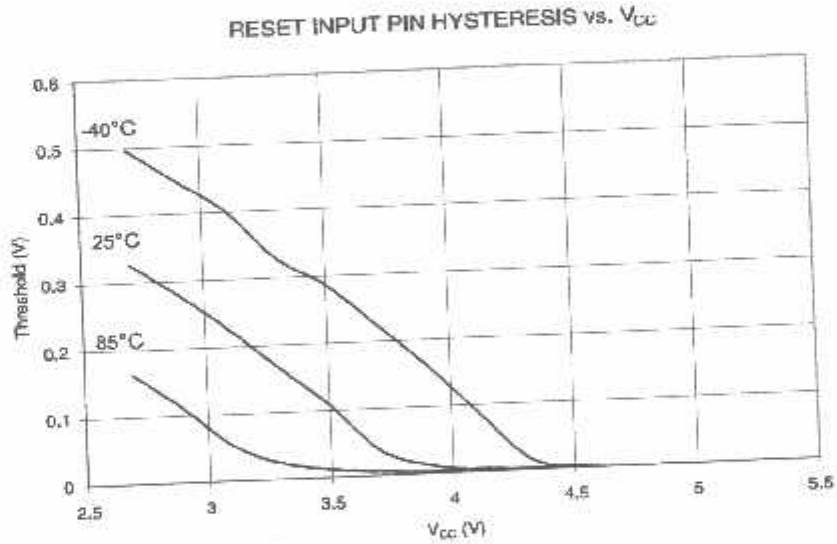


Figure 169. BOD Thresholds vs. Temperature (BOD Level Is 4.0V)

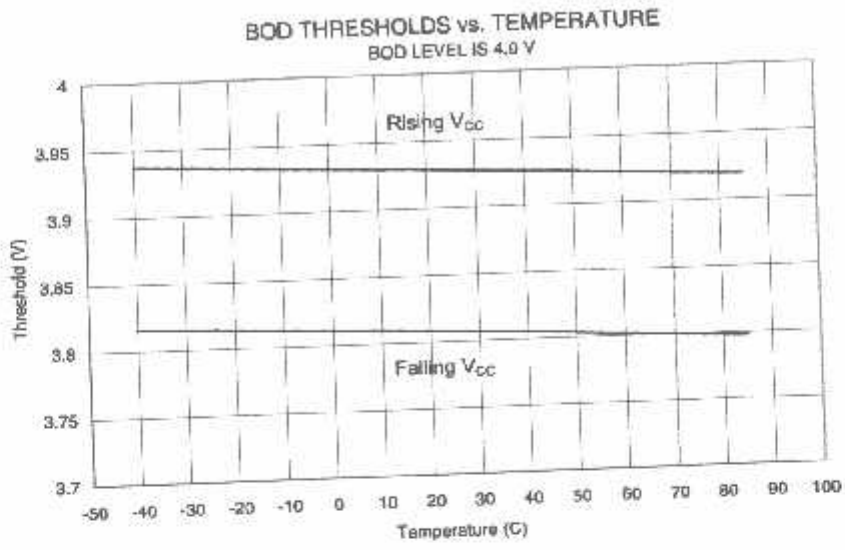


Figure 170. BOD Thresholds vs. Temperature (BOD Level is 2.7V)

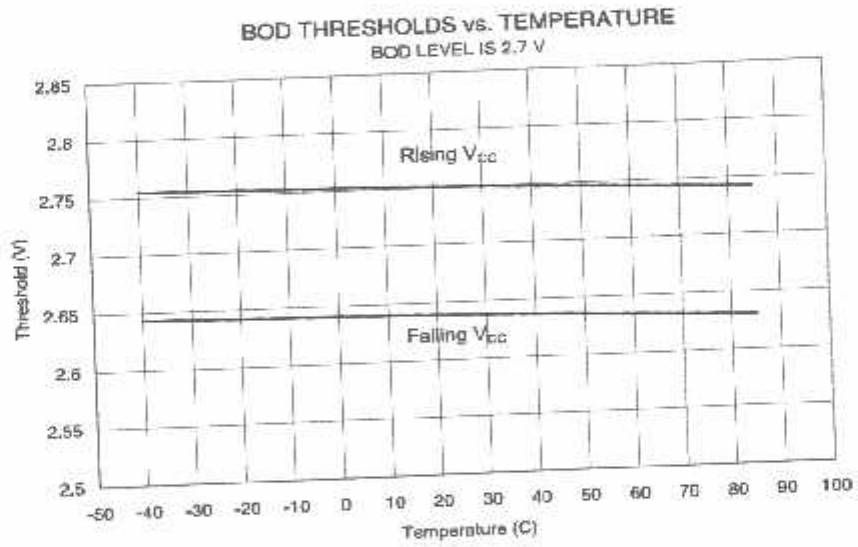


Figure 171. Bandgap Voltage vs.  $V_{CC}$

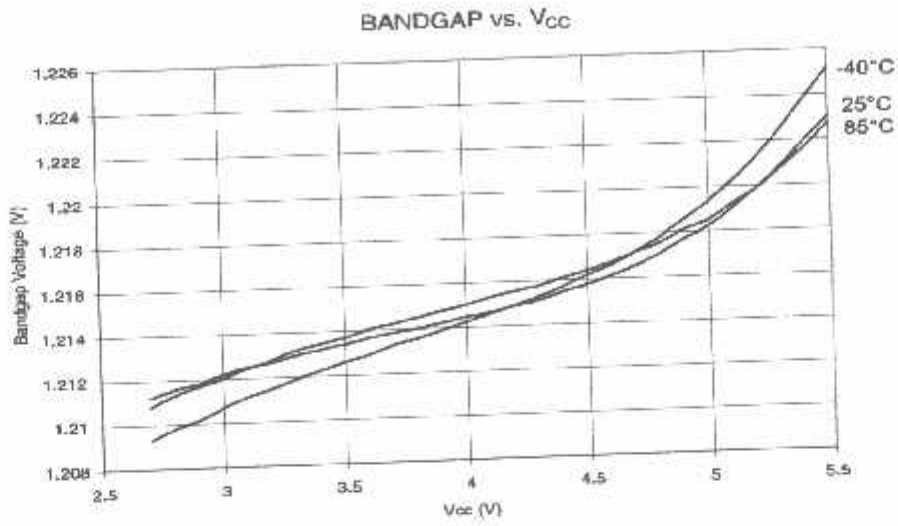
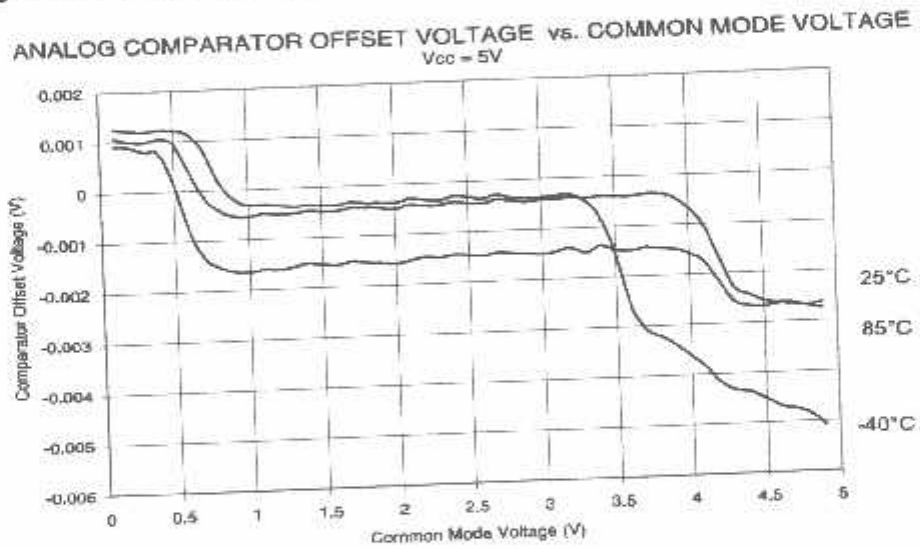
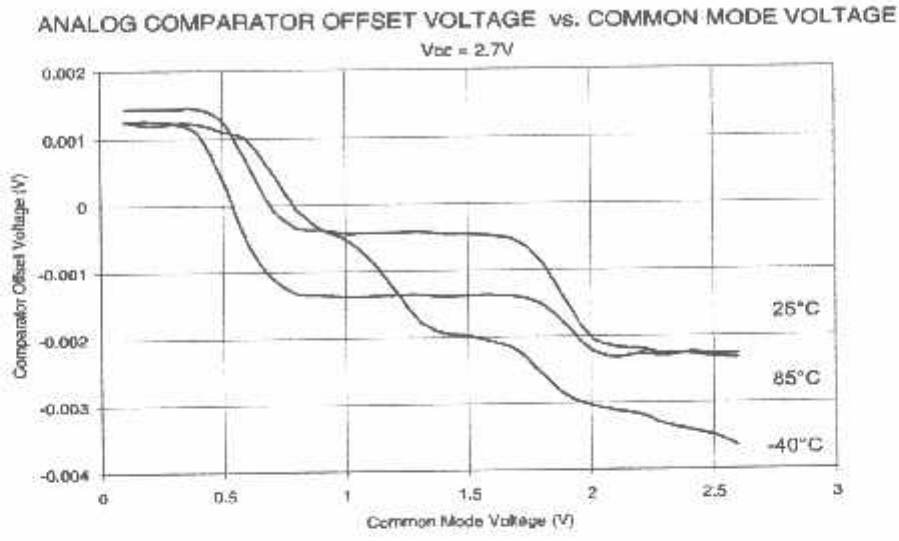


Figure 172. Analog Comparator Offset Voltage vs. Common Mode Voltage ( $V_{CC} = 5V$ )



**Figure 173.** Analog Comparator Offset Voltage vs. Common Mode Voltage ( $V_{CC} = 2.7V$ )



Internal Oscillator Speed

**Figure 174.** Watchdog Oscillator Frequency vs.  $V_{CC}$

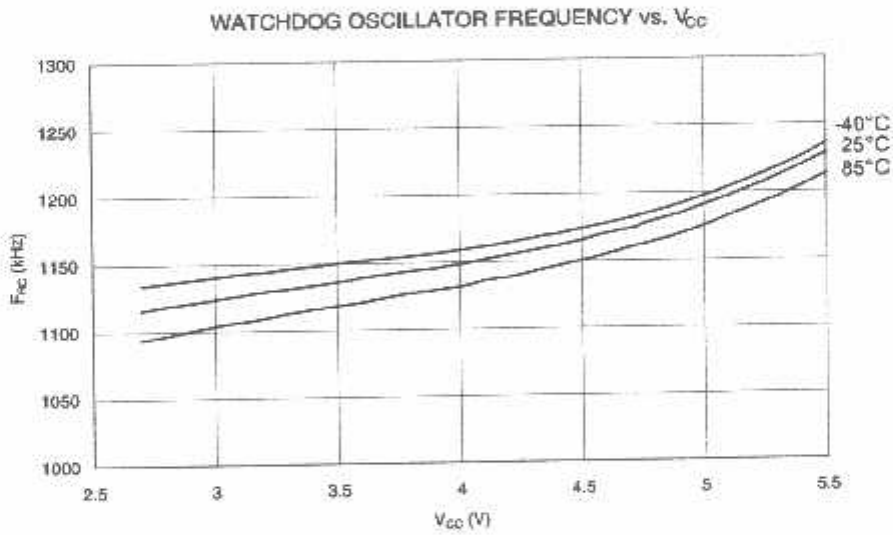


Figure 175. Calibrated 8 MHz RC Oscillator Frequency vs. Temperature

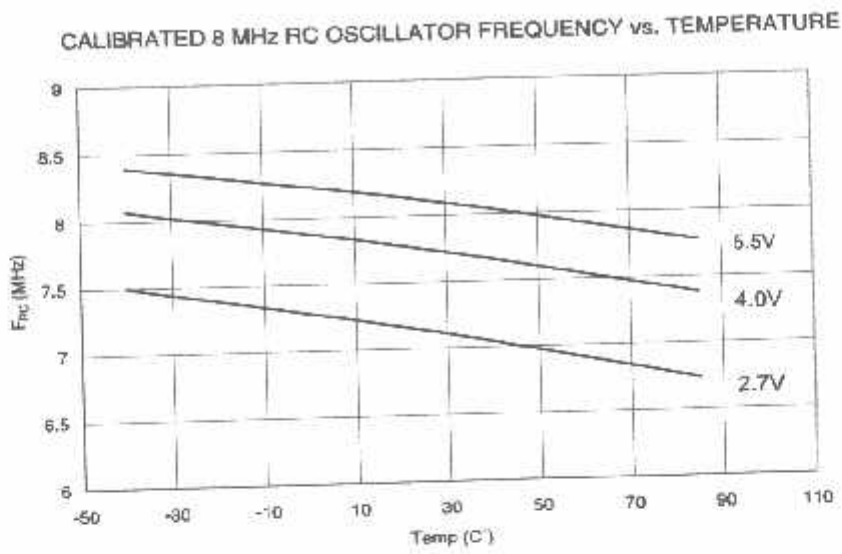
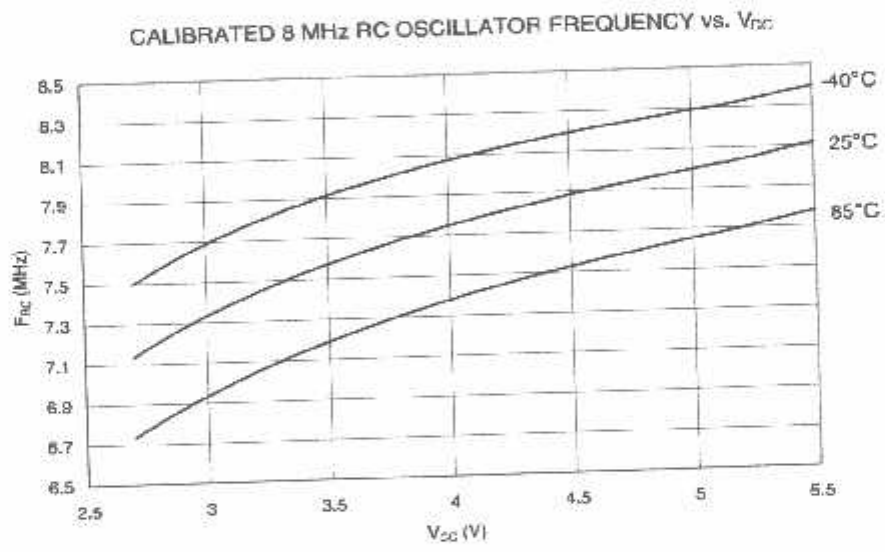
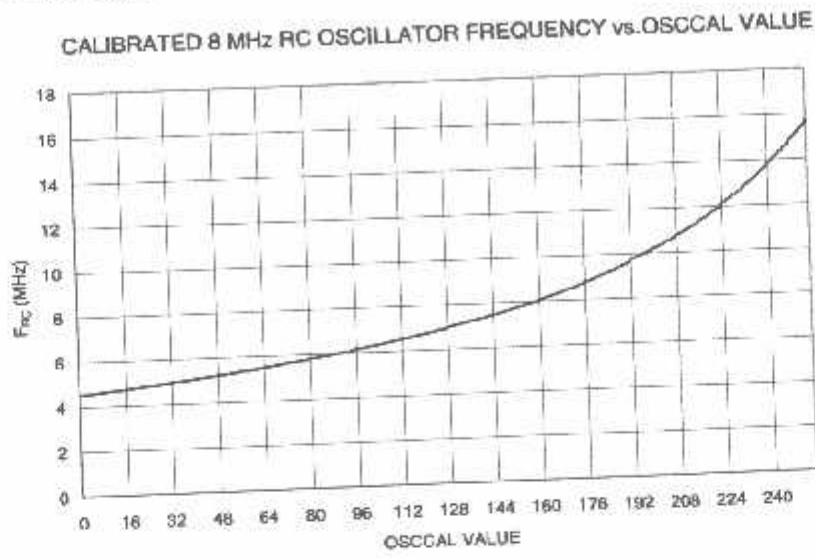


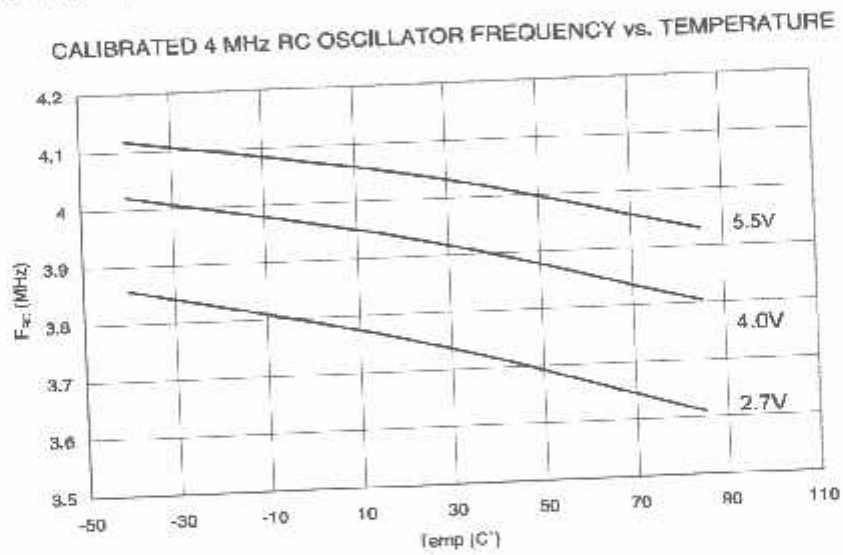
Figure 176. Calibrated 8 MHz RC Oscillator Frequency vs. V<sub>CC</sub>



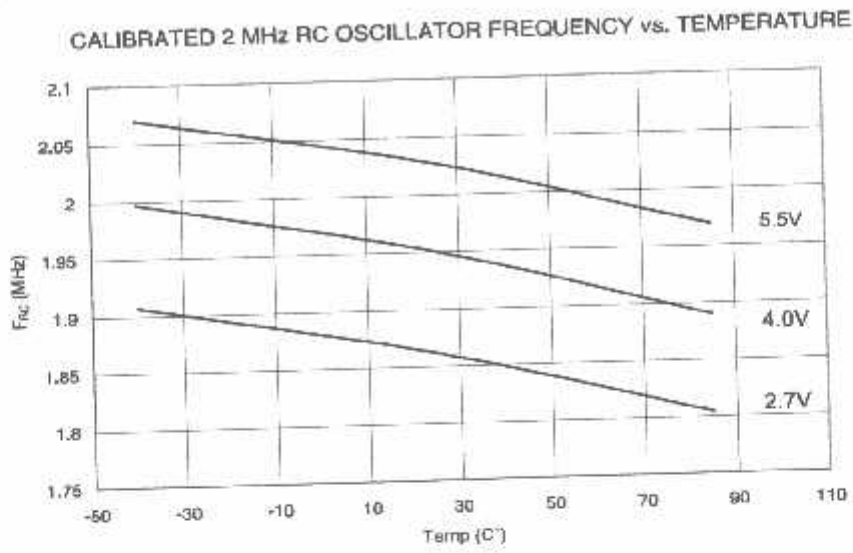
**Figure 177.** Calibrated 8 MHz RC Oscillator Frequency vs. Oscal Value



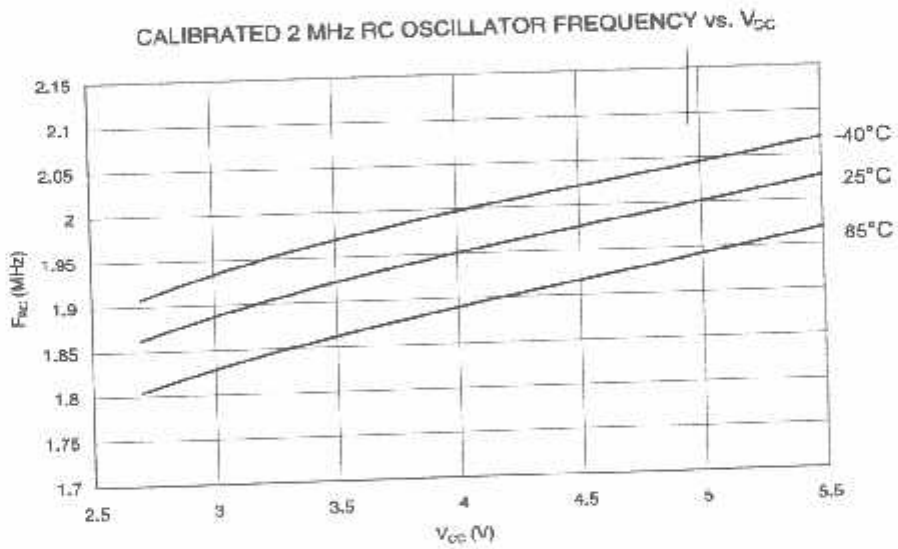
**Figure 178.** Calibrated 4 MHz RC Oscillator Frequency vs. Temperature



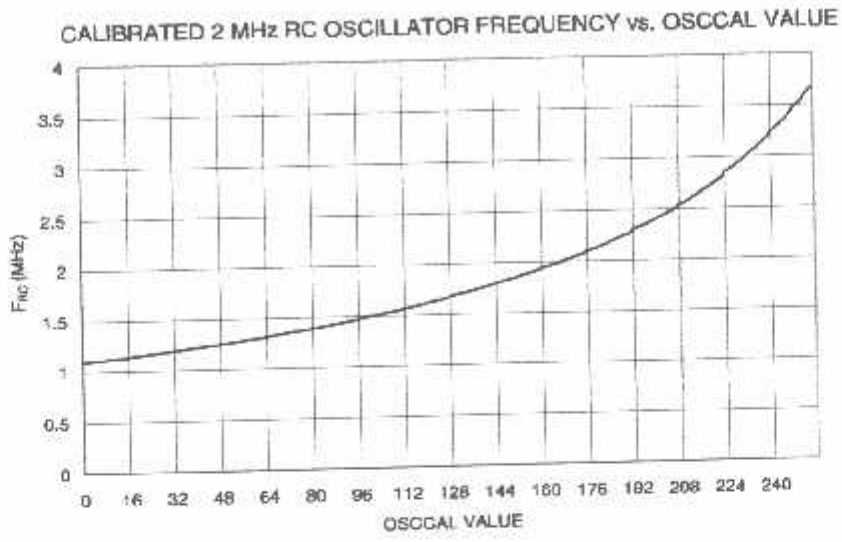
**Figure 181.** Calibrated 2 MHz RC Oscillator Frequency vs. Temperature



**Figure 182.** Calibrated 2 MHz RC Oscillator Frequency vs.  $V_{CC}$



**Figure 183.** Calibrated 2 MHz RC Oscillator Frequency vs. OSCCAL Value



**Figure 184.** Calibrated 1 MHz RC Oscillator Frequency vs. Temperature

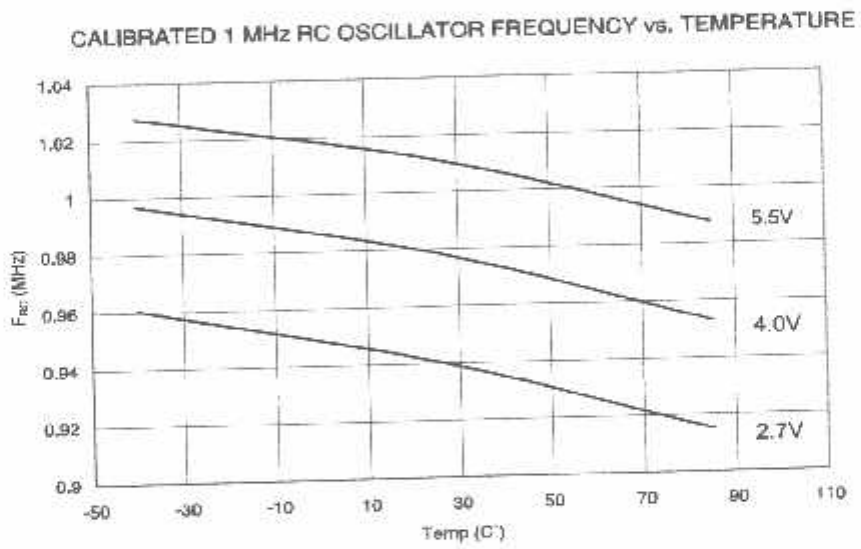






Figure 185. Calibrated 1 MHz RC Oscillator Frequency vs.  $V_{CC}$

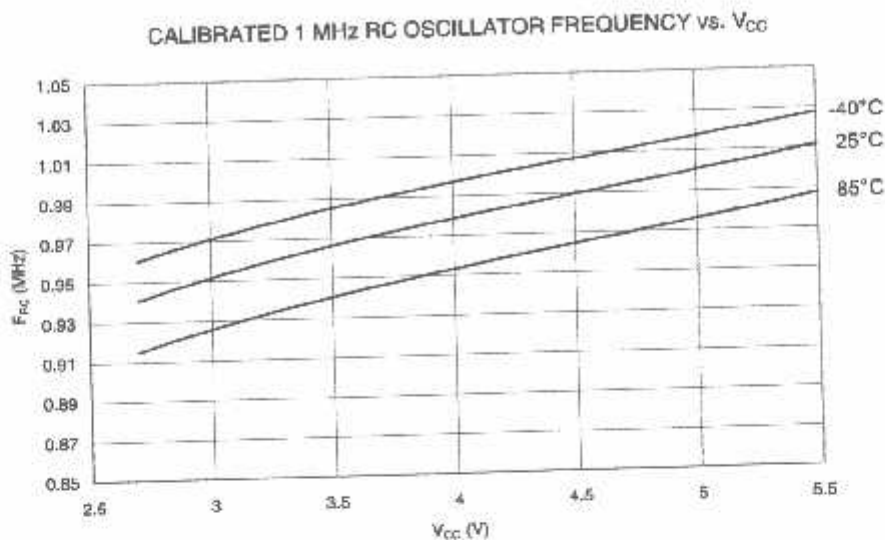


Figure 186. Calibrated 1 MHz RC Oscillator Frequency vs. Oscal Value

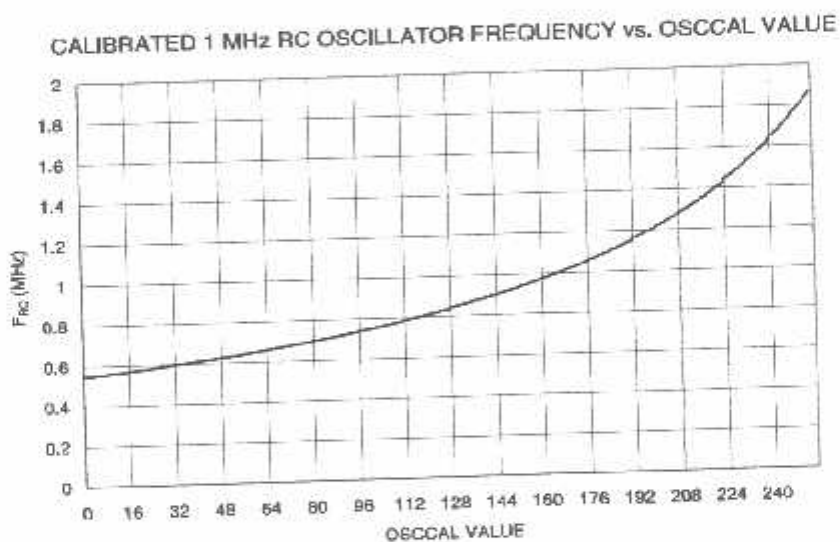


Figure 187. Brownout Detector Current vs.  $V_{CC}$

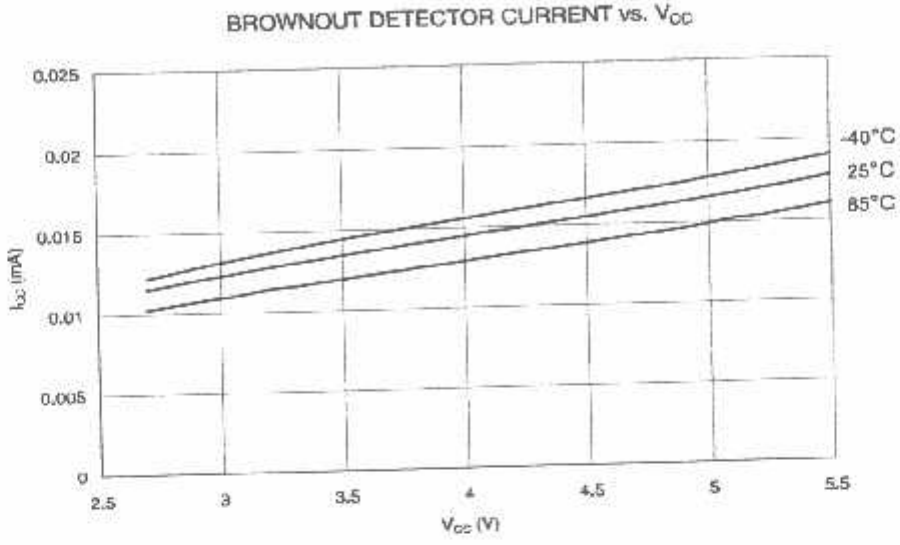


Figure 188. ADC Current vs.  $V_{CC}$  (AREF = AVCC)

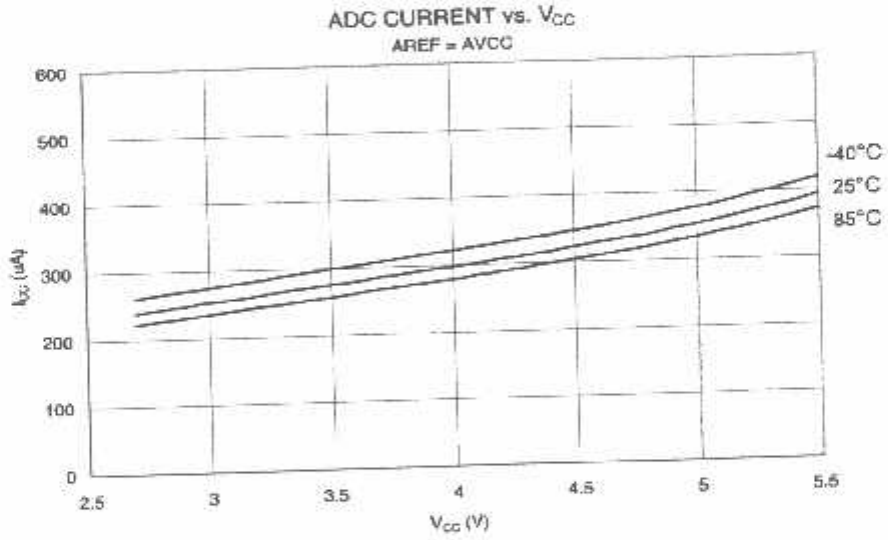


Figure 189. AREF External Reference Current vs.  $V_{CC}$

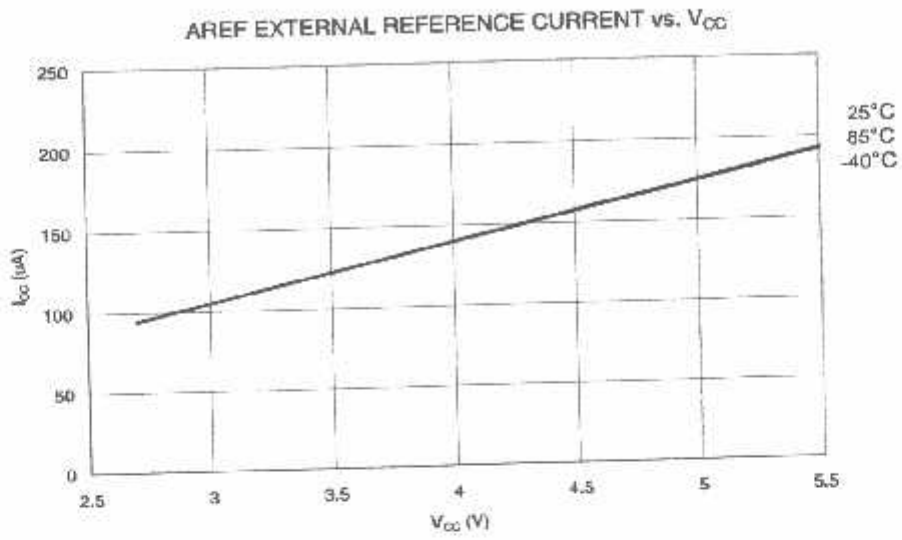
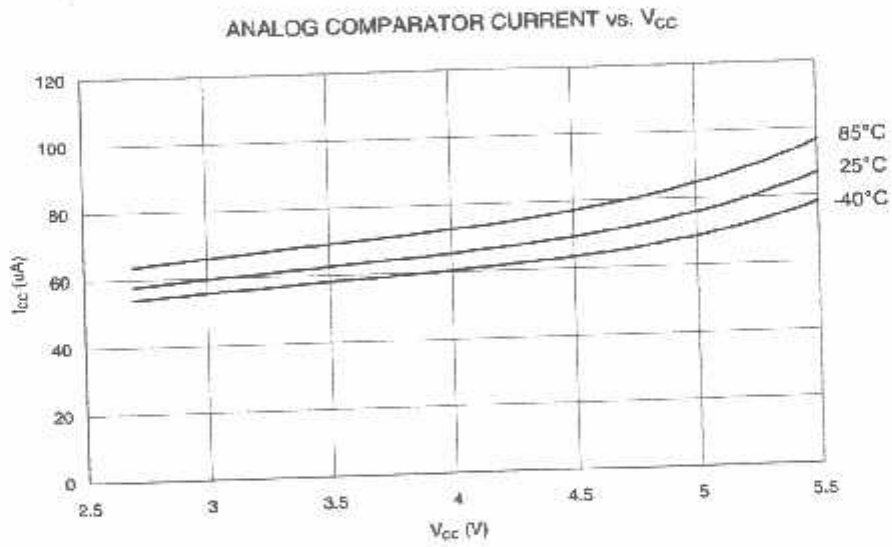
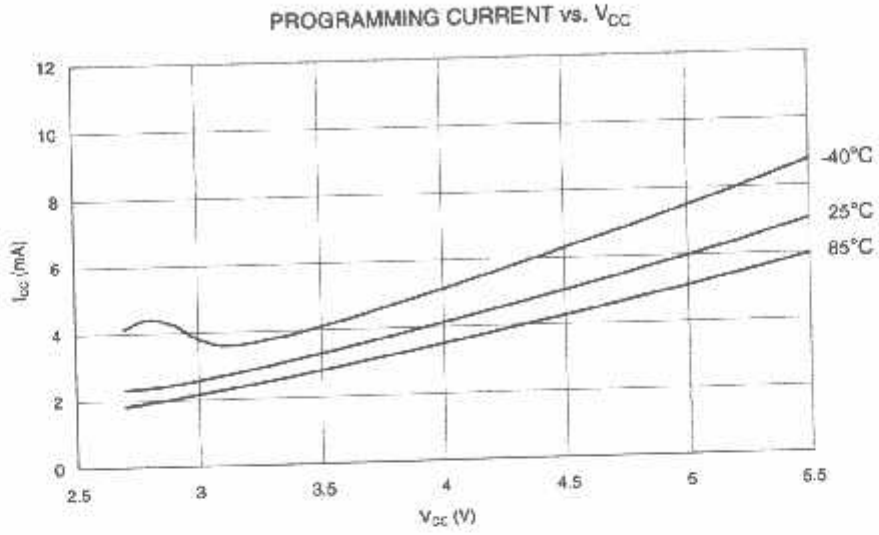


Figure 190. Analog Comparator Current vs.  $V_{CC}$



**Figure 191.** Programming Current vs.  $V_{CC}$



urrent Consumption In  
set And Reset Pulsewidth

**Figure 192.** Reset Supply Current vs.  $V_{CC}$  (0.1 - 1.0 MHz, Excluding Current Through The Reset Pull-up)

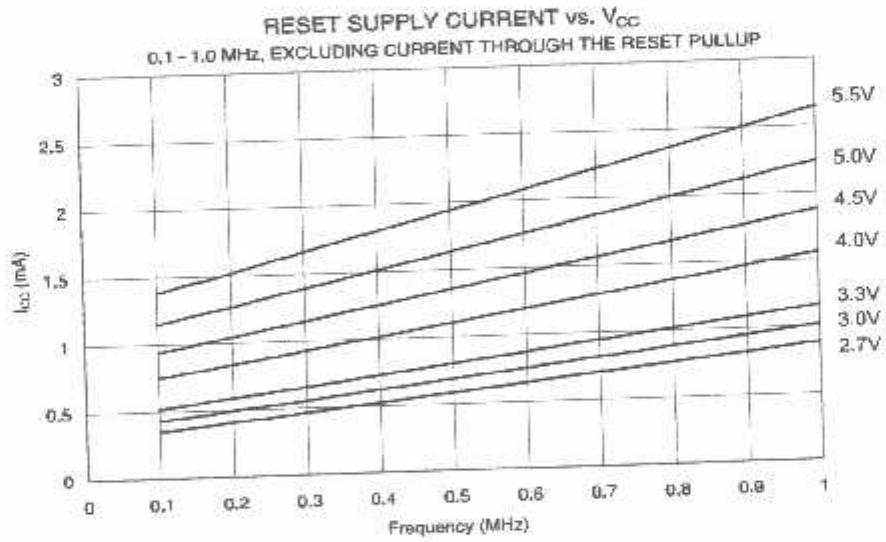




Figure 193. Reset Supply Current vs.  $V_{CC}$  (1 - 20 MHz, Excluding Current Through The Reset Pull-up)

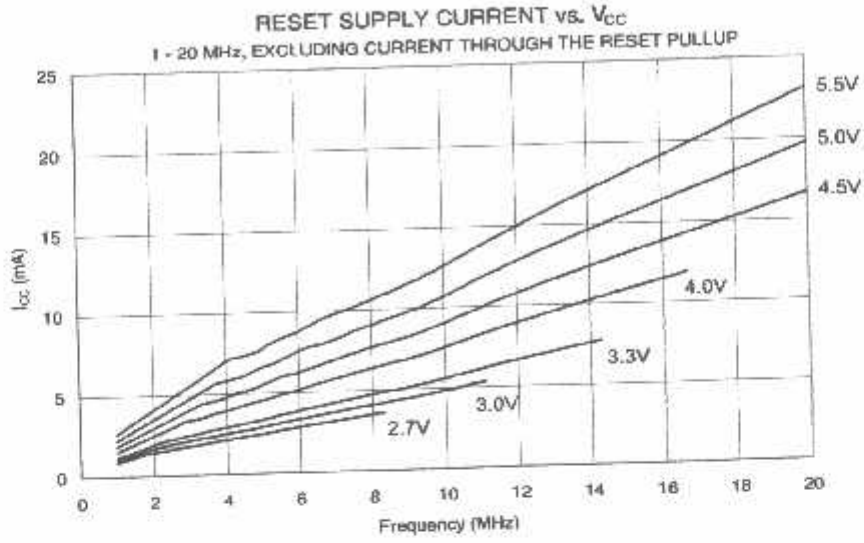
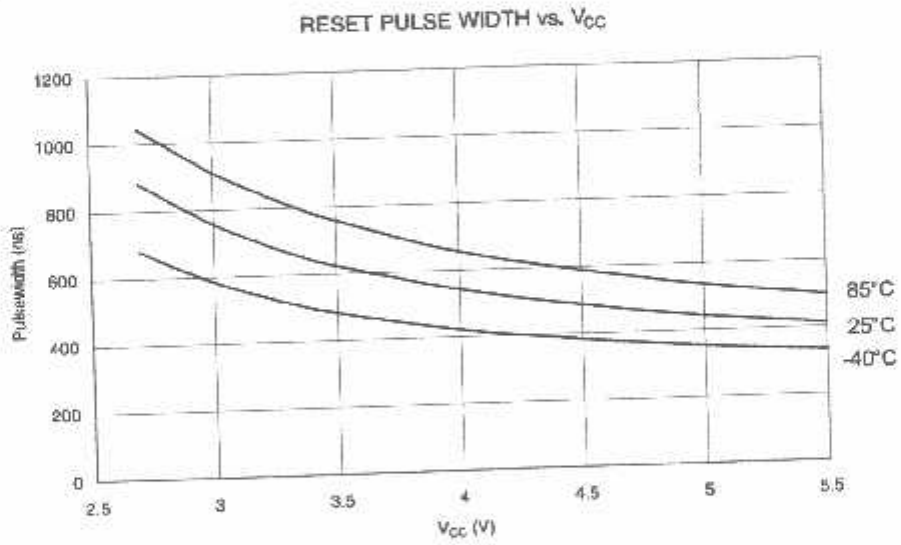


Figure 194. Reset Pulse Width vs.  $V_{CC}$



## Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0F (0x0F)	SREG	I	T	H	S	V	N	Z	C	9
3E (0x3E)	BPH	-	-	-	-	-	-	SP9	SP8	11
3D (0x3D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	11
2C (0x2C)	OCR0	Timer/Counter0 Output Compare Register								48, 68
3B (0x3B)	GICR	INT1	INT0	INT2	-	-	-	IVSEL	WCE	68
3A (0x3A)	GIFR	INTF1	INTF0	INTF2	-	-	-	-	-	68
39 (0x39)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	84, 114, 132
38 (0x38)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	85, 115, 133
37 (0x37)	SPMCR	SPMIE	RWWBRR	-	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	226
36 (0x36)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE	173
35 (0x35)	MCUCR	BM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	31, 67
34 (0x34)	MCUCSR	-	ISC2	-	-	WDPF	BORF	EXTRF	PORF	38, 68
33 (0x33)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	82
32 (0x32)	TCNT0	Timer/Counter0 (8 Bits)								84
31 (0x31)	OBSCAL	Oscillator Calibration Register								29
30 (0x30)	SPOR	ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10	58, 87, 104, 201, 221
2F (0x2F)	TCCR1A	COM1A1	OCM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	109
2E (0x2E)	TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	112
2D (0x2D)	TCNT1H	Timer/Counter1 - Counter Register High Byte								113
2C (0x2C)	TCNT1L	Timer/Counter1 - Counter Register Low Byte								113
2B (0x2B)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte								113
2A (0x2A)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte								113
29 (0x29)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte								113
28 (0x28)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte								113
27 (0x27)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								113
26 (0x26)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								113
25 (0x25)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	129
24 (0x24)	TCNT2	Timer/Counter2 (8 Bits)								130
23 (0x23)	OCR2	Timer/Counter2 Output Compare Register								130
22 (0x22)	ASSR	-	-	-	-	AS2	TCN2UB	OCR2UB	TCR2UB	41
21 (0x21)	WDTCR	-	-	-	-	WDCR	WDF2	WDF1	WDF0	107
20 (0x20)	UBRRH	URSEL	-	-	-	-	UBRR[11:8]		UCPOL	186
1F (0x1F)	UCSRB	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	EEAR8	18
1E (0x1E)	EEARH	-	-	-	-	-	-	-	-	18
1D (0x1D)	EEARL	EEPROM Address Register Low Byte								18
1C (0x1C)	EEDR	EEPROM Data Register								18
1B (0x1B)	EEDR	-	-	-	-	EERIE	EEMWE	EEWE	EERE	18
1A (0x1A)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	65
19 (0x19)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	65
18 (0x18)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	65
17 (0x17)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	65
16 (0x16)	DDRB	DDR7	DDR6	DDR5	DDR4	DDR3	DDR2	DDR1	DDR0	65
15 (0x15)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	65
14 (0x14)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	66
13 (0x13)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	66
12 (0x12)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	66
11 (0x11)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	66
10 (0x10)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	66
0F (0x0F)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	141
0E (0x0E)	SPDR	SPI Data Register								141
0D (0x0D)	SPSR	SPIF	WCOL	-	-	-	-	-	SP2X	139
0C (0x0C)	SPCR	SPIE	SPE	DORF	MSTR	CPOL	CPHA	SPR1	SPR0	139
0B (0x0B)	UDR	USART I/O Data Register								163
0A (0x0A)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	164
09 (0x09)	UCSRB	RXCIE	TXCIE	UDRIF	RXEN	TXEN	UCS22	RXB8	TXB8	167
08 (0x08)	UBRRL	USART Baud Rate Register Low Byte								201
07 (0x07)	ACSR	ACD	ACBG	ACO	AC	ACIE	ACIC	ACIS1	ACIS0	217
06 (0x06)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	218
05 (0x05)	ADCSRA	ADEN	ADSC	ADIF	ADIF	ADIF	ADPS2	ADPS1	ADPS0	220
04 (0x04)	ADCH	ADC Data Register High Byte								220
03 (0x03)	ADCL	ADC Data Register Low Byte								181
02 (0x02)	TWDR	Two-Wire Serial Interface Data Register								181
01 (0x01)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	181
00 (0x00)	TWBR	TWB6	TWB5	TWB4	TWB3	TWB2	-	TWP61	TWP60	181





## Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
00 (0x20)	TWBRH	Two-wire Serial Interface Bit Rate Register								179

- Notes:
1. Refer to the USART description for details on how to access UBRRH and UCSRC.
  2. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
  3. Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

Instruction Set Summary

Mnemonic	Operands	Description	Operation	Flags	#Clocks
<b>ARITHMETIC AND LOGIC INSTRUCTIONS</b>					
ADD	Rd, Rr	Add two Registers	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
ADDC	Rd, Rr	Add with Carry two Registers	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
ADIW	Rd,K	Add Immediate to Word	$RdH:RdL \leftarrow RdH:RdL + K$	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	$Rd \leftarrow Rd - Rr$	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	$Rd \leftarrow Rd - K$	Z,C,N,V,H	1
SUBC	Rd, Rr	Subtract with Carry two Registers	$Rd \leftarrow Rd - Rr - C$	Z,C,N,V,H	1
SUBI_C	Rd, K	Subtract with Carry Constant from Reg.	$Rd \leftarrow Rd - K - C$	Z,C,N,V,S	2
SUBW	Rd,K	Subtract immediate from Word	$RdH:RdL \leftarrow RdH:RdL - K$	Z,N,V	1
AND	Rd, Rr	Logical AND Registers	$Rd \leftarrow Rd \& Rr$	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	$Rd \leftarrow Rd \& K$	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	$Rd \leftarrow Rd \vee Rr$	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	$Rd \leftarrow Rd \vee K$	Z,N,V	1
EXOR	Rd, Rr	Exclusive OR Registers	$Rd \leftarrow Rd \oplus Rr$	Z,N,V	1
COM	Rd	One's Complement	$Rd \leftarrow 0xFF - Rd$	Z,C,N,V,H	1
NEG	Rd	Two's Complement	$Rd \leftarrow 0x00 - Rd$	Z,C,N,V,H	1
ANDK	Rd,K	Set BIt(s) in Register	$Rd \leftarrow Rd \vee K$	Z,N,V	1
ANDK	Rd,K	Clear BIt(s) in Register	$Rd \leftarrow Rd \& (0xFF - K)$	Z,N,V	1
INC	Rd	Increment	$Rd \leftarrow Rd + 1$	Z,N,V	1
DEC	Rd	Decrement	$Rd \leftarrow Rd - 1$	Z,N,V	1
BR	Rd	Test for Zero or Minus	$Rd \leftarrow Rd \& Rd$	Z,N,V	1
CLR	Rd	Clear Register	$Rd \leftarrow 0xFF$	None	1
SET	Rd	Set Register	$Rd \leftarrow 0xFF$	Z,C	2
MUL	Rd, Rr	Multiply Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
LS	Rd, Rr	Multiply Signed	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
LSU	Rd, Rr	Multiply Signed with Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
UL	Rd, Rr	Fractional Multiply Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
ULS	Rd, Rr	Fractional Multiply Signed	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
ULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
<b>BRANCH INSTRUCTIONS</b>					
BR	k	Relative Jump	$PC \leftarrow PC + k + 1$	None	2
BRW		Indirect Jump to Z	$PC \leftarrow Z$	None	2
BR		Relative Subroutine Call	$PC \leftarrow PC + k + 1$	None	3
CALL	k	Indirect Call to Z	$PC \leftarrow Z$	None	3
CALL		Subroutine Return	$PC \leftarrow STACK$	None	4
CALL		Interrupt Return	$PC \leftarrow STACK$	I	4
CP	Rd,Rr	Compare, Skip if Equal	$if (Rd = Rr) PC \leftarrow PC + 2 \text{ or } 3$	None	1 / 2 / 3
CP	Rd,Rr	Compare	$Rd - Rr$	Z, N, V, C, H	1
CP	Rd,Rr	Compare with Carry	$Rd - Rr - C$	Z, N, V, C, H	1
CP	Rd,K	Compare Register with Immediate	$Rd - K$	Z, N, V, C, H	1
SC	Rr, b	Skip if Bit in Register Cleared	$if (Rr(b)=0) PC \leftarrow PC + 2 \text{ or } 3$	None	1 / 2 / 3
SC	Rr, b	Skip if Bit in Register is Set	$if (Rr(b)=1) PC \leftarrow PC + 2 \text{ or } 3$	None	1 / 2 / 3
SC	P, b	Skip if Bit in I/O Register Cleared	$if (P(b)=0) PC \leftarrow PC + 2 \text{ or } 3$	None	1 / 2 / 3
SC	P, b	Skip if Bit in I/O Register is Set	$if (P(b)=1) PC \leftarrow PC + 2 \text{ or } 3$	None	1 / 2 / 3
SB	s, k	Branch if Status Flag Set	$if (SREG(s) = 1) then PC \leftarrow PC - k + 1$	None	1 / 2
SB	s, k	Branch if Status Flag Cleared	$if (SREG(s) = 0) then PC \leftarrow PC - k + 1$	None	1 / 2
BC	k	Branch if Carry Set	$if (Z = 1) then PC \leftarrow PC + k + 1$	None	1 / 2
BC	k	Branch if Carry Cleared	$if (Z = 0) then PC \leftarrow PC + k + 1$	None	1 / 2
CC	k	Branch if Carry Set	$if (C = 1) then PC \leftarrow PC + k + 1$	None	1 / 2
CC	k	Branch if Carry Cleared	$if (C = 0) then PC \leftarrow PC + k + 1$	None	1 / 2
CH	k	Branch if Same or Higher	$if (C = 0) then PC \leftarrow PC + k + 1$	None	1 / 2
CH	k	Branch if Same or Higher	$if (C = 1) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Lower	$if (N = 1) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Lower	$if (N = 0) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Minus	$if (N \oplus V = 0) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Minus	$if (N \oplus V = 1) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Greater or Equal, Signed	$if (H = 1) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Greater or Equal, Signed	$if (H = 0) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Half Carry Flag Set	$if (T = 1) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Half Carry Flag Cleared	$if (T = 0) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if T Flag Set	$if (T = 1) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if T Flag Cleared	$if (T = 0) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Overflow Flag is Set	$if (V = 1) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Overflow Flag is Cleared	$if (V = 0) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Interrupt Enabled	$if (I = 1) then PC \leftarrow PC + k + 1$	None	1 / 2
CL	k	Branch if Interrupt Disabled	$if (I = 0) then PC \leftarrow PC + k + 1$	None	1 / 2
<b>DATA TRANSFER INSTRUCTIONS</b>					







Symbolics	Operands	Description	Operation	Flags	#Clocks
	Rd, Rr	Move Between Registers	$Rd \leftarrow Rr$	None	1
LD	Rd, Rr	Copy Register Word	$Rd \leftarrow Rr$	None	1
	Rd, K	Load Immediate	$Rd \leftarrow K$	None	1
	Rd, X	Load Indirect	$Rd \leftarrow (X)$	None	2
	Rd, X+	Load Indirect and Post-Inc.	$Rd \leftarrow (X), X \leftarrow X + 1$	None	2
	Rd, -X	Load Indirect and Pre-Dec.	$X \leftarrow X - 1, Rd \leftarrow (X)$	None	2
	Rd, Y	Load Indirect	$Rd \leftarrow (Y)$	None	2
	Rd, Y+	Load Indirect and Post-Inc.	$Rd \leftarrow (Y), Y \leftarrow Y + 1$	None	2
	Rd, -Y	Load Indirect and Pre-Dec.	$Y \leftarrow Y - 1, Rd \leftarrow (Y)$	None	2
	Rd, Y+q	Load Indirect with Displacement	$Rd \leftarrow (Y + q)$	None	2
	Rd, Z	Load Indirect	$Rd \leftarrow (Z)$	None	2
	Rd, Z+	Load Indirect and Post-Inc.	$Rd \leftarrow (Z), Z \leftarrow Z + 1$	None	2
	Rd, -Z	Load Indirect and Pre-Dec.	$Z \leftarrow Z - 1, Rd \leftarrow (Z)$	None	2
	Rd, Z+q	Load Indirect with Displacement	$Rd \leftarrow (Z + q)$	None	2
	Rd, k	Load Direct from SRAM	$Rd \leftarrow (k)$	None	2
	X, Rr	Store Indirect	$(X) \leftarrow Rr$	None	2
	X+, Rr	Store Indirect and Post-Inc.	$(X) \leftarrow Rr, X \leftarrow X + 1$	None	2
	-X, Rr	Store Indirect and Pre-Dec.	$X \leftarrow X - 1, (X) \leftarrow Rr$	None	2
	Y, Rr	Store Indirect	$(Y) \leftarrow Rr$	None	2
	Y+, Rr	Store Indirect and Post-Inc.	$(Y) \leftarrow Rr, Y \leftarrow Y + 1$	None	2
	-Y, Rr	Store Indirect and Pre-Dec.	$Y \leftarrow Y - 1, (Y) \leftarrow Rr$	None	2
	Y+q, Rr	Store Indirect with Displacement	$(Y + q) \leftarrow Rr$	None	2
	Z, Rr	Store Indirect	$(Z) \leftarrow Rr$	None	2
	Z+, Rr	Store Indirect and Post-Inc.	$(Z) \leftarrow Rr, Z \leftarrow Z + 1$	None	2
	-Z, Rr	Store Indirect and Pre-Dec.	$Z \leftarrow Z - 1, (Z) \leftarrow Rr$	None	2
	Z+q, Rr	Store Indirect with Displacement	$(Z + q) \leftarrow Rr$	None	2
	k, Rr	Store Direct to SRAM	$(k) \leftarrow Rr$	None	2
		Load Program Memory	$R0 \leftarrow (Z)$	None	3
	Rd, Z	Load Program Memory	$Rd \leftarrow (Z)$	None	3
	Rd, Z+	Load Program Memory and Post-Inc.	$Rd \leftarrow (Z), Z \leftarrow Z + 1$	None	3
		Store Program Memory	$(Z) \leftarrow R0$	None	-
	Rd, P	In Port	$Rd \leftarrow P$	None	1
	P, Rr	Out Port	$P \leftarrow Rr$	None	1
SHR	Rr	Push Register on Stack	$STACK \leftarrow Rr$	None	2
POP	Rd	Pop Register from Stack	$Rd \leftarrow STACK$	None	2
<b>AMD BIT-TEST INSTRUCTIONS</b>					
	P, b	Set Bit in I/O Register	$IOP(b) \leftarrow 1$	None	2
	P, b	Clear Bit in I/O Register	$IOP(b) \leftarrow 0$	None	2
	Rd	Logical Shift Left	$Rd(n+1) \leftarrow Rd(n), Rd(0) \leftarrow 0$	Z, C, N, V	1
	Rd	Logical Shift Right	$Rd(n) \leftarrow Rd(n+1), Rd(7) \leftarrow 0$	Z, C, N, V	1
	Rd	Rotate Left Through Carry	$Rd(0) \leftarrow C, Rd(n+1) \leftarrow Rd(n), C \leftarrow Rd(7)$	Z, C, N, V	1
	Rd	Rotate Right Through Carry	$Rd(7) \leftarrow C, Rd(n) \leftarrow Rd(n+1), C \leftarrow Rd(0)$	Z, C, N, V	1
	Rd	Arithmetic Shift Right	$Rd(n) \leftarrow Rd(n+1), n=0..6$	Z, C, N, V	1
AP	Rd	Swap Nibbles	$Rd(3..0) \leftrightarrow Rd(7..4), Rd(7..4) \leftrightarrow Rd(3..0)$	None	1
SET	a	Flag Set	$SREG(s) \leftarrow 1$	SREG(s)	1
CLR	a	Flag Clear	$SREG(s) \leftarrow 0$	SREG(s)	1
ST	Rr, b	Bit Store from Register to T	$T \leftarrow Rr(b)$	T	1
LD	Rd, b	Bit Load from T to Register	$Rd(b) \leftarrow T$	None	1
		Set Carry	$C \leftarrow 1$	C	1
		Clear Carry	$C \leftarrow 0$	C	1
		Set Negative Flag	$N \leftarrow 1$	N	1
		Clear Negative Flag	$N \leftarrow 0$	N	1
		Set Zero Flag	$Z \leftarrow 1$	Z	1
		Clear Zero Flag	$Z \leftarrow 0$	Z	1
		Global Interrupt Enable	$I \leftarrow 1$	I	1
		Global Interrupt Disable	$I \leftarrow 0$	I	1
		Set Signed Test Flag	$S \leftarrow 1$	S	1
		Clear Signed Test Flag	$S \leftarrow 0$	S	1
		Set Two's Complement Overflow	$V \leftarrow 1$	V	1
		Clear Two's Complement Overflow	$V \leftarrow 0$	V	1
		Set T in SREG	$T \leftarrow 1$	T	1
		Clear T in SREG	$T \leftarrow 0$	T	1
		Set Half Carry Flag in SREG	$H \leftarrow 1$	H	1
		Clear Half Carry Flag in SREG	$H \leftarrow 0$	H	1
<b>CONTROL INSTRUCTIONS</b>					
		No Operation		None	1

Mnemonic	Operands	Description	Operation	Flags	#Clocks
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDPRF		Watchdog Reset	(see specific descr. for WDRF/mr)	None	1
EA		Break	For On-chip Debug Only	None	N/A



## Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package <sup>(1)</sup>	Operation Range	
8	2.7 - 5.5V	ATmega8535L-8AC	44A	Commercial (0°C to 70°C)	
		ATmega8535L-8PC	40P6		
		ATmega8535L-8JC	44J		
		ATmega8535L-8MC	44M1		
		ATmega8535L-8AI	44A		Industrial (-40°C to 85°C)
		ATmega8535L-8PI	40P6		
		ATmega8535L-8JI	44J		
		ATmega8535L-8MI	44M1		
	ATmega8535L-8AU <sup>(2)</sup>	44A			
	ATmega8535L-8PU <sup>(2)</sup>	40P6			
	ATmega8535L-8MU <sup>(2)</sup>	44M1			
	16	4.5 - 5.5V	ATmega8535-16AC	44A	Commercial (0°C to 70°C)
			ATmega8535-16PC	40P6	
			ATmega8535-16JC	44J	
ATmega8535-16MC			44M1		
ATmega8535-16AI			44A	Industrial (-40°C to 85°C)	
ATmega8535-16PI			40P6		
ATmega8535-16JI			44J		
ATmega8535-16MI			44M1		
ATmega8535-16AU <sup>(2)</sup>	44A				
ATmega8535-16PU <sup>(2)</sup>	40P6				
ATmega8535-16MU <sup>(2)</sup>	44M1				

1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging alternative, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.

Package Type	
44A	44-lead, Thin (1.0 mm) Plastic Gull Wing Quad Flat Package (TQFP)
40P6	40-pin, 0.600" Wide, Plastic Dual In-line Package (PDIP)
44J	44-lead, Plastic J-leaded Chip Carrier (PLCC)
44M1	44-pad, 7 x 7 x 1.0 mm body, lead pitch 0.50 mm, Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)

## ATmega8535(L)

2502G-AVR-04/06

**SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
BCD-TO-SEVEN-SEGMENT DECODERS/DRIVERS**

SDLS111 - MARCH 1974 - REVISED MARCH 1988

**'46A, '47A, 'LS47  
feature**

- Open-Collector Outputs Drive Indicators Directly
- Lamp-Test Provision
- Leading/Trailing Zero Suppression

**'48, 'LS48  
feature**

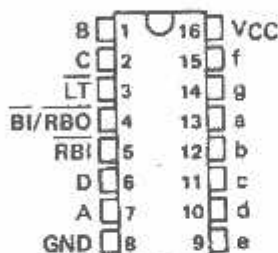
- Internal Pull-Ups Eliminate Need for External Resistors
- Lamp-Test Provision
- Leading/Trailing Zero Suppression

**'LS49  
feature**

- Open-Collector Outputs
- Blanking Input

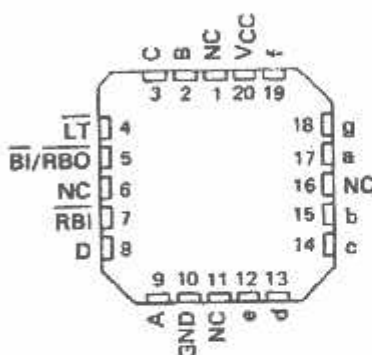
SN5446A, SN5447A, SN54LS47, SN5448,  
SN54LS48 . . . J PACKAGE  
SN7446A, SN7447A,  
SN7448 . . . N PACKAGE  
SN74LS47, SN74LS48 . . . D OR N PACKAGE

(TOP VIEW)



SN54LS47, SN54LS48 . . . FK PACKAGE

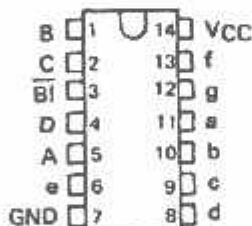
(TOP VIEW)



SN54LS49 . . . J OR W PACKAGE

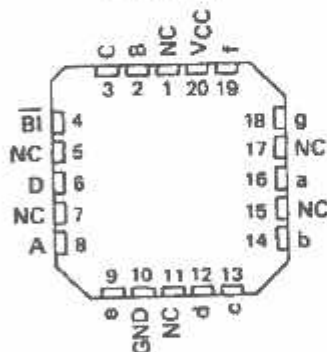
SN74LS49 . . . D OR N PACKAGE

(TOP VIEW)



SN54LS49 . . . FK PACKAGE

(TOP VIEW)



NC - No internal connection

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

**TEXAS  
INSTRUMENTS**  
POST OFFICE BOX 655305 • DALLAS, TEXAS 75265

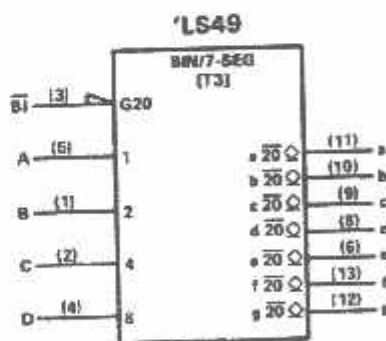
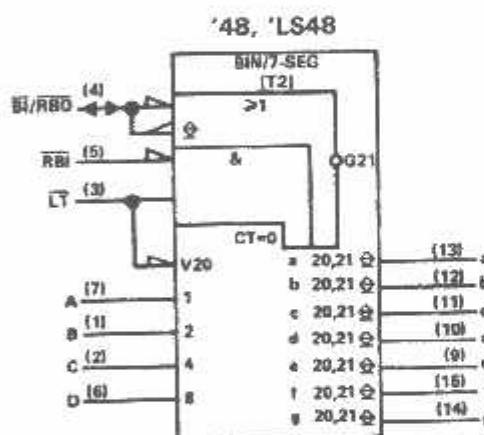
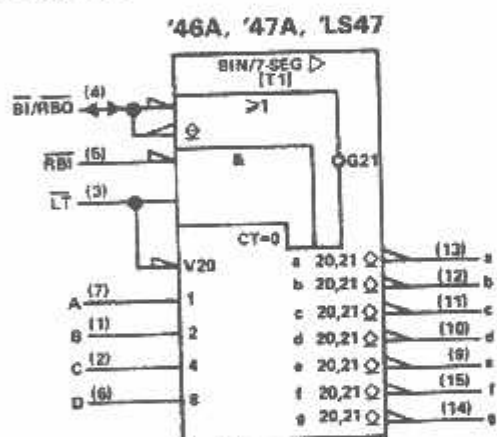
Copyright © 1988, Texas Instruments Incorporated

SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
 SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
 CD-TO-SEVEN-SEGMENT DECODERS/DRIVERS  
 LS111 - MARCH 1974 - REVISED MARCH 1988

• All Circuit Types Feature Lamp Intensity Modulation Capability

TYPE	DRIVER OUTPUTS				TYPICAL POWER DISSIPATION	PACKAGES
	ACTIVE LEVEL	OUTPUT CONFIGURATION	SINK CURRENT	MAX VOLTAGE		
SN5446A	low	open-collector	40 mA	30 V	320 mW	J, W
SN5447A	low	open-collector	40 mA	15 V	320 mW	J, W
SN5448	high	2-k $\Omega$ pull-up	6.4 mA	5.5 V	265 mW	J, W
SN54LS47	low	open-collector	12 mA	15 V	35 mW	J, W
SN54LS48	high	2-k $\Omega$ pull-up	2 mA	5.5 V	125 mW	J, W
SN54LS49	high	open-collector	4 mA	5.5 V	40 mW	J, W
SN7448A	low	open-collector	40 mA	30 V	320 mW	J, N
SN7447A	low	open-collector	40 mA	15 V	320 mW	J, N
SN7448	high	2-k $\Omega$ pull-up	6.4 mA	5.5 V	265 mW	J, N
SN74LS47	low	open-collector	24 mA	15 V	35 mW	J, N
SN74LS48	high	2-k $\Omega$ pull-up	6 mA	5.5 V	125 mW	J, N
SN74LS49	high	open-collector	8 mA	5.5 V	40 mW	J, N

logic symbols†



†These symbols are in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12. Pin numbers shown are for D, J, N, and W packages.

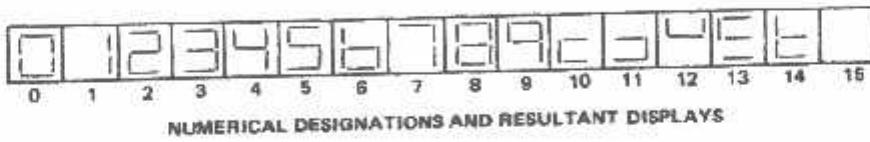
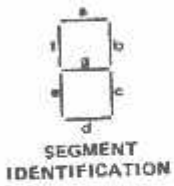
**SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
BCD-TO-SEVEN-SEGMENT DECODERS/DRIVERS**  
SOLS111 - MARCH 1974 - REVISED MARCH 1986

**description**

The '46A, '47A, and 'LS47 feature active-low outputs designed for driving common-anode LEDs or incandescent indicators directly. The '48, 'LS48, and 'LS49 feature active-high outputs for driving lamp buffers or common-cathode LEDs. All of the circuits except 'LS49 have full ripple-blanking input/output controls and a lamp test input. The 'LS49 circuit incorporates a direct blanking input. Segment identification and resultant displays are shown below. Display patterns for BCD input counts above 9 are unique symbols to authenticate input conditions.

The '46A, '47A, '48, 'LS47, and 'LS48 circuits incorporate automatic leading and/or trailing-edge zero-blanking control ( $\overline{RBI}$  and  $\overline{RBO}$ ). Lamp test (LT) of these types may be performed at any time when the  $\overline{BI}/\overline{RBO}$  node is at a high level. All types (including the '49 and 'LS49) contain an overriding blanking input ( $\overline{BI}$ ), which can be used to control the lamp intensity by pulsing or to inhibit the outputs. Inputs and outputs are entirely compatible for use with TTL logic outputs.

The SN54246/SN74246 and '247 and the SN54LS247/SN74LS247 and 'LS248 compose the 6 and the 9 with tails and were designed to offer the designer a choice between two indicator fonts.



'46A, '47A, 'LS47 FUNCTION TABLE (T1)

DECIMAL OR FUNCTION	INPUTS						$\overline{BI}/\overline{RBO}^1$	OUTPUTS							NOTE
	LT	$\overline{RBI}$	D	C	B	A		a	b	c	d	e	f	g	
0	H	H	L	L	L	L	H	ON	ON	ON	ON	ON	ON	OFF	
1	H	X	L	L	L	H	H	OFF	ON	ON	OFF	OFF	OFF	OFF	
2	H	X	L	L	H	L	H	ON	ON	OFF	ON	ON	OFF	ON	
3	H	X	L	L	H	H	H	ON	ON	ON	ON	OFF	OFF	ON	
4	H	X	L	H	L	L	H	OFF	ON	ON	OFF	OFF	ON	ON	
5	H	X	L	H	L	H	H	ON	OFF	ON	ON	OFF	ON	ON	
6	H	X	L	H	H	L	H	OFF	OFF	ON	ON	ON	ON	ON	
7	H	X	L	H	H	H	H	ON	ON	ON	OFF	OFF	OFF	OFF	1
8	H	X	H	L	L	L	H	ON	ON	ON	ON	ON	ON	ON	
9	H	X	H	L	L	H	H	ON	ON	ON	OFF	OFF	ON	ON	
10	H	X	H	L	H	L	H	OFF	OFF	OFF	ON	ON	OFF	ON	
11	H	X	H	L	H	H	H	OFF	OFF	ON	ON	OFF	OFF	ON	
12	H	X	H	H	L	L	H	OFF	ON	OFF	OFF	OFF	ON	ON	
13	H	X	H	H	L	H	H	ON	OFF	OFF	ON	OFF	ON	ON	
14	H	X	H	H	H	L	H	OFF	OFF	OFF	ON	ON	ON	ON	
15	H	X	H	H	H	H	H	OFF	OFF	OFF	OFF	OFF	OFF	OFF	
$\overline{BI}$	X	X	X	X	X	X	L	OFF	OFF	OFF	OFF	OFF	OFF	OFF	2
$\overline{RBI}$	H	L	L	L	L	L	L	OFF	OFF	OFF	OFF	OFF	OFF	OFF	3
LT	L	X	X	X	X	X	H	ON	ON	ON	ON	ON	ON	ON	4

H = high level, L = low level, X = irrelevant

- NOTES: 1. The blanking input ( $\overline{BI}$ ) must be open or held at a high logic level when output functions 0 through 15 are desired. The ripple-blanking input ( $\overline{RBI}$ ) must be open or high if blanking of a decimal zero is not desired.  
 2. When a low logic level is applied directly to the blanking input ( $\overline{BI}$ ), all segment outputs are off regardless of the level of any other input.  
 3. When ripple-blanking input ( $\overline{RBI}$ ) and inputs A, B, C, and D are at a low level with the lamp test input high, all segment outputs go off and the ripple-blanking output ( $\overline{RBO}$ ) goes to a low level (response condition).  
 4. When the blanking input/ripple blanking output ( $\overline{BI}/\overline{RBO}$ ) is open or held high and a low is applied to the lamp-test input, all segment outputs are on.

<sup>1</sup>  $\overline{BI}/\overline{RBO}$  is wire AND logic serving as blanking input ( $\overline{BI}$ ) and/or ripple-blanking output ( $\overline{RBO}$ ).

SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
 SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
 CD-TO-SEVEN-SEGMENT DECODERS/DRIVERS  
 LS111 - MARCH 1974 - REVISED MARCH 1988

'48, 'LS48  
 FUNCTION TABLE (T2)

DECIMAL OR FUNCTION	INPUTS						$\overline{BI}/\overline{RBO}^1$	OUTPUTS							NOTE
	LT	RBI	D	C	B	A		a	b	c	d	e	f	g	
0	H	H	L	L	L	L	H	H	H	H	H	H	L		
1	H	X	L	L	L	H	H	L	H	H	L	L	L		
2	H	X	L	L	H	L	H	H	H	L	H	H	L		
3	H	X	L	L	H	H	H	H	H	H	L	L	H		
4	H	X	L	H	L	L	H	L	H	H	L	L	H		
5	H	X	L	H	L	H	H	H	L	H	H	L	H		
6	H	X	L	H	H	L	H	L	L	H	H	H	H		
7	H	X	L	H	H	H	H	H	H	H	L	L	L		
8	H	X	H	L	L	L	H	H	H	H	H	H	H		
9	H	X	H	L	L	H	H	H	H	H	L	L	H		
10	H	X	H	L	H	L	H	L	L	L	H	H	L		
11	H	X	H	L	H	H	H	L	L	H	H	L	L		
12	H	X	H	H	L	L	H	L	H	L	L	L	H		
13	H	X	H	H	L	H	H	H	L	L	H	L	H		
14	H	X	H	H	H	L	H	L	L	L	H	H	H		
15	H	X	H	H	H	H	H	L	L	L	L	L	L		
BI	X	X	X	X	X	X	L	L	L	L	L	L	L		
RBI	H	L	L	L	L	L	L	L	L	L	L	L	L		
LT	L	X	X	X	X	X	H	H	H	H	H	H	H		

- H = high level, L = low level, X = irrelevant
- NOTES: 1. The blanking input ( $\overline{BI}$ ) must be open or held at a high logic level when output functions 0 through 15 are desired. The ripple-blanking input ( $\overline{RBI}$ ) must be open or high, if blanking of a decimal zero is not desired.  
 2. When a low logic level is applied directly to the blanking input ( $\overline{BI}$ ), all segment outputs are low regardless of the level of any other input.  
 3. When ripple-blanking input ( $\overline{RBI}$ ) and inputs A, B, C, and D are at a low level with the lamp-test input high, all segment outputs go low and the ripple-blanking output ( $\overline{RBO}$ ) goes to a low level (response condition).  
 4. When the blanking input/ripple-blanking output ( $\overline{BI}/\overline{RBO}$ ) is open or held high and a low is applied to the lamp-test input, all segment outputs are high.

$\overline{BI}/\overline{RBO}$  is wire-AND logic serving as blanking input ( $\overline{BI}$ ) and/or ripple-blanking output ( $\overline{RBO}$ ).

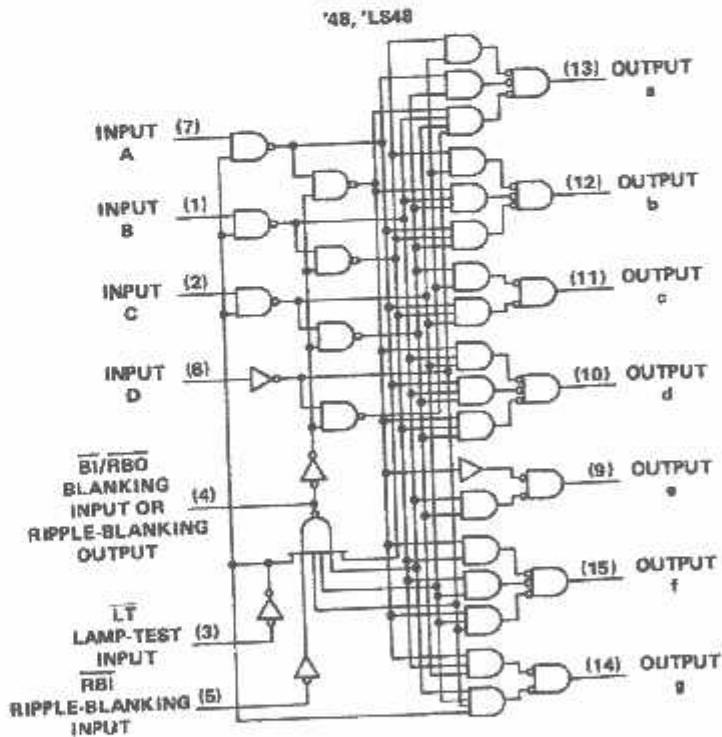
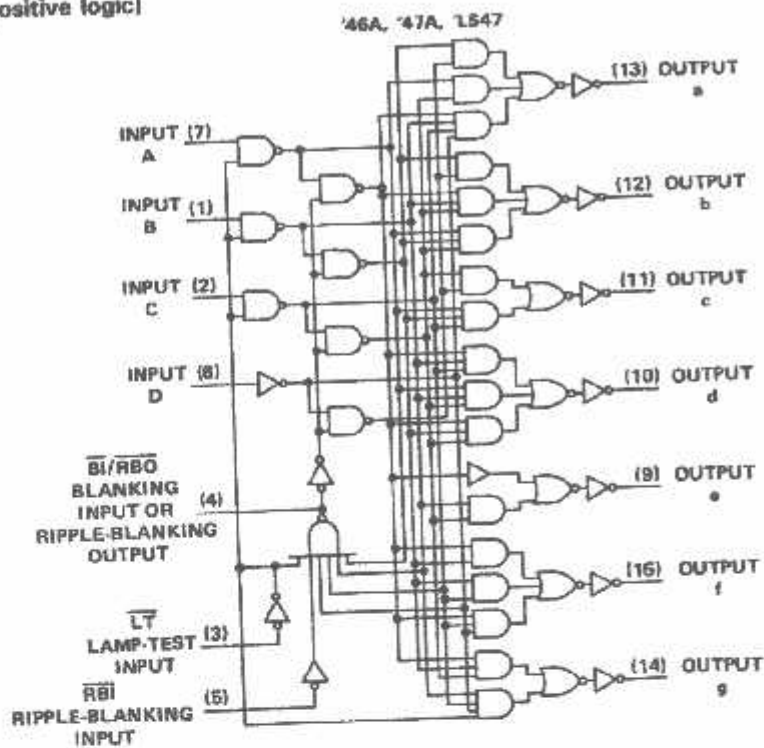
'LS49  
 FUNCTION TABLE (T3)

DECIMAL OR FUNCTION	INPUTS						$\overline{BI}$	OUTPUTS							NOTE
	D	C	B	A	$\overline{BI}$	a		b	c	d	e	f	g		
0	L	L	L	L	H	H	H	H	H	H	L	L	L		
1	L	L	L	H	H	H	H	H	L	L	L	L	L		
2	L	L	H	L	H	H	H	H	H	L	H	L	H		
3	L	L	H	H	H	H	H	H	H	H	L	L	H		
4	L	H	L	L	H	H	H	H	L	H	L	H	H		
5	L	H	L	H	H	H	H	H	L	H	H	L	H		
6	L	H	H	L	H	H	H	H	L	L	H	H	H		
7	L	H	H	H	H	H	H	H	L	L	L	L	L		
8	H	L	L	L	H	H	H	H	H	H	L	L	H		
9	H	L	L	H	H	H	H	H	L	L	L	L	H		
10	H	L	H	L	H	H	H	H	L	L	L	H	L		
11	H	L	H	H	H	H	H	H	L	L	H	H	L		
12	H	H	L	L	H	H	H	H	L	H	L	L	H		
13	H	H	L	H	H	H	H	H	L	L	L	H	H		
14	H	H	H	L	H	H	H	H	L	L	L	H	H		
15	H	H	H	H	H	H	H	H	L	L	L	L	L		
BI	X	X	X	X	X	X	L	L	L	L	L	L	L		

- H = high level, L = low level, X = irrelevant
- NOTES: 1. The blanking input ( $\overline{BI}$ ) must be open or held at a high logic level when output functions 0 through 15 are desired.  
 2. When a low logic level is applied directly to the blanking input ( $\overline{BI}$ ), all segment outputs are low regardless of the level of any other input.

SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
 SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
 BCD-TO-SEVEN-SEGMENT DECODERS/DRIVERS  
 SDLS111 - MARCH 1974 - REVISED MARCH 1988

logic diagrams (positive logic)



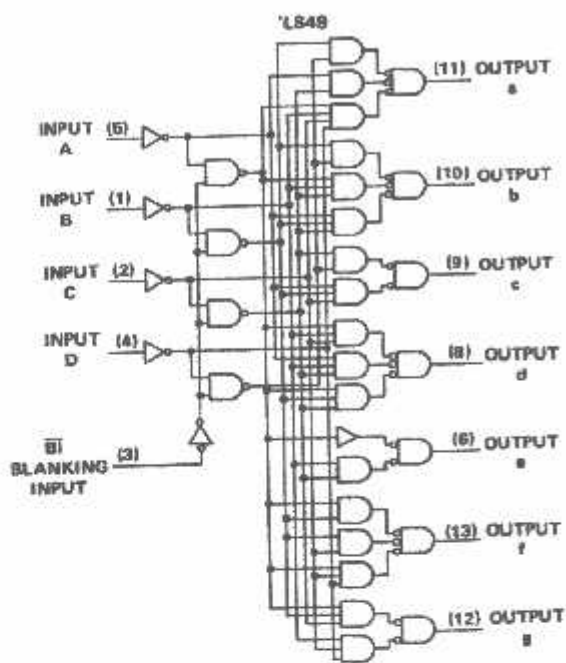
Pin numbers shown are for D, J, N, and W packages.



POST OFFICE BOX 355303 • DALLAS, TEXAS 75265



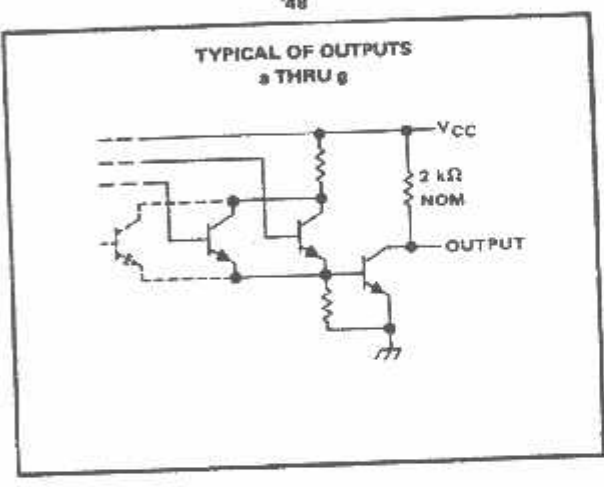
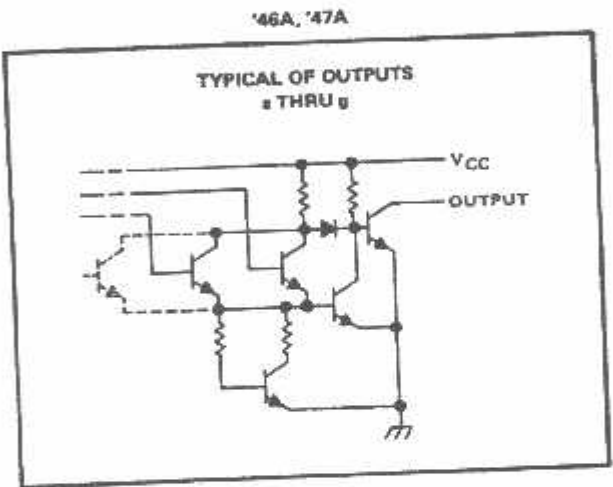
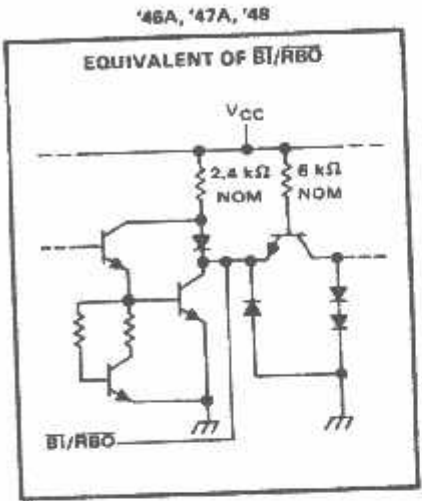
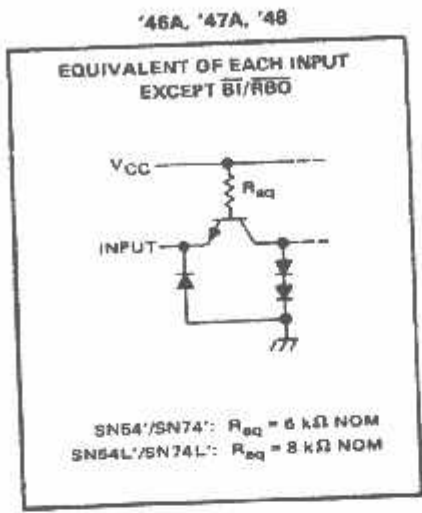
Logic diagrams (continued)



Pin numbers shown are for D, J, N, and W packages.

SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
 SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
 BCD-TO-SEVEN-SEGMENT DECODERS/DRIVERS  
 SDLS111 - MARCH 1974 - REVISED MARCH 1966

schematics of inputs and outputs

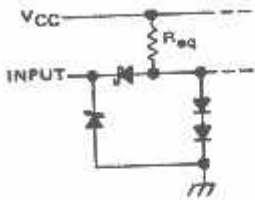


SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
 SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
 CD-TO-SEVEN-SEGMENT DECODERS/DRIVERS  
 LS111 - MARCH 1974 - REVISED MARCH 1988

Schematics of inputs and outputs

'LS47, 'LS48, 'LS49

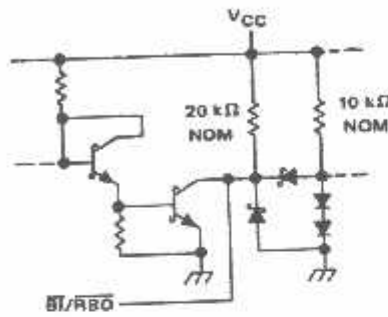
EQUIVALENT OF EACH INPUT  
 EXCEPT  $\overline{BI}/\overline{R\overline{B}0}$



$\overline{CT}$  and  $\overline{RB1}$  ('LS47, 'LS48):  $R_{eq} = 20\text{ k}\Omega$  NOM  
 $\overline{BI}$  ('LS48):  $R_{eq} = 20\text{ k}\Omega$  NOM  
 A, B, C, and D:  $R_{eq} = 25\text{ k}\Omega$  NOM

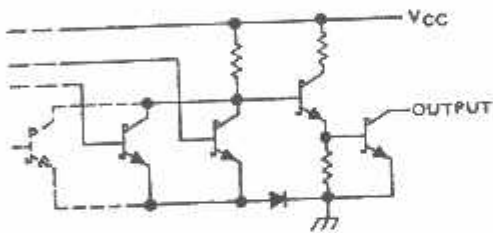
'LS47, 'LS48, 'LS49

EQUIVALENT OF  $\overline{BI}/\overline{R\overline{B}0}$



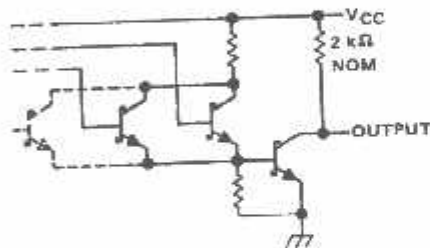
'LS47

TYPICAL OF OUTPUTS  
 a THRU g



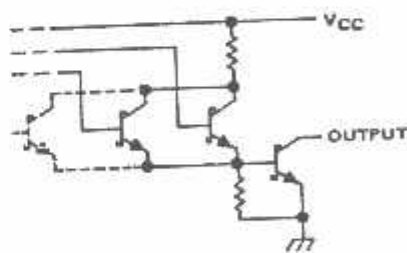
'LS48

TYPICAL OF OUTPUTS  
 a THRU g



'LS49

TYPICAL OF OUTPUTS  
 a THRU g



 **TEXAS  
 INSTRUMENTS**

POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

**SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
BCD-TO-SEVEN-SEGMENT DECODERS/DRIVERS**  
SDLS111 - MARCH 1974 - REVISED MARCH 1986

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, $V_{CC}$ (see Note 1)	7 V
Input voltage	5.5 V
Current forced into any output in the off state	1 mA
Operating free-air temperature range: SN5446A, SN5447A	-55°C to 125°C
SN7446A, SN7447A	0°C to 70°C
Storage temperature range	-65°C to 150°C

NOTE 1: Voltage values are with respect to network ground terminal.

recommended operating conditions

	SN5446A			SN5447A			SN7446A			SN7447A			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, $V_{CC}$	4.5	5	5.5	4.5	5	5.5	4.75	5	5.25	4.75	5	5.25	V
Off-state output voltage, $V_{O(off)}$	a thru g			15			30			15			V
On-state output current, $I_{O(on)}$	a thru g			40			40			40			mA
High-level output current, $I_{OH}$	$\overline{BI}/\overline{RBO}$			-200			-200			-200			$\mu$ A
Low-level output current, $I_{OL}$	$\overline{BI}/\overline{RBO}$			8			8			8			mA
Operating free-air temperature, $T_A$	-65		125	-65		125	0		70	0		70	°C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS†	MIN	TYP‡	MAX	UNIT	
$V_{IH}$	High-level input voltage			2		V	
$V_{IL}$	Low-level input voltage				0.8	V	
$V_{IK}$	Input clamp voltage				-1.5	V	
$V_{OH}$	High-level output voltage	$V_{CC} = \text{MIN.}, I_1 = -12 \text{ mA}$				V	
$V_{OL}$	Low-level output voltage	$V_{CC} = \text{MIN.}, V_{IH} = 2 \text{ V}, V_{IL} = 0.8 \text{ V}, I_{OH} = -200 \mu\text{A}$	2.4	3.7		V	
$I_{O(off)}$	Off-state output current	$V_{CC} = \text{MIN.}, V_{IH} = 2 \text{ V}, V_{IL} = 0.8 \text{ V}, I_{OL} = 8 \text{ mA}$		0.27	0.4	$\mu$ A	
$V_{O(on)}$	On-state output voltage	$V_{CC} = \text{MAX.}, V_{IH} = 2 \text{ V}, V_{IL} = 0.8 \text{ V}, V_{O(off)} = \text{MAX}$			250	$\mu$ A	
$I_1$	Input current at maximum input voltage	$V_{CC} = \text{MIN.}, V_{IH} = 2 \text{ V}, V_{IL} = 0.8 \text{ V}, I_{O(on)} = 40 \text{ mA}$		0.3	0.4	V	
$I_{IH}$	High-level input current	Any input except $\overline{BI}/\overline{RBO}$			1	mA	
$I_{IL}$	Low-level input current	Any input except $\overline{BI}/\overline{RBO}$			40	$\mu$ A	
$I_{OS}$	Short-circuit output current	$\overline{BI}/\overline{RBO}$			-1.6	mA	
$I_{CC}$	Supply current	$V_{CC} = \text{MAX.}$			-4	mA	
		See Note 2			-4	mA	
					64	85	mA
					64	103	mA

†For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

‡All typical values are at  $V_{CC} = 5 \text{ V}, T_A = 25^\circ\text{C}$ .

NOTE 2:  $I_{CC}$  is measured with all outputs open and all inputs at 4.5 V.

switching characteristics,  $V_{CC} = 5 \text{ V}, T_A = 25^\circ\text{C}$

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{off}$	Turn-off time from A input	$C_L = 15 \text{ pF}, R_L = 120 \Omega,$ See Note 3			100	ns
$t_{on}$	Turn-on time from A input				100	ns
$t_{off}$	Turn-off time from $\overline{RBI}$ input				100	ns
$t_{on}$	Turn-on time from $\overline{RBI}$ input				100	ns

NOTE 3: Load circuits and voltage waveforms are shown in Section 7.



POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
 SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
 8-TO-SEVEN SEGMENT DECODERS/DRIVERS  
 S111 - MARCH 1974 - REVISED MARCH 1988

Absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, $V_{CC}$ (see Note 1)	7 V
Input voltage	5.5 V
Operating free-air temperature range: SN5448	-55°C to 125°C
SN7448	0°C to 70°C
Storage temperature range	-65°C to 150°C

NOTE 1: Voltage values are with respect to network ground terminal.

Recommended operating conditions

PARAMETER	SN5448			SN7448			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, $V_{CC}$	4.5	5	5.6	4.75	5	5.25	V
High-level output current, $I_{OH}$	a thru g		-400			-400	$\mu$ A
	$\overline{BI}/\overline{RBO}$		-200			-200	
Low-level output current, $I_{OL}$	a thru g		6.4			6.4	mA
	$\overline{BI}/\overline{RBO}$		8			8	
Operating free-air temperature, $T_A$	-55		125	0		70	°C

Electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS†	MIN	TYP‡	MAX	UNIT
$V_{IH}$	High-level input voltage			2		V
$V_{IL}$	Low-level input voltage				0.8	V
$V_{IK}$	Input clamp voltage				-1.5	V
$V_{OH}$	High-level output voltage	a thru g	$V_{CC} = \text{MIN}, I_I = -12 \text{ mA}$	2.4	4.2	V
		$\overline{BI}/\overline{RBO}$	$V_{CC} = \text{MIN}, V_{IH} = 2 \text{ V}, V_{IL} = 0.8 \text{ V}, I_{OH} = \text{MAX}$	2.4	3.7	
$I_O$	Output current	a thru g	$V_{CC} = \text{MIN}, V_O = 0.85 \text{ V}$ , Input conditions as for $V_{OH}$	-1.3	-2	mA
$V_{OL}$	Low-level output voltage		$V_{CC} = \text{MIN}, V_{IH} = 2 \text{ V}, V_{IL} = 0.8 \text{ V}, I_{OL} = \text{MAX}$	0.27	0.4	V
$I_I$	Input current at maximum input voltage	Any input except $\overline{BI}/\overline{RBO}$	$V_{CC} = \text{MAX}, V_I = 5.5 \text{ V}$		1	mA
$I_{IH}$	High-level input current	Any input except $\overline{BI}/\overline{RBO}$	$V_{CC} = \text{MAX}, V_I = 2.4 \text{ V}$		40	$\mu$ A
$I_{IL}$	Low-level input current	Any input except $\overline{BI}/\overline{RBO}$	$V_{CC} = \text{MAX}, V_I = 0.4 \text{ V}$		-1.6	mA
		$\overline{BI}/\overline{RBO}$			-4	
$I_{OS}$	Short-circuit output current	$\overline{BI}/\overline{RBO}$	$V_{CC} = \text{MAX}$		53	76
$I_{CC}$	Supply current		$V_{CC} = \text{MAX}$ , See Note 2		53	90
				SN5448		
				SN7448		90

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.  
 ‡ All typical values are at  $V_{CC} = 5 \text{ V}, T_A = 25^\circ \text{C}$ .  
 NOTE 2:  $I_{CC}$  is measured with all outputs open and all inputs at 4.5 V.

Switching characteristics,  $V_{CC} = 5 \text{ V}, T_A = 25^\circ \text{C}$

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{PHL}$	$C_L = 15 \text{ pF}, R_L = 1 \text{ k}\Omega$ See Note 3			100	ns
$t_{PLH}$				100	ns
$t_{PHL}$				100	ns
$t_{PLH}$				100	ns

NOTE 3: Load circuits and voltage waveforms are shown in Section 1.



POST OFFICE BOX 655303 • DALLAS, TEXAS 75285

**SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
BCD-TO-SEVEN-SEGMENT DECODERS/DRIVERS**  
SDLS111 - MARCH 1974 - REVISED MARCH 1988

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, $V_{CC}$ (see Note 1)	7 V
Input voltage	7 V
Peak output current ( $t_w \leq 1$ ms, duty cycle $\leq 10\%$ )	200 mA
Current forced into any output in the off state	1 mA
Operating free-air temperature range: SN54LS47	-55°C to 125°C
SN74LS47	0°C to 70°C
Storage temperature range	-65°C to 150°C

NOTE 1: Voltage values are with respect to network ground terminal.

recommended operating conditions

	SN54LS47			SN74LS47			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, $V_{CC}$	4.5	5	5.5	4.75	5	5.25	V
Off-state output voltage, $V_{O(off)}$	a thru g			15			V
On-state output current, $I_{O(on)}$	a thru g			12			mA
High-level output current, $I_{OH}$	B1/RB0			-50			$\mu$ A
Low-level output current, $I_{OL}$	B1/RB0			1.6			mA
Operating free-air temperature, $T_A$	-55	125		0	70		°C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS†	SN54LS47		SN74LS47		UNIT
			MIN	TYP‡	MAX	MIN	
$V_{IH}$	High-level input voltage		2		2		V
$V_{IL}$	Low-level input voltage		0.7		0.8		V
$V_{IK}$	Input clamp voltage	$V_{CC} = \text{MIN}, I_I = -18$ mA	-1.5		-1.5		V
$V_{OH}$	High-level output voltage	$V_{CC} = \text{MIN}, V_{IH} = 2$ V, $V_{IL} = V_{IL \text{ max}}, I_{OH} = -50$ $\mu$ A	2.4	4.2	2.4	4.2	V
$V_{OL}$	Low-level output voltage	$V_{CC} = \text{MIN}, I_{OL} = 1.6$ mA $V_{IH} = 2$ V, $V_{IL} = V_{IL \text{ max}}, I_{OL} = 3.2$ mA	0.25	0.4	0.25	0.4	V
$I_{O(off)}$	Off-state output current	a thru g $V_{CC} = \text{MAX}, V_{IH} = 2$ V, $V_{IL} = V_{IL \text{ max}}, V_{O(off)} = 15$ V	250		250		$\mu$ A
$V_{O(on)}$	On-state output voltage	a thru g $V_{CC} = \text{MIN}, I_{O(on)} = 12$ mA $V_{IH} = 2$ V, $V_{IL} = V_{IL \text{ max}}, I_{O(on)} = 24$ mA	0.25	0.4	0.25	0.4	V
$I_I$	Input current at maximum input voltage	$V_{CC} = \text{MAX}, V_I = 7$ V	0.1		0.1		mA
$I_{IH}$	High-level input current	$V_{CC} = \text{MAX}, V_I = 2.7$ V	20		20		$\mu$ A
$I_{IL}$	Low-level input current	Any input except B1/RB0 B1/RB0 $V_{CC} = \text{MAX}, V_I = 0.4$ V	-0.4		-0.4		mA
$I_{OS}$	Short-circuit output current	B1/RB0 $V_{CC} = \text{MAX}$	-0.3	-2	-0.3	-2	mA
$I_{CC}$	Supply current	$V_{CC} = \text{MAX}$ , See Note 2	7	13	7	13	mA

†For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

‡All typical values are at  $V_{CC} = 5$  V,  $T_A = 25^\circ$  C.

NOTE 2:  $I_{CC}$  is measured with all outputs open and all inputs at 4.5 V.

switching characteristics,  $V_{CC} = 5$  V,  $T_A = 25^\circ$  C

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{off}$	Turn-off time from A input	100			ns
$t_{on}$	Turn-on time from A input	100			
$t_{off}$	Turn-off time from RB1 input, outputs (a-f) only	100			ns
$t_{on}$	Turn-on time from RB1 input, outputs (a-f) only	100			

$C_L = 15$  pF,  $R_L = 665$   $\Omega$ ,  
See Note 3

NOTE 3: Load circuits and voltage waveforms are shown in Section 1.

SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
 SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
 CD-TO-SEVEN-SEGMENT DECODERS/DRIVERS  
 D.S111 - MARCH 1974 - REVISED MARCH 1986

Absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, $V_{CC}$ (see Note 1)	7 V
Input voltage	7 V
Operating free-air temperature range: SN54LS48	-55°C to 125°C
SN74LS48	0°C to 70°C
Storage temperature range	-65°C to 150°C

NOTE 1: Voltage values are with respect to network ground terminal.

Recommended operating conditions

	SN54LS48			SN74LS48			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, $V_{CC}$	4.5	5	5.5	4.75	5	5.25	V
High-level output current, $I_{OH}$	a thru g		-100			-100	$\mu$ A
	$\overline{BI}/\overline{RBO}$		-50			-50	
Low-level output current, $I_{OL}$	a thru g		2			6	mA
	$\overline{BI}/\overline{RBO}$		1.6			3.2	
Operating free-air temperature, $T_A$	-55		125	0		70	°C

Electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS†	SN54LS48			SN74LS48			UNIT	
		MIN	TYP‡	MAX	MIN	TYP‡	MAX		
$V_{IH}$ High-level input voltage		2			2			V	
$V_{IL}$ Low-level input voltage				0.7			0.8	V	
$V_{IK}$ Input clamp voltage				-1.5			-1.5	V	
$V_{OH}$ High-level output voltage	a thru g and $\overline{BI}/\overline{RBO}$ $V_{CC} = \text{MIN}, V_{IH} = 2 \text{ V}, V_{IL} = V_{IL \text{ max}}, I_{OH} = \text{MAX}$	2.4	4.2		2.4	4.2		V	
$I_O$ Output current	a thru g $V_{CC} = \text{MIN}, V_O = 0.86 \text{ V},$ Input conditions as for $V_{OH}$	-1.3	-2		-1.3	-2		mA	
$V_{OL}$ Low-level output voltage	a thru g $V_{CC} = \text{MIN}, V_{IH} = 2 \text{ V}, V_{IL} = V_{IL \text{ max}}$	$I_{OL} = 2 \text{ mA}$	0.25	0.4	0.25	0.4		V	
		$I_{OL} = 6 \text{ mA}$			0.35	0.5			
	$\overline{BI}/\overline{RBO}$ $V_{CC} = \text{MIN}, V_{IH} = 2 \text{ V}, V_{IL} = V_{IL \text{ max}}$	$I_{OL} = 1.6 \text{ mA}$	0.25	0.4	0.25	0.4		V	
		$I_{OL} = 3.2 \text{ mA}$			0.35	0.5			
$I_I$ Input current at maximum input voltage	Any input except $\overline{BI}/\overline{RBO}$ $V_{CC} = \text{MAX}, V_I = 7 \text{ V}$			0.1			0.1	mA	
$I_{IH}$ High-level input current	Any input except $\overline{BI}/\overline{RBO}$ $V_{CC} = \text{MAX}, V_I = 2.7 \text{ V}$			20			20	$\mu$ A	
$I_{IL}$ Low-level input current	Any input except $\overline{BI}/\overline{RBO}$ $V_{CC} = \text{MAX}, V_I = 0.4 \text{ V}$			-0.4			-0.4	mA	
	$\overline{BI}/\overline{RBO}$			-1.2			-1.2		
$I_{OS}$ Short-circuit output current	$\overline{BI}/\overline{RBO}$ $V_{CC} = \text{MAX}$	-0.3		-2	-0.3		-2	mA	
$I_{CC}$ Supply current	$V_{CC} = \text{MAX},$ See Note 2			25	38		25	38	mA

For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

† All typical values are at  $V_{CC} = 5 \text{ V}, T_A = 25^\circ \text{C}.$

‡ NOTE 2:  $I_{CC}$  is measured with all outputs open and all inputs at 4.5 V.

Switching characteristics,  $V_{CC} = 5 \text{ V}, T_A = 25^\circ \text{C}$

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{PHL}$ Propagation delay time, high-to-low-level output from A input	$C_L = 15 \text{ pF}, R_L = 4 \text{ k}\Omega,$ See Note 3			100	ns
$t_{PLH}$ Propagation delay time, low-to-high-level output from A input				100	
$t_{PHL}$ Propagation delay time, high-to-low-level output (a-f only) from $\overline{RBI}$ input	$C_L = 15 \text{ pF}, R_L = 8 \text{ k}\Omega,$ See Note 3			100	ns
$t_{PLH}$ Propagation delay time, low-to-high-level output (a-f only) from $\overline{RBI}$ input				100	

NOTE 3: Load circuits and voltage waveforms are shown in Section 1.

**SN5446A, '47A, '48, SN54LS47, 'LS48, 'LS49  
SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49  
BCD-TO-SEVEN-SEGMENT DECODERS/DRIVERS**  
SDLS111 - MARCH 1974 - REVISED MARCH 1988

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, $V_{CC}$ (see Note 1)	7 V
Input voltage	7 V
Current forced into any output in the off state	1 mA
Operating free-air temperature range: SN54LS49	-55°C to 125°C
SN74LS49	0°C to 70°C
Storage temperature range	-65°C to 150°C

NOTE 1: Voltage values are with respect to network ground terminal.

recommended operating conditions

	SN54LS49			SN74LS49			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, $V_{CC}$	4.5	5	5.5	4.75	5	5.25	V
High-level output voltage, $V_{OH}$			5.5			5.5	V
Low-level output current, $I_{OL}$			4			8	mA
Operating free-air temperature, $T_A$	-55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS†	SN54LS49		SN74LS49		UNIT	
		MIN	TYP‡	MAX	MIN		TYP‡
$V_{IH}$ High-level input voltage		2		2		V	
$V_{IL}$ Low-level input voltage			0.7		0.8	V	
$V_{IK}$ Input clamp voltage			-1.5		-1.5	V	
$I_{OH}$ High-level output current	$V_{CC} = \text{MIN}$ , $I_I = -18 \text{ mA}$ $V_{IL} = V_{IL \text{ max}}$ , $V_{OH} = 5.5 \text{ V}$		250		250	µA	
$V_{OL}$ Low-level output voltage	$V_{CC} = \text{MIN}$ , $V_{IH} = 2 \text{ V}$ , $V_{IL} = V_{IL \text{ max}}$		0.25	0.4	0.25	0.4	V
	$I_{OL} = 4 \text{ mA}$				0.35	0.5	
	$I_{OL} = 8 \text{ mA}$						
$I_I$ Input current at maximum input voltage	$V_{CC} = \text{MAX}$ , $V_I = 7 \text{ V}$		0.1		0.1	mA	
$I_{IH}$ High-level input current	$V_{CC} = \text{MAX}$ , $V_I = 2.7 \text{ V}$		20		20	µA	
$I_{IL}$ Low-level input current	$V_{CC} = \text{MAX}$ , $V_I = 0.4 \text{ V}$		-0.4		-0.4	mA	
$I_{CC}$ Supply current	$V_{CC} = \text{MAX}$ , See Note 2		8	15	8	15	mA

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

‡ All typical values are at  $V_{CC} = 5 \text{ V}$ ,  $T_A = 25^\circ \text{C}$ .

NOTE 2:  $I_{CC}$  is measured with all outputs open and all inputs at 4.5 V.

switching characteristics,  $V_{CC} = 5 \text{ V}$ ,  $T_A = 25^\circ \text{C}$

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{PHL}$ Propagation delay time, high-to-low-level output from A input	$C_L = 15 \text{ pF}$ , $R_L = 4 \text{ k}\Omega$ See Note 3			100	ns
$t_{PLH}$ Propagation delay time, low-to-high-level output from A input				100	
$t_{PHL}$ Propagation delay time, high-to-low-level output (a-f only) from RBI input	$C_L = 15 \text{ pF}$ , $R_L = 6 \text{ k}\Omega$ See Note 3			100	ns
$t_{PLH}$ Propagation delay time, low-to-high-level output (a-f only) from RBI input				100	

NOTE 3: Load circuits and voltage waveforms are shown in Section 1.



### IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

## BIODATA PENULIS



### **Bambang Husono**

Penulis Lahir di Pulau Bunyu pada tanggal 10 Februari 1984 (Bunyu Island). Anak dari Saelan Sumokarto sebagai anak pertama dari 5 bersaudara (*it's a great family, love you all*). Penulis merupakan seorang yang menyenangkan dan sukanya merempoti orang lain, (*Ok Coy*) paling tidak suka melihat teman-temannya bersedih apalagi sakit hati, dan memiliki motto "Don't give up before you try". Penulis mempunyai hobi bermain dan bersantai (*it's All gratify*). Sampai saat ini penulis tinggal di Pulau Bunyu. Dan selama menempuh kuliah di ITN Malang, penulis kost di Jl. Bendungan Sutami No.18C & Jl. Perusahaan Gang 5 No. 32.

Riwayat Pendidikan formal yang pernah di tempuh penulis:

- TK Tunas Teratai Pulau Bunyu tamat tahun 1990
  - SD Negeri 008 Pulau Bunyu tamat tahun 1996
  - SMP Negeri 1 Pulau Bunyu tamat tahun 1999
  - SMK Negeri 2 Tarakan tamat tahun 2002
  - ITN malang S-1 Elektronika tamat tahun 2009 (*Lama Banget Coy*)
- Riwayat Pendidikan Informal yang pernah ditempuh penulis:
- Kursus komputer "ITC dan Microsoft Word" tahun 1999
  - Kursus komputer "LOTUS 123R24" tahun 2001

No Hp : 085646477444

: 081233244869

Email : [husono\\_club@yahoo.com](mailto:husono_club@yahoo.com)

: [husonohandayani@ymail.com](mailto:husonohandayani@ymail.com)

Pada hari Senin tanggal 23 Maret 2009 mengikuti Ujian Akhir Komprehensif sebagai salah satu persyaratan untuk mendapatkan gelar Sarjana Teknik (S-1) Institut Teknologi Nasional Malang.