SKRIPSI

PERANCANGAN DAN PEMBUATAN ALAT PEMUTAR LAGU DIGITAL YANG DISIMPAN PADA MEDIA KARTU MEMORI MMC (MULTI MEDIA CARD) BERBASIS MIKROKONTROLLER ATMEGA 8

Disusun Oleh:

MARTHEN LEUNARD SIMAELA
01.17.025

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

2009

INSTITUT TEKNOLOGI NASIONAL
Jl. Raya Karanglo KM 2
MALANG

BERITA ACARA UJIAN SKRIPSI
FAKULTAS TEKNOLOGI INDUSTRI

1.	Nama Mahasiswa	: Marthen Leunard Simaela
2.	NIM	: 0117025
3.	Jurusan	: Teknik Elektro S-1
4.	Konsenstrasi	: Teknik Elektronika
5.	Judul Skripsi	: Perancangan Dan Pembuatan Alat Pemutar Lagu Digital Yang Disimpan Pada Media Kartu MMC (Multi Media Card) Berbasis Mikrokontroller ATMEGA 8

Dipertahankan di hadapan Tim Penguji Skripsi Jenjang Strata Satu (S1) pada :

Hari	: Selasa
Tanggal	: 24 Maret 2009
Dengan Nilai	: 73,4 (B+)

Panitia Ujian Skripsi

Ketua	Sekretaris
Ir. Sidik Noertjahjono, MT	Ir. F. Yudi Limpraptono, MT
NIP.Y 1028700163	NIP.Y 1039500274

Anggota Penguji

Penguji Pertama	Penguji Kedua
Joseph Dedy Irawan, ST. MT	Bambang Prio Hartono, ST
NIP. 132315178	NIP. Y 1028400082

PERANCANGAN DAN PEMBUATAN ALAT PEMUTAR LAGU DIGITAL YANG DISIMPAN PADA MEDIA KARTU MEMORY MMC (MULTI MEDIA CARD) BERBASIS MIKROKONTROLLER ATMEGA 8

Marthen Leunard Simaela
01.17.025
Jurusan Teknik Elektronika – Institut Teknologi Nasional Malang
Jln. Raya Karanglo Km 2 Malang
ml.simaela@gmail.com

Dosen Pembimbing : Ir.F.Yudi Limpraptono, MT.

Perkembangan pesat bidang komunikasi membuat komunikasi digital semakin populer. Di dalam segi penyampaian informasi tersebut ada banyak cara yang digunakan, contohnya pemanfaatan teknologi MMC (Multi Media Card). Dengan memanfaatkan teknologi MMC sebagai suatu media komunikasi yang berfungsi sebagai penyimpan data dengan kemampuan akses integrasi acak yang sangat cepat untuk mengoptimalkan transmisi data (berupa file lagu), kita dapat mendengarkan musik dengan praktis, ekonomis dan efisien serta kita dapat mendengarkan musik seberapa banyak musik yang ingin kita dengarkan dan bermacam-macam jenis musiknya

Prinsip kerja alat ini adalah pada saat sistem pada kondisi ON, mikrokontroler menginisialisasi LCD dan MMC. Kemudian mikrokontroler menampilkan judul alat dan menu pada layar LCD. Ketika tombol PLAY ditekan maka mikrokontroler akan mulai membaca data MMC. Data MMC akan diolah terlebih dulu oleh mikrokontroler sebelum dikeluarkan menjadi bentuk data PWM (Pulse Width Modulation). Pengolahan data yang dimaksud disini berhubungan dengan volume dan kecepatan data suara yang akan dikeluarkan. Pengaturan volume didapat dengan mengatur duty cycle pulsa PWM yang dikeluarkan, sedangkan kecepatan diatur dengan mengatur waktu interval saat mengeluarkan data menjadi PWM. Dengan ditambahkannya LPF (Low Pass Filter) maka sinyal frekuensi pembawa PWM akan dipangkas dan dihasilkan sinyal suara yang dimodulasikannya. Sinyal hasil keluaran sudah berupa sinyal analog, jadi tinggal menguatkan saja dengan amplifier analog dan kemudian diumpankankan ke loudspeaker sehingga bisa didengar oleh telinga kita.

Dari hasil pengujian, suara yang dihasilkan mono, lagu yang di baca hanya berformat WAV. Pada hasil pengujian sinyal MMC dan sinyal PWM , apabila diberi sinyal input, maka sinyal PWM akan berubah/ bergetar sesuai perubahan sinyal input, dan apabila hasil pengujian sinyal keluaran tidak diberi filter, suara analog yang dihasilkan tidak jelas.

Kata Kunci : *MMC, Mikrokontroller Atmega 8, LCD, PWM*

KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa yang telah memberikan Rahmat kepada kita semua dan khususnya penulis sehingga dapat menyelesaikan laporan Skripsi ini dengan judul :

PERANCANGAN DAN PEMBUATAN ALAT PEMUTAR LAGU DIGITAL YANG DISIMPAN PADA MEDIA KARTU MEMORY MMC (MULTI MEDIA CARD) BERBASIS MIKROKONTROLLER ATMEGA 8

Pembuatan skripsi ini guna memenuhi syarat akhir kelulusan pendidikan jenjang strata-1 di Program Studi Elektonika Jurusan Teknik Elektro Institut Teknologi Nasional Malang.

Dengan segala kerendahan hati atas keberhasilan penyelesaian laporan Skripsi ini, Penyusun mengucapan terima kasih kepada :

1. Kedua Orang Tua dan kakak atas segala doa dan dukungannya.
2. Bapak Dr. Ir. Abraham Lomi, MSEE. selaku Rektor Institut Teknologi Nasional Malang
3. Bapak Ir. F. Yudi Limpraptono, MT, selaku Ketua Jurusan Teknik Elektro Institut Teknologi Malang.
4. Bapak Ir. F. Yudi Limpraptono, MT selaku Dosen Pembimbing yang telah memberikan bimbingan, saran dan pemikiran dalam menyelesaikan laporan skripsi ini.

5. Seseorang yang paling aku sayangi, Irma Rusli Lestari yang selalu membantu dan memberikan semangat dalam penyelesaian skripsi ini
6. Teman–teman teknik elektronika S-1 yang telah membantu dalam penyelesaian skripsi ini saya ucapkan terima kasih.
7. Teman–teman di Goku Camp di Jl. Raya Candi II/ 76, terimakasih atas dukungannya.

Penulis menyadari bahwa laporan ini masih banyak yang perlu disempurnakan. Oleh sebab itu kritik dan saran yang bersifat membangun sangat diharapkan dari pembaca khususnya mahasiswa Teknik Jurusan Elekto ITN Malang, agar di masa datang akan lebih baik lagi. Harapan dari penulis semoga skripsi ini dapat bermanfaat.

Malang, Mei 2009

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

PENDAHULUAN

1.1. Latar Belakang

Dalam era globalisasi dan modern seperti saat ini, kemajuan teknologi sangat diperlukan manusia dalam menciptakan kemudahan hidup dalam segala bidang. Kemajuan teknologi seiring bejalannya waktu, manusia dituntut agar bisa mengikuti perkembangan teknologi tersebut

Banyak sekali kita jumpai kemajuan teknologi di mana-mana bahkan di sekitar kita sendiri. Dengan teknologi , hidup manusia menjadi lebih muda. Sebagai contoh media penyimpan musik. Dulu kita masih menggunakan kaset untuk mendengarkan music, dimana kaset tersebut hanya bisa memuat beberapa lagu dan tidak bisa bertahan lama, begirtu juga CD, CD mungkin bisa memuat banyak lagu, tapi kendalanya juga sama, tidak bisa bertahan lama.

Faktor inilah yang mendorong kami untuk mengembangkan suatu system pelayanan yang lebih praktis, hemat dan efisien. Di sini akan dicoba memperkenalkan sebuah alat pemutar musik digital yang disimpan pada media kartu memori mmc (multi media card) yang berbasis microcontroller ATMEGA8. Di mana dengan menggunakan sistem ini, kita dapat mendengarkan musik dengan praktis, ekonomis dan efisien serta kita dapat mendengarkan musik seberapa banyak musik yang ingin kita dengarkan dan bermacam-macam jenis musiknya.

1.2. Rumusan Masalah

Permasalahan yang di bahas dalam penyusuan skripsi ini adalah ;

1. Bagaimana merancang dan membuat perangkat keras yang digunakan.
2. Bagaimana merancang dan membuat mikrokontroller yang digunakan untuk mengendalikan seluruh sistem.
3. Bagaimana merancang perangkat lunak yang digunakan

1.3. Batasan Masalah

Dalam penyusunan skripsi ini pembahasan dibatasi meliputi hal berikut ;

1. Audio yang disimpan harus berupa PCM data 8 bit (file yang ber-extention WAV)
2. Tidak membahas tentang FAT (File Allocation Table)
3. Tidak membahas Power Supply

1.4. Tujuan

Tujuan dari penyusunan skripsi ini adalah untuk merancang dan membuat alat pemutar musik yang lebih efisien, efektif dan ekonomis.

1.5.. Metodologi Penelitian

Metodologi yang digunakan dalam penyusunan skripsi ini adalah sebagai berikut ;

1. Studi literature tentang proses penghitungan, cara kerja mikorokontroller, MMC, yang meliputi pengumpulan, dan analisa data.
2. Perancangan dan pembuatan alat ;
    - Pembuatan Hardware
    - Sistem pengolahan data pada Mikrokontroller ATMEGA 8
3. Pengujian alat yang meliputi
    - Cara kerja MMC (MULTI MEDIA CARD)
    - Mikrokontroller AMEGA 8, keypad dan LCD
    - Sinyal audio digital PWM (Pulse Width Modulation)

- LPF (Low Pass Filter) untuk mengubah sinyal PWM menjadi sinyal audio analog.
- Sistem alat keseluruhan.

1.6. Sistematika Tulisan

Sistematika dari penulisan Skripsi ini adalah

- BAB I ; Pendahuluan

  Membahas tentang latar belakang permasalahan , tujuan dan

  Manfaat, batasan masalah, dan sistematika tulisan.
- BAB II ; Teori Penunjang

  Merupakan teori penunjang dari alat dan program yang dibuat.
- BAB III , Perancangan dan Pembuatan Program dan Alat

  Membahas tentang perancangan alat dan program yang digunakan , mulai dari pembuatan hardware, software. dan cara kerja alat.
- BAB IV ; Pengujian Program dan Alat

  Membahas pengujian alat program yang telah selesai dengan

  menjalankan program tersebut dan mengamati hasilnya.
- BAB V ; Kesimpulan

  Membahas kesimpulan dari pengujian suatu alat dan program

BAB II

LANDASAN TEORI

Landasan teori sangat membantu untuk dapat memahami suatu sistem. Landasan teori juga dapat digunakan sebagai acuan di dalam merencanakan suatu sistem. Dengan pertimbangan hal-hal tersebut, maka landasan teori merupakan bagian yang harus dipahami untuk pembahasan lebih lanjut. Dalam landasan teori ini akan dibahas teori dasar yang berhubungan dengan Mikrokontroller ATmega 8, MMC, Filter, PWM, Amplifier, Speaker dan LCD.

2.1. Mikrokontroller AVR ATMega8

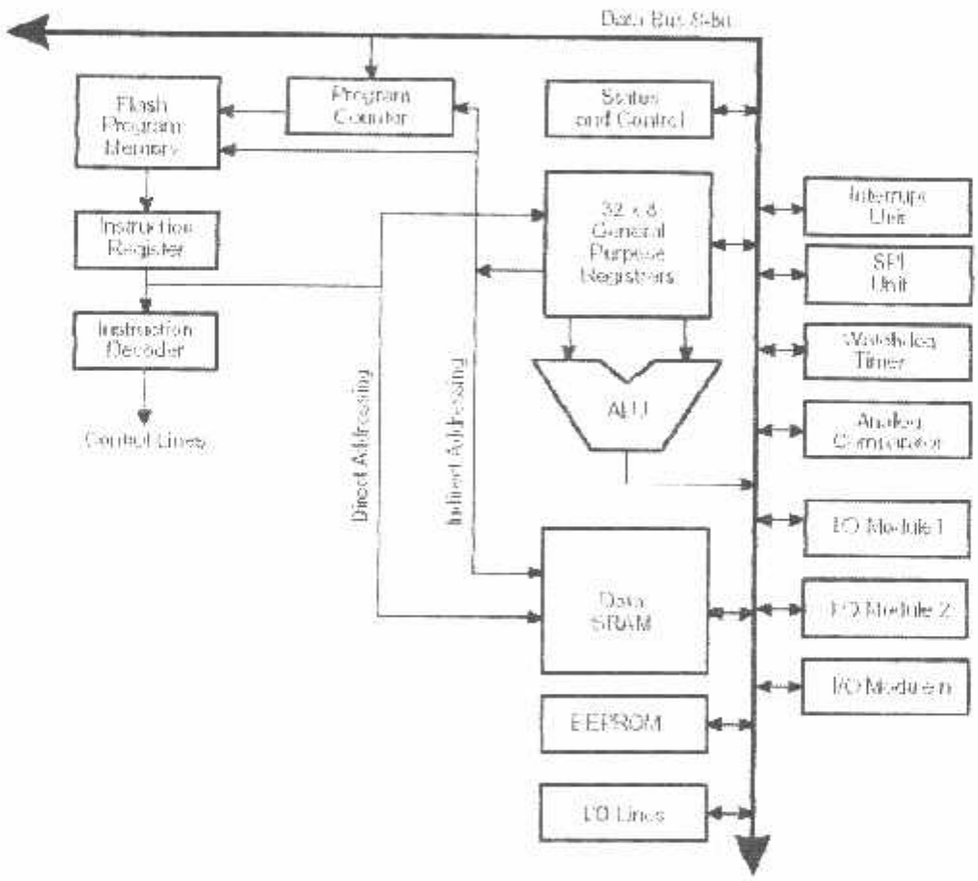
2.1.1. Arsitektur

AVR Atmega8 adalah mikrokontroler 8-bit CMOS, *low-power* yang berdasarkan pada bentuk arsitektur AVR RISC (*Reduced Instruction Set Komputer*), yang hampir semua instruksinya selesai dikerjakan dalam satu siklus *clock*. AVR ATmega8 menggunakan instruksi tunggal (*Single Clock Cycle*), yaitu sistem mikrokontroler yang frekuensi kerja dalam chip sama dengan frekuensi kristal untuk osilator tanpa memerlukan rangkaian pembagi frekuensi setelah osilator yang diperlukan untuk memperoleh perbedaan fase dari *clock*, sehingga AVR 12 kali lebih cepat dibanding MCS51.

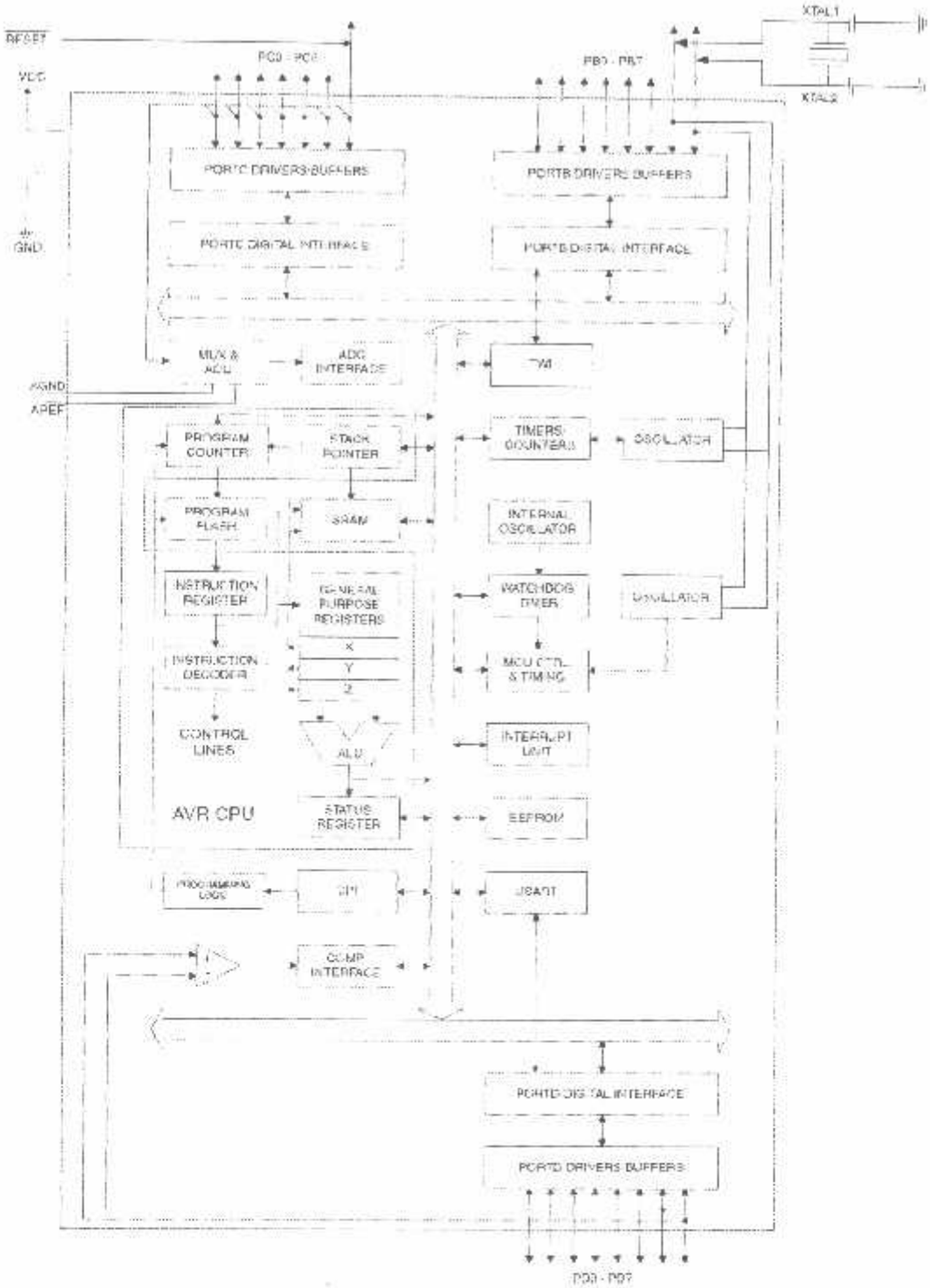
Berbagai karakteristik yang tersedia dalam IC ATmega8 adalah sebagai berikut:

- 8K *bytes In-Sistem Programable Flash*
- 512 *bytes* EEPROM (*Electrical Erasable Programable Read Only Memory*)

- 512 *bytes* SRAM (*Static Random Access Memory*)
- 23 jalur I/O *general-purpose*
- 32 x 8 *general-purpose working register*
- *Timer/ Counter* yang fleksibel dengan *mode* pembanding
- Interupsi internal dan eksternal
- Pemrograman serial UART (*Universal Asynchronous Receiver and Transmitter)*
- *Serial Port* SPI (*Serial Peripheral Interface*)



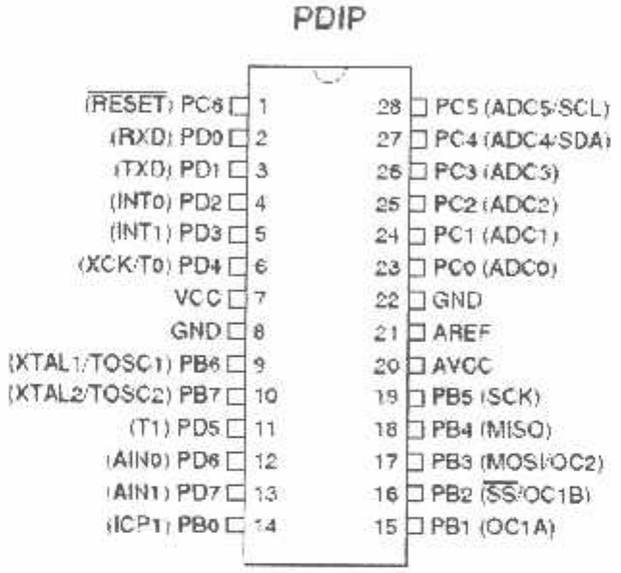
Gambar 2.1 Arsitektur AVR ATmega8[1]



Gambar 2.2 Blok Diagram AVR ATmega8[11]

2.1.2. Konfigurasi Pin-Pin Mikrokontroller ATmega8

Mikrokontroller ATmega8 mempunyai 28 pin seperti pada gambar di bawah ini:



Gambar 2.3 Konfigurasi Pin ATmega8[1]

Fungsi tiap pin-nya adalah sebagai berikut:

a. Vcc: Tegangan *Supply*

b. Gnd: *Ground*

c. *Port* A (PA0-PA7): *Port* dua arah I/O 8-bit, kaki *port*nya dapat menyediakan *resistor pull-up* internal (dipilih untuk masing-masing bit). *Port* A juga dapat mengendalikan tampilan LED secara langsung.

d. *Port* B (PB0-PB7: *Port* dua arah I/O 8-bit dengan *resistor pull-up internal*, digunakan pada fungsi-fungsi khusus dari karakteristik ATmega8.

e. *Port* C (PC0-PC7): *Port* dua arah I/O 7-bit dengan *resistor pull-up internal*.

f. PC6/$\overline{\text{RESET}}$: Jika fuse RSTDISBL sudah diprogram, PC6 digunakan sebagai suatu pin I/O. Jika fuse RSTDISBL belum diprogram, PC6 digunakan sebagai inputan *reset* dimana *level low* dari pin ini lebih panjang dari pulsa minimum yang dihasilkan *reset*.

g. *Port* D (PD0-PD7): *Port* dua arah /O 7-bit dengan *resistor pull-up internal*. Sebagai input, *port* D menggunakan eksternal *pull low* dengan sumber arus jika *pull up resistor* diaktifkan.

h. $\overline{\text{RESET}}$: Input *reset*. *Level low*-nya untuk lebih panjang dari pulsa minimum yang dihasilkan *reset*, meskipun *clock* tidak bekerja.

i. AV_{CC}: sebagai *supplay* tegangan untuk A/D *konverter port* C (3..0), dan ADC (7..6). Pin ini harus dihubungkan dengan V_{CC} melalui *low-pass filter*.

j. AREF: Pin analog referensi untuk A/D *konverter*.

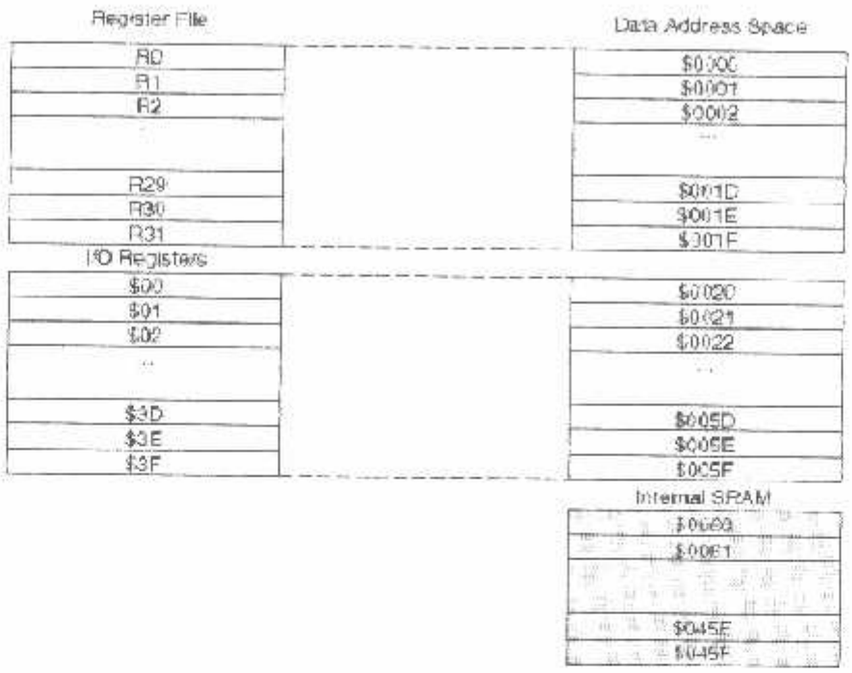
k. ADC 7..6 (TQFT and MLF): Pada TQFP dan MLP, ADC 7..6 bekerja sebagai input *analog* untuk A/D *konverter*. Pin-pinnya mendapat daya dari power *supplay analog* dan dapat melayani 10 bit saluran ADC.

2.1.3. Peta Memori

AVR ATmega8 memiliki ruang pengalamatan memori data dan memori program yang terpisah. Memori data terbagi menjadi 3 bagian, yaitu 32 register umum, 64 buah register I/O, dan 1024 *byte* SRAM *Internal*.

Register keprluan umum menempati *space* data pada alamat terbawah, yaitu $00 sampai$1F. Sementara itu, register khusus unutk menangani I/O dan kontrol terhadap mikrokontroller menempati 64 alamat berikutnya, yaitu mulai dari $20 hingga $5F. register tersebut merupakan register yang khusus digunakan untuk mengatur fungsi terhadap peripheral mikrokontroller, seperti kontrol

register, *timer/ counter*, fungsi-fungsi I/O, dan sebagainya. Alamat memori berikutnya digunakan untuk SRAM 1024 *byte*, yaitu pada lokasi $60 sampai dengan $45F. Konfigurasi memori data ditunjukkan pada gambar dibawah ini.

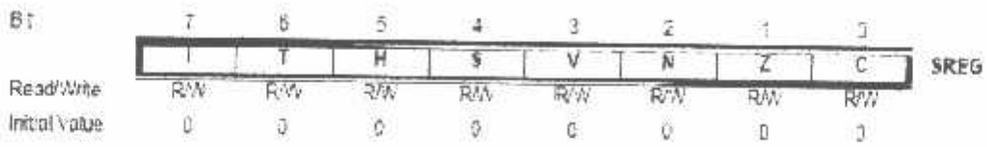


Gambar 2.4 Konfigurasi Memori Data AVR ATmega8[1]

Memori program yang terletak dalam *Flash Sistem Reprogrammable Flash* mempunyai 8K *byte* karena setiep instruksi memiliki lebar 16-bit atau 32-bit. AVR ATmega memiliki 4K *byte* x 16-bit *Flash* dengan alamat mulai dari $00 sampai $FFF. AVR tersebut memiliki 12-bit *Program Counter* sehingga mampu mengalamati isi *Flash*. Selain itu,AVR ATmega8 juga memiliki memori data berupa EEPROM 8-bit sebanyak 512 *byte*. Alamt EEPROM dimulai dari $000 sampai $1FF.

2.1.4. Status Register (SREG)

Status register adalah register berisis status yang dihasilkan pada setiap operasi yang dilakukan ketika suatu instruksi dieksekusi. SREG merupakan bagian dari inti CPU mikrokontroller.



Gambar 2.5 Status Register Atmega8[1]

Keteranga dari bit SREG adalah:

a. Bit7 - I: *Global Interrupt Enable*

   Bit harus diset untuk meng-*enable* interupsi. Setelah itu, baru dapat mengaktifkan interupsi mana yang akan digunakan dengan cara meng-*enable* bit kontrol register yang bersangkutan secara individu. Bit akan di-*clear* apabila terjadi suatu interupsi, serta akan diset kembali oleh intruksi RETI.

b. Bit 6 – T: *Bit Copy Storage*

   Intruksi BLD dan BST menggunakan bit-T sebagai sumber atau tujuan dalam operasi bit. Suatu bit dalam suatu register GPR dapat disalin ke bit T menggunakan instrusi BST, dan sebaliknya bit T dapat disalin kembali ke suatu bit dengan register GPR menggunakan instruksi BLD.

c. Bit 5 – H: *Half Carry Flag*

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

Bit-S merupakan hasil operasi EOR antara *flag*-N (*negative*) dan *flag* V (komplemen dua *overflow*).

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

Bit berguna untuk mendukung operasi aritmatika.

f. Bit 2 – N: *Negative Flag*

Apabila suatu operasi menghasilkan bilangan negative, maka *flag*-N akan diset.

g. Bit 1 – Z: *Zero Flag*

Bit akan diset bila hasil operasi yang diperoleh adalah nol.

h. Bit 0 – C: *Carry Flag*

Apabila suatu operasi menghasilkan *carry*, maka bit akan diset.]

2.1.5. Register I/O

Semua *port* pada AVR memiliki kebenaran fungsional *read-modify-write* ketika digunakan sebagai *port* I/O umum. Ini berarti bahwa arah dari satu pin *port* dapat diubah tanpa bermaksud mengubah arah dari pin yang lain. Logika *port* I/O dapat diubah-ubah dalam program secara *byte* atau hanya bit tertentu. Mengubah sebuah keluaran bit I/O dapat dilakukan menggunakan perintah cbi (clear bit I/O) untuk menhasilkan output *low* dan perintah sbi (set bit I/O) untuk menghasilkan output *high*. Pengubahan secara *byte* dilakukan dengan perintah *in* atau *out* yang menggunakan register bantu.

a. *Port* A

Tiga lokasi alamat memori I/O dilokasikan pada *port* A, masing-masing adalah register data-*Port* A, $1B ($3B), register data *direction* (register pengarah data)-DDRA, $1A ($3A), dan pin input *port* A-PIN A, $19

($39). Pin-pin *port* A memiliki fungsi alternatif yang terhubung pada plihan data eksternal SRAM. *Port* A dapat dikonfigurasikan menjadi multiplexed *low order* alamat/data bus selama akses ke data memori eksternal. (blok diagram dapat dilihat pada lampiran).

b. *Port* B

Tiga lokasi alamat memori I/O yang dilokasikan pada *port* D, masing-masing adalah register data-*port* B, $18 ($38), register pengarah data-DDRB, $17 ($37), dan pin input *port* B-PINB, $16 ($36). (Blok diagram *port* B dan fungsi *Timer/ Counter 2* pinnya dapat dilihat pada lampiran).

c. *Port* C

Tiga lokasi alamat memori I/O yang dilokasikan pada *port* C, masing-masing adalah register data-*port* C, $15 ($35), register pengarah data-DDRC, $14 ($34), dan pin input *port* C-PINC, $13 ($33). (Blok diagram skematik dapat dilihat pada lampiran).

d. *Port* D

Tiga lokasi alamat memori I/O yang dilokasikan pada *port* D, masing-masing adalah register data-*port* D, $12 ($32), register pengarah data-DDRD, $11 ($31), dan pin input *port* D-PIND, $10 ($30). (Blok diagram skematik *port* D dan fungsi alternatif pinnya dapat dilihat pada lampiran).

2.1.6. Timer/ Counter 2

Timer/ Counter 2 adalah 8 bit Timer/ Counter yang multifungsi. Deskripsi Timer/ counter 2 pada ATmega8 adalah sebagai berikut:

a. Sebagai *Counter* 1 kanal.

b. *Timer* dinolkan saat *match compare* (*auto reload*).

c. Dapat menghasilkan gelombang PWM dengan *glitch-free*.

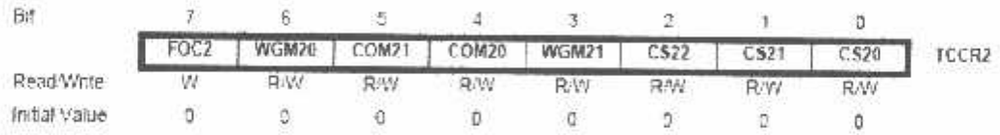
d. *Frekuensi generator*.

e. *Prescaler* 10 bit untuk timer.

f. Interupsi *timer* yang disebabkan *timer overflow* dan *match compare* (TOV2 dan OCF2).

g. Dapat menggunakan *clock* dari kristal *independent* luar sebesar 32 kHz pada I/O *clock*.

Pengaturan Timer/*Counter*2 diatur oleh TCCR2 (*Timer/ Counter* Kontrol Register 0) yang dapat dilihat pada gambar berikut:



Gambar 2.6 Register TCCR2[11]

Penjelasan untuk setiap bit adalah:

a. bit 7 – FOC2: *Force Output Compare*

b. bit 6,3 – WGM21: WGM21:WGM20: *Waveform Generator Unitl*.

Bit tersebut mengontrol kenaikan *counter*, sumber dari nilai maksimum *counter*, dan tipe dari jenis *Timer/ Counter* yang dihasilkan, yaitu *mode normal, clear timer, mode compare match*, dan dua tipe dari PWM (*Pulse Width Modulation*). Berikut tabel *setting* pada bit untuk menghasilkan *mode* tertentu:

Tabel 2.1. Konfigurasi Bit WGM21 dan WGM20[1]

Mode	WGM21 (CTC2)	WGM20 (PWM2)	Timer/Counter Mode of Operation[1]	TOP	Update of OCR2	TOV2 Flag Set
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

c. Bit 5,4 – COM01:COM00: *Compare Match Output Mode*

Bit mengontrol pin OC0 (*Ouput Compare Pin*). Apabila kedua bit tersebut nol atau *clear*, maka pin CO0 berfungsi sebagai pin biasa. Namun, jika salah satu bit *set*, maka fungsi bit tergantung pada *setting* bit pada WGM00 dan WGM01. Berikut daftar tabel *setting* bit pada WGM00 dan WGM01:

Tabel 2.2. Konfigurasi Bit COM21 dan COM20 *Compare Output Mode* non-PWM[1]

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

Tabel 2.3. Konfigurasi Bit COM21 dan COM20 Compare Output *Mode* Fast PWM[1]

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

Tabel 2.4. Konfigurasi Bit COM21 dan COM20 *Compare Output Phase Correct* PWM[1]

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 downcounting.
1	1	Set OC2 on Compare Match when up-counting. Clear OC2 on Compare Match when downcounting.

d. Bit 2, 1, 0 – CS22, CS21, CS20

Ketiga bit tersebut memilih sumber *clock* yang akan digunakan oleh Timer/*Counter*. Berikut *list* tabelnya:

Tabel 2.5. Konfigurasi Bit *Clock Select* Untuk Memilih Sumber *Clock*[1]

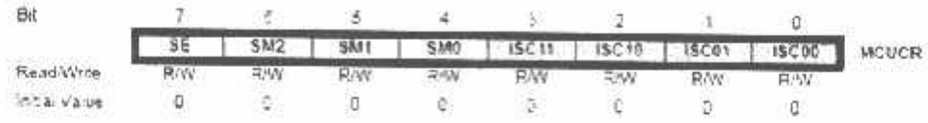
CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler)
1	1	1	clk_{T2S}/1024 (From prescaler)

2.1.7. Interupsi

Interupsi adalah kondisi yang membuat CPU berhenti dari rutinitas yang sedang dikerjakan (rutin utama) untuk mengerjakan rutin lain (rutin intrupsi). AVR ATMega8 memiliki 19 sumber interupsi.

a. Pada AVR terdapat 3 pin untuk interupsi eksternal, yaitu INT0, INT1, INT2. Interupsi eksternal dapat dibangkitkan apabila terdapat perubahan logika atau logika 0 pada pin interupsi. pengaturan kondisi

keadaan yang menyebabkan terjadinya interupsi eksternal diatur oleh register MCUCR (MCU Kontrol Register). Yang terlihat pada gambar dibawah ini:



Gambar 2.7 Register MCUCR[1]

b. Bit ISC11 dan ISC10 bersama-sama menentukan kondisi yang dapat menyebabkan interupsi eksternal pada pin INT1. Keadaan selengkapnya dapat dilihat pada tabel dibawah ini:

c.

Tabel 2.6. Beberapa *Setting* Kondisi Yang Menyebabkan Interupsi Eksternal I[1]

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.

d. Bit ISC01 dan ISC00 bersama-sama menentukan kondisi yang dapat menyebabkan interupsi eksternal pada pin INT0. keadaan selengkapnya dapat dilihat pada tabel dibawah ini:

e.

Tabel 2.7 Beberapa *Setting* Kondisi Yang Menyebabkan Interupsi Eksternal 0[1]

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.

Pemilihan pengaktifan interupsi eksternal diatur oleh register GICR (General Interupsi Kontrol Register) yang terlihat seperti gambar berikut:

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

Gambar 2.8. General Interrupt Kontrol Register[1]

Bit penyusun dapat dijelaskan sebagai berikut:

a. Bit INT1 adalah bit untuk mengaktifkan intrupsi eksternal 1. apabila bit tersebut diberi logika 1 dan bit-I pada SREG (status register) juga satu, maka interupsi eksternal 1 akan aktif.

b. Bit INT0 adalah bit untuk mengaktifkan intrupsi eksternal 0. apabiala bit tersebut diberi logika 1 dan bit-I pada SREG (status register) juga satu, maka interupsi eksternal 0 akan aktif.

c. Bit INT2 adalah bit untuk mengaktifkan interupsi eksternal 2 apabila bit tersebut diberi logika 1 dan bit-I pada SREG (status register) juga satu, maka eksternal 2 akan aktif.

Program interupsi dari masing-masing jenis interupsi eksternal akan dimulai dari vektor interupsi pada masing-masing jenis. Alamatnya dapat dilihat pada tabel:

Tabel 2.8. Macam Sumber Interupsi pada AVR Atmega8[1]

Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x000(1)	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	SPM_RDY	Store Program Memory Ready

Untuk inisialiasasi awal interupsi, perlu dituliskan terlebih dahulu vektor interupsi dari interupsi yang terdapat pada sistem. Vektor interupsi adalah nilai yang disimpan ke *program counter* pada saat terjadi interupsi sehingga program akan menuju ke alamat yang ditunjukkan oleh *program counter*. alamat interupsi eksternal 0 pada alamat 001H dan interupsi terima serial pada alamat 00B masing-masing alamat vektor memililki jarak yang berdekatan sehingga akan timbul masalah jika diperlukan rutin layanan interupsi yang panjang oleh sebab itu

layanan interupsi ekternal 0 akan melompat ke alamat *ext int0* dan inetrupsi terima serial pada alamat USART RXC.

Pengaktifan interupsi eksternal dilakukan dengan memberikan logika satu pada register GICR. Dengan demikian, pada pengaktifan interupsi ekternal 0, akan diberikan logika satu pada bit ke 6 register GICR. pengaktifan interupsi terima serial dilakukan dengan memberikan logika 1 pada bit ke 7 register UCSRA. Terakhir, berikan perintah sei untuk menagaktifkan *global interupt*.

Interupsi dapat muncul kapan pun (kecuali jika bit *enable interupsi* dalam SREG *clear*) dengan demikian, interupsi juga dapat mencul ketika program sedang melakukan kalkulasi. Kalkulasi tersebut merubah *flags* dalam status register yang digunakan untuk *next step* dari kalkulasi atau untuk beberapa percabangan program. Jika ISR mengubah *flags* dalam SREG, maka kalkulasi yang sedang ditempatkan dalam program yang berjalan normal dapat di-*corupt*. Oleh sebab itu, perlu pengamanan SREG pada setiap subrutin interupsi.

2.1.8. Register TIMSK

Selain register di atas terdapat pula register TIMSK (*Timer/ Counter* Interrupt Mask Register).

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	TIMSK
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	

Gambar 2.9 Register TIMSK[1]

Penjelasan Untuk setiap bit adalah:

a. Bit 0 – TOIE0: *Timer/ Counter* 0 *Overflow Interrup Enable*

   Jika bit tersebut diberi logika satu dan bit i SREG juga set, maka dilakukan *enable* interupsi Overflow *Timer/ Counter* 0.

b. Bit 1 – OCIE0: *Timer/ Counter* 0, *Output Compare Match Interrupt Enable*

Jika bit tersebut diberi logika satu dan bit i SREG juga set, maka bias dilakukan *enable* interupsi *Output Compare Match Timer/ Counter* 0.

c. Bit 2 – TOIE1: *Timer/ Counter* 1 *Overflow Interrup Enable*

Jika bit tersebut diberi logika satu dan bit i SREG juga set, maka dilakukan *enable* interupsi Overflow *Timer/ Counter* 1.

d. Bit 3 – OCIE1B: *Timer/ Counter* 1, *Output Compare* B *Match Interrupt Enable*

Jika bit tersebut diberi logika satu dan bit i SREG juga set, maka dilakukan *enable* interupsi Overflow Compare Match B *Timer/ Counter* 1.

e. Bit 4 – OCIE1A: *Timer/ Counter* 1, *Output Compare* A *Match Interrupt Enable*

Jika bit tersebut diberi logika satu dan bit i SREG juga set, maka dilakukan *enable* interupsi *Overflow Compare Match* A *Timer/ Counter* 1.

f. Bit 5 – TICIE1: *Timer/ Counter* 1 *Input Capture Interrupt Enable*

g. Bit 6 – TOIE2: *Timer/ Counter* 2, *Overflow Interrupt Enable*

Jika bit tersebut diberi logika satu dan bit i SREG juga set, maka dilakukan *enable* interupsi *Overflow Timer/ Counter* 2.

h. Bit 7 – OCIE2: *Timer/ Counter* 2, *Output Compare Match Interurpt Enable*

Jika bit tersebut diberi logika satu dan bit i SREG juga set, maka bias dilakukan *enable* interupsi *Output Compare Match Timer/ Counter* 2.

2.1.9. Osilator

Sumber clock dapat diatur dengan dua cara yaitu osilator internal dan osilator eksternal. Pengaturan osilator eksternal dilakukan dengan menambahan kristal keramik sesuai kebutuhan. Untuk osilator internal Atmega8 memiliki 4 nilai yaitu 1, 2, 4, 8 MHz. Penggunaan osilator internal menggunakan register OSCCAL, untuk 1 MIz alamat regiternya adalah 0X0000, untuk 2 MHz alamat regiternya adalah 0X0001, untuk 4 MHz alamat regiternya adalah 0X0002, untuk 8 MHz alamat regiternya adalah 0X0003.

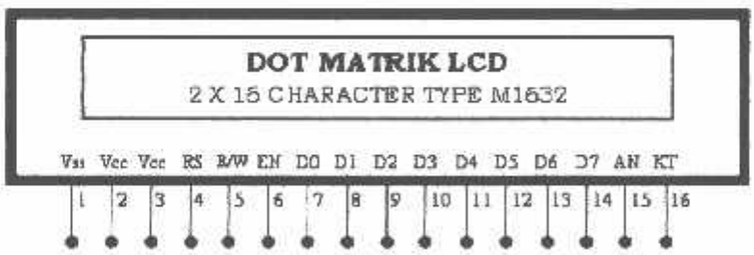
2.2. LCD (Liquid Crystal Display)

Liquid Crystal Display atau LCD merupakan komponen optoelektronik yaitu komponen yang bekerja atau dipengaruhi oleh sinar (optolistrik), komponen pembangkit cahaya (light emiting) dan komponen-komponen yang akan mengubah sinar. LCD terbuat dari bahan kristal cair yang merupakan suatu komponen organik dan mempunyai sifat optik seperti benda padat meskipun bahan tetap cair.

Sel kristal cair terdiri dari selapis bahan kristal cair yang diapit antara dua kaca tipis yang transparan. Antara dua lembar kaca tersebut diberi bahan kristal cair *(liquid crystal)* yang tembus cahaya. Permukaan luar dari masing-masing keping kaca mempunyai lapisan penghantar tembus cahaya seperti oxida timah *(tin oxide)* atau oxida indium *(indium oxide)*. Sel mempunyai ketebalan sekitar 1 x 10^{-5} meter dan diisi dengan kristal cair.

Karena sel-sel kristal cair merefleksikan cahaya dan bukan membangkitkan cahaya maka konsumsi daya yang dibutuhkan relatif rendah. Energi yang dipergunakan hanya untuk mengaktifkan kristal cair. Pada dasarnya LCD bekerja pada tegangan rendah (3 – 15 Vrms), frekuensi rendah (25 – 60 Hz) sinyal AC dan memakai arus listrik yang sangat kecil (25 - 300 μA). LCD seringkali ditata sebagai tampilan *seven segment* untuk menampilkan angka tetapi juga memiliki keistimewaan lain, yaitu kemampuan untuk menampilkan karakter dan berbagai macam simbol.

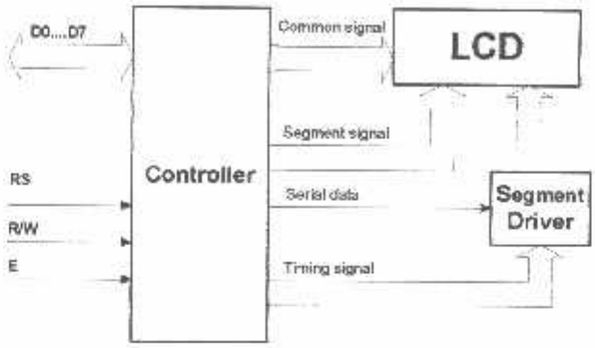
Salah satu jenis LCD diantaranya adalah LCD M1632, suatu jenis piranti dengan konsumsi daya yang rendah, disusun dari dot matrik dan dikontrol oleh ROM atau RAM generator karakter dan RAM data display. Pengontrolan utamanya adalah pada ROM generator dan display data RAM yang menghasilkan kode ASCII jika padanya diberikan input ASCII. Untuk dapat difungsikan dengan baik maka perlu diperhatikan proses analisasi yang telah ditentukan oleh pabrik pembuatnya. Timing penganalisasian sangat dipertimbangkan, karena jika meleset sampai ordo *milisecon* maka dapat dipastikan LCD tidak dapat berfungsi.



Gambar 2.10 Konfigurasi Kaki LCD[8]

Masukan yang diperlukan untuk mengendalikan modul berupa bus data yang masih termutiplek dengan bus alamat serta 3 bit sinyal kontrol. Sementara

pengendalian LCD dilakukan secara internal oleh kontroler yang sudah terpasang dalam modul LCD. Diagram blok untuk LCD dapat dilihat dalam Gambar 2.11



Gambar 2.11 Blok Diagram LCD[8]

Adapun karakteristik dari LCD M1632 antara lain :

- Dengan 16 karakter – 2 baris dalam bentuk dotmatrik 5x7 dan cursor
- *Duty ratio* 1/16
- Memiliki ROM pembangkitan karakter untuk 192 jenis karakter
- RAM untuk data display sebanyak 80x8 bit
- Dapat dirangkai dengan MPU 8 bit/4 bit
- RAM data display dan RAM pembangkit karakter dapat dibaca oleh MPU
- Memiliki fungsi instruksi antara lain *display on/off, Cursor on/off, display* karakter *blink, cursor shift* dan *display shift*
- Memiliki rangkaian osilator sendiri
- Catu tegangan tunggal yaitu ± 5 V
- Memiliki rangkaian reset otomatis pada catu daya yang dihidupkan
- Temperatur operasi 0^0 - 50^0

LCD memiliki 16 pin yang masing-masing mempunyai fungsi sebagai berikut :

Tabel 2.9. Fungsi Tiap Pin LCD[8]

No. Pin	Simbol	Level	Fungsi	
1	V_{SS}	-	Power Supply	0 V (GND)
2	V_{CC}	-		5 V ± 10%
3	V_{DD}	-		For LCD Drive
4	RS	H/L	Sinyal seleksi register H ; Data Input [register data (write/read)] L ; Instruction Input [register instruksi (write), busy flag dan address counter (read)]	
5	R/W	H/L	H ; Read L ; Write	
6	E	H	Enable Signal [sinyal penanda mulai operasi, aktif saat operasi write atau read]	
7	DB0	H/L	4 bit bus data lower 2 arah, dapat dibaca atau ditulis terhadap mikrokontroler	
8	DB1	H/L		
9	DB2	H/L		
10	DB3	H/L		
11	DB4	H/L	4 bit bus data upper 2 arah, dapat dibaca atau ditulis terhadap mikrokontroler, DB7 juga sebagai busy flag	
12	DB5	H/L		
13	DB6	H/L		
14	DB7	H/L		

15	V+BL	-	Back Light Supply	4 - 4,2 V 50 – 200 mA
16	V-BL	-		0 V (GND)

2.2.1. Instruksi Operasi

Tabel 2.10. Instruksi Pada LCD[8]

Instruksi	**RS**	**RW**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Display Clear	0	0	0	0	0	0	0	0	0	1
Cursor Home	0	0	0	0	0	0	0	0	1	*
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display On/Off	0	0	0	0	0	0	1	D	C	B
Cursor Display Shift	0	0	0	0	0	1	S/C	R/L	*	*
Function Set	0	0	0	0	1	DL	1	*	*	*
CG RAM Address Set	0	0	0	1	A_{CG}					
DD RAM Address Set	0	0	1	A_{DD}						
BF/Address Read	0	1	BF	AC						
Data Write to CG RAM	1	0	Write Data							
Data Read from CG RAM	1	1	Read Data							

*Invalid Bit

A_{CG} ; CG RAM Address

A_{DD} ; DD RAM Address

Sumber: Liquid Crystal Display Module M1632 User Manual

2.2.2. Operasi Dasar

2.2.2.1 Register

Kontrol dari LCD memiliki 2 buah register 8 bit yaitu register instruksi (IR) dan register data (DR). IR memiliki instruksi seperti display, clear, cursor shift dan display data (DD RAM) serta karakter (CG RAM). DR menyimpan data untuk ditulis ke DD RAM ataupun membaca data dari DD RAM dan CG RAM. Ketika data ditulis ke DD RAM atau CG RAM maka DR secara otomatis menulis data ke DD RAM atau CG RAM. Ketika data pada CG RAM atau DD RAM akan dibaca maka alamat data ditulis pada IR. Sedangkan data akan dimasukkan melalui DR sehingga dapat dibaca oleh mikrokontroler.

Tabel 2.11. Tabel Register Pada LCD[8]

RS	RW	Operasi
0	0	Seleksi IR, IR Write Display Clear
0	1	Busy Flag (DB7), @ Counter (DB0-DB7) Read
1	0	Seleksi DR, DR Write
1	1	Seleksi DR, DR Read

2.2.2.2 Busy Flag

Busy Flag menunjukkan bahwa modul siap untuk menerima instruksi selanjutnya sebagaimana terlihat pada tabel diatas. Register seleksi sinyal akan

melalui DB7 jika RS=0 dan R/W=1. Jika bernilai 1 maka sedang melakukan kerja internal dan instruksi tidak akan dapat diterima, oleh karena itu status dari flag harus diperiksa sebelum melaksanakan instruksi selanjutnya.

2.2.2.3. Address Counter (AC)

AC menunjukkan lokasi memori dalam modul LCD. Pemilihan lokasi alamat lewat Ac diberikan lewat register instruksi (IR) ketika data pada A, maka AC secara otomatis menaikkan atau menurunkan alamat tergantung dari Entry Mode Set.

2.2.2.4. Display Data RAM

Pada LCD, masing-masing line memiliki range alamat tersendiri. Alamat itu diekspresikan dengan bilangan hexadesimal. Untuk line 1 range alamat berkisar antara 40_H-$4F_H$.

2.2.2.5. Character Generator ROM (CG ROM)

CG ROM memiliki tipe dot matrik 5x7, dimana pada LCD telah tersedian ROM sebagai pembangkit karakter dalam kode ASCII.

2.2.2.6. Character Generator RAM (CG RAM)

CG RAM dipakai untuk pembuatan karakter tersendiri melalui program.

CG RAM digunakan untuk pembuatan karakter tersendiri melalui program.

Tabel 2.12 Fungsi Terminal Pada LCD[8]

Nama Signal	Jml Term	I/O	Tujuan	Fungsi
DB0-DB3	4	I/O	MPU	Sebagai lalu lintas data dan intruksi ke atau dari MPU *Low Byte*

DB4-DB7	4	I/O	MPU	Sebagai lalu lintas data atau intruksi 2 arah *upper byte*. DB7 sebagai *busy flag*
E	1	I	MPU	Sinyal Start *(read/write)*
R/W	1	I	MPU	Seleksi Sinyal *0 = write* *1 = read*
RS	1	I	MPU	Seleksi Register
VLS	1	-	PS	0 = *intruksi reg* (wr) *Busy flag addr counter* (rd) 1 – *data reg* (wr dan rd)
7	1	-	PS	Mengatur Tampilan LCD
Vss	1	-	PS	+5 volt

(Sumber : LCD Manual Data Book)

Adapun bentuk fsik dari LCD M1632 adalah pada gambar berikut :

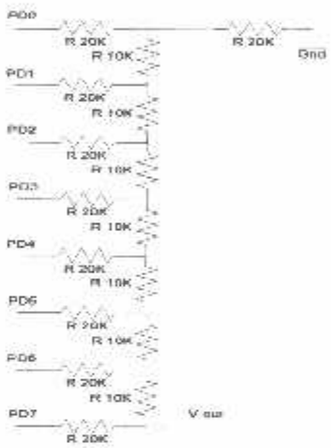
Gambar 2.12 Liquid Crystal Display[8]

2.3. Digital To Analog Converter (DAC)

Keluaran-keluaran yang berbentuk sinyal analog dari suatu system computer dapat diperoleh dengan menggunakan converter digital ke analog, yang luas

dikenal dengan istilah DAC *(Digital to Analoque Converter)*. DAC akan mengkonversi sebuah sinyal digital menjadi bentuk sinyal analog.

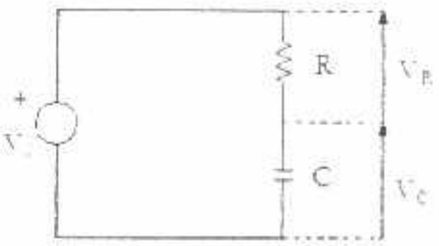
Salah satu susunan rangkaian *converter* digital ke analog adalah DAC dengan pembobotan biner (R-2R). Dalam implementasi pembobotan biner kedua, hanya dua nilai resistor yang berbeda yang digunakan untuk memperoleh arus-arus pembobotan biner. Seperti tampak pada gambar 2.11 implementasi DAC pembobotan biner dapat dilakukan dengan menggunakan rangkaian R-2R, arus-arus pembobotan biner dapat mengalir ke resistor umpan balik atau ke terminal *ground*. arus yang mengalir ke resistor umpan balik akan berkontribusi pada tegangan keluaran rangkaian.



Gambar 2.13 Rangkaian R-2R

Keuntungan dari rangkaian R-2R adalah hanya memerlukan dua nilai resistor. pencapaian kondisi matching di antara sejumlah resistor yang memiliki nilai R dan 2R akan lebih mudah.

2.4. Filter RC (Resistor Capasitor)

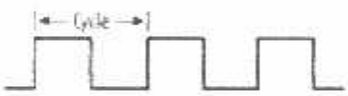


Gambar 2.14 Rangkaian filter RC

Rangkaian filter RC ini mengacu pada sistem pengolahan data MMC yang berupa sinyal digital menjadi sinyal analog melalui sistem PWM (Pulse Width Modulation) dengan tegangan 5 volt

2.5. Pulse Width Modulation (PWM)

PWM (Pulse Width Modulation) adalah teknik mendapatkan efek sinyal analog dari sebuah sinyal digital yang terputus-putus. PWM dapat dibangkitkan hanya dengan menggunakan digital i/o yang difungsikan sebagai output.



Gambar 2.15 Gelombang PWM dengan duty cycle 50%

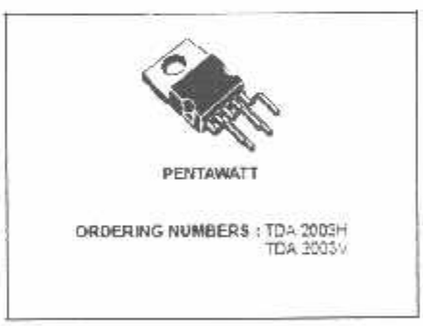
Pada contoh di atas, sinyal high (1) dan sinyal low (0), waktunya sama dengan duty cycle 50%. Jika amplitudo PWM 5 volt, maka tegangan rata-rata 2,5 volt. Berikut di bawah ini juga ditampilkan gambar gelombang jika duty cyclenya 10%

Gambar 2.16 Gelombang PWM dengan duty cycle 10%

Pada ATMEGA8 ada 2 cara membangkitkan PWM, yang pertama PWM dapat dibangkitkan dari port input/outputnya yang difungsikan sebagai output. Yang kedua adalah dengan memanfaatkan fasilitas PWM dari fungsi timer/counter yang telah disediakan. Dengan adanya fasilitas ini proses pengaturan waktu high/low sinyal digital tidak akan mengganggu urutan program lain yang sedang dieksekusi oleh processor. Selain itu, dengan menggunakan fasilitas ini kita tinggal memasukkan berapa porsi periode waktu on dan off gelombang PWM pada sebuah register. **OCR1A**, **OCR1B** dan **OCR2** adalah register tempat mengatur duty cycle PWM.

2.6. Amplifier TDA 2003

TDA 2003 merupakan pengembangan dari konfigurasi TDA 2002, yaitu dengan penambahan fiture seperti pengurangan jumlah komponen eksternal , biaya perawatan murah. Kapasitas arus keluran tinggi, 3.5 ampere sehingga tidak terjadi distorsi cross over. Melindungi dari arus pendek DC maupun AC. Load-dump voltage surge up 40 volt dan fortuitous open



Gambar 2.17 Amlpifier TDA 2003[7]

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Peak supply voltage (50msl	40	V
V_S	DC supply voltage	28	V
V_S	Operating supply voltage	18	V
I_O	Output peak current (repetitive)	3.5	A
I_O	Output peak current (non repetitive)	4.5	A
Ptot	Power dissipation at Tcase = 90°C	20	W
T_{stg}, T_j	Storage and junction temeperature	-40 to 150	°C

Tabel 2.13 Absolut Maximum Rating[7]

2. 7. Multi Media Card (MMC)

2.7.1. Arsitektur

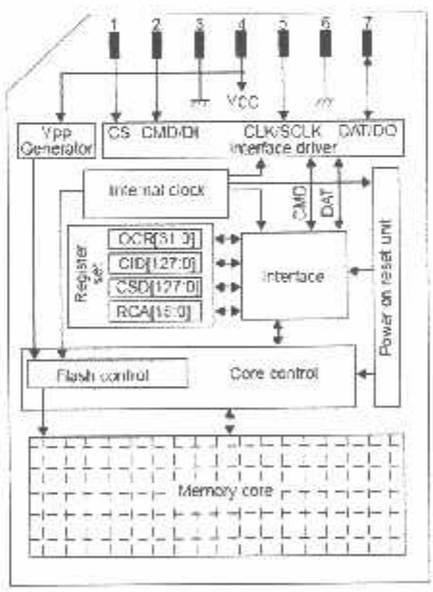
MMC adalah sebuah suatu media komunikasi yang berfungsi sebagai penyimpan data yang terintgrasi sangat cepat dengan kemamapuan akses acak dan serial untuk menoptimalkan transmisi data. MMC dirancang untuk mencakup area yang luas seperti telepon cerdas, kamera, organisator, PDAs, digital recorder, MP3 players, pagers, mainan elktronik dan lain2. MMC terdiri dari 7pin penghubung dan dua serial data, yaitu mode MMC dan mode SPI (Serial Peripherial Interface). Frekuensi yang digunakan pada dua mode tersebut adalah 20khz. MMC bertegangan antara 2 volt hingga 3,6 volt. Memori aksesnya antara 2,7volt hingga 3,6 volt. Kapasitasnya mulai 4MB hingga GB.

Spesifikasi MMC:

1. Frekuensi yang digunakan adalah 20 khz
2. Tegangan antara 2,7 volt - 3,6 volt
3. Kapasitas memori 4Mbit hingga Gbit
4. Pasword akses data terlindungi
5. SPI sebagai pendukung dalam pembacaan dan penulisan pengoperasian blok

6. Kecepatan tinggi alat penghubung serial dengan akses acak
    - Kecepatan membaca : 13,7 Mbit/s (multi-blok read) (for HB28E016/D032/D064/B128MM2) dan 20 Mbit/s (one blok)
    - Kecepatan penulisan : 6,4 Mbit/s (multi-blok read) (for HB28E016/D032MM2) dan 20 Mbit/s (one blok)
    - Akses waktu : 300 ms (typ) (at 20 MHz, VCC = 2.7 to 3.6V)
7. Daya yang dikeluarkan rendah
    - Kecepatan tinggi : 216 mW (max) (at 20 MHz, VCC = 3.6 V): HB28E016/D032MM2
    - Kecepatan tinggi : 288 mW (max) (at 20 MHz, VCC = 3.6 V): HB28E016/D032MM2
8. Ukuran blok yang dibaca program antara 1 – 2048 bit

Berikut adalah gambar blok diagram dan table pin MMC



Gambar 2.18 Blok Diagram Multi Media Card[5]

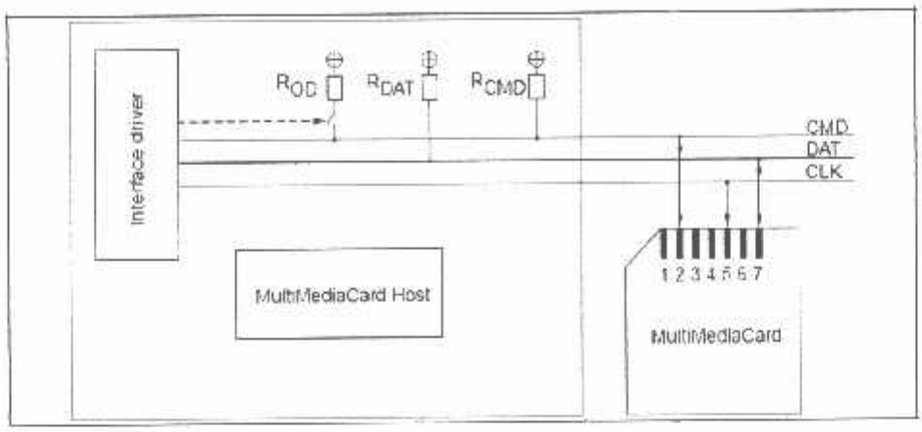
Pin	Name	Type	Function
1	CS#	Input	Chip select (active low)
2	Din	Input	Data input
3	VSS1	Power	GND
4	VDD	Power	VCC
5	CLK	Input	Clock input
6	VSS2	Power	GND
7	Dout	Output	Data output

Tabel 2.14 Tabel Pin-Pin MMC[2]

2.7.2. Multi Media Card Mode

Pada Multi Media Card mode, semua data di transfer menjadi beberapa bagian :

- CLK : yang mana masing-masing putaran pada sinyal ini, memerintahkan suatu bit dan bentuk data dikerjakan. Frekuensi antar o dan frekuuensi clok maksimum. Bus master Multi Media Card bebas untuk menghasilkan siklus ini tanpa pembatasan dalam kisaran 0 – 20 Mhz.
- CMD : sebuah pengarahan pelaksnaan saluran yang digunakan untuk menginisialisasikan kartu dan pelaksanaan tranfer data. Sinyal CMD mempunyai 2 mode operasi : membuka saluran selama menginisialisaikan mode dan tarik mendorong selama pelaksanaan tranfer cepat. Pelaksanaannya yaitu mengirimkan dari Bus master Multi Media Card ke Multi Media Card mode dan hubungan sebaliknya.
- DAT : Pelaksnaaan saluran data dengan lebar dari 1 garis. Sinyal DAT dari Multi Media Card dioperasikan pada mode tarik dorong
- RSV : pencabutant dengan resisitor (2MW typ) di Multi Media Card. Pencabutan resitor eksternal dibolehkan jika sistemnya diperlukan.

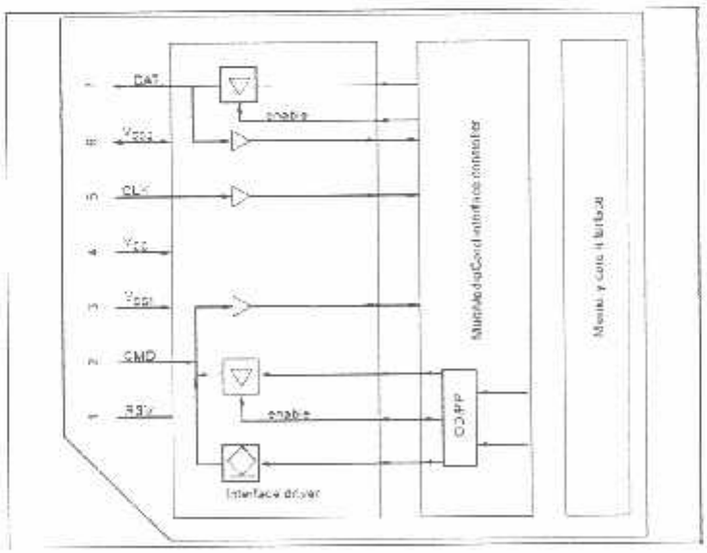


Gambar 2.19 Multi Media card Mode[5]

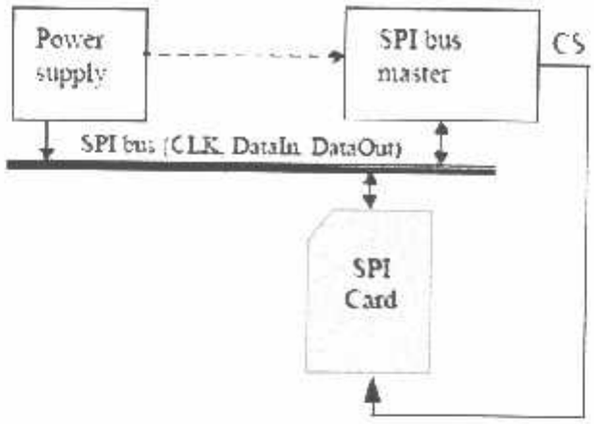
Pin No.	Name	Type*1	Description
1	RSV	NC	No connection
2	CMD	I/O/PP/OD	Command/Response
3	V_{SS1}	S	Ground
4	V_{DD}	S	Power supply
5	CLK	I	Clock
6	V_{SS2}	S	Ground
7	DAT	I/O/PP	Data

Note 1 S: power supply; I: input; O: output; PP: push-pull; OD: open-drain; NC: No connection or $V_{...}$

Tabel 2.15 MultiMediaCard Mode Pad Definition[5]



Gambar 2.20 MultiMediaCard Mode I/O-drivers[5]

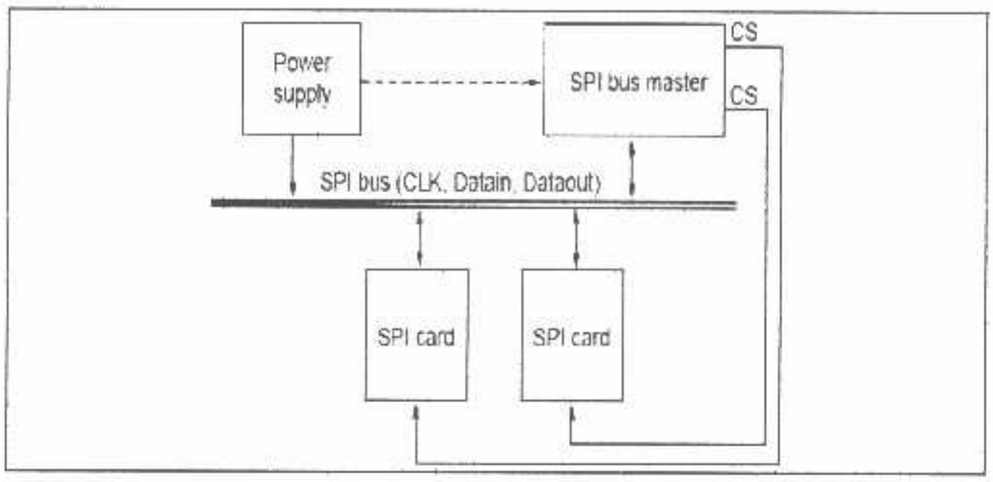


Gambar 2.21 Multi Media Card Bus[6]

2.7.3. SPI (Serial Peripheral Interface) Mode

SPI terdiri dari suatu protokol komunikasi sekunder yang mana ditawarkan oleh kecepatan dasar Multi Media Card. Model ini merupakan bagian dari Multi Media Card protokol, didesain untuk komunikasi dengan saluran SPI. Interface diseleksi pertama kali reset setelah daya dalam kondisi on (CMD0).

Standart SPI hanya menggambarkan bentuk mata rantai dan data tranfer protokol tidak lengkap. Implikasi Multi Media Card SPI menggunakan set tambahan dari Multi Media Card protokol. MMC SPI sangat dibutuhkna dalam sebuah sistem dimana menggunakan satu kartu dan mempunyai tafsiran tranfer data yang rendah, dibandingkan MMC Protokol- sistem dasar. Dari aplikasi yang kita lihat, keuntungan dari model SPI adalah kemampuan dalam menggunakan an off-the-shelf host, sehingga mengolah desain menjadi lebih simple. Kerugian SPI : SPI mode masih kalah dengan Multi Media Card mode (kecepatan tarnfer data rendah, perangkat keras CS (dll).



Gambar 2.22 SPI Bus Sistem[5]

SPI Interface Pin Configuration

	MultiMediaCard			SPI		
Pin No.	Name	Type*1	Description	Name	Type	Description
1	RSV	NC	Reserved for future use	CS	I	Chip select (neg true)
2	CMD	I/O/PP/OD	Command/Response	DI	I	Data in
3	V_{SS1}	S	Ground	V_{SS}	S	Ground
4	V_{DD}	S	Power supply	V_{DD}	S	Power supply
5	CLK	I	Clock	SCLK	I	Clock
6	V_{SS2}	S	Ground	V_{SS2}	S	Ground
7	DAT	I/O/PP	Data	DO	O/PP	Data out

Note: 1. S: power supply; I: Input; O: output; PP: push-pull; OD: open-drain; NC: No connection or V_{SS}

Tabel 2.16 SPI Interface Pin Configuration[5]

		MMC Interface Mode			SPI Interface Mode		
Comparison of system specification	Interface	Ten-wire bus(CLK,CMD, DAT0-8)			Three-wire serial data bus(CLK,DI,DO)+CS		
	Frequency	0-20MHz, 0-26MHz, 0-52MHz			0-20MHz		
	Card selection	Card is selected by MMC bus protocol. Host sends the relative card address to select the card which has the same one.			Card is selected by the CS signal		
	Access mode	Single block access, Multiple block access, Stream access			Single block access, Multi block access		
Pin Arrangement	Pin No.	Name	Type	Description	Name	Type	Description
	1	DAT3	I/O/PP	Data	CS	Input	Chip Select
	2	CMD	I/O/PP/OD	Command/ Response	DI	I/PP	Data In
	3	VSS1	-	GND	VSS	-	GND
	4	VDD	-	VCC	VDD	-	VCC
	5	CLK	Input	Clock	SCLK	Input	Clock
	6	VSS2	-	GND	VSS2	-	GND
	7	DAT0	I/O/PP	Data	DO	O/PP	Data Out
	8	DAT1	I/O/PP	Data	Not Used		
	9	DAT2	I/O/PP	Data	Not Used		
	10-13	DAT4-DAT7	I/O/PP	Data	Not Used		

Tabel 2.17. MultiMediaCard Interface Pin Configuration[4]

MultiMediaCard Mode	SPI Mode
Ten-wire bus (clock, 1 bit command, 8 bit data bus)	Three-wire serial data bus (clock, dataIn, dataOut) + card specific CS signal
Card selection is done through an assigned unique card address to maintain backwards compatibility to prior versions of the specification	Card selection via a hardware CS signal
One card per MultiMediaCard bus	Card requires a dedicated CS signal.
Easy identification and assignment of session address	Not available. Card selection via a hardware CS signal
Error-protected data transfer	Optional. A non-protected data transfer mode is available.
Sequential and Single/Multiple block Read/Write commands	Single/Multiple block Read/Write commands

Tabel 2.18 MMC Sistem Operational Modes[6]

BAB III

PERANCANGAN ALAT

3.1 Pendahuluan

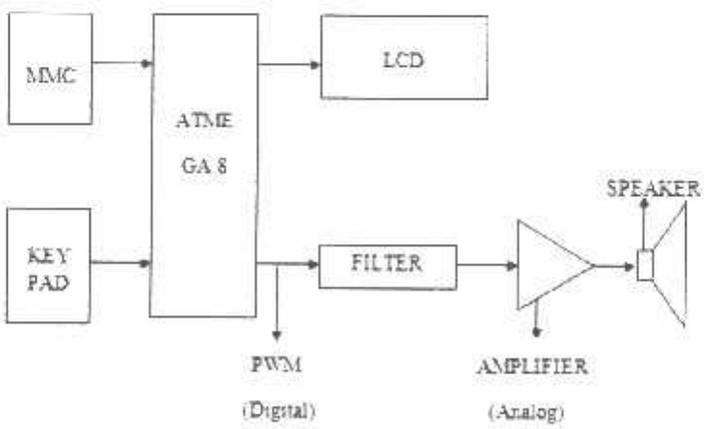
Dalam bab ini akan dibahas mengenai perencanaan dan pembuatan alat mulai dari perencanaan perangkat keras (Hardware) hingga perencanaan perangkat lunak (Software). Pembahasan dilakukan pada tiap blok rangkaian penyusun sistem meliputi cara kerja masing-masing blok rangkaian, fungsi masing-masing blok rangkaian, serta prinsip kerja sistem keseluruhan.

Pada perancangan perangkat keras akan meliputi seluruh peripheral yang digunakan pada sistem ini. Pada perancangan perangkat lunak akan meliputi diagram alir dari software secara umum. Kedua perangkat ini dalam kerjanya akan saling menunjang satu sama lain.

3.2. Perancangan Perangkat Keras

3.2.1 Diagram Blok

Diagram blok sistem secara umum terdiri dari kartu memori (MMC), LCD (Liquid Crystal Display), keypad, filter, amplifier, speaker



Gambar 3.1 Blok Diagram Sistem

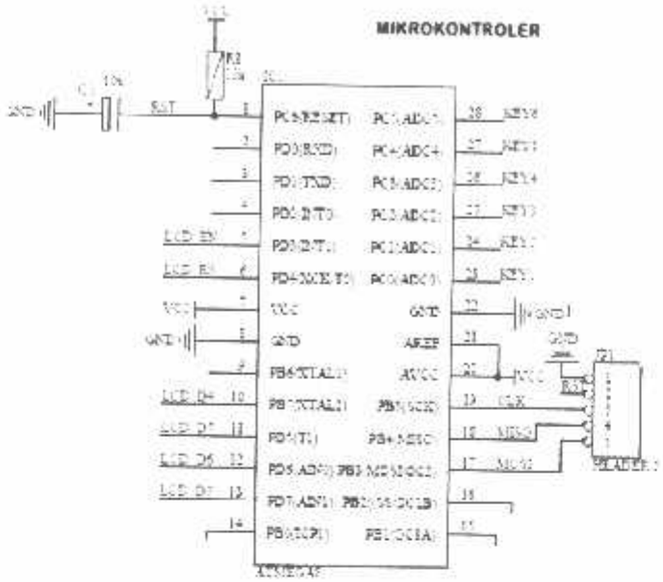
Penjelasan Singkat Dari Blok Diagram

1. MMC merupakan kartu penyimpan data, dalam hal ini data yang dimaksud adalah berupa musik.
2. Mikrokontroller ATMEGA 8 merupakan single chip mikrokontroller yang berfungsi untuk menjalankan dan menginstruksikan suatu sistem dari alat ini.
3. LCD (ukuran 16 x 2 character) sebagai penampil data agar kita sebagai pemakai bisa melihat kejadian yang diinginkan misalnya saat menekan tombol volume, tombol next, stop, play dll.
4. Filter merupakan sistem penyaringan gelombang frekuensi dasar PWM (Pulse Width Modulation) agar dihasilkan frekuensi suara analog yang di modulasikan.
5. Amplifier merupakan penguat sinyal audio analog agar bisa di umpankan ke beban loud speaker.
6. Speaker merupakan alat pengeras suara.
7. Keypad merupakan tombol inputan

3.2.2. Perancangan Rangkaian Mikrokontroler ATmega8

3.2.2.1. Rangkaian ATMEGA 8

Pada rangkaian ini komponen utamanya adalah unit Mikrokontroler ATmega8. Komponen ini merupakan sebuah *chip* tunggal sebagai pengolah data dan pengontrolan alat. Sebagai pengolah data dan pengontrolan sistem, pin-pin mikrokontroler ATmega8 dihubungkan pada rangkaian pendukung membentuk suatu sistem, yang ditunjukkan pada gambar 3.2



Gambar 3.2 Rangkaian Mikrokontroller ATMEGA8[1]

3.2.2.2. Rangkaian Reset

Rangkaian reset dirancang agar mempunyai kemampuan power on reset yaitu reset yang terjadi pada saat system di nyalakan untuk pertama kalinya. Reset juga dapat dilakukan secara manual dengan menambahkan tombol reset berupa switch button

Untuk me-*reset* mikrokontroler atmega8, maka pin Reset diberi logika tinggi selama sekurangnya dua siklus mesin (24 periode osilator). Untuk membangkitkan sinyal *reset* kapasitor dihubungkan dengan V_{CC} dan sebuah resistor yang dihubungkan ke *ground*.

Karena kristal yang digunakan mempunyai frekuensi sebesar 4 MHz, maka satu periode dapat dihitung dari persamaan :

$$T = \frac{1}{f_{XTAL}} = \frac{1}{4MHz}\, s = 0{,}25 \times 10^{-6}\,\mathrm{s}$$

Sehingga waktu minimal logika tinggi yang dibutuhkan untuk *mereset* mikrokontroller adalah :

reset(min) = T × periode yang dibutuhkan

$= 0{,}25 \times 10^{-6} \times 24 = 6\ \mu s$

Jadi mikrokontroller membutuhkan waktu minimal 6 µs untuk *mereset*. Waktu minimal inilah yang dijadikan pedoman untuk menentukan nilai R dan C. Dengan menentukan nilai R = 10 kΩ maka :

$T = 2 \times 3{,}14 \times R \times C$

$C = T / 2 \times 3{,}14 \times R$

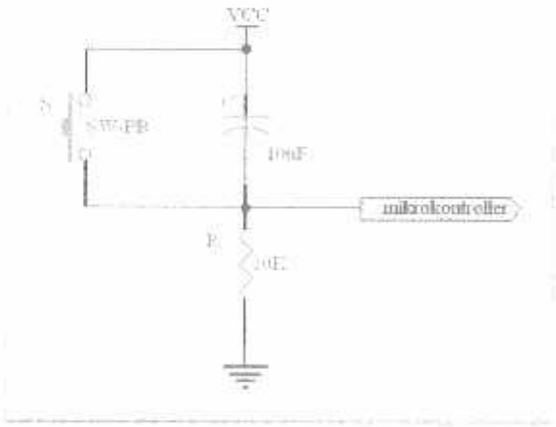
$C = \frac{0{,}25 \times 10^{-6}}{2 \times 3{,}14 \times 10 \times 10^{3}}$

$C = 0.0039 \times 10^{-9}$

C = 390 pF

Jadi nilai komponen R = 10 kΩ dan C = 390 pf, namun C yang dipakai pada alat sebesar 10 µF, karena dengan C yang lebih besar maka akan lebih baik untuk menghilangkan bouncing.

Rangkaian reset ditunjukkan dalam gambar di bawah ini :



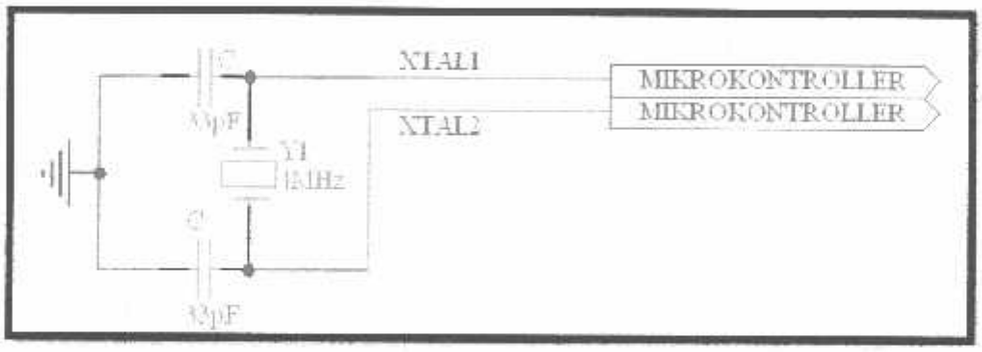
Gambar 3.3 Perancangan Rangkaian Reset[11]

3.2.2.3. Rangkaian Clock

Kecepatan proses yang diperlukan oleh mikrokontroller atmega8 ditentukan oleh sumber *clock* yang mengendalikan mikrokontroller tersebut. Untuk kristal *clock* dipasang Kristal dan resonator keramik yang berfungsi sebagai pembangkit *clock* osilator yang ada pada mikrokontroller.

Rangkaian ini terdiri dari dua buah kapasitor dan sebuah kristal. Untuk mengendalikan frekuensi osilatornya cukup dengan menghubungkan Kristal pada pin 9 (P47/X_{out}) dan pin 10 (P46/X_{in}) serta dua buah kapasitor ke *ground*.

Dalam minimum kristal ini, menggunakan kristal 4 Mhz dan $C_1 = C_2$ yaitu sebesar 33 pF. Dengan rangkaian sebagai berikut :



Gambar 3.4 Perancangan Rangkaian *Clock*

Dengan menggunakan nilai kristal dan kapasitor di atas maka dapat dihitung waktu yang diperlukan untuk 1 siklus mesin yaitu :

Diketahui : F = 4 MHz

T = 1 / f

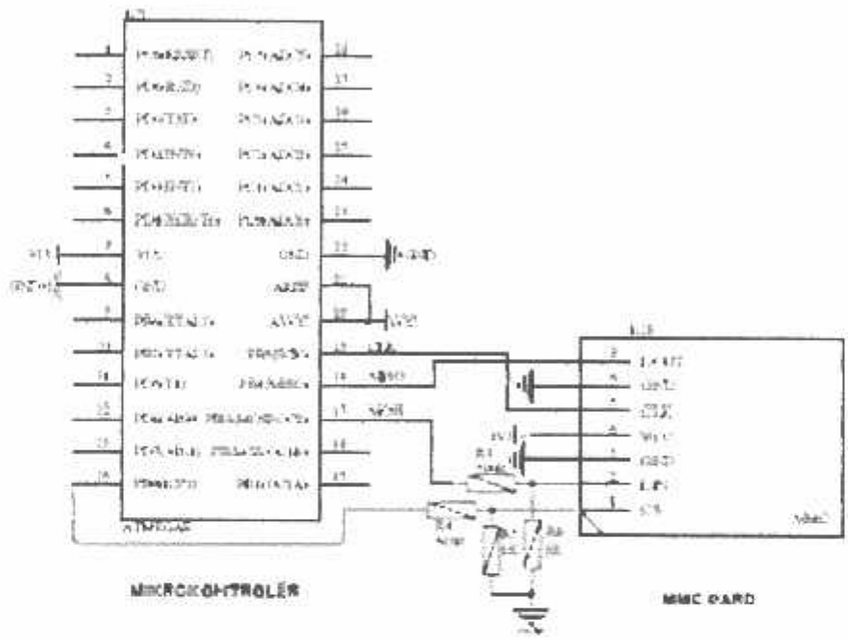
Maka T = 1 /4 MHz = - µs

Maka untuk 1 siklus mesin dari mikrokontroller atmega8 adalah sebesar:

$$T_m = 4 \times T$$

$$= 4 \times \frac{1}{4}\mu s = 1\ \mu s$$

3.2.3. Perancangan Rangkaian MMC



Gambar 3.5 Perancangan Rangkaian MMC

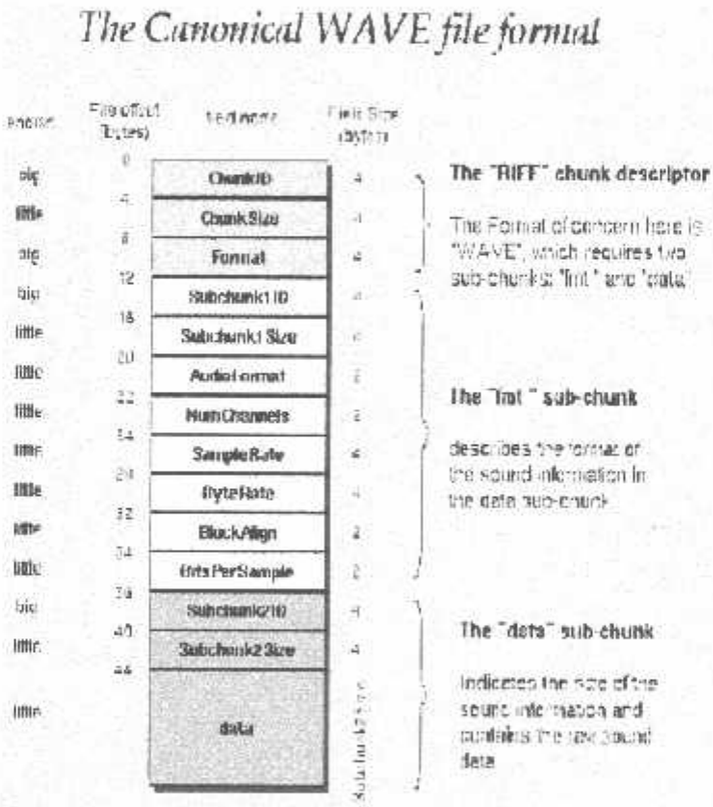
Mikrokontoller ATMEGA8 memerintahkan SPI untuk memindahkan pin data in dari MMC. Pin data in MMC dhubungkan ke mosi (SPI). Data tersebut juga ditulis dalam MMC melalui sinyal data in MMC. Berdasarkan received command, MMC mengirimkan respon atau data melalui pin data out. Pin dat out MMC dikoneksikan dengan miso dari port SPI mikrokontroller ATMEGA8. Port SPI menggunakan salah satu pin programmable Flag untuk mengatur CS (Chip

Select) dari MMC. Komunikasi diaktifkan oleh perintah yang berbeda yang dikirim dari mikrokontroller ATMEGA8 ke MMC.

3.2.3.1. WAV File Format

Sering disebut juga sebagai WAVE file, yaitu bagian dari spesifikasi file RIFF yang dikeluarkan oleh microsoft. RIFF sendiri adalah format file yang ditujukan untuk menyimpan macam-macam data khususnya multimedia data terutama audio dan video.

Berikut ini adalah susunan dari file WAV :

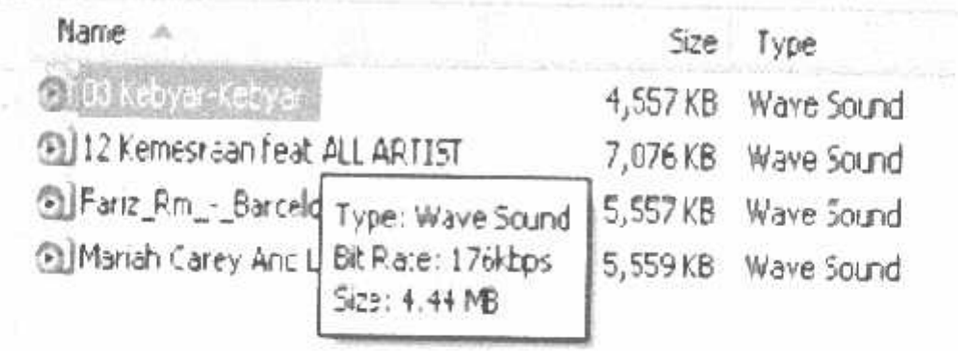


Gambar 3.6 Susunan Wave File

sumber : http://www.sonicspot.com/guide/wavefiles.html#wave_format

3.2.3.2. Analisa Data File Lagu

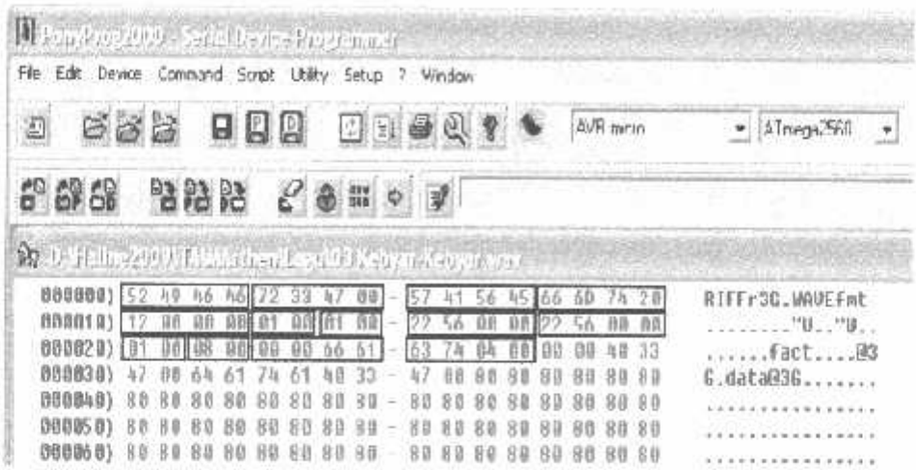
Gambar 3.6. menunjukkan file lagu yang nampak pada folder di PC, sedangkan Gambar 3.7 adalah properties untuk lagu berjudul Kebyar-Kebyar



Gambar 3.7 List file lagu yang nampak pada folder di PC

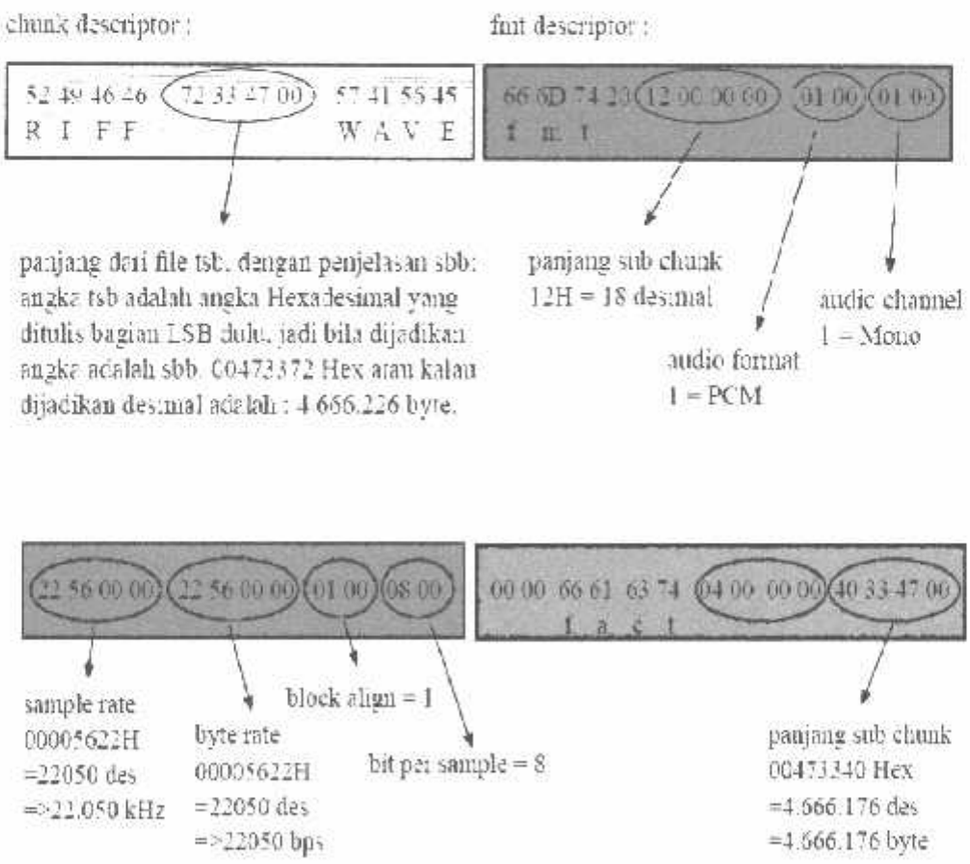
Gambar 3.8 Properties untuk lagu berjudul Kebyar-Kebyar

Berikut ini adalah header dari file bernama Kebyar-Kebyar.WAV



Gambar 3.9 Header dari file "Kebyar-kebyar.wav"

Analisa file Kebyar-Kebyar.WAV adalah sbb:



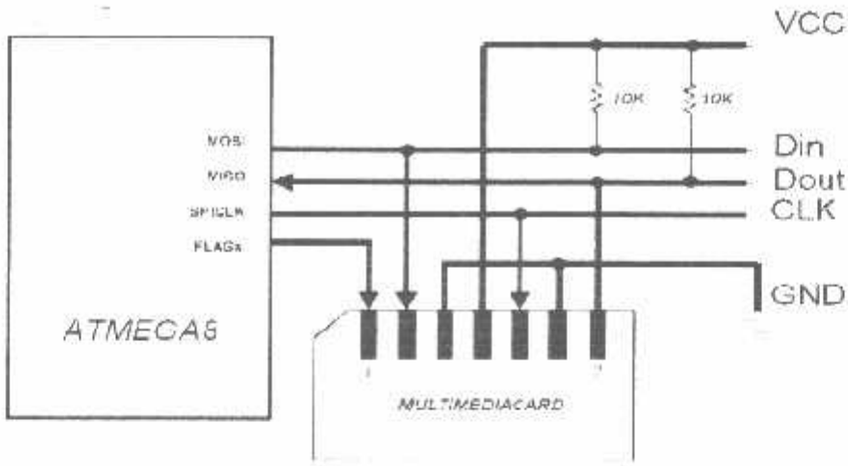
Dengan demikian menurut analisa file diatas, file lagu yang berupa PCM data dimulai dari urutan file ke 3A H atau ke 58.

3.2.3.3. Fungsi PWM

Data yang disimpan untuk aplikasi tugas akhir ini yaitu berupa file WAV didalam kartu MMC dalam bentuk data digital, berupa data PCM (pulse code modulation) dengan lebar data 8 bit , mono (1channel audio) , 22050 sampling rate *(lihat keterangan diatas)*.Dengan demikian data tersebut harus diubah menjadi analog sebelum diberikan ke audio amplifier (penguat audio), dalam hal ini digunakan DAC (digital to analog conversion). Ada banyak macam DAC, dan yang digunakan pada perencanaan ini adalah DAC 1 bit berupa modulasi data PWM (pulse width modulation). Jadi mikrokontroler akan mengeluarkan pulsa PWM untuk diberikan ke rangkaian filter terlebih dulu supaya dihasilkan sinyal analog audio yang bisa diteruskan ke rangkaian penguat audio.

1. **Proses Pembacaan Data MMC Oleh Mikrokontroller**

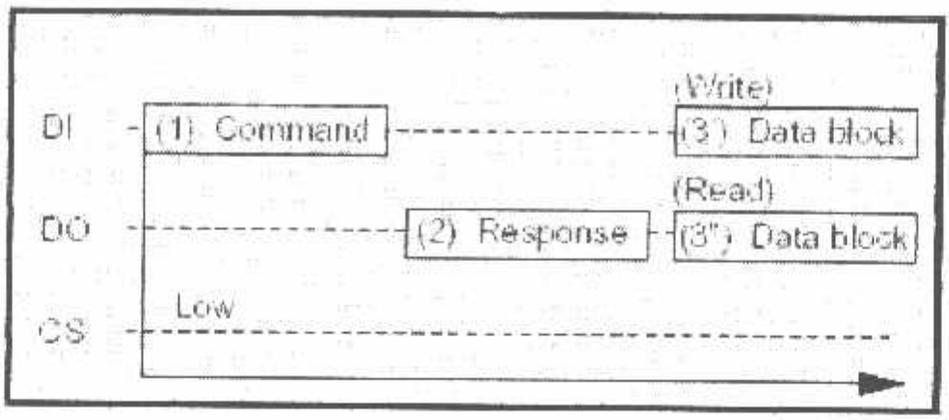
   Berikut ini adalah bagaimana sebuah kartu memory MMC disambungkan dengan mikrokontroler dan cara-cara pembacaan data dari MMC ke mikrokontroler.



Gambar 3.10 Multi Media Card Interface with SPI Port

Menunjuk pada gambar 3.9 diatas, kartu MMC disambungkan ke mikrokontroler pada kaki SPI yang dalam perencanaan ini digunakan ATMEG8. SPI sendiri terdapat 3 kaki penting yaitu :

- MOSI (Master output slave input) adalah kaki I/O dimana kaki mikrokontroler (master) sebagai output data dan kaki MMC (slave) berupa input.
- MISO (Master input slave output) adalah kaki I/O dimana kaki mikrokontroler (master) difungsikan sebagai input data kaki MMC (slave) difungsikan sebagai output.
- SPICLK atau CLK adalah clock atau pulsa untuk mengatur jalannya data yang lewat MISO dan MOSI.

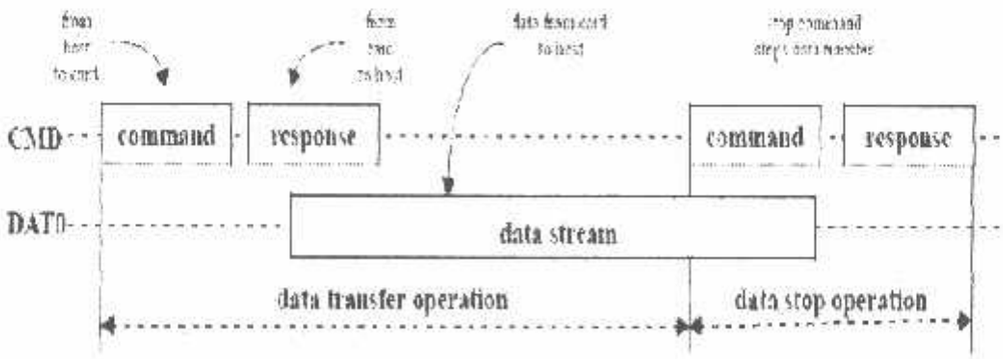


Gambar 3.11. MultiMediaCard Transfer Protocol

Gambar 3.10 adalah diagram aliran data untuk berhubungan dengan kartu MMC, hal yang perlu dilakukan yaitu:

- Aktifkan CS (chip select) dari MMC tersebut, yaitu kaki MMC no. 1 dibuat logika nol "0"
- Kirim data ke MMC ke kaki DI atau MOSI berupa command yang isi datanya bisa dilihat pada lampiran.
- MMC akan merespon pengiriman data melalui kaki DO atau (MISO).

Atau lebih jelasnya untuk pengiriman data streaming audio bisa digambarkan sbb:



Gambar 3.12. Sequential Read Operation

Dari sisi software bisa dijelaskan sbb:

1. Inisialisasi MMC

   MMC perlu diinisialisasi sebelum dilakukan baca/tulis, berikut adalah program untuk keperluan ini:

```
char MMC_Init(void)
{ // init SPI
      char i;
      PORTB |= (1 << SPICS); // disable MMC
      // start MMC in SPI mode
      for(i=0; i < 10; i++) SPI(0xFF); // send
10*8=80 clock pulses
      PORTB &= ~(1 << SPICS); // enable MMC
         if (Command1(0x40,0,0,0x95) != 1) goto
mmcerror; // reset MMC

st: // if there is no MMC, prg. loops here
      if (Command1(0x41,0,0,0xFF) !=0) goto st;
      return (1);
mmcerror:
      return (0);
}
```

   Inti dari program diatas adalah MMC di set pada mode SPI, lalu dikirim pulsa sebanyak 80 kali untuk sinkronisasi awal, setelah diberi command(0x40,0,0,0x95) maka MMC harus menjawab 1, bila tidak maka proses inisialisasi diatas gagal atau terjadi error.

2. Membaca streaming data audio dari MMC

Ini adalah proses yang terpenting pada perencanaan Tugas Akhir ini, yaitu bagaimana data audio yang tersimpan berupa file WAV atau dalam bentuk urutan data PCM bisa dibaca oleh mikrokontroler Atmel ATMEGA8.

Berikut ini adalah potongan program tsb:

```
for(j=0; j < 512; j++)
{
        SPDR = (0xFF); temp = vol >> 4; if (temp==0)
temp=1;
        while(!(SPSR & (1<<SPIF))); if (temp < 14) pwm
= SPDR/temp;
       // this instruction for delay only !
        speed++; speed++; speed++;
        speed++; speed++; speed++;
}
```

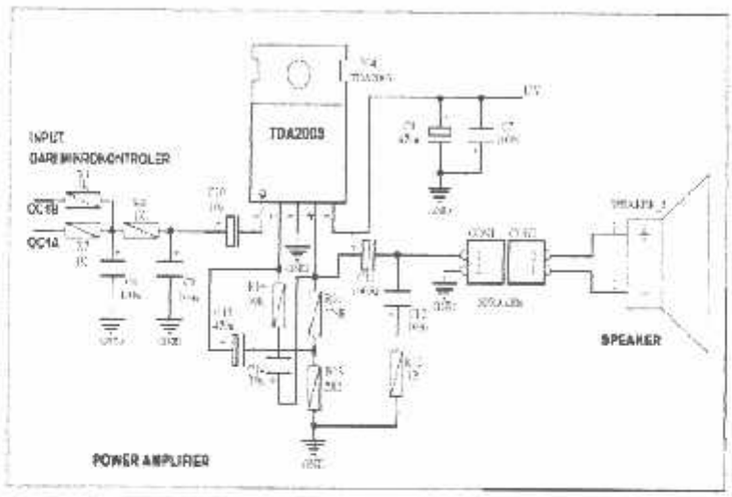
Inti dari program diatas adalah membaca data dari MMC per paket, dimana tiap paket data terdiri dari 512 data. Data hasil pembacaan berada pada register SPDR, dan data inilah data PCM audio yang tersimpan pada MMC tsb. Data ini diberikan ke register untuk mengaturan PWM setelah di olah dengan posisi volume saat itu dengan rumus :

```
pwm = SPDR / [Volume]
```

jadi sinyal PWM yang dihasilkan nanti tergantung dari data PCM dan nilai Volume.

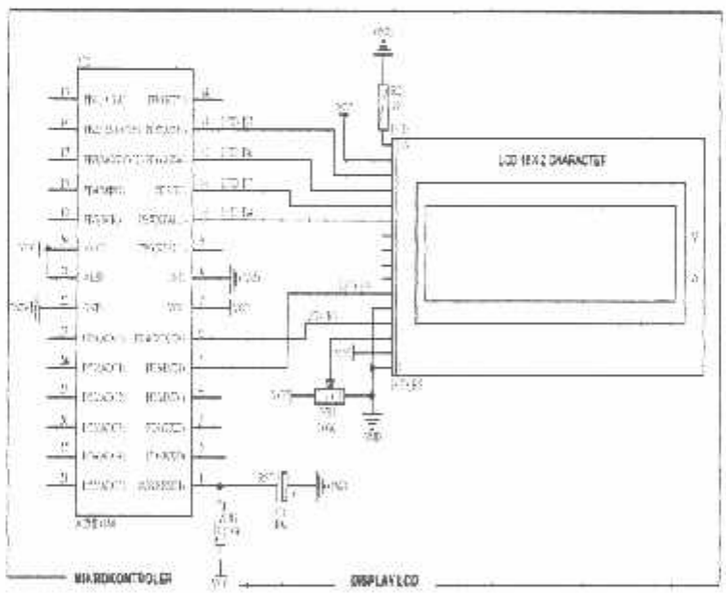
3.2.4. Perancangan Rangkaian Amplifier

Pada perancangan ini Amplifier yang digunakan adalah TDA 2003, dibawah ini adalah gambar rangkaian Amplifier.



Gambar 3.13 Perancangan Rangkaian Amplifier

3.2.5. Perancangan Rangkain LCD



Gambar 3.14 Perancangan Rangkaian LCD

LCD dot matrik ini membutuhkan sepuluh buah pin masukan/keluaran dari mikrokontroler. Adapun dua buah pin yakni port PD4 terminal RS yang

digunakan sebagai sinyal pemilih register dan port PD3 pada terminal Enable yang digunakan sebagai sinyal operasi awal. Sinyal enable ini mengaktifkan data tulis atau baca oleh mikrokontroler, pin DB0-DB7 langsung dihubungkan ke mikrokontroller untuk menampilkan karakter yang dikehendaki oleh mikrokontroler tersebut. Ketika terdapat data pada jalur data, data tersebut akan ditahan dengan memberikan *clock* pin E pada LCD. Pin RS menentukan apakah data yang ditahan akan digunakan sebagai instruksi untuk mengatur *setting* tampilan pada LCD atau sebagai kode karakter yang diperlukan LCD untuk menampilkan suatu karakter. Sedangkan untuk pin R/W pada LCD dihubungkan ke *ground* karena dalam hal ini LCD hanya melakukan operasi write atau operasi menampilkan karakter. Untuk pin Vcc pada LCD dihubungkan ke supply +Vcc dan Vss dihubungkan ke *ground*. Pin V_{EE} beserta pin Vcc dan Vss dihubungkan ke *trimer potensio* atau kadang disebut dengan *trimpot*. *Trimpot* ini digunakan untuk mengatur kontras dari tampilan LCD dengan cara mengubah tegangan pada pin V_{EE}. Daftar tabel fungsi penyemat pada LCD dapat dilihat dalam Tabel 3.1.

Tabel 3.1 Fungsi penyemat LCD[3]

Penyemat	**Fungsi**
DB0 – DB7	Merupakan saluran data, berisi perintah dan data yang akan ditampilkan di LCD.
Enable	Sinyal operasi awal, sinyal ini mengaktifkan data tulis atau baca
R/W	Sinyal seleksi tulis atau baca 0: tulis 1: baca
RS	Sinyal pemilih *register* 0: masukan data

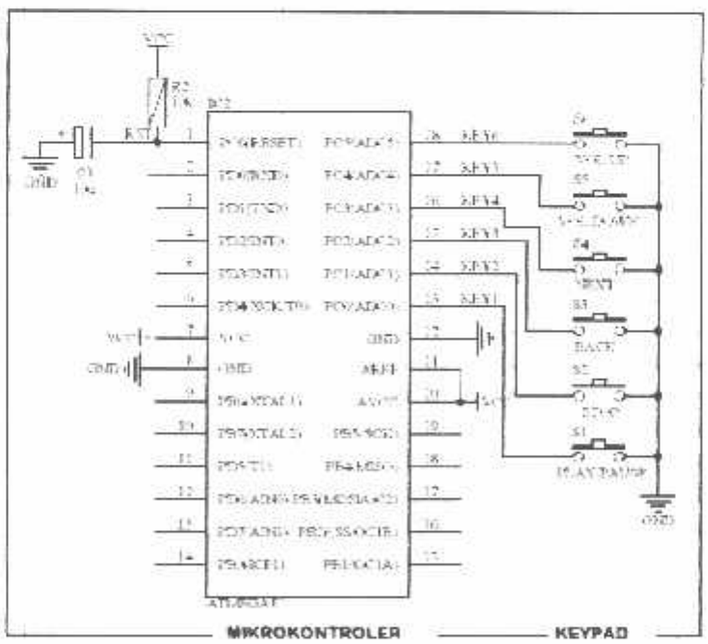
	1: masukan instruksi

Seperti telah disebutkan sebelumnya bahwa data yang terdapat pada jalur data selain dianggap sebagai kode karakter dapat digunakan sebagai suatu perintah instruksi untuk mengatur setting dari tampilan LCD. Cara pemakaian data antara sebagai instruksi dengan kode karakter berbeda. Perbedaan hanyalah keadaan pin RS ketika data yang ada dijalur data ditahan oleh LCD dengan memberikan *clock* pada pin E.

Pin – pin yang digunakan adalah

- Pin DB0-DB7 terhubung pada mikrokontroller Port A
- Pin Enable pada LCD terhubung pada Mikrokontroller yaitu Port PD3
- Pin RS pada LCD terhubung pada Mikrokontroller yaitu Port PD4

3.2.6. Perancangan Rangkaian Keypad

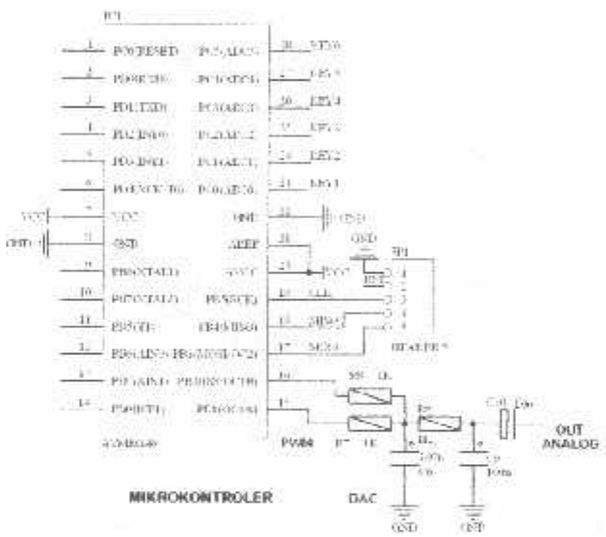


Gambar 3.15 Perancangan Rangkian Keypad

Rangkaian keypad dalam perancangan ini menggunakan push button yang langsung dihubungkan ke kaki mikrokontroler terhadap ground. Bila tombol ditekan maka kaki mikrokontroler terhubung dengan ground dan akan terbaca sebagai logika nol, sedangkan bila tidak ditekan akan terbaca sebagai logika satu yang disebabkan karena adanya internal pull up di dalam mikrokontroler tersebut.

Rangkaian ini terdiri dari 6 buah keyapad, yang mana apabila kita tekan keypad tersebut akan terbentuk suatu instruksi yang berupa layout (play/pause, next, previous, stop/mode, increase, decrease).

3.2.7. Perancangan Rangkaian DAC

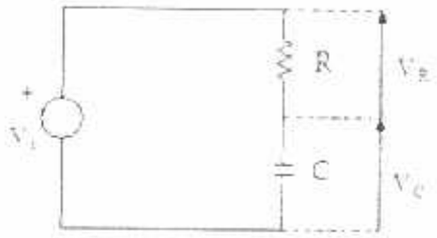


Gambar 3.16 Perancangan Rangkaian DAC (I Bit DAC)

Keluaran-keluaran yang berbentuk sinyal analog dari suatu system computer dapat diperoleh dengan menggunakan converter digital ke analog, DAC akan mengkonversi sebuah sinyal digital menjadi bentuk sinyal analog. Salah satu metoda yang paling banyak dipakai sekarang khususnya untuk aplikasi audio adalah DAC 1 bit seperti terlihat pada gambar diatas. DAC disusun dari R7, R8,

R9, C6 dan C9. Keluaran dari mikrokontroler berupa sinyal digital PWM, dan setelah melalui DAC yang sekaligus berfungsi sebagai filter tersebut akan dihasilkan sinyal analog pada kaki keluaran C10.

3.2.8. Perancangan Rangkaian Filter RC

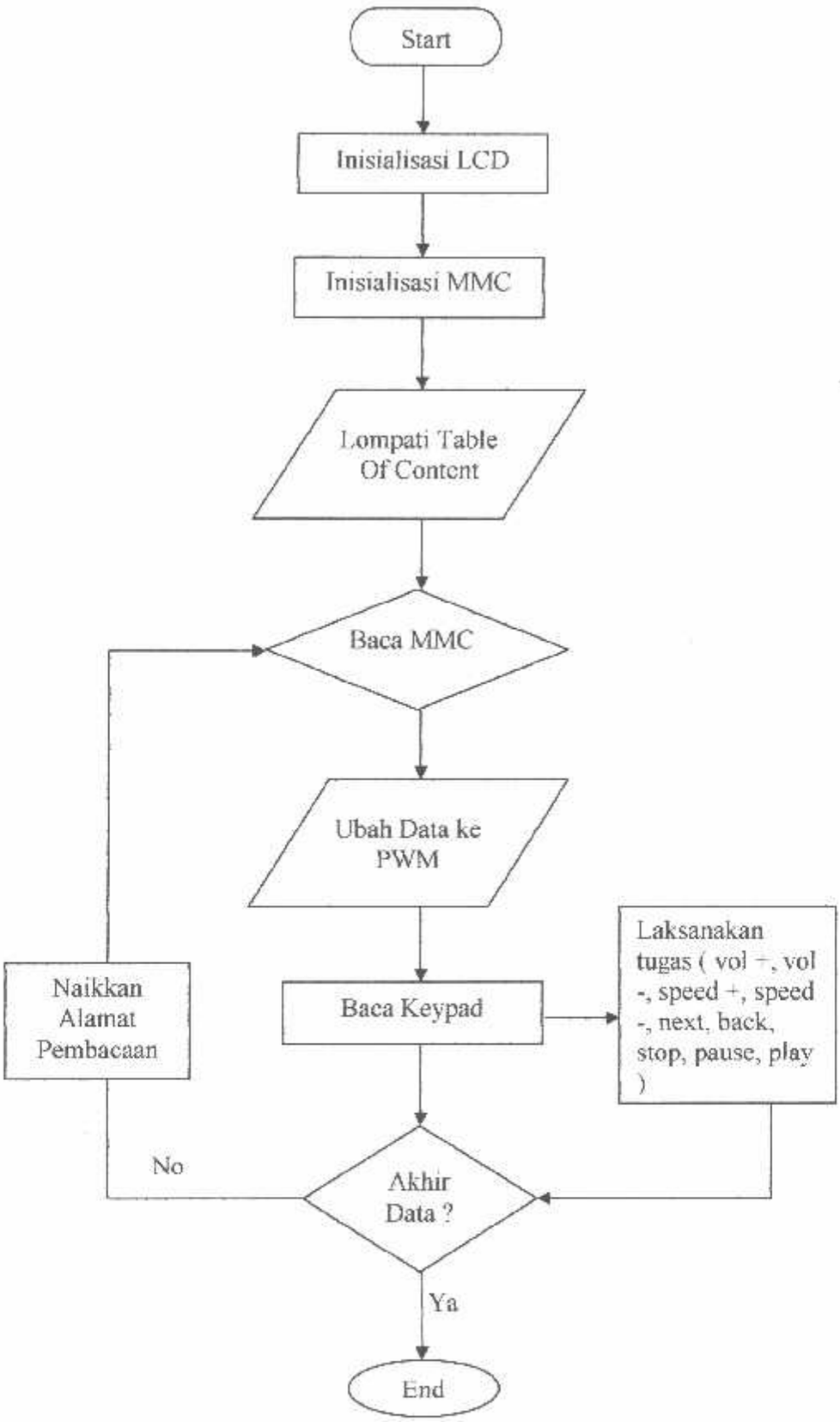


Gambar 3.17 Rangkaian filter RC

Rangkaian filter RC ini mengacu pada sistem pengolahan data MMC yang berupa sinyal digital menjadi sinyal analog melalui sistem PWM (Pulse Width Modulation) dengan tegangan 5 volt

3.3. Perancangan Perangkat Lunak

Setelah semua perencanaan perangkat keras telah selesai dikerjakan, pada tahap selanjutnya perangkat lunak (software) yang akan menangani system rangkaian. Pada perangkat lunak inilah kita dapat menentukan bagaimana sistem rangkaian ini akan bekerja. Pemrograman pada mikrokontroler ATmega8 menggunakan bahasa C yaitu Code Vision AVR.



Gambar 3.18. Flowchart Sistem

BAB IV

PENGUJIAN ALAT

Setelah perangkat keras dan perangkat lunak yang direncanakan selesai dibuat, selanjutnya dilakukan pengujian terahadap alat yang dibuat. Tahap pengujian alat ini perlu untuk dilakukan untuk mendapatkan hasil pengukuran serta kerja alat sesuai dengan yang diharapkan.

4.1. Pengujian LCD Dan Mikrokontroler

Pengujian ini dimaksudkan untuk menguji bagimana sebuah LCD 16x2 Character dengan type M1632 atau yang kompatibel disambungkan dengan sebuah mikrokontroler ATMEGA8 dibantu dengan program sederhana agar bisa menampilkan tulisan pada layar LCD tersebut.

4.1.1 Tujuan

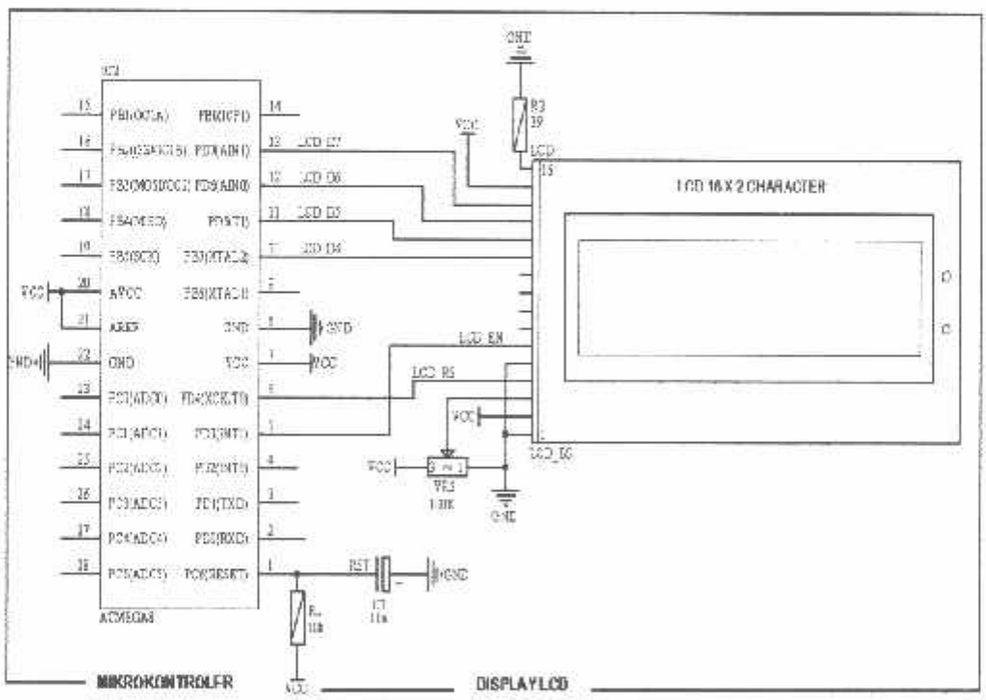
Untuk mengetahui apakah LCD dapat menampilkan data yang diinginkan dengan baik.

4.1.2 Peralatan Yang Digunakan

- Minimum system ATMEGA 8
- LCD EL1602 atau yang kompatibel dengan type M1632
- Power Supply 5V DC

4.1.3. Prosedur pengujiannya :

1. Menyusun rangkaian seperti gambar 4.1
2. Menulis program dalam bahasa C menggunakan program bantu Code Vision AVR seperti listing program dibagian bawah ini.
3. Meng-kompile program yang telah ditulis pada no.2 dan hasilnya dimasukkan/dituliskan ke IC mikrokontroler ATMEGA8 dengan menggunakan kabel ISP dan program bantu downloader Ponyprog 2000.
4. Menjalankan program untuk menampilkan tulisan ke LCD
5. Mengamati keluaran pada LCD



Gambar 4.1. Gambar Pengujian Rangkain LCD

Listing program :

```
' ==================================
Routine untuk menjalankan LCD 16x2 Character
type L1632 atau M1632 compatible
data bus      : 4 bit
wr/rd mode    : write only
author        : MARTHEN LEONARD
 NIM          : 0117025
 Teknik Elektro ITN Malang
 Indonesia
 ================================== */

#include <delay.h>

#define     LCDDATA7        PORTD.7
#define     LCDDATA6        PORTD.6
#define     LCDDATA5        PORTD.5
#define     LCDDATA4        PORTB.1
#define     LCD_RS          PORTD.4
#define     LCD_EN          PORTD.3

void LCD_data(char c,char dat)
{
    LCD_RS = c;
    if ((dat & 0x80)==0x80) LCDDATA7=1; else LCDDATA7=0;
    if ((dat & 0x40)==0x40) LCDDATA6=1; else LCDDATA6=0;
    if ((dat & 0x20)==0x20) LCDDATA5=1; else LCDDATA5=0;
    if ((dat & 0x10)==0x10) LCDDATA4=1; else LCDDATA4=0;
    LCD_EN = 0;             LCD_EN = 1;
    if ((dat & 0x08)==0x08) LCDDATA7=1; else LCDDATA7=0;

    if ((dat & 0x04)==0x04) LCDDATA6=1; else LCDDATA6=0;
```

4.1.4. Hasil Pengujian

Dari hasil pengujian didapatkan bahwa rangkaian LCD dapat menampilkan karakter-karakter, sesuai dengan program yang dibuat.

Gambar 4.2. Tampilan hasil pengujian LCD

4.2. Pengujian tatap muka (interfacing) MMC dengan mikrokontroller ATMEGA8

Pengujian ini dimaksudkan untuk menguji bagimana sebuah MMC disambungkan dengan sebuah mikrokontroler ATMEGA8 dibantu dengan program sederhana agar data pada kartu MMC bisa dibaca.

4.2.1. Tujuan

Untuk menguji data pada kartu MMC bisa dibaca ulang oleh mikrokontroler ATMEGA8.

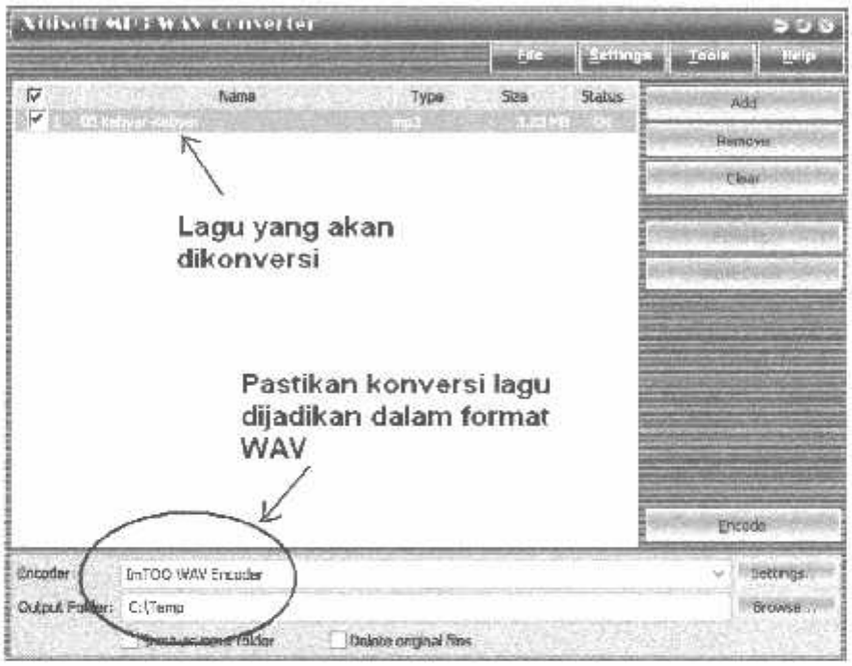
4.2.2. Peralatan Yang Digunakan

- Rangkaian Pengujian berisi mikrokontroler dan kartu MMC.
- Oscilloscope
- Power supply 5V DC
- PC / Laptop
- Universal Card Reader

```
    if ((dat & 0x02)==0x02) LCDDATA5=1; else LCDDATA5=0;
    if ((dat & 0x01)==0x01) LCDDATA4=1; else LCDDATA4=0;
    LCD_EN = 0;          LCD_EN = 1;
    delay_ms(2);
}

void Tulis_LCD(char a,char flash *dat)
{
    char i = 0;
    LCD_data(0,a); while(dat[i] != 0) { LCD_data(1,dat[i]); i++; }
}

void LCD_test()
{
      DDRD=0x2F;
      DDRB.7=1;
      // Initialisasi LCD
      delay_ms(2000);
      LCD_data(0,0x33); delay_ms(100); LCD_data(0,0x3F);delay_ms(100);
      LCD_data(0,0x38);        LCD_data(0,0x08);        LCD_data(0,0x01);
LCD_data(0,0x06);
      LCD_data(0,0x2F); LCD_data(0,0x0E); LCD_data(0,0x06);

      Tulis_LCD(0x80," MARTHEN LEONARD ");
      Tulis_LCD(0xC0," NIM : 0117025 ");
      while(1);
}
```

4.2.3. Prosedur Pengujian:

1. Mengubah file lagu dengan bantuan program konversi audio seperti misalnya Xilisof MP3 WAV Converter, tampilan program seperti pada Gambar dibawah ini :

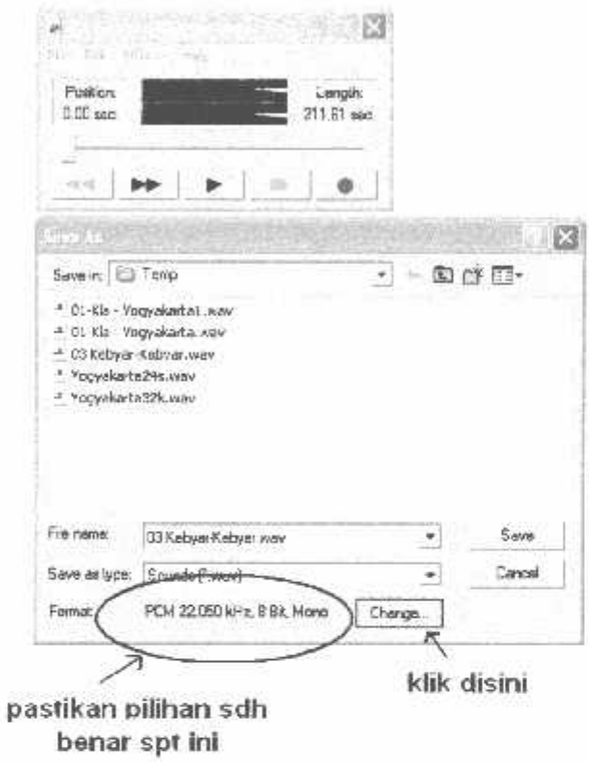


Gambar 4.3. Program MP3 WAV Converter

Dalam perencanaan ini format lagu harus berupa PCM data atau kalau dalam bentuk file berupa WAV (Wave format) dan mempunyai lebar data 8 bit serta kecepatan sampling 22kHz. Sehingga file diatas perlu dikonversi lagi dengan bantuan program sound recorder yang sudah ada ketika program windows operating system di install.

Langkah awal adalah klik Start → klik All Programs → klik Accessories → klik Entertainment → klik sound recorder, tampilan lihat Gambar 4.4 dibawah ini.

Buka file yang hendak dikonversi, kemudian save as dan lakukan klik change terlebih dulu kemudian pilih PCM 22.050kHz, 8bit, mono lalu klik save. File yang disimpan ini sudah siap dimasukkan ke dalam MMC.

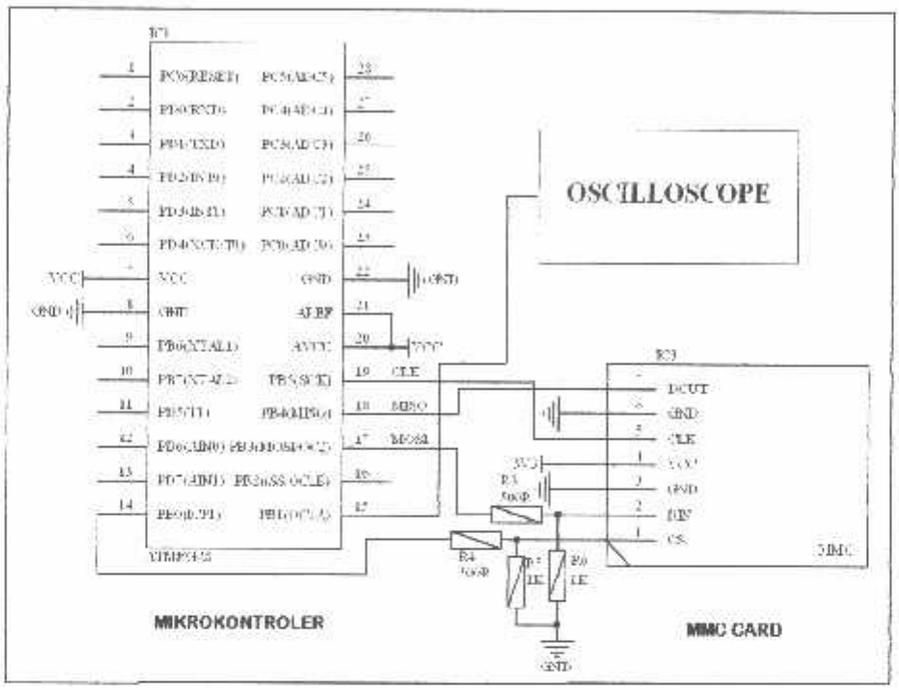


Gambar 4.4. Konversi file WAV dengan menggunakan Sound Recorder

2. Gunakan universal card reader untuk memindahkan data file lagu wav tadi ke dalam kartu MMC, sebelum dipindahkan, pastikan kartu MMC telah diformat dengan mode FAT16.

   Pengujian ini dimaksudkan untuk menguji bagimana sebuah LCD 16x2 Character dengan type M1632 atau yang kompatibel disambungkan dengan sebuah mikrokontroler ATMEGA8 dibantu dengan program sederhana agar bisa menampilkan tulisan pada layar LCD tersebut.

3. Rangkaian untuk pengujian kartu MMC adalah sbb:



Gambar 4.5. Rangkaian pengujian pembacaan kartu MMC

Listing Program

```
/*****************************************************
Chip type           : ATmega8
Program type        : Application
Clock frequency     : 8.000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************************/

#include <mega8.h>

// Declare your global variables here
```

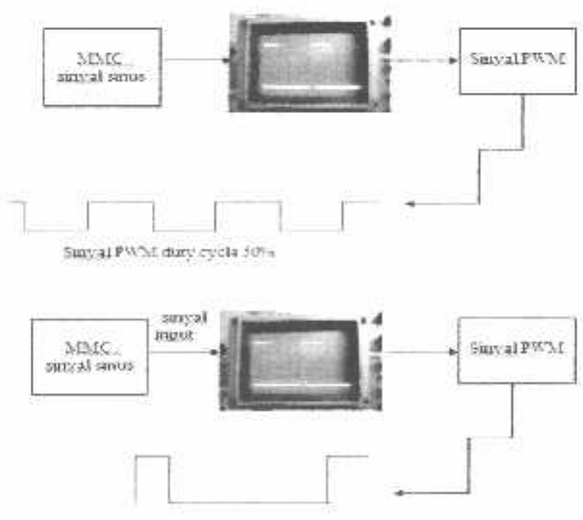
```
void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port B initialization
//  Func0=In  Func1=Out  Func2=Out  Func3=In  Func4=In  Func5=In  Func6=In
Func7=In
//  State0=T  State1=0  State2=0  State3=T  State4=T  State5=T  State6=T
State7=T
PORTB=0x00;
DDRB=0x06;

// Port C initialization
// Func0=In Func1 In Func2=In Func3=In Func4=In Func5=In Func6=In
// State0=T State1=T State2=T State3=1 State4=T State5=T State6=T
PORTC=0x08;
DDRC=0x00;

// Port D initialization
//  Func0=In  Func1=In  Func2=In  Func3=In  Func4=In  Func5=In  Func6=In
Func7=In
//  State0=T  State1=T  State2=T  State3=T  State4=T  State5=T  State6=T
State7=T
PORTD=0x00;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
TCCR0=0x00;
TCNT0=0x00;
```

```
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 8000.000 kHz
// Mode: Fast PWM top=OCR1A
// OC1A output: Non-Inv.
// OC1B output: Inverted
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0xB3;
TCCR1B=0x19;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
MCUCR=0x00;
```

```
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0xC0;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;
SFIOR=0x00;

while (1)
      {
      // Place your code here

      };
}
```

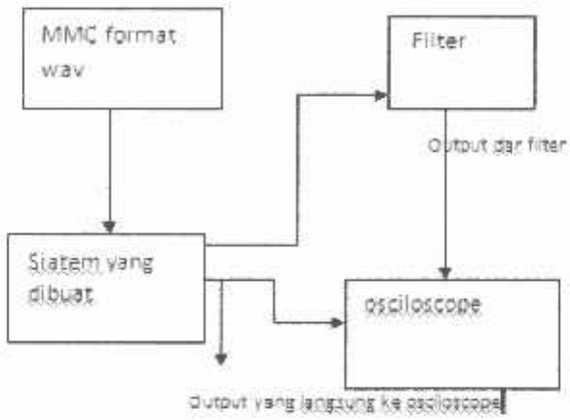
4.2.4. Hasil Pengujian

4.2.4.1. Sinyal MMC dan Sinyal PWM

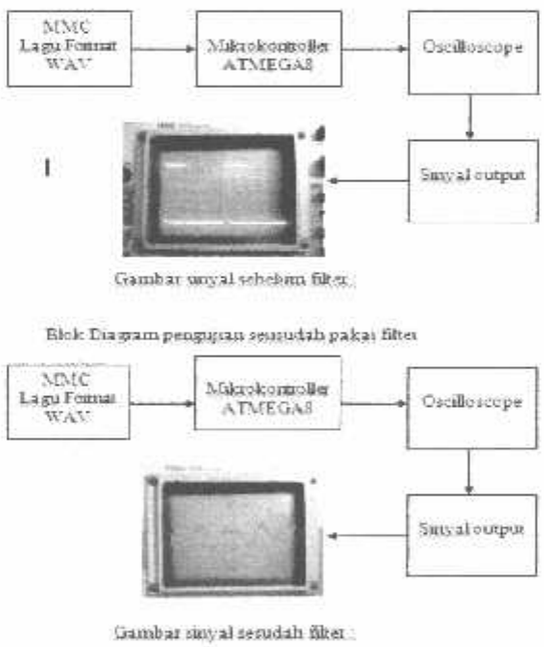


Gambar 4.6 Sinyal MMC dan Sinyal PWM

4.2.4.2. Sinyal Keluaran Dari sistem Yang Direncanakan



Gambar 4.7. Diagram Hasil Pengujian sinyal yang direncanakan



Gambar 4.8 Pengujian Sinyal Sebelum Dan Sesudah Ada Filter

4.4. Pengujian Hasil Seluruh Sistem

4.4.1. Tujuan

Untuk mengetahui system alat ini bekerja dengan baik atau tidak, dari data MMC yang dbaca oleh mikrokontroller Atmega 8 yang kemudian ditampilkan di LCD.

4.4.2. Alat-alat yang digunakan

- Amplifier
- Power Supply 5 volt
- Card Reader

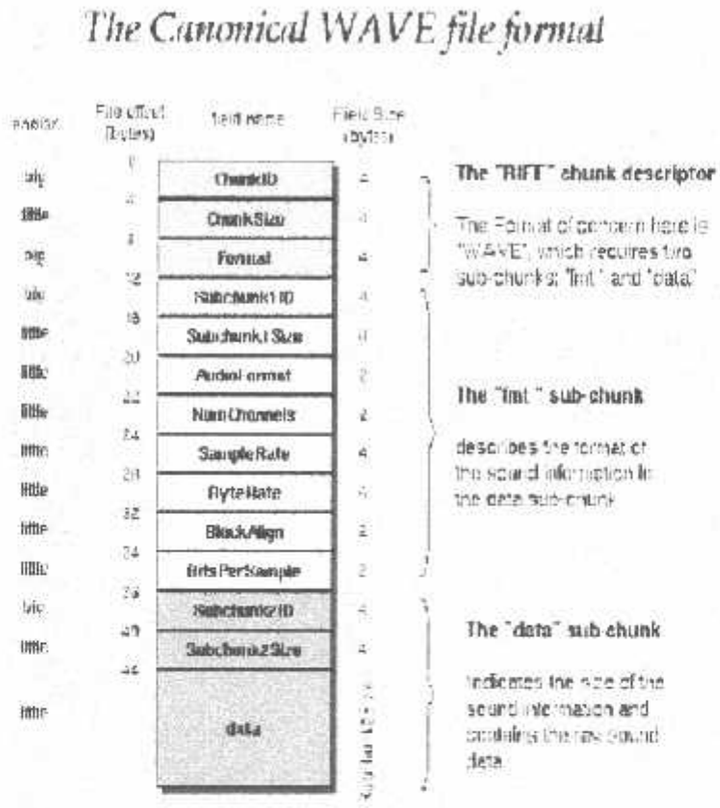
4.4.3. Prosedur Pengujian

Mikrokontoller ATMEGA8 memerintahkan SPI untuk memindahkan pin data in dari MMC. Pin data in MMC dhubungkan ke mosi (SPI). Data tersebut juga ditulis dalam MMC melalui sinyal data in MMC. Berdasarkan received command, MMC mengirimkan respon atau data melalui pin data out. Pin dat out MMC dikoneksikan dengan miso dari port SPI mikrokontroller ATMEGA8. Port SPI menggunakan salah satu pin programmable Flag untuk mengatur CS (Chip Select) dari MMC. Komunikasi diaktifkan oleh perintah yang berbeda yang dikirim dari mikrokontroller ATMEGA8 ke MMC.

4.4.3.1. WAV File Format

Sering disebut juga sebagai WAVE file, yaitu bagian dari spesifikasi file RIFF yang dikeluarkan oleh microsoft. RIFF sendiri adalah format file yang ditujukan untuk menyimpan macam-macam data khususnya multimedia data terutama audio dan video.

Berikut ini adalah susunan dari file WAV :

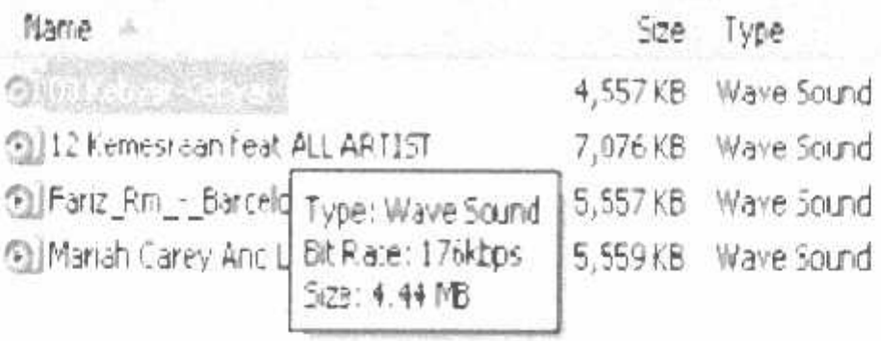


Gambar 4.9 Susunan Wave File

sumber :

4.3.3.2. Analisa Data File Lagu

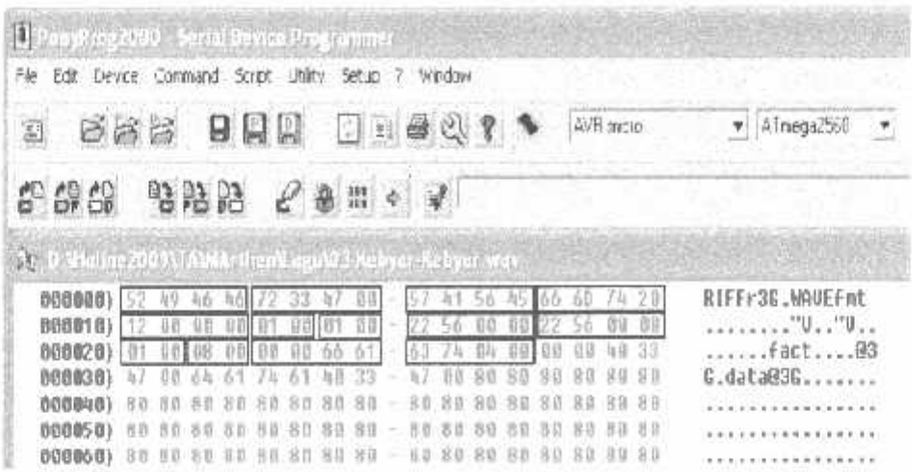
Gambar 4.9. menunjukkan file lagu yang nampak pada folder di PC, sedangkan Gambar 4.10 adalah properties untuk lagu berjudul Kebyar-Kebyar



Gambar 4.10 List file lagu yang nampak pada folder di PC

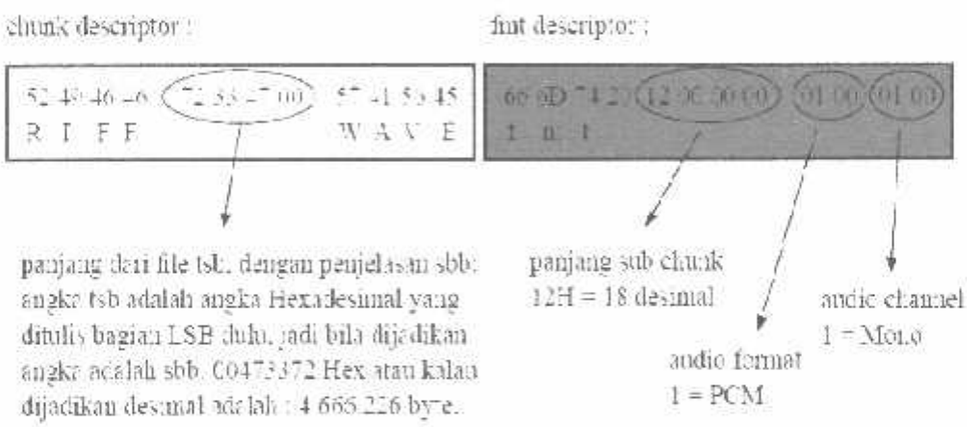
Gambar 4.11 Properties untuk lagu berjudul Kebyar-Kebyar

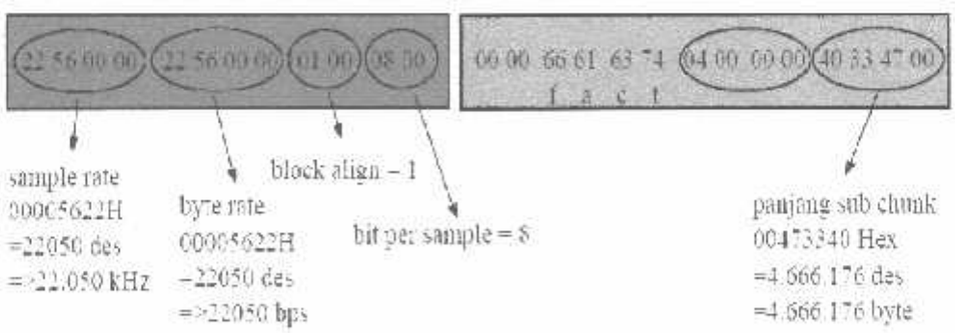
Berikut ini adalah header dari file bernama Kebyar-Kebyar.WAV

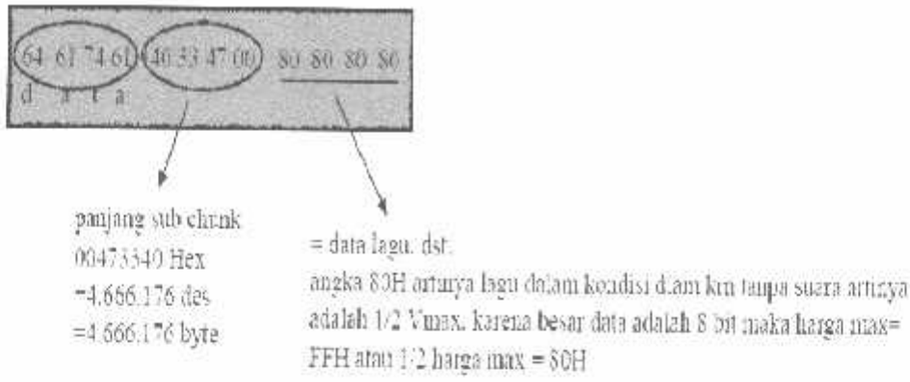


Gambar 4.12 Header dari file "Kebyar-kebyar.wav"

Analisa file Kebyar-Kebyar.WAV adalah sbb:







Dengan demikian menurut analisa file diatas, file lagu yang berupa PCM data dimulai dari urutan file ke 3A H atau ke 58.

4.3.3.3. Fungsi PWM

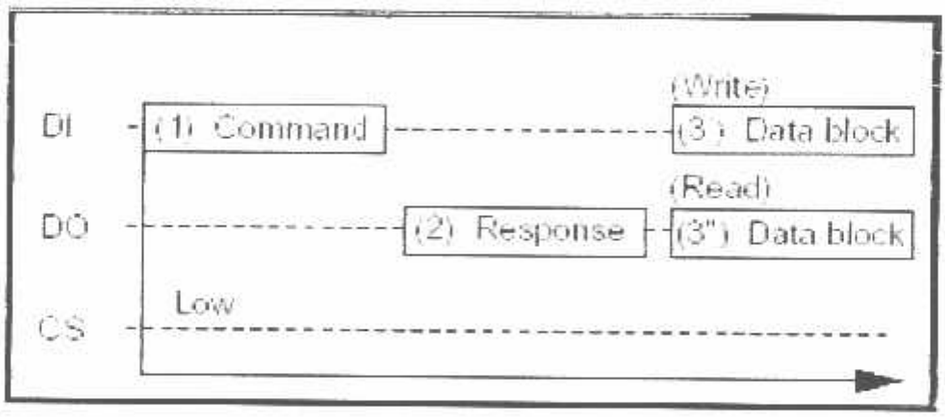
Data yang disimpan untuk aplikasi tugas akhir ini yaitu berupa file WAV didalam kartu MMC dalam bentuk data digital, berupa data PCM (pulse code modulation) dengan lebar data 8 bit , mono (1channel audio) , 22050 sampling rate *(lihat keterangan diatas).*Dengan demikian data tersebut harus diubah menjadi analog sebelum diberikan ke audio amplifier (penguat audio), dalam hal ini digunakan DAC (digital to analog conversion). Ada banyak macam DAC, dan yang digunakan pada perencanaan ini adalah DAC 1 bit berupa modulasi data PWM (pulse width modulation). Jadi mikrokontroler akan mengeluarkan pulsa PWM untuk diberikan ke rangkaian filter terlebih dulu supaya dihasilkan sinyal analog audio yang bisa diteruskan ke rangkaian penguat audio.

1. **Proses Pembacaan Data MMC Oleh Mikrokontroller**

   Berikut ini adalah bagaimana sebuah kartu memory MMC disambungkan dengan mikrokontroler dan cara-cara pembacaan data dari MMC ke mikrokontroler.

Menunjuk pada gambar 3.9 diatas, kartu MMC disambungkan ke mikrokontroler pada kaki SPI yang dalam perencanaan ini digunakan ATMEG8. SPI sendiri terdapat 3 kaki penting yaitu :

- MOSI (Master output slave input) adalah kaki I/O dimana kaki mikrokontroler (master) sebagai output data dan kaki MMC (slave) berupa input.
- MISO (Master input slave output) adalah kaki I/O dimana kaki mikrokontroler (master) difungsikan sebagai input data kaki MMC (slave) difungsikan sebagai output.
- SPICLK atau CLK adalah clock atau pulsa untuk mengatur jalannya data yang lewat MISO dan MOSI.



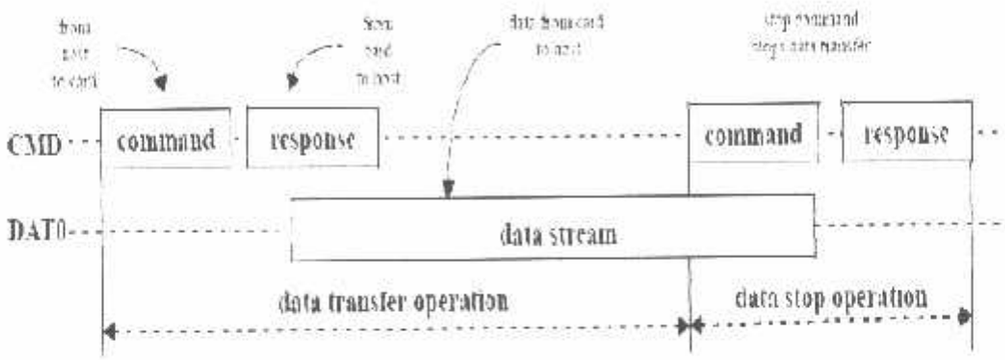
Gambar 4.13. MultiMediaCard Transfer Protocol

Gambar 3.10 adalah diagram aliran data untuk berhubungan dengan kartu MMC, hal yang perlu dilakukan yaitu:

- Aktifkan CS (chip select) dari MMC tersebut, yaitu kaki MMC no. 1 dibuat logika nol "0"

- Kirim data ke MMC ke kaki DI atau MOSI berupa command yang isi datanya bisa dilihat pada lampiran.
- MMC akan merespon pengiriman data melalui kaki DO atau (MISO).

Atau lebih jelasnya untuk pengiriman data streaming audio bisa digambarkan sbb:



Gambar 4.14. Sequential Read Operation

Dari sisi software bisa dijelaskan sbb:

1. Inisialisasi MMC

   MMC perlu diinisialisasi sebelum dilakukan baca/tulis, berikut adalah program untuk keperluan ini:

```
char MMC_Init(void)
{ // init SPI
      char i;
      PORTB |= (1 << SPICS); // disable MMC
      // start MMC in SPI mode
      for(i=0; i < 10; i++) SPI(0xFF); // send
10*8=80 clock pulses
      PORTB &= ~(1 << SPICS); // enable MMC
```

```
        if (Command1(0x40,0,0,0x95) != 1) goto
mmcerror; // reset MMC

st: // if there is no MMC, prg. loops here
        if (Command1(0x41,0,0,0xFF) !=0) goto st;
        return (1);
mmcerror:
        return (0);
}
```

Inti dari program diatas adalah MMC di set pada mode SPI, lalu dikirim pulsa sebanyak 80 kali untuk sinkronisasi awal, setelah diberi command(0x40,0,0,0x95) maka MMC harus menjawab 1, bila tidak maka proses inisialisasi diatas gagal atau terjadi error.

2. Membaca streaming data audio dari MMC

Ini adalah proses yang terpenting pada perencanaan Tugas Akhir ini, yaitu bagaimana data audio yang tersimpan berupa file WAV atau dalam bentuk urutan data PCM bisa dibaca oleh mikrokontroler Atmel ATMEGA8.

Berikut ini adalah potongan program tsb:

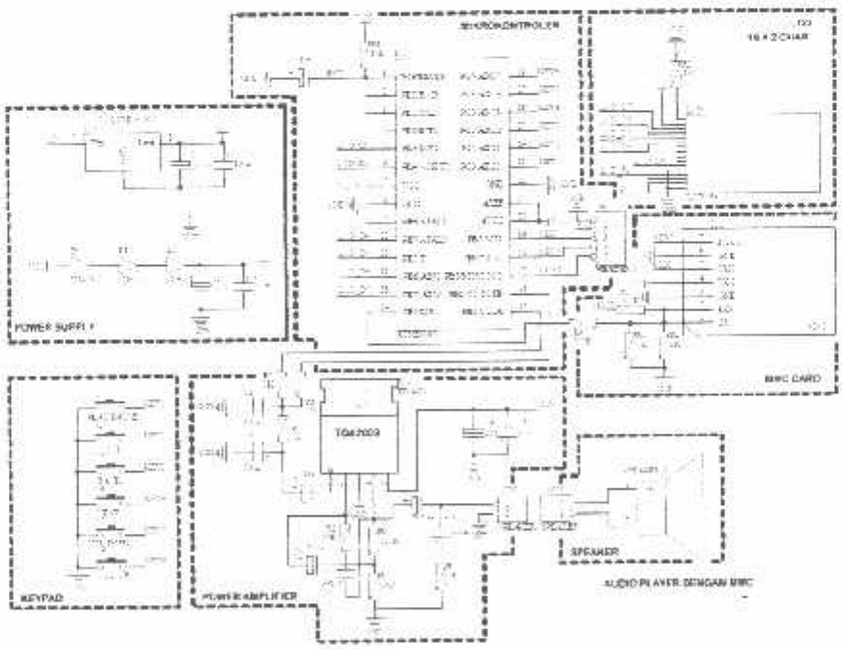
```
for(j=0; j < 512; j++)
{
        SPDR = (0xFF); temp = vol >> 4; if (temp==0)
temp=1;
        while(!(SPSR & (1<<SPIF))); if (temp < 14) pwm
= SPDR/temp;
```

```
        // this intruction for delay only !
         speed++; speed++; speed--;
         speed--; speed--; speed++;
}
```

Inti dari program diatas adalah membaca data dari MMC per paket, dimana tiap paket data terdiri dari 512 data. Data hasil pembacaan berada pada register SPDR, dan data inilah data PCM audio yang tersimpan pada MMC tsb. Data ini diberikan ke register untuk mengaturan PWM setelah di olah dengan posisi volume saat itu dengan rumus :

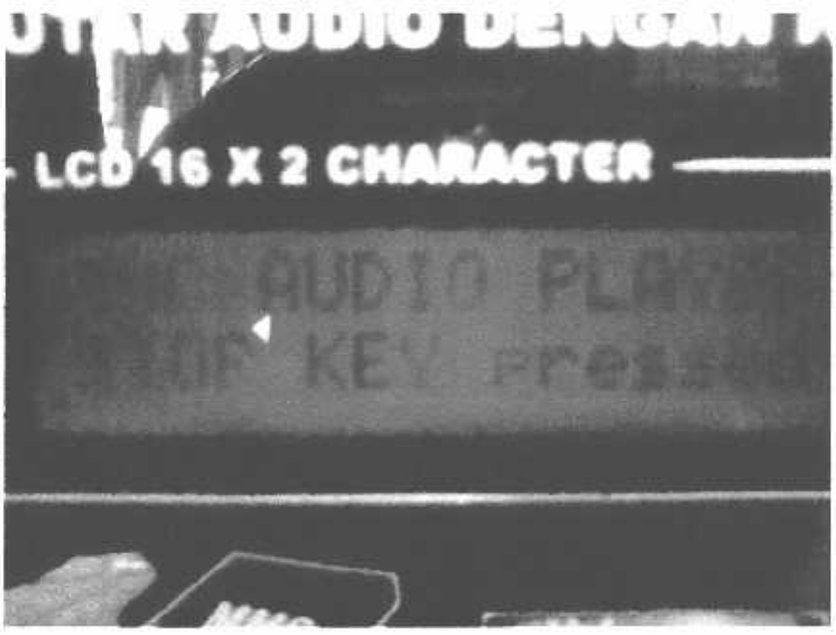
```
pwm = SPDR / [Volume]
```

jadi sinyal PWM yang dihasilkan nanti tergantung dari data PCM dan nilai Volume.



Gambar 4.15 Diagram Blok Keseluruhan

4.3.4. Hasil Pengujian



BAB V

KESIMPULAN

5.1 Kesimpulan

Dari hasil pengujian dan dari alat yang telah dibuat maka dapat disimpulkan:

- Pada hasil pengujian sinyal MMC dan sinyal PWM, apabila diberi sinyal inputan, maka sinyal PWM pada osciloscope akan berubah/ bergetar sesuai perubahan sinyal input.
- Pada hasil pengujian sinyal keluaran dan sistem yang direncanakan, bahwa apabila tidak diberi filter, maka bentuk sinyalnya konstan , dan suara analog yang dihasilkan tidak jelas.
- Suara analog yang dihasilkan berupa mono
- Karena menggunakan filter RC maka suara analaog yang dihasilkan tidak begitu jernih.
- File Lagu hany berformat Wave.
- **5.2. Saran**
  - Agar alat ini dapat bekerja dengan baik, diperlukan komponen yang lebih baik.
  - Untuk perancangan ke depan diharapkan alat ini dapat disempurnakan untuk memperoleh hasil yang lebih baik.
  - Diperlukan ketelitian untuk mendapatkan hasil yang memuaskan.

5. Seseorang yang paling aku sayangi, Irma Rusli Lestari yang selalu membantu dan memberikan semangat dalam penyelesaian skripsi ini.
6. Teman–teman teknik elektronika S-1 yang telah membantu dalam penyelesaian skripsi ini saya ucapkan terima kasih.
7. Teman–teman di Goku Camp di Jl. Raya Candi II/ 76, terimakasih atas dukungannya.

Penulis menyadari bahwa laporan ini masih banyak yang perlu disempurnakan. Oleh sebab itu kritik dan saran yang bersifat membangun sangat diharapkan dari pembaca khususnya mahasiswa Teknik Jurusan Elekto ITN Malang, agar di masa datang akan lebih baik lagi. Harapan dari penulis semoga skripsi ini dapat bermanfaat.

Malang, Mei 2009

Penulis

DAFTAR PUSTAKA

1. www.atmel.com, Data Sheet Mikrokontroller Atmega 8, Atmel Corp.
2. www.analog.com/processors, *ADSP-2126x SHARC DSP Peripherals Manual*, Rev 2.0, January 2004, Analog Devices, Inc
3. www.sharpsma.com, The LH 79520 Universal Microcontroller User's Guide – by Sharp Microelectronics of the Americas
4. www.samsung.com, Semiconductor Flash Memory Product Palnning & Application
5. www.hitachisemiconductor.com, **HB28E016MM2/HB28D032MM2** MultiMediaCard™ ; 16 MByte/32 MByte/64 MByte/128 MByte
6. MultiMediaCard System Summary Version 4.1 Official Release (c) April 2005 MMCA
7. www.st.com, **TDA2003;** 10W CAR RADIO AUDIO AMPLIFIER
8. Dot Matrix Liquid Crystal Display Modules Seiko Instruments GmbH

LAMPIRAN

INSTITUT TEKNOLOGI NASIONAL
JL. BENDUNGAN SIGURA-GURA 2
MALANG

FORMULIR BIMBINGAN SKRIPSI

Nama : MARTHEN L. S.
Nim : 01.17.025
Masa Bimbingan : 16 FEBRUARI 2009 s/d 16 AGUSTUS 2009
Judul Skripsi : **PERENCANAAN DAN PEMBUATAN ALAT PEMUTAR MUSIK DIGITAL YANG DISIMPAN PADA MEDIA KARTU MEMORI MMC (MULTI MEDIA CARD) BERBASIS MIKROKONTROLLER ATMEGA 8**

No.	Tanggal	Uraian	Paraf Pembimbing
1.	16/02 2009	Bab I	
2.	23/02 2009	Bab II	
3.	9/03 2009	Bab III	
4.	16/03 2009	Bab IV	
5.	18/03 2009	Bab V	
6.	21/03 2009	Selesai	
7.			
8.			
9.			
10.			

Malang,
Dosen Pembimbing

Ir. F. Yudi Limpraptono, MT
NIP. 103 950 0274

INSTITUT TEKNOLOGI NASIONAL
FAKULTAS TEKNOLOGI INDUSTRI
JURUSAN TEKNIK ELEKTRO S1
KONSENTRASI TEKNIK ELEKTRONIKA

PERSETUJUAN PERBAIKAN SKRIPSI

Dari hasil Ujian Kompreherensip Jenjang Strata Satu (S-1) Jurusan Teknik Elektro Konsentrasi Elektronika yang diselenggarakan pada :

Hari : Selasa
Tanggal : 24 Maret 2009

Telah dilaksanakan Perbaikan skripsi oleh saudara :

Nama : Marthen Leunard Simaela
N I M : 01.17.025

Perbaikan tersebut meliputi :

No	Materi Perbaikan	Paraf Dosen
1	File Format WAV, Fungsi PWM, Proses Pembacaan File dari MMC	

Malang, April 2009

Disetujui Oleh

Penguji I

Joseph Dedy Irawan ST,MT
NIP.132315178

Mengetahui

Dosen Pembimbing

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

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 : MARTHEN
NIM : 01.17.025
Perbaikan meliputi :

-) PROSES PEMBACAAN FILE dr MMC
-) FORMAT FILE WAVE
-) FUNGSI PWM,
-) EAT → JIKA Lagu-nalum kurus dilatar.

Malang, 200

( )

INSTITUT TEKNOLOGI NASIONAL
FAKULTAS TEKNOLOGI INDUSTRI
JURUSAN TEKNIK ELEKTRO S1
KONSENTRASI TEKNIK ELEKTRONIKA

PERSETUJUAN PERBAIKAN SKRIPSI

Dari hasil Ujian Kompreherensip Jenjang Strata Satu (S-1) Jurusan Teknik Elektro Konsentrasi Elektronika yang diselenggarakan pada :

Hari : Selasa
Tanggal : 24 Maret 2009

Telah dilaksanakan Perbaikan skripsi oleh saudara :

Nama : Marthen Leunard Simaela
N I M : 01.17.025

Perbaikan tersebut meliputi :

No	Materi Perbaikan	Paraf Dosen
1	Judul Abstrak	
2	Abstrak Disempurnakan	
3	Kesimpulan Disempurnakan	

Malang, April 2009

Disetujui Oleh

Penguji II

Bambanag Prio Hartono, ST
NIP.Y.1028400082

Mengetahui

Dosen Pembimbing

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

INSTITUT TEKNOLOGI NASIONAL MALANG
FAKULTAS TEKNOLOGI INDUSTRI
JURUSAN TEKNIK ELEKTRO

Formulir Perbaikan Ujian Skripsi

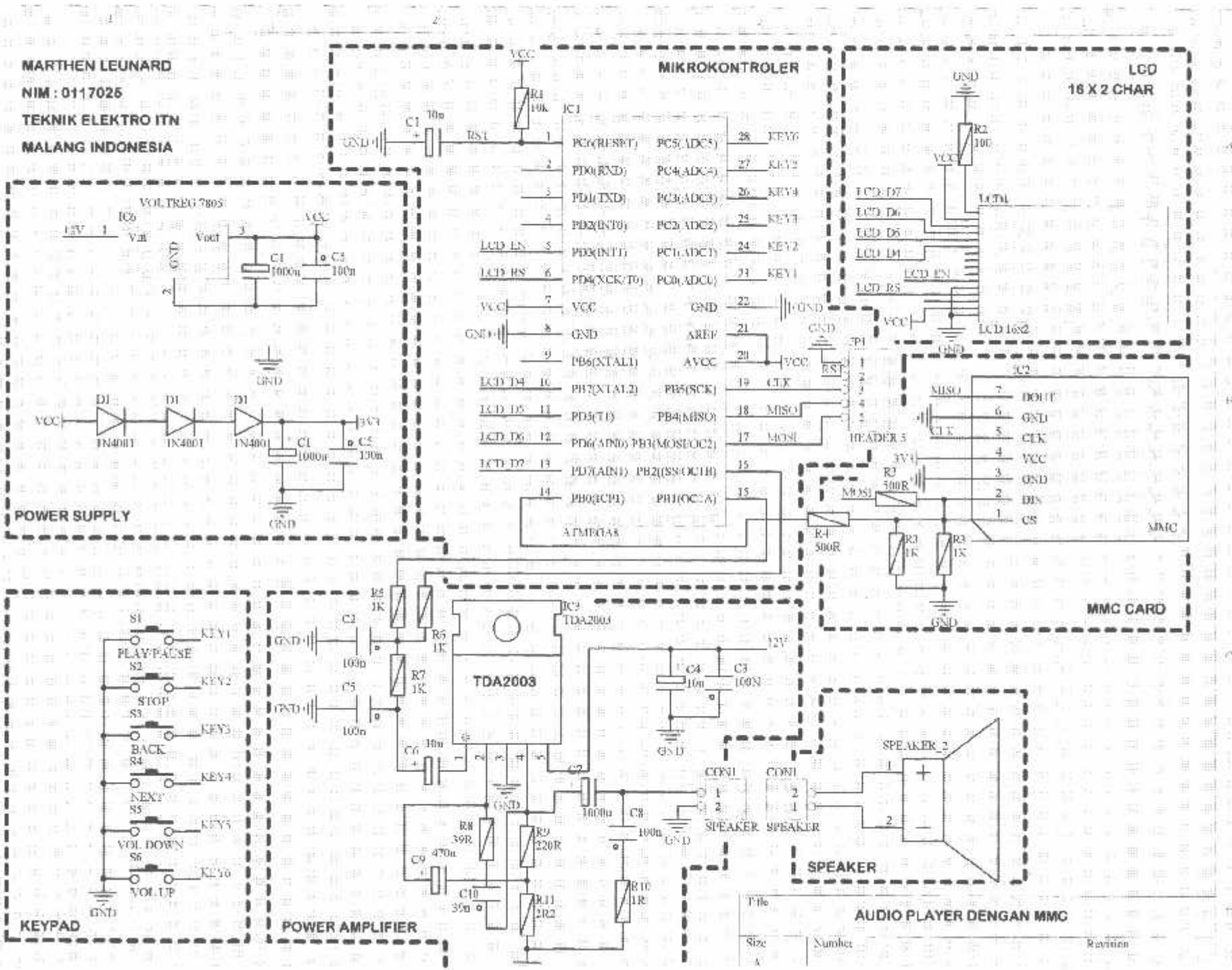
Dalam pelaksanaan Ujian Skripsi Jenjang Strata 1 Jurusan Teknik Elektro Konsentrasi T. Energi Listrik / T. Elektronika / T.Infokom, maka perlu adanya perbaikan skripsi untuk mahasiswa :

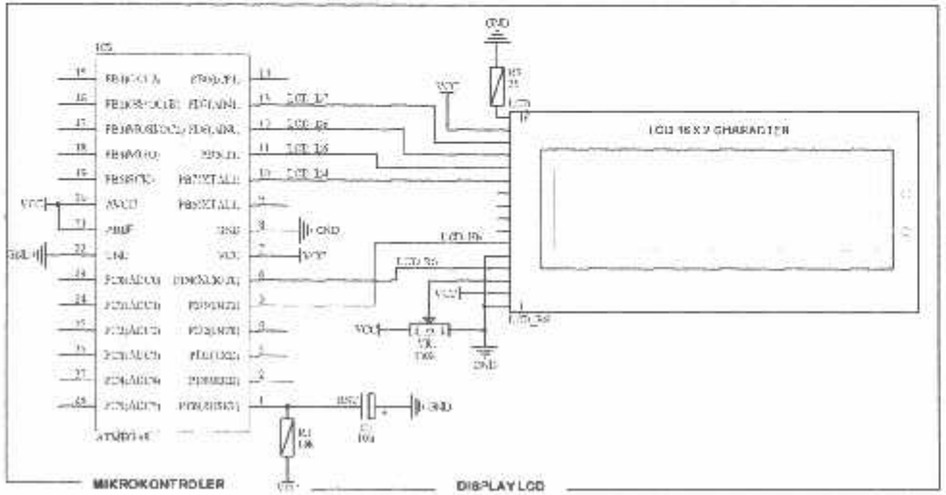
NAMA : Norther L.S
NIM : 0117025
Perbaikan meliputi :

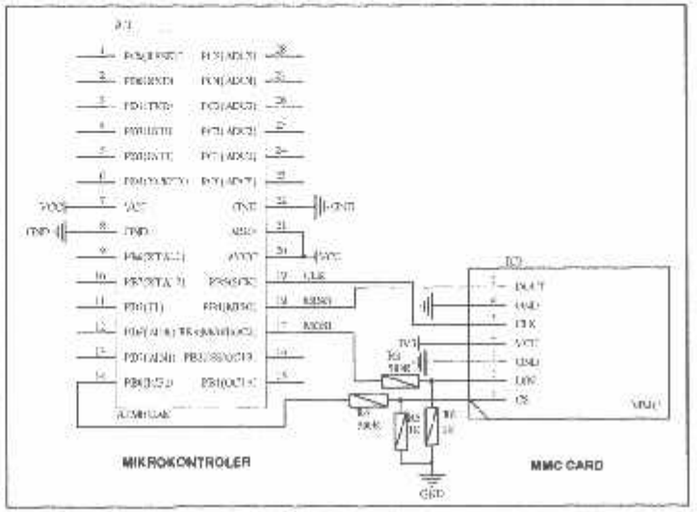
- Judul Abstrak.
- Abstrak disemparnakan?
- latar belakang. di beri Pustaka ?
- Ref 16 untuk Ket. Tabel ispari.
- Kesimpulan di semparnakan ?

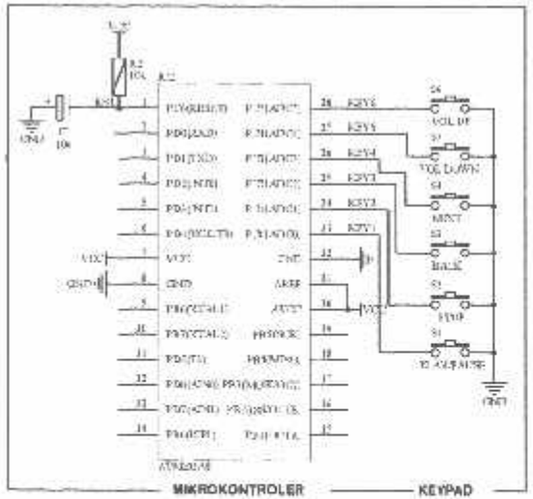
Malang, 200

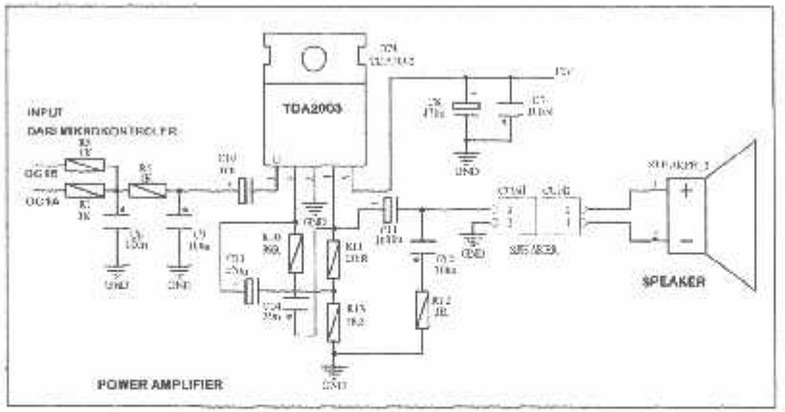
(_____)


MARTHEN LEUNARD
NIM : 0117025
TEKNIK ELEKTRO ITN
MALANG INDONESIA
MIKROKONTROLER
LCD
16 X 2 CHAR
LCD 16x2
POWER SUPPLY
VOLTREG 7805
IC6
1N4001
KEYPAD
S1
PLAY/PAUSE
S2
STOP
S3
BACK
S4
NEXT
S5
VOL DOWN
S6
VOL UP
POWER AMPLIFIER
TDA2003
IC3
ATMEGA8
MMC CARD
MMC
HEADER 5
SPEAKER
SPEAKER_2
AUDIO PLAYER DENGAN MMC


LCD 16 X 2 CHARACTER
MIKROKONTROLER
DISPLAY LCD


MIKROKONTROLER
MMC CARD


MIKROKONTROLER
KEYPAD


INPUT
DARI MIKROKONTROLER
TDA2003
SPEAKER
POWER AMPLIFIER

atures

igh-performance, Low-power AVR® 8-bit Microcontroller
dvanced 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

onvolatile 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
- 1K Byte Internal SRAM
- Programming Lock for Software Security

eripheral Features
- Two 8-bit Timer/Counters with Separate Prescaler, one Compare Mode
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Real Time Counter with Separate Oscillator
- Three PWM Channels
- 8-channel ADC in TQFP and QFN/MLF package
  Eight Channels 10-bit Accuracy
- 6-channel ADC in PDIP package
  Eight Channels 10-bit Accuracy
- 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

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

O and Packages
- 23 Programmable I/O Lines
- 28-lead PDIP, 32-lead TQFP, and 32-pad QFN/MLF

perating Voltages
- 2.7 - 5.5V (ATmega8L)
- 4.5 - 5.5V (ATmega8)

peed Grades
- 0 - 8 MHz (ATmega8L)
- 0 - 16 MHz (ATmega8)

ower Consumption at 4 Mhz, 3V, 25°C
- Active: 3.6 mA
- Idle Mode: 1.0 mA
- Power-down Mode: 0.5 µA

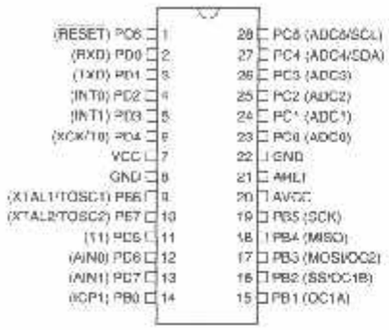
8-bit AVR® with 8K Bytes In-System Programmable Flash

ATmega8
ATmega8L

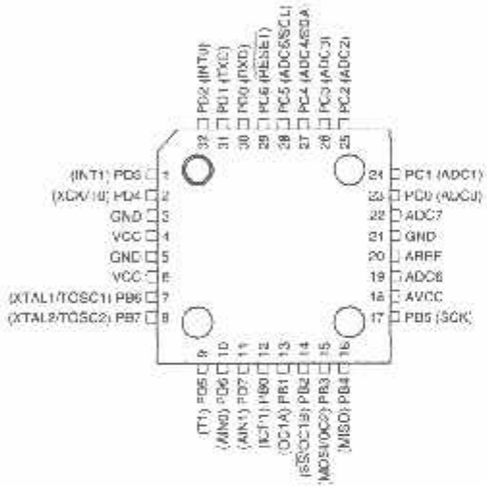
2486P–AVR–02/06

n Configurations

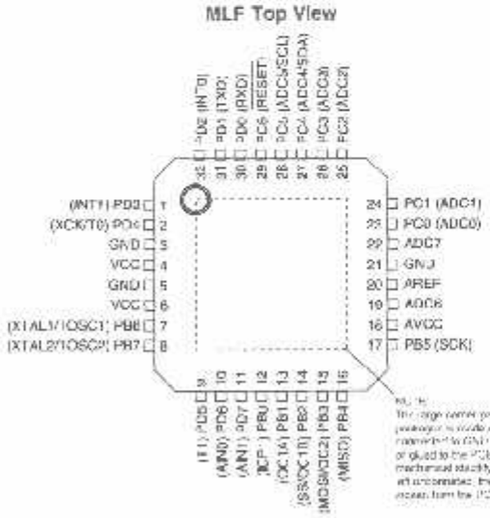
PDIP



TQFP Top View



MLF Top View

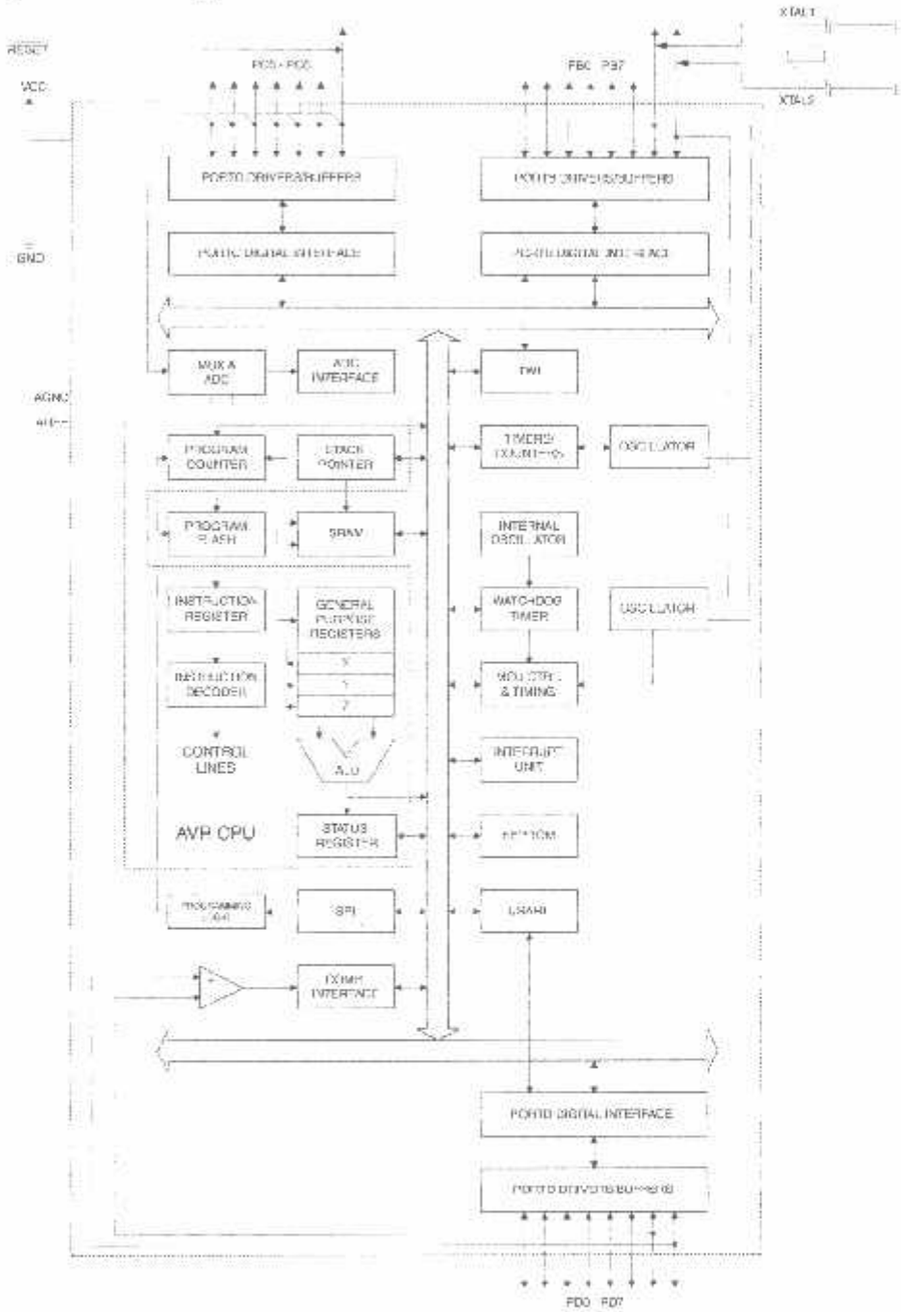


verview

The ATmega8 is a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega8 achieves throughputs approaching 1 MIPS per MHz, allowing the system designer to optimize power consumption versus processing speed.

ock Diagram

Figure 1. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 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 ATmega8 provides the following features: 8K bytes of In-System Programmable Flash with Read-While-Write capabilities, 512 bytes of EEPROM, 1K byte of SRAM, 23 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, a 6-channel ADC (eight channels in TQFP and QFN/MLF packages) with 10-bit accuracy, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and five 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.

The device is manufactured using Atmel's high density non-volatile memory technology. The Flash Program memory can be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile 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 ATmega8 is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many embedded control applications.

The ATmega8 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.

scialmer

Typical values contained in this datasheet 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.

1 Descriptions

C	Digital supply voltage.
D	Ground.
t B (PB7..PB0) AL1/XTAL2/TOSC1/TOSC2	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. Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit. Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier. If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set. The various special features of Port B are elaborated in "Alternate Functions of Port B" on page 58 and "System Clock and Clock Options" on page 25.
t C (PC5..PC0)	Port C is an 7-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.
S/$\overline{\text{RESET}}$	If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C. If the RSTDISBL Fuse is unprogrammed, PC6 is used as a 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 38. Shorter pulses are not guaranteed to generate a Reset. The various special features of Port C are elaborated on page 61.
t D (PD7..PD0)	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. Port D also serves the functions of various special features of the ATmega8 as listed on page 63.
SET	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 38. Shorter pulses are not guaranteed to generate a reset.

cc	AV_{CC} is the supply voltage pin for the A/D Converter, Port C (3..0), and ADC (7..6). It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter. Note that Port C (5..4) use digital supply voltage, V_{CC}.
EF	AREF is the analog reference pin for the A/D Converter.
C7..6 (TQFP and QFN/MLF :kage Only)	In the TQFP and QFN/MLF package, ADC7..6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

esources

A comprehensive set of development tools, application notes and datasheets are available for download on http://www.atmel.com/avr.

bout Code
:amples

This datasheet 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.

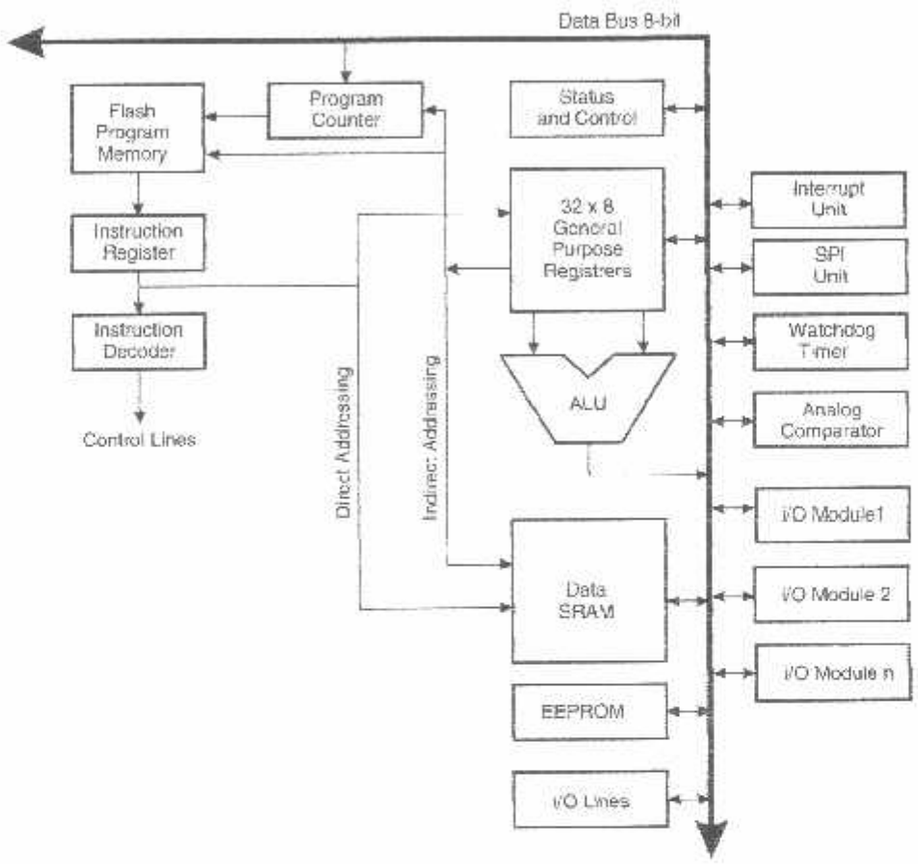
/R CPU Core

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.

chitectural Overview

Figure 2. 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 Reprogrammable 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 the 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-register, 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.

The 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.

thmetic Logic Unit – U

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.

tus 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 ⊕ 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.

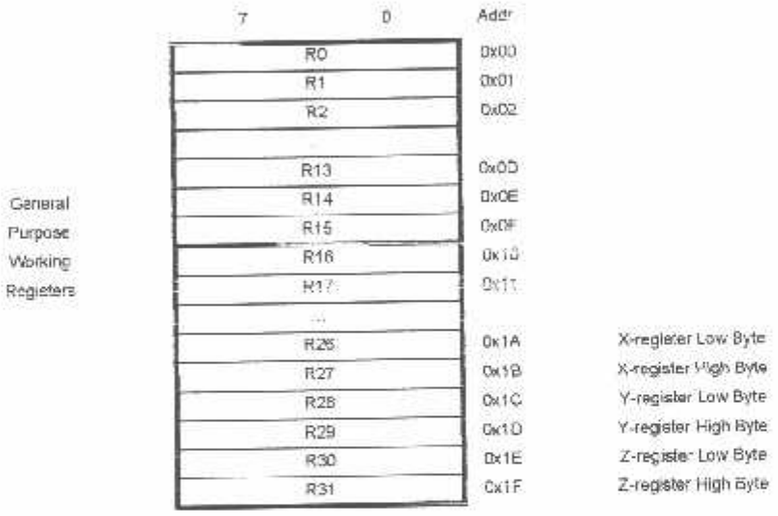
neral Purpose gister File

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

Figure 3. AVR CPU General Purpose Working Registers



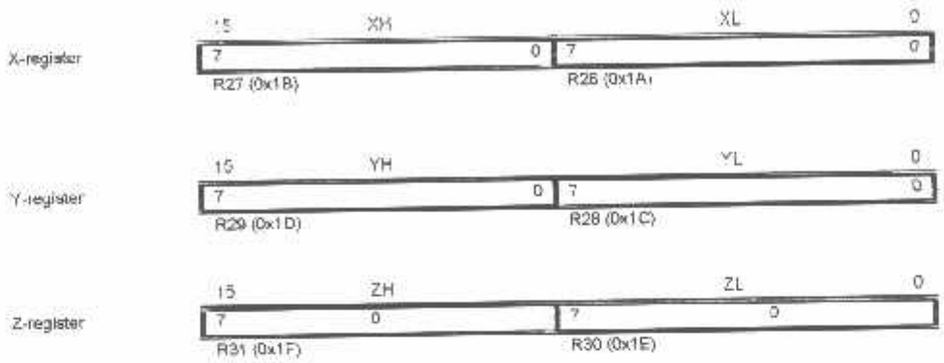
Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

As shown in Figure 3, 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 4.

Figure 4. 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 address 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	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	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	

ıtruction Execution ning

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 5 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 5. The Parallel Instruction Fetches and Instruction Executions

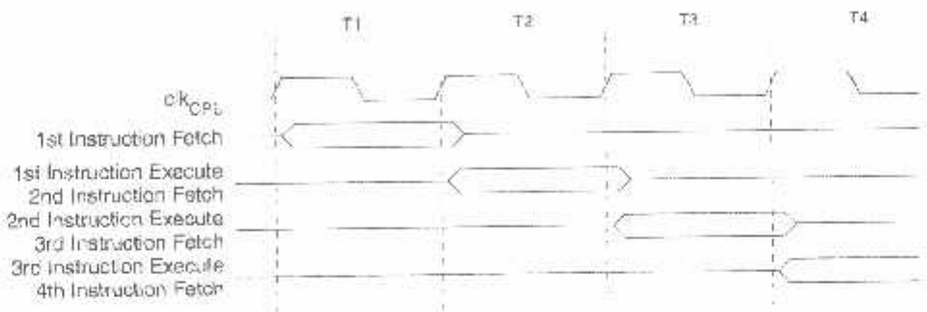
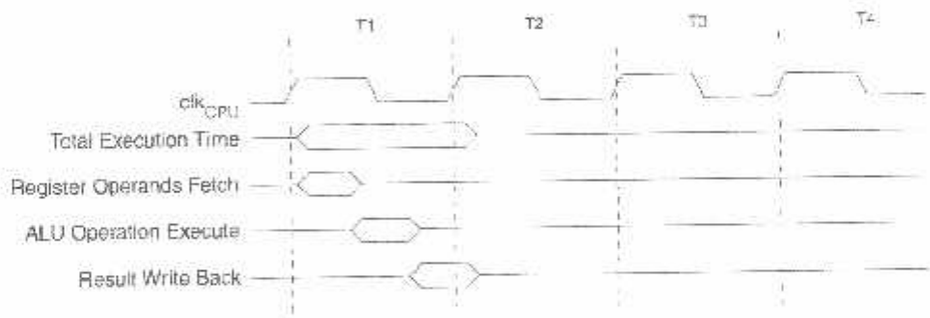


Figure 6 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 6. Single Cycle ALU Operation



set and Interrupt ndling

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 222 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 46. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0. The Interrupt Vectors can be moved to the start

of the boot Flash section by setting the Interrupt Vector Select (IVSEL) bit in the General Interrupt Control Register (GICR). Refer to "Interrupts" on page 46 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 209.

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

```
in r16, SREG     ; store SREG value
cli    ; disable interrupts during timed sequence
sbi EECR, EEMWE  ; start EEPROM write
sbi EECR, EEWE
out SREG, r16    ; restore SREG value (I-bit)
```

C Code Example

```
char 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 the following example.

Assembly Code Example

```
sei  ; set global interrupt enable
sleep; enter sleep, waiting for interrupt
; note: will enter sleep before any pending
; interrupt(s)
```

C Code Example

```
_SEI(); /* set global interrupt enable */
_SLEEP(); /* enter sleep, waiting for interrupt */
/* note: will enter sleep before any pending interrupt(s) */
```

errupt 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 4-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 (2 bytes) is popped back from the Stack, the Stack Pointer is incremented by 2, and the I-bit in SREG is set.

/R ATmega8 emories

This section describes the different memories in the ATmega8. The AVR architecture has two main memory spaces, the Data memory and the Program Memory space. In addition, the ATmega8 features an EEPROM Memory for data storage. All three memory spaces are linear and regular.

System programmable Flash ogram Memory

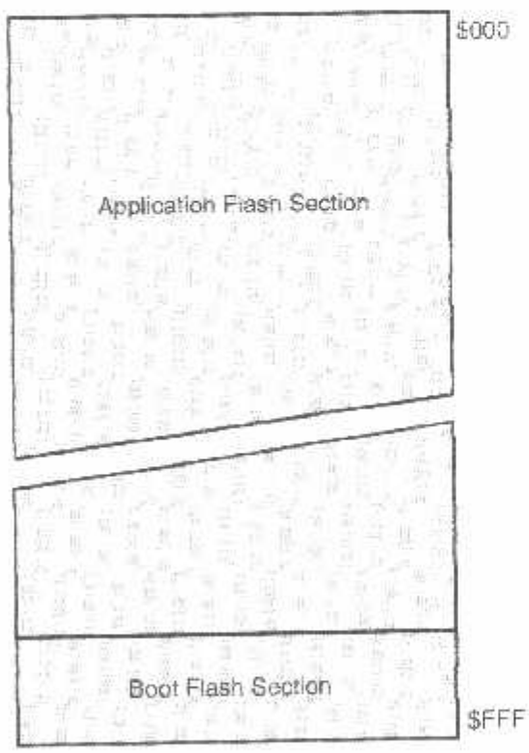
The ATmega8 contains 8K bytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16- or 32-bits wide, the Flash is organized as 4K x 16 bits. For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section.

The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega8 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 209. "Memory Programming" on page 222 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 14.

Figure 7. Program Memory Map



AM Data Memory

Figure 8 shows how the ATmega8 SRAM Memory is organized.

The lower 1120 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 1024 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.

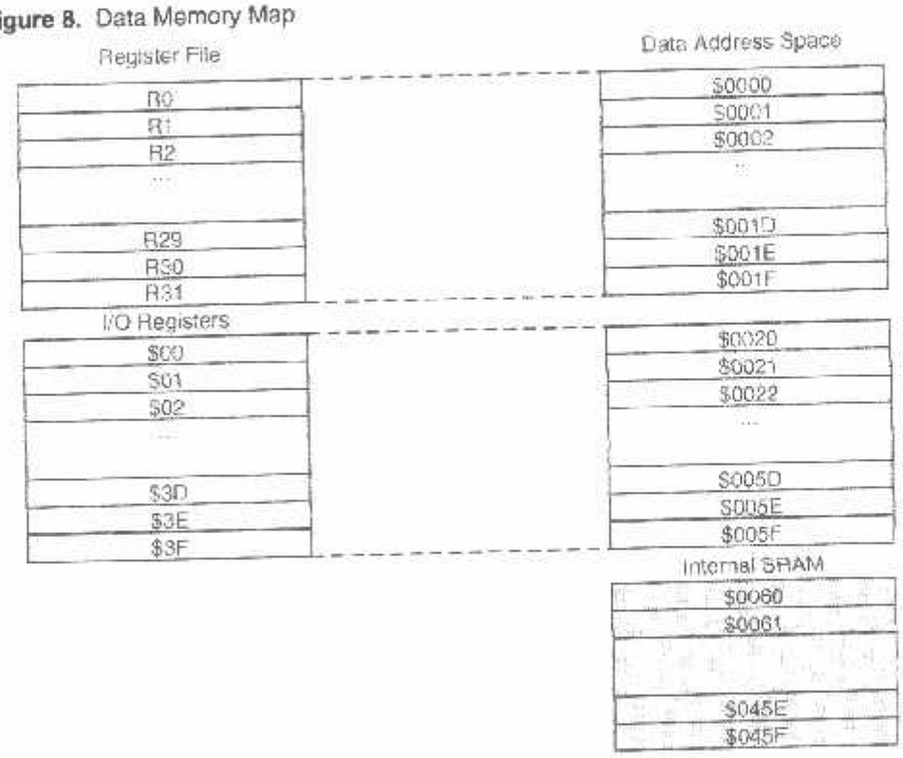
The direct addressing reaches the entire data space.

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 1024 bytes of internal data SRAM in the ATmega8 are all accessible through all these addressing modes. The Register File is described in "General Purpose Register File" on page 12.

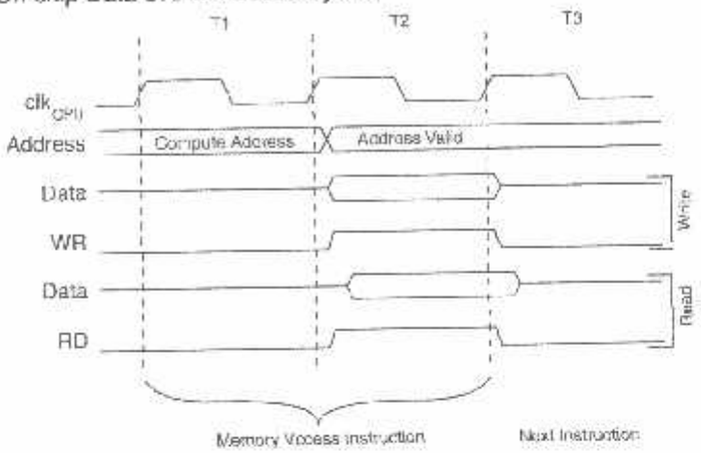
Figure 8. Data Memory Map



ta Memory Access nes

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 9.

Figure 9. On-chip Data SRAM Access Cycles



EPROM Data Memory

The ATmega8 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 bellow, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

"Memory Programming" on page 222 contains a detailed description on EEPROM Programming in SPI- or Parallel Programming mode.

PROM 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 on page 21. 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 23. 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
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	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	X	
	X	X	X	X	X	X	X	X	

- **Bits 15..9 – Res: Reserved Bits**

These bits are reserved bits in the ATmega8 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 ATmega8 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

The MultiMediaCard

Based On System Specification Version 4.1

System Summary

MMCA Technical Committee

1 General Description

The MultiMediaCard is an universal low cost data storage and communication media. It is designed to cover a wide area of applications as smart phones, cameras, organizers, PDAs, digital recorders, MP3 players, pagers, electronic toys, etc. Targeted features are high mobility and high performance at a low cost price. These features include low power consumption and high data throughput at the memory card interface.

The MultiMediaCard communication is based on an advanced 13-pin bus. The communication protocol is defined as a part of this standard and referred to as the MultiMediaCard mode. To ensure compatibility with existing controllers, the cards may offer, in addition to the MultiMediaCard mode, an alternate communication protocol which is based on the SPI standard.

To provide for the forecasted migration of CMOS power (V_{DD}) requirements and for compatibility and integrity of MultiMediaCard systems, two types of MultiMediaCards are defined in this standard specification, which differ only in the valid range of system V_{DD}. These two card types are referred to as High Voltage MultiMediaCard and Dual Voltage MultiMediaCard.

The purpose of the system specification is the definition of the MultiMediaCard, its environment and handling. It gives guidelines for a system designer. The system specification also defines a tool box (a set of macro functions and algorithms) which contributes to reducing the design-in costs.

As used in this document, "shall" or "will" denotes a mandatory provision of the standard. "Should" denotes a provision that is recommended but not mandatory. "May" denotes a feature whose presence does not preclude compliance, that may or may not be present at the option of the implementor

2 System Features

The MultiMediaCard System has a wide variety of system features, whose comprehensive elements serves several purposes, which include:

- Covering a broad category of applications from smart phones and PDAs to digital recorders and toys
- Facilitating the work of designers who seek to develop applications with their own advanced and enhanced features
- Maintainng compatibility and compliance with current electronic, communication, data and error handling standards

The following list identifies the main features of the MultiMediaCard System, which:

- **Is targeted for portable and stationary applications**
- **Has these System Voltage (V_{DD}) Ranges:**

	High Voltage MultiMediaCard	Dual Voltage MultiMediaCard
Communication	2.7 - 3.6	1.65 - 1.95, 2.7 - 3.6[1]
Memory Access	2.7 - 3.6	1.65 - 1.95, 2.7 - 3.6

1) V_{DD} range: 1.95V - 2.7V is not supported.

- **Includes MMCplus and MMCmobile definitions**
- **Is designed for read-only, read/write and I/O cards**
- **Supports card clock frequencies of 0-20MHz, 0-26MHz or 0-52MHz**
- **Has a maximum data rate up to 416Mbits/sec.**
- **Has a defined minimum performance**
- **Maintains card support for three different data bus width modes: 1bit (default), 4bit and 8bit**
- **Includes password protection of data**
- **Supports basic file formats for high data interchangeability**
- **Includes application specific commands**
- **Enables correction of memory field errors**
- **Has built-in write protection features, which may be permanent or temporary**
- **Includes a simple erase mechanism**
- **Maintains full backward compatibility with previous MultiMediaCard systems (1 bit data bus, multi-card systems)**
- **Ensures that new hosts retain full compatibility with previous versions of MultiMediaCards (backward compatibility).**
- **Supports two form factors: Normal size(24mm x 32mm x 1.4mm) and Reduced size (24mm**

x 18mm x 1.4mm)

Supports multiple command sets

Includes attributes of the available operation modes:

MultiMediaCard Mode	SPI Mode
Ten-wire bus (clock, 1 bit command, 8 bit data bus)	Three-wire serial data bus (clock, dataIn, dataOut) + card specific CS signal.
Card selection is done through an assigned unique card address to maintain backwards compatibility to prior versions of the specification	Card selection via a hardware CS signal
One card per MultiMediaCard bus	Card requires a dedicated CS signal.
Easy identification and assignment of session address	Not available. Card selection via a hardware CS signal
Error-protected data transfer	Optional. A non-protected data transfer mode is available.
Sequential and Single/Multiple block Read/Write commands	Single/Multiple block Read/Write commands

Table 1: MMC System Operational Modes

5 The MultiMediaCard Bus

The MultiMediaCard bus has ten communication lines and three supply lines:

- CMD: Command is a bidirectional signal. The host and card drivers are operating in two modes, open drain and push/pull.
- DAT0-7: Data lines are bidirectional signals. Host and card drivers are operating in push-pull mode
- CLK: Clock is a host to card signal. CLK operates in push-pull mode
- V_{DD}: V_{DD} is the power supply line for all cards.
- V_{SS1}, V_{SS2} are two ground lines.

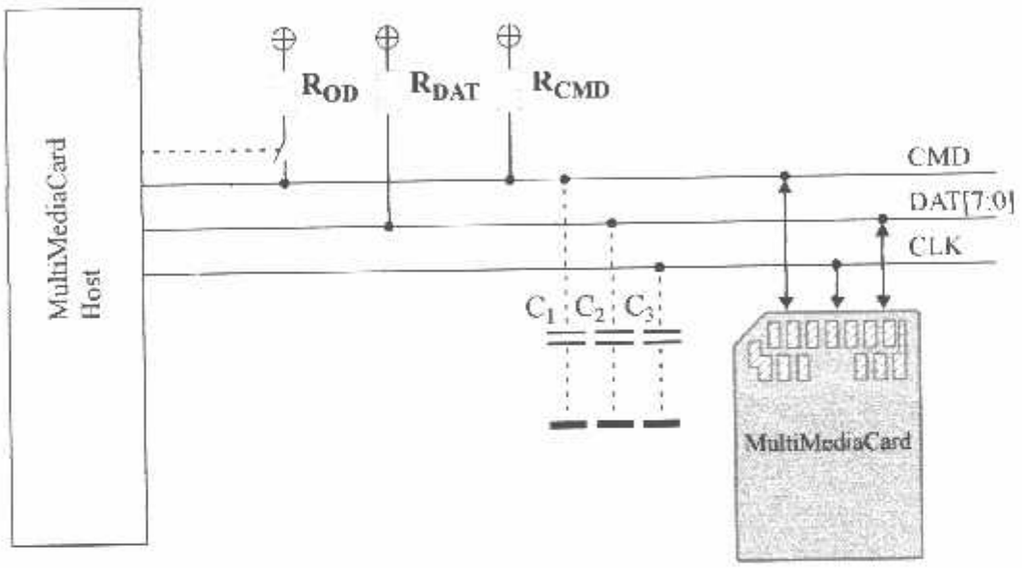


Figure 17: Bus Circuitry Diagram

The R_{OD} is switched on and off by the host synchronously to the open-drain and push-pull mode transitions. The host does not have to have open drain drivers, but must recognize this mode to switch on the R_{OD}. R_{DAT} and R_{CMD} are pull-up resistors protecting the CMD and the DAT lines against bus floating when no card is inserted or when all card drivers are in a high-impedance mode.

A constant current source can replace the R_{OD} by achieving a better performance (constant slopes for the signal rising and falling edges). If the host does not allow the switchable R_{OD} implementation, a fixed R_{CMD} can be used. Consequently the maximum operating frequency in the open drain mode has to be reduced if the used R_{CMD} value is higher than the minimal one.

To guarantee the proper sequence of card pin connection during hot insertion, the use of either a special hot-insertion capable card connector or an auto-detect loop on the host side (or some similar mechanism) is mandatory.

No card shall be damaged by inserting or removing a card into the MultiMediaCard bus even when the power (V_{DD}) is up. Data transfer operations are protected by CRC codes, therefore any bit changes induced by card insertion and removal can be detected by the MultiMediaCard bus master.

6 SPI Mode

6.1 Introduction

The SPI mode consists of a secondary, optional communication protocol which is offered by Flash-based MultiMediaCards. This mode is a subset of the MultiMediaCard protocol, designed to communicate with a SPI channel, commonly found in Motorola's (and lately a few other vendors') microcontrollers. The interface is selected during the first reset command after power up (CMD0) and cannot be changed once the part is powered on.

The SPI standard only defines the physical link and not the complete data transfer protocol. The MultiMediaCard SPI implementation uses a subset of the MultiMediaCard protocol and command set. It is intended to be used by systems which typically require one card and have lower data transfer rates, compared to MultiMediaCard-protocol-based systems. From the application point of view, the advantage of the SPI mode is the capability of using an off-the-shelf host, hence, reducing the design-in effort to minimum. The disadvantage is the loss of performance of the SPI mode versus MultiMediaCard mode (lower data transfer rate, hardware CS, etc.).

6.2 SPI Interface Concept

The Serial Peripheral Interface (SPI) is a general purpose synchronous serial interface originally found on certain Motorola microcontrollers. A virtually identical interface can now be found on certain TI and SGS Thomson microcontrollers as well.

The MultiMediaCard SPI interface is compatible with SPI hosts available on the market. As in any other SPI device, the MultiMediaCard SPI channel consists of the following four signals:

CS: Host to card Chip Select signal.

CLK: Host to card clock signal

DataIn: Host to card data signal

DataOut: Card to host data signal.

Another SPI common characteristic is byte transfers, which is implemented in the card as well. All data tokens are multiples of bytes (8 bit) and always byte aligned to the CS signal.

6.3 SPI Bus Topology

The card identification and addressing methods are replaced by a hardware Chip Select (CS) signal. There are no broadcast commands. For every command, a card (slave) is selected by asserting (active low) the CS signal (see Figure 18).

The CS signal must be continuously active for the duration of the SPI transaction (command, response and data). The only exception occurs during card programming, when the host can de-assert the CS signal without affecting the programming process.

The bidirectional CMD and DAT lines are replaced by unidirectional *dataIn* and *dataOut* signals. This eliminates the ability of executing commands while data is being read or written and, therefore, makes the sequential and multi block read/write operations obsolete. Only single block read/write commands are supported by the SPI channel.

The MultiMediaCard pin assignment in SPI mode (compared to MultiMediaCard mode) is given in Table 4.

Name	Available in SPI mode	Width [Bytes]	Description
CID	Yes	16	Card identification data (serial number, manufacturer ID, etc.)
RCA	No		
DSR	No		
CSD	Yes	16	Card-specific data, information about the card operation conditions.
EXT_CSD	Yes	512	Extended Card-specific data, information about the card supported properties and configured modes
OCR	Yes	32	Operation condition register.

Table 5: MultiMediaCard Registers In SPI Mode

6.5 SPI Bus Protocol

While the MultiMediaCard channel is based on command and data bit streams which are initiated by a start bit and terminated by a stop bit, the SPI channel is byte oriented. Every command or data block is built of 8-bit bytes and is byte aligned to the CS signal (i.e. the length is a multiple of 8 clock cycles).

Similar to the MultiMediaCard protocol, the SPI messages consist of command, response and data-block tokens. All communication between host and card is controlled by the host (master). The host starts every bus transaction by asserting the CS signal low.

The response behavior in the SPI mode differs from the MultiMediaCard mode in the following three aspects:

- The selected card always responds to the command.
- Additional (8, 16 & 40 bit) response structures are used.
- When the card encounters a data retrieval problem, it will respond with an error response (which replaces the expected data block) rather than by a time-out, as in the MultiMediaCard mode.

Only single and multiple block read/write operations are supported in SPI mode (sequential mode is not supported). In addition to the command response, every data block sent to the card during write operations will be responded to with a special data response token. A data block may be as big as one card write block and as small as a single byte. Partial block read/write operations are enabled by card options specified in the CSD register.

6.6 SPI Mode Transaction Packets

SPI mode transaction packets can be described by one of the following tokens:

- **Command tokens**: various formats and classes that support a set of card functions
- **Response tokens**: signals acknowledging the commands sent
- **Data tokens**: representing data transmission
- **Data error tokens**: identifying data read failure
- **Clearing Status bits**: a SPI mode status returned to the host

6.7 Card Registers

In SPI mode, only the OCR, CSD and CID registers are accessible. Their format is identical to the format in the MultiMediaCard mode. However, a few fields are irrelevant in SPI mode.

5.8 SPI Bus Timing Diagrams

The host must keep the clock running for at least N_{CR} clock cycles after receiving the card response. This restriction applies to both command and data response tokens. Timing diagrams for SPI bus mode use a well-specified set of schematics and abbreviations.

5.9 SPI Electrical Interface

Identical to MultiMediaCard mode with the exception of the programmable card output drivers option, which not supported in SPI mode.

5.10 SPI Bus Operating Conditions

The bus operating conditions are identical to MultiMediaCard mode.

5.11 Bus Timing

Identical to MultiMediaCard mode. The timing of the CS signal is the same as any other card input.

o obtain a license under or to any Third Party IP Rights, you must contact the applicable third party nd enter into a separate agreement with that party covering its Third Party IP Rights. All such ctivity, and all terms, conditions, warranties or representations associated with such activity, is olely between you and the applicable third party. MMCA SHALL HAVE NO OBLIGATION OR ESPONSIBILITY TO ASSIST YOU IN NEGOTIATING, EXECUTING, ADMINISTERING OR NFORCING ANY SUCH AGREEMENTS, AND MMCA SHALL HAVE NO LIABILITY RISING OUT OF ANY SUCH AGREEMENTS OR YOUR FAILURE TO OBTAIN ANY ECESSARY LICENSES FOR ANY THIRD PARTY IP RIGHTS.

SE OF THE SPECIFICATIONS IS SUBJECT TO, AND GOVERNED BY, THE TERMS OF MCA'S LICENSE AGREEMENT.

HE SPECIFICATIONS ARE PROVIDED ON AN "AS IS" BASIS AND WITHOUT ARRANTY OF ANY TYPE OR KIND. MMCA HAS NOT CONDUCTED AN INDEPENDENT NTELLECTUAL PROPERTY RIGHTS REVIEW WITH RESPECT TO THE SPECIFICATIONS ND MAKES NO REPRESENTATIONS OR WARRANTIES REGARDING ANY THIRD ARTY INTELLECTUAL PROPERTY RIGHTS, INCLUDING, WITHOUT LIMITATION, ANY ATENT RIGHTS, COPYRIGHT RIGHTS OR TRADE SECRET RIGHTS. MMCA HEREBY ISCLAIMS AND EXCLUDES ALL WARRANTIES, WHETHER STATUTORY, EXPRESS OR MPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF ERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-FRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL MMCA BE LIABLE OR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, STATUTORY OR INDIRECT AMAGES OF ANY TYPE OR KIND (INCLUDING, BUT NOT LIMITED TO, ANY LOSS OF OFITS, REVENUE, SAVINGS, USE, OR OTHER ECONOMIC ADVANTAGE) ARISING T OF OR IN ANY WAY IN CONNECTION WITH THESE SPECIFICATIONS, GARDLESS OF WHETHER THE CLAIM FOR DAMAGES IS BASED IN CONTRACT, RT (INCLUDING NEGLIGENCE), STRICT LIABILITY, OR ANY OTHER LEGAL OR UITABLE THEORY, EVEN IF MMCA HAS BEEN ADVISED IN ADVANCE OF THE SIBILITY OF SUCH DAMAGES.

HB28E016MM2/HB28D032MM2
HB28D064MM2/HB28B128MM2

MultiMediaCard™
16 MByte/32 MByte/64 MByte/128 MByte

HITACHI

ADE-203-1294C (Z)
Rev. 3.0
Apr. 26, 2002

Description

These Hitachi MultiMediaCard™s, HB28E016MM2, HB28D032MM2, HB28D064MM2 and HB28B128MM2, are highly integrated flash memories with serial and random access capability. It is accessible via a dedicated serial interface optimized for fast and reliable data transmission. This interface allows several cards to be stacked by through connecting their peripheral contacts. These Hitachi MultiMediaCards are fully compatible to a new consumer standard, called the MultiMediaCard system standard defined in the MultiMediaCard system specification [1]. The MultiMediaCard system is a new mass-storage system based on innovations in semiconductor technology. It has been developed to provide an inexpensive, mechanically robust storage medium in card form for multimedia consumer applications. MultiMediaCard allows the design of inexpensive players and drives without moving parts. A low power consumption and a wide supply voltage range favors mobile, battery-powered applications such as audio players, organizers, palmtops, electronic books, encyclopedia and dictionaries. Using very effective data compression schemes such as MPEG, the MultiMediaCard will deliver enough capacity for all kinds of multimedia data: software/programs, text, music, speech, images, video etc.

Note: MultiMediaCard™ is a trademark of Infineon Technologies AG.

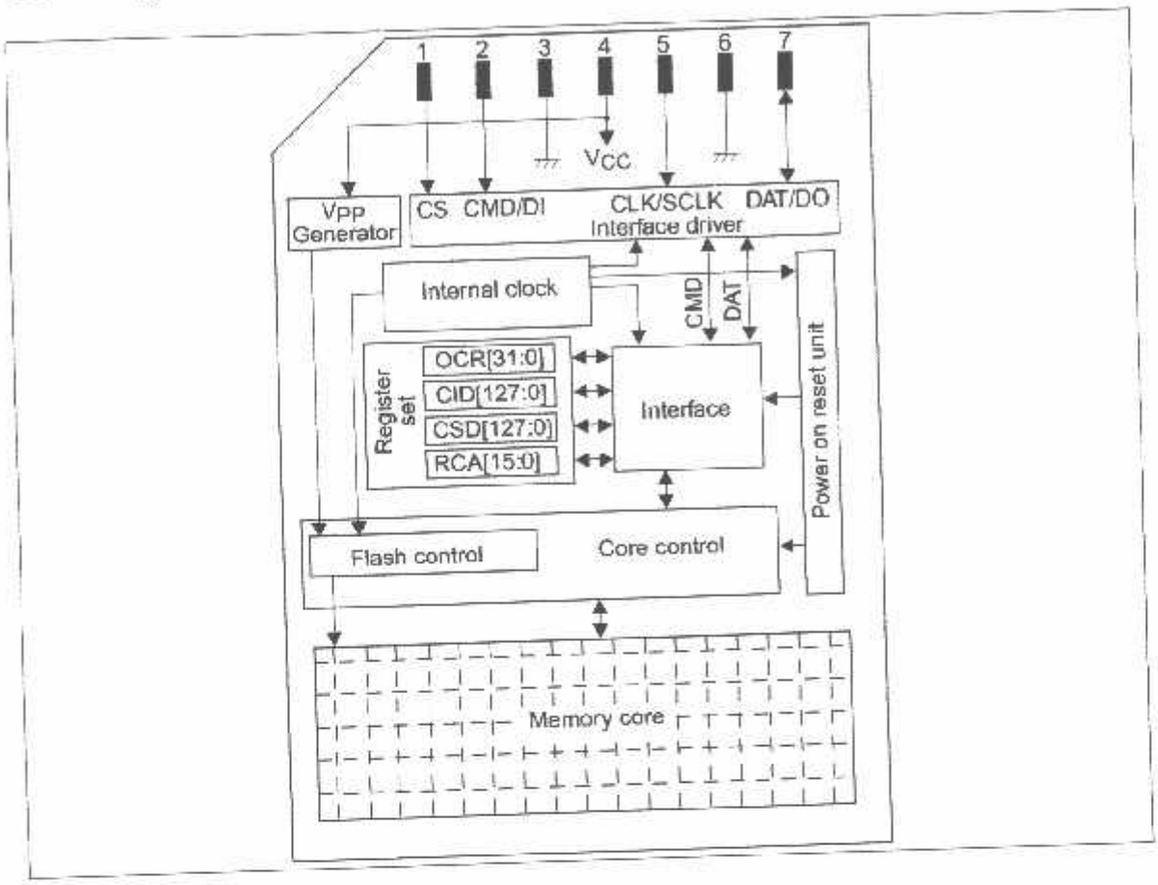
Features

- 16 MByte/32 MByte/64 MByte/128 MByte memory capacity
- On card error correction

Hitachi semiconductor

- MultiMediaCard system standard compatibility
  - — System specification version 3.1 compliant
  - — SPI mode supports the single and multiple block read and write operations.
  - — Block and partial block read supported (Command classes 2)
  - — Stream read supported (Command class 1)
  - — Block write and erase supported (Command classes 4 and 5)
  - — Group write protection (Command classes 6)
  - — Stream write supported (Command classes 3)
  - — Password data access protection
  - — Small erase block size of 512 bytes, tagged erase supported
  - — Read block size programmable between 1 and 2048 bytes
  - — V_{CC} = 2.7 V to 3.6 V operation voltage range (V_{CC} = 2.0 V to 3.6 V for the interface)
  - — No external programming voltage required
  - — Damage free powered card insertion and removal (no operation)
  - — 4kV ESD protection (Contact Pads)
- High speed serial interface with random access
  - — Read speed: sustained : 13.7 Mbit/s (multi-block read) (for HB28E016/D032/D064/B128MM2)
    burst (one block): 20 Mbit/s
  - — Write speed: sustained: 6.4 Mbit/s (multi-block write) (for HB28E016/D032MM2)
    12.8 Mbit/s (multi-block write) (for HB28D064/B128MM2)
    burst (one block): 20 Mbit/s
  - — Up to 10 stacked card (at 20 MHz, V_{CC} = 2.7 to 3.6V)
  - — Access time: 300 µs (typ) (at 20 MHz, V_{CC} = 2.7 to 3.6V)
- Low power dissipation
  - — High speed: 216 mW (max) (at 20 MHz, V_{CC} = 3.6 V): HB28E016/D032MM2
  - — High speed: 288 mW (max) (at 20 MHz, V_{CC} = 3.6 V): HB28D064/B128MM2

Block Diagram



All units in these Hitachi MultiMediaCards are clocked by an internal clock generator. The Interface driver unit synchronizes the DAT and CMD signals from external CLK to the internal used clock signal. The card is controlled by the three line MultiMediaCard interface containing the signals: CMD, CLK, DAT (refer to Chapter "Interfaces"). For the identification of the MultiMediaCard in a stack of MultiMediaCards, a card identification register (CID) and a relative card address register (RCA) is foreseen. An additional register contains different types of operation parameters. This register is called card specific data register (CSD). The communication using the MultiMediaCard lines to access either the memory field or the registers is defined by the MultiMediaCard standard (refer to Chapter "Communication"). The card has its own power on detection unit. No additional master reset signal is required to setup the card after power on. It is protected against short circuit during insertion and removal while the MultiMediaCard system is powered up (refer to Chapter "Power Supply"). No external programming voltage supply is required. The programming voltage is generated on card.

These Hitachi MultiMediaCards support a second interface operation mode the SPI interface mode. The SPI mode is activated if the CS signal is asserted (negative) during the reception of the reset command (CMD0) (refer to Chapter "SPI Communication").

Interface

These Hitachi MultiMediaCards' interface can operate in two different modes:

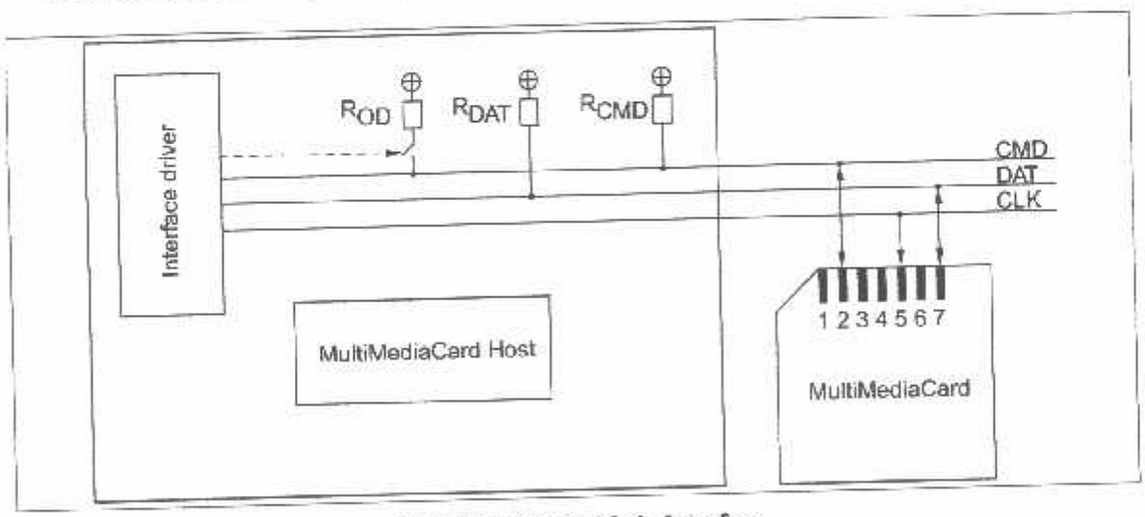
- MultiMediaCard mode
- SPI mode

Both modes are using the same pins. The default mode is the MultiMediaCard mode. The SPI mode is selected by activating (= 0) the CS signal (Pin1) and sending the CMD0.

MultiMediaCard Mode

In the MultiMediaCard mode, all data is transferred over a minimal number of lines:

- CLK: with each cycle of this signal a one-bit transfer on the command and data lines is done. The frequency may vary between zero and the maximum clock frequency. The MultiMediaCard bus master is free to generate these cycles without restrictions in the range of 0 to 20 MHz.
- CMD: is a bidirectional command channel used for card initialization and data transfer commands. The CMD signal has two operation modes: open drain for initialization mode and push pull for fast command transfer. Commands are sent from the MultiMediaCard bus master to the MultiMediaCard and responses vice versa.
- DAT: is a bidirectional data channel with a width of one line. The DAT signal of the MultiMediaCard operates in push pull mode.
- RSV: is pulled up with resistor (2 MΩ typ) in the MultiMediaCard. The external pull-up resistor should be recommended if the system requires.



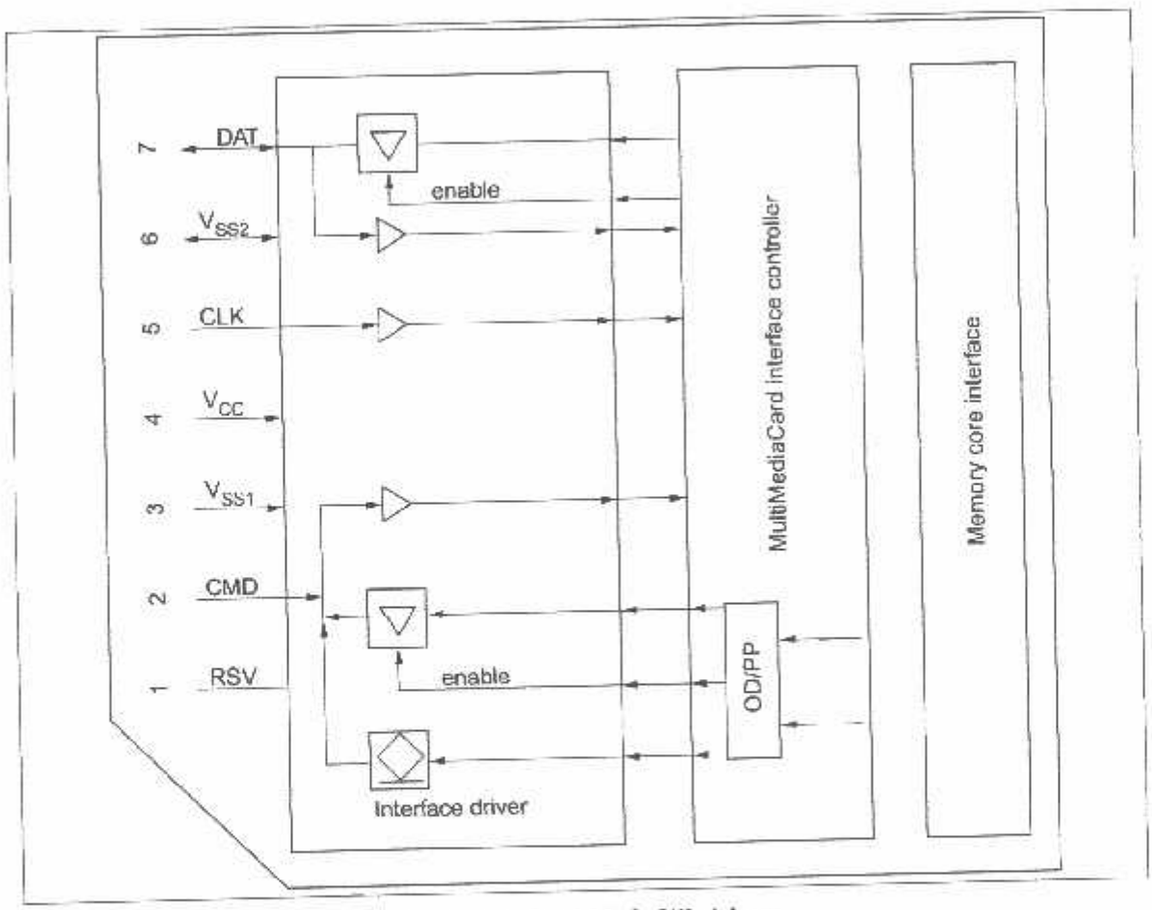
MultiMediaCard Mode Interface

All MultiMediaCards are connected directly to the lines of the MultiMediaCard bus. The following table defines the card contacts.

MultiMediaCard Mode Pad Definition

Pin No.	Name	Type*1	Description
1	RSV	NC	No connection
2	CMD	I/O/PP/OD	Command/Response
3	V_{SS1}	S	Ground
4	V_{CC}	S	Power supply
5	CLK	I	Clock
6	V_{SS2}	S	Ground
7	DAT	I/O/PP	Data

Note: 1. S: power supply; I: input; O: output; PP: push-pull; OD: open-drain; NC: No connection or V_{IH}



MultiMediaCard Mode I/O-drivers

SPI Mode

The Serial Peripheral Interface (SPI) is a general-purpose synchronous serial interface originally found on certain Motorola microcontrollers. The MultiMediaCard SPI interface is compatible with SPI hosts available on the market. As any other SPI device the MultiMediaCard SPI interface consists of the following four signals:

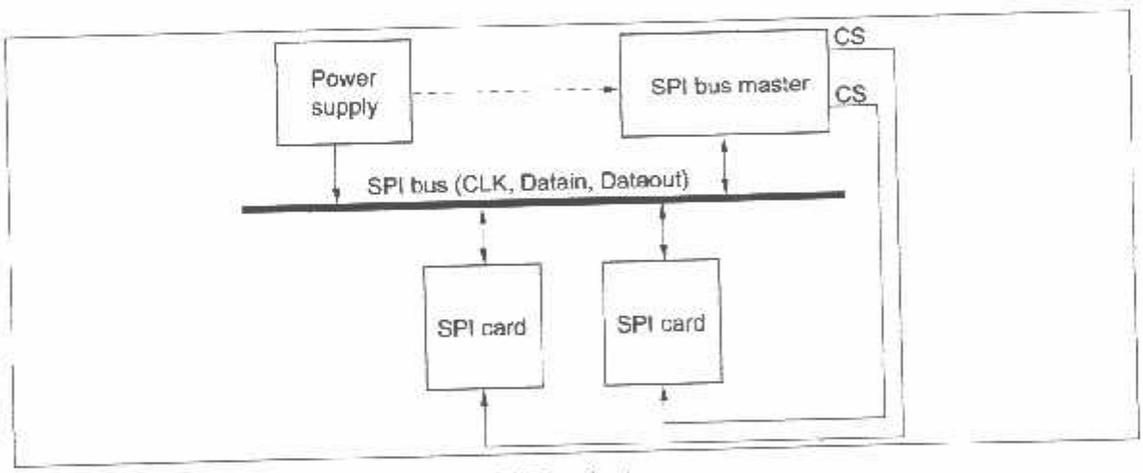
CS: Host to card Chip Select signal.

CLK: Host to card clock signal

Data in: Host to card data signal.

Data out: Card to host data signal.

The MultiMediaCard card identification and addressing methods are replaced by a hardware Chip Select (CS) signal. There are no broadcast commands. For every command, a card (slave) is selected by asserting (active low) the CS signal (refer to Figure "SPI Bus System"). The CS signal must be continuously active for the duration of the SPI transaction (command, response and data). The only exception occurs during card programming, when the host can de-assert the CS signal without affecting the programming process. The bidirectional CMD and DAT lines are replaced by unidirectional data in and data out signals. This eliminates the ability of executing commands while data is being read or written and, therefore, makes the sequential and multi block read/write operations obsolete. The single and multiple block read/write commands are supported by the SPI channel. The SPI interface uses the same seven signals of the standard MultiMediaCard bus (refer to Table "SPI Interface Pin Configuration").



SPI Bus System

SPI Interface Pin Configuration

	MultiMediaCard			SPI		
Pin No.	Name	Type*1	Description	Name	Type	Description
1	RSV	NC	Reserved for future use	CS	I	Chip select (neg true)
2	CMD	I/O/PP/OD	Command/Response	DI	I	Data in
3	V_{SS1}	S	Ground	V_{SS}	S	Ground
4	V_{CC}	S	Power supply	V_{CC}	S	Power supply
5	CLK	I	Clock	SCLK	I	Clock
6	V_{SS2}	S	Ground	V_{SS2}	S	Ground
7	DAT	I/O/PP	Data	DO	O/PP	Data out

Note: 1. S: power supply; I: input; O: output; PP: push-pull; OD: open-drain; NC: No connection or V_{IH}

SPI Communication

The SPI mode consists of a secondary communication protocol. This mode is a subset of the MultiMediaCard protocol, designed to communicate with a SPI channel, commonly found in Motorola's (and lately a few other vendors') microcontrollers. The interface is selected during the first reset command after power up (CMD0) and cannot be changed once the part is powered on. The SPI standard defines the physical link only, and not the complete data transfer protocol. The MultiMediaCard SPI implementation uses a subset of the MultiMediaCard protocol and command set. It is intended to be used by systems which require a small number of card (typically one) and have lower data transfer rates (compared to MultiMediaCard protocol based systems). From the application point of view, the advantage of the SPI mode is the capability of using an off-the-shelf host, hence reducing the design-in effort to minimum. The disadvantage is the loss of performance of the SPI system versus MultiMediaCard (lower data transfer rate, fewer cards, hardware CS per card etc.). While the MultiMediaCard channel is based on command and data bitstreams which are initiated by a start bit and terminated by a stop bit, the SPI channel is byte oriented. Every command or data block is built of 8-bit bytes and is byte aligned to the CS signal (i.e. the length is a multiple of 8 clock cycles). Similar to the MultiMediaCard protocol, the SPI messages consist of command, response and data-block tokens (refer to Chapter "Commands" and Chapter "Responses" for a detailed description). All communication between host and cards is controlled by the host (master). The host starts every bus transaction by asserting the CS signal low. The response behavior in the SPI mode differs from the MultiMediaCard mode in the following three aspects:

- The selected card always responds to the command.
- An additional (8 bit) response structure is used
- When the card encounters a data retrieval problem, it will respond with an error response (which replaces the expected data block) rather than by a time-out as in the MultiMediaCard mode.

Single block and multiple read and write operations are supported in SPI mode. In addition to the command response, every data block sent to the card during write operations will be responded with a special data response token. A data block may be as big as one card sector and as small as a single byte. Partial block read/write operations are enabled by card options specified in the CSD register.

Mode Selection

The MultiMediaCard wakes up in the MultiMediaCard mode. It will enter SPI mode if the reset command (CMD0) is received during the CS signal is asserted (negative). Selecting SPI mode is not restricted to Idle state (the state the card enters after power up) only. Every time the card receives CMD0, including while in Inactive state, CS signal is sampled.

If the card recognizes that the MultiMediaCard mode is required it will not respond to the command and remain in the MultiMediaCard mode. If SPI mode is required the card will switch to SPI and respond with the SPI mode R1 response.

The only way to return to the MultiMediaCard mode is by entering the power cycle. In SPI mode the MultiMediaCard protocol state machine is not observed. All the MultiMediaCard commands supported in SPI mode are always available.

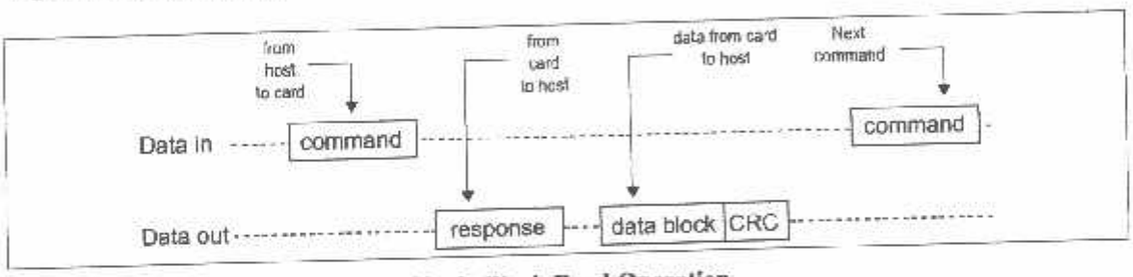
Bus Transfer Protection

Every MultiMediaCard token transferred on the bus is protected by CRC bits. In SPI mode, the MultiMediaCard offers a non-protected mode which enables systems built with reliable data links to exclude the hardware or firmware required for implementing the CRC generation and verification functions. In the non-protected mode the CRC bits of the command, response and data tokens are still required in the tokens. However, they are defined as "don't care" for the transmitter and ignored by the receiver. The SPI interface is initialized in the non protected mode. However, the RESET command (CMD0), which is used to switch the card to SPI mode, is received by the card while in MultiMediaCard mode and, therefore, must have a valid CRC field.

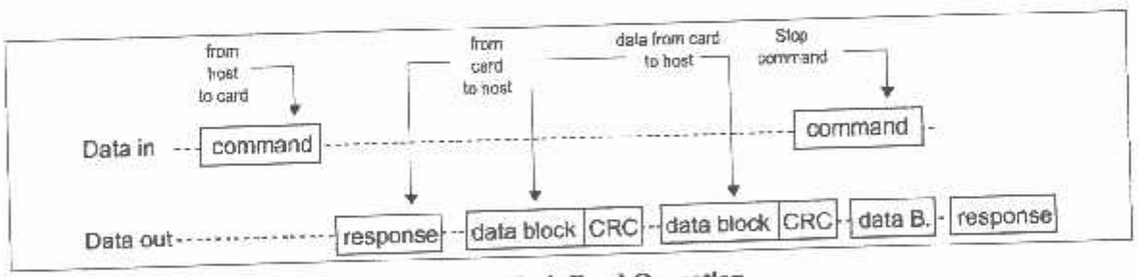
The host can turn this option on and off using the CRC_ON_OFF command (CMD59).

Data Read Overview

The SPI mode supports single and multiple block read operations (CMD17 and CMD18 in the MultiMediaCard protocol). The main difference between SPI and MultiMediaCard modes is that the data and the response are both transmitted to the host on the DataOut signal (refer to Figure 43 and Figure 45). Therefore the card response to the STOP_COMMAND may cut-short and replace the last data block (refer to Figure "Read Operation").

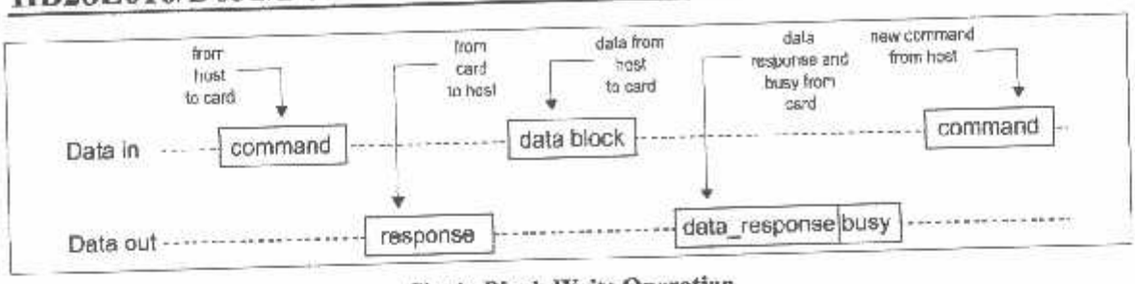


Single Block Read Operation



Multiple Block Read Operation

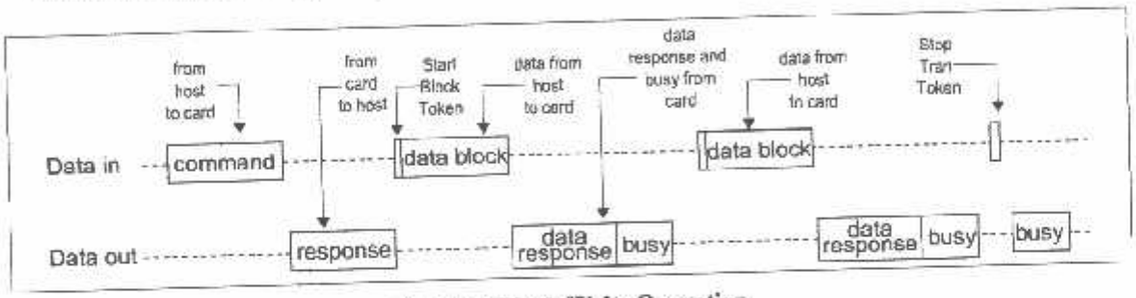
The basic unit of data transfer is a block whose maximum size is defined in the CSD (READ_BL_LEN). If READ_BL_PARTIAL is set, smaller blocks whose starting and ending addresses are entirely contained within one physical block (as defined by READ_BL_LEN) may also be transmitted. A 16-bit CRC is appended to the end of each block ensuring data transfer integrity (also refer to chapter "Cyclic Redundancy Check (CRC)"). CMD17 (READ_SINGLE_BLOCK) initiates a single block read. CMD18



Single Block Write Operation

After a data block has been received, the card will respond with a data-response token. If the data block has been received without errors, it will be programmed. As long as the card is busy programming, a continuous stream of busy tokens will be sent to the host (effectively holding the DataOut line low).

In Multiple Block write operation the stop transmission will be done by sending 'Stop Tran' token instead of 'Start Block' token at the beginning of the next block.



Multiple Block Write Operation

Two types of multiple block write transactions, identical to the multiple block read, are defined (the host can use either one at any time):

- Open-ended Multiple block write

  The number of blocks for the write multiple block operation is not defined. The card will continuously accept and program data blocks until a 'Stop Tran' token is received.

- Multiple block write with pre-defined block count

  The card will accept the requested number of data blocks and terminate the transaction. 'Stop Tran' token is not required at the end of this type of multiple block write, unlessterminated with an error. In order to start a multiple block write with pre-defined block count, the host must use the SET_BLOCK_COUNT command (CMD23) immediately preceding the WRITE_MULTIPLE_BLOCK (CMD25) command. Otherwise the card will start an open-ended multiple block write which can be stopped using the 'Stop Tran' token.

The host can abort writing at any time, within a multiple block operation, regardless of the its type.

Transaction abort is done by sending the 'Stop Tran' token. If a multiple block write with predefined block count is aborted, the data in the remaining blocks is not defined.

If the card detects a CRC error or a programming error (e.g. write protect violation, out of range, address misalignment, internal error, etc.) during a multiple block write operation (both types) it will report the failure in the data-response token and ignore any further incoming data blocks. The host must than abort the operation by sending the 'Stop Tran' token.

If the host uses partial blocks whose accumulated length is not block aligned and block misalignment is not allowed (CSD parameter WRITE_BLK_MISALIGN is not set), the card shall detect the block misalignment error before the beginning of the first misaligned block and respond with an error indication in the data response token and ignore any further incoming data blocks. The host must than abort the operation by sending the 'Stop Tran' token.

Once the programming operation is completed, the host must check the results of the programming using the SEND_STATUS command (CMD13). Some errors (e.g. address out of range, write protect violation etc.) are detected during programming only. The only validation check performed on the data block and communicated to the host via the data-response token is the CRC.

If the host sends a 'Stop Tran' token after the card received the last block of a multiple block operation with pre-defined number of blocks, it will be responded to as the beginning of an illegal command and responded accordingly.

While the card is busy, resetting the CS signal will not terminate the programming process. The card will release the DataOut line (tri-state) and continue with programming. If the card is reselected before the programming is finished, the DataOut line will be forced back to low and all commands will be rejected.

Resetting a card (using CMD0) will terminate any pending or active programming operation. This may destroy the data formats on the card. It is in the responsibility of the host to prevent it.

Error Conditions

Unlike the MultiMediaCard protocol, in the SPI mode the card will always respond to a command. The response indicates acceptance or rejection of the command. A command may be rejected if it is not supported (illegal opcode), if the CRC check failed, if it contained an illegal operand, or if it was out of sequence during an erase sequence.

Memory Array Partitioning

Same as for MultiMediaCard mode.

Card Lock/Unlock

Usage of card lock and unlock commands in SPI mode is identical to MultiMediaCard mode. In both cases the command response is of type R1b. After the busy signal clears, the host should obtain the result of the operation by issuing a GET_STATUS command. Please refer to Chapter "Card lock/unlock operation" for details.

Commands

All the MultiMediaCard commands are 6 bytes long. The command transmission always starts with the left bit of the bitstring corresponding to the command codeword. All commands are protected by a CRC7. The commands and arguments are listed in Table

Bit position	[47]	[46]	[45:40]	[39:8]	[7:1]	[0]
Width (bits)	1	1	6	32	7	1
Value	'0'	'1'	×	×	×	'1'
Description	start bit	transmission bit	command index	argument	CRC7	end bit

The following table provides a detailed description of the SPI bus commands. The responses are defined in Chapter "Responses". The Table "Commands and Arguments" lists all MultiMediaCard commands. A "yes" in the SPI mode colon indicates that the command is supported in SPI mode. With these restrictions, the command class description in the CSD is still valid. If a command does not require an argument, the value of this field should be set to zero. The reserved commands are reserved in MultiMediaCard mode as well. The binary code of a command is defined by the mnemonic symbol. As an example, the content of the command index field is (binary) '000000' for CMD0 and '100111' for CMD39.

SPI Bus Timing

All timing diagrams use the following schematics and abbreviations:

H: Signal is high (logical '1')

L: Signal is low (logical '0')

X: Don't care

Z: High impedance state (-> = 1)

*: Repeater

Busy: Busy Token

Command: Command token

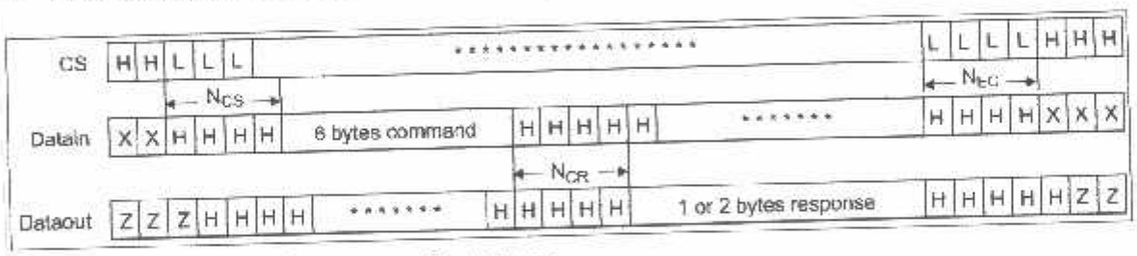
Response: Response token

Data block: Data token

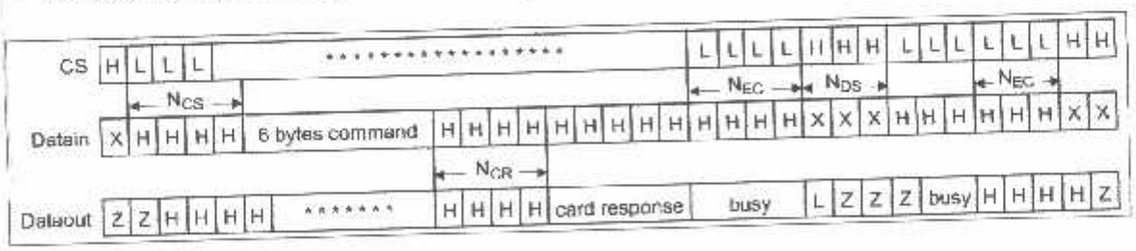
All timing values are defined in Table "Timing Values". The host must keep the clock running for at least N_{CR} clock cycles after receiving the card response. This restriction applies to both command and data response tokens.

Command/Response

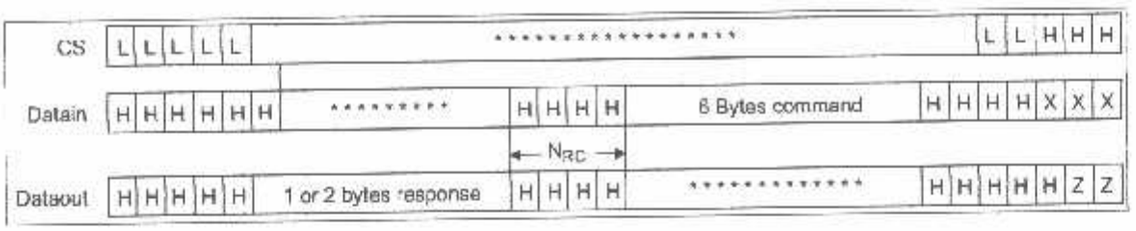
- Host Command to Card Response - Card is ready



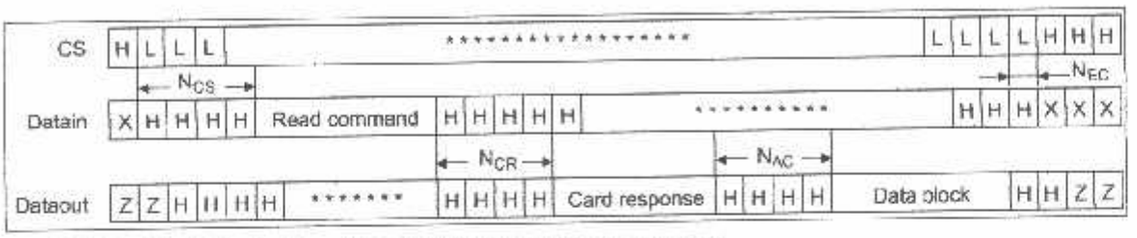
- Host Command to Card Response - Card is busy



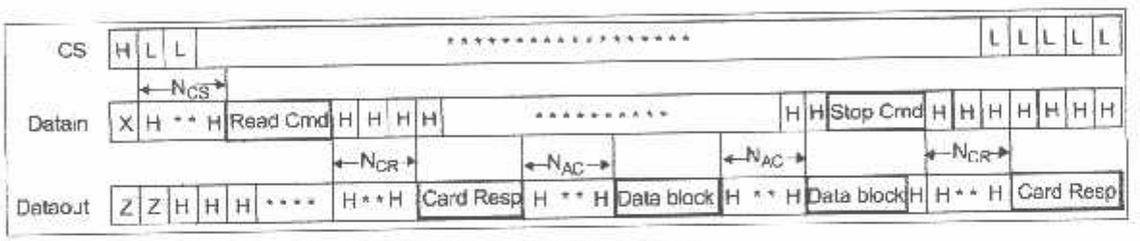
- Card Response to Host Command



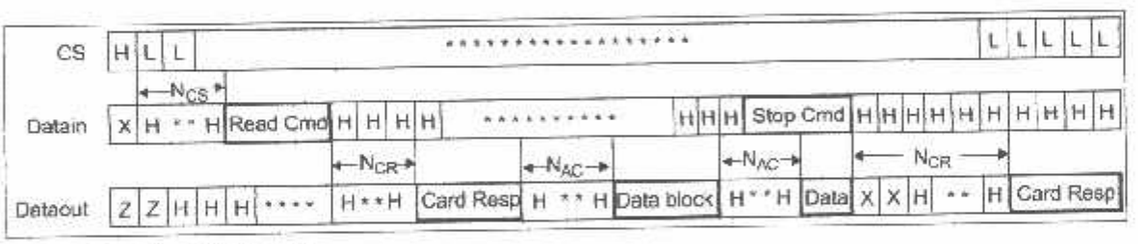
- Single Block Read



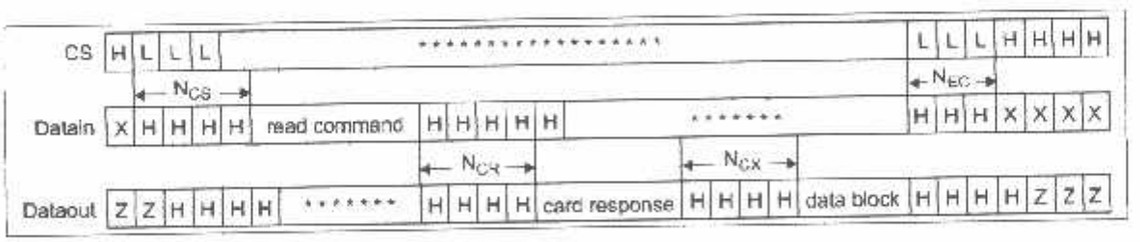
- Multiple Block Read – Stop Transmission is sent between blocks



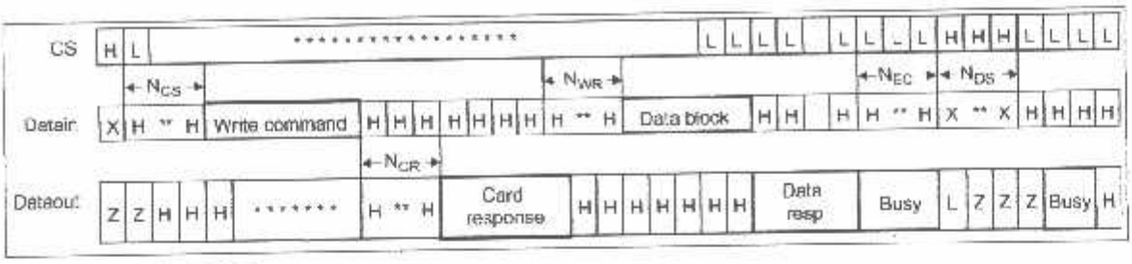
- Multiple Block Read – Stop Transmission is sent within a block



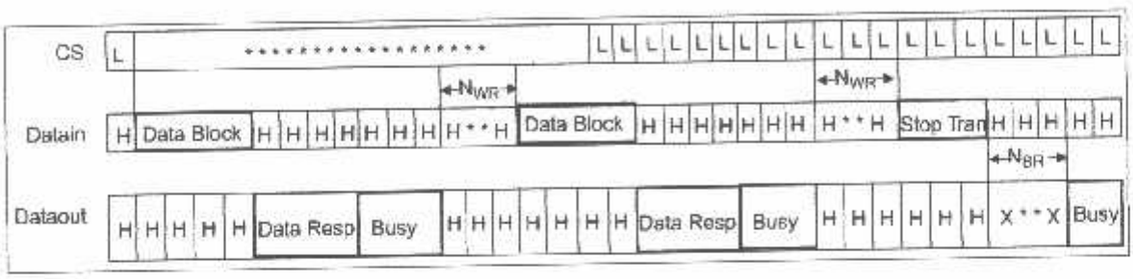
- Reading The CSD Register



- Single Block Write



- Multiple Block Write



Timing Values

Symbol	Min	Max	Unit
N_{CS}	0	—	8 clock cycles
N_{CR}	1	8	8 clock cycles
N_{CX}	0	8	8 clock cycles
N_{RC}	1	—	8 clock cycles
N_{AC}	1	spec. in the CSD*1	8 clock cycles
N_{WR}	1	—	8 clock cycles
N_{EC}	0	—	8 clock cycles
N_{DS}	0	—	8 clock cycles
N_{BR}	1	1	8 clock cycles

Note: 1. Refer to Chapter "Time-out Condition".

ANALOG DEVICES
Technical notes on using Analog Devices DSPs, processors and development tools
Contact our technical support at dsp.support@analog.com and at dsptools.support@analog.com
Or visit our on-line resources http://www.analog.com/ee-notes and http://www.analog.com/processors

Interfacing MultiMediaCard™ with ADSP-2126x SHARC® Processors

Contributed by Aseem Vasudev Prabhugaonkar and Jagadeesh Rayala

Rev 1 March 11, 2005

Introduction

This application note describes how to implement the interface between an ADSP-2126x SHARC® processor and a MultiMediaCard™ (MMC). The application note also describes the MMC command format and demonstrates with example code how an MMC card can be interfaced seamlessly with the SHARC processor's SPI port. Example code supplied with this application note implements the most commonly used commands of MultiMediaCard.

About MultiMediaCard

The MMC was introduced in 1998 and had an amazing reduction in cubic capacity compared with CompactFlash™. MMC cards are now widely used in digital cameras, smart cell phones, PDAs, and portable MP3 players. Their intended use is to store information and content.

The MMC consists of a 7-pin interface and supports two serial data transfer protocols viz. the MMC (MultiMediaCard) mode and SPI (Serial Peripheral Interface) mode. The maximum operating clock frequency used for serial communication in both modes can go up to 20 MHz. The data written in any of these modes can be read by host in either mode. The advantage of MMC supporting SPI mode is that MMC can be interfaced seamlessly to many controllers or DSP processors, which have on-chip support for SPI. Most MMCs have a communication voltage from 2.0 to 3.6 V, a memory access voltage of 2.7 to 3.6 V, and a capacity from 4 MB to the gigabyte range.

About the ADSP-2126x SPI Port

The ADSP-2126x processor is equipped with a synchronous serial peripheral interface port that is compatible with the industry-standard Serial Peripheral Interface (SPI). The SPI port supports communication with a variety of peripheral devices including codecs, data converters, sample rate converters, S/PDIF or AES/EBU digital audio transmitters and receivers, LCDs, shift registers, micro-controllers, and FPGA devices with SPI emulation capabilities.

Important features of ADSP-2126x SPI port include:

- Simple four-wire interface, consisting of two data pins, a device select pin, and a clock pin
- Full duplex operation, allowing simultaneous data transmission and reception on the same SPI port
- Data formats to accommodate little and big endian data, different word lengths, and packing modes
- Master and slave modes as well as multimaster mode in which the ADSP-2126x processor can be connected to up to four other SPI devices
- Open drain outputs to avoid data contention and to support multimaster scenarios

- Programmable bit rates, clock polarities, and phases
- DMA capability, allowing transfer of data without core overhead
- Master or slave booting from a master SPI device

The MultiMediaCard Interface

In SPI mode, four signals (clock, data in, data out and chip select) are used for the interface. The clock is used to drive data out on the data out pin and receive data on the data in pin. The host drives commands and data to the MMC over the MMC's data in pin. The host receives response and data from the MMC on its data out pin. The chip select signal is used to enable the MMC during data and command transfer. The chip select signal is also used initially to drive the MMC in SPI mode. Note that in SPI mode, data is transferred in units of eight clock cycles.

Pin	Name	Type	Function
1	CS#	Input	Chip select (active low)
2	Din	Input	Data input
3	VSS1	Power	GND
4	VDD	Power	VCC
5	CLK	Input	Clock input
6	VSS2	Power	GND
7	Dout	Output	Data output

Table 1. MultiMediaCard Pin Assignment

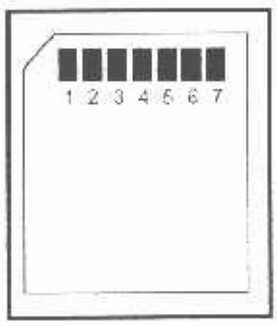


Figure 1. MultiMediaCard Pin Assignments

The MMC pin assignments in SPI mode are shown in Table 1 and Figure 1. Figure 2 shows the MMC interface with the ADSP-2126x SPI port.

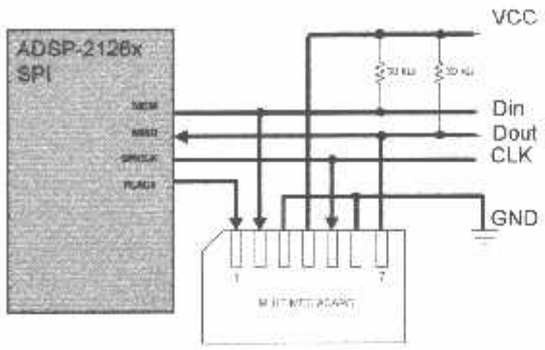


Figure 2. MultiMediaCard Interface with SPI Port

The MultiMediaCard Protocol

The SHARC processor's SPI issues commands to the MMC over the data in (DIN) pin of the MMC. The data in pin of the MMC is connected to MOSI of the SPI. The data is also written to the MMC over the data in signal of the MMC. Based on the received command, the MMC sends response or data on the data out (DOUT) pin. The data out pin of the MMC is connected to MISO of the SHARC processor's SPI port. The processor's SPI port uses one of the Programmable Flag pins (FLAGx) to drive CS# of the MMC. The communication is initiated by different commands sent from the SHARC processor to the MMC. All commands are six bytes long and are transmitted MSB first. Refer to Figure 3 for generic transfer protocol between the MMC and the SHARC processor's SPI port.

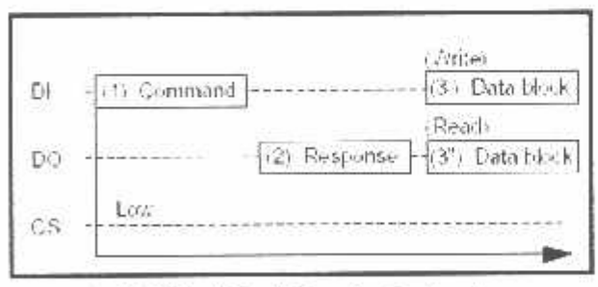


Figure 3. MultiMediaCard Transfer Protocol

Commands and Responses

Table 2 lists the MultiMediaCard's most commonly used commands in SPI mode. The command format is shown in Figure 6.

Each MMC command consists of 48 bits (6 bytes) comprising a start bit (always 0), a transfer bit (always 1), a 6-bit command field, a 4-byte (32-bit) argument field, a 7-bit CRC field, and an end bit (always 1). The argument field contains the necessary information (card relative address, read address, write address, etc.) for issuing that command. For every received command (except the SEND_STATUS command), the MMC responds with a token value.

The R1 response token is 1-byte long and its MSB is always 0. The other bits in the response indicate error conditions. The structure of an R1 response is shown in Figure 4. The R1 format byte description is given in Table 3.

BIT	7	6	5	4	3	2	1	0
FIELD	0	Parameter Error	Address Error	Erase Seq Error	CRC Error	Illegal Command	Erase Reset	In Idle State

Figure 4. The MultiMediaCard R1 Response Format

Response format R2 is 2 bytes long. The response token is sent by the card as a response to the SEND_STATUS command. The format of the R2 status is shown in Figure 5.

Figure 5. The MultiMediaCard R2 Response Format

The R2 format description is given in Table 4.

CMD INDEX	ARGUMENT	RESPONSE	ABBREVIATION	COMMAND DESCRIPTION
CMD0	None	R1	GO_IDLE_STATE	Resets the MultiMediaCard
CMD1	None	R1	SEND_OP_COND	Activates the card initialization process
CMD13	None	R2	SEND_STATUS	Asks the selected card to send its status register
CMD16	[31:0]block length	R1	SET_BLOCKLEN	Selects a block length (in bytes) for all following block commands (read and write).
CMD17	[31:0]data address	R1	READ_SINGLE_BLOCK	Reads a block of size selected by the SET_BLOCKLEN command
CMD24	[31:0]data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command
CMD32	[31:0]data address	R1	TAG_SECTOR_START	Sets the address of the first sector of the erase group
CMD33	[31:0]data address	R1	TAG_SECTOR_END	Sets the address of the last sector in a continuous range within the selected erase group, or the address of a single sector to be selected for erase.
CMD34	[31:0]data address	R1	UNTAG_SECTOR	Removes one previously selected sector from the erase selection
CMD38	[31:0]don't care	R1b	ERASE	Erases all previously selected sectors
CMD59	[31:1]don't care [0:0]CRC option	R1	CRC_ON_OFF	Turns the CRC option on or off. A '1' in the CRC option bit will turn the option on. A '0' will turn it off.

Table 2. Most Commonly Used MultiMediaCard Commands

Serial data → [47] Command [0]

Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bit)	1	1	6	32	7	1
Value	0	1	X	X	X	1
Description	Start bit	Transmission bit	Command index	Argument	CRC7	End bit

Figure 6. The MultiMediaCard Command Format

BIT	NAME	DESCRIPTION
7	0	Fixed to '0'
6	Parameter Error	The command's argument (e.g. address, block length) was out of the allowed range for this card
5	Address Error	A misaligned address, which did not match the block length, was used in the command
4	Erase Seq Error	An error in the sequence of erase commands occurred
3	Com CRC Error	The CRC check of the last command failed
2	Illegal Command	An illegal command code was detected
1	Erase Reset	An erase sequence was cleared before executing because an out of erase sequence command was received
0	In Idle State	The card is in idle state and running initializing process

Table 3. MultiMediaCard R1 Response Format Description

BIT	NAME	DEFINITION
15	0	Fixed to '0'
14	Parameter Error	The command's argument (e.g. address, block length) was out of the allowed range for this card
13	Address Error	A misaligned address, which did not match the block length was used in the command
12	Erase Seq Error	An error in the sequence of erase commands occurred
11	Com CRC Error	The CRC check of the last command failed
10	Illegal Command	An illegal command code was detected
9	Erase Reset	An erase sequence was cleared before executing because an out of erase sequence command was received
8	In Idle State	The card is in idle state and running initializing process
7	Out of Range	
6	Erase Param	An invalid selection, sectors or groups, for erase
5	WP Violation	The command tried to write a write protected block
4	Card ECC Failed	Card internal ECC was applied but failed to the corrected data
3	CC Error	Internal card controller error
2	Error	A general or an unknown error occurred during the operation
1	WP Erase Skip	Only partial address space was erased due to existing WP blocks
0	0	Fixed to '0'

Table 4. MultiMediaCard R2 Response Format Description

Interfacing a MultiMediaCard to the LH79520 System-On-Chip

Jun Li, Applications Engineer

NTRODUCTION

The MultiMediaCard (MMC) is a low-cost data storge media widely used in MP3 players, digital recorders, smart phones, PDAs, and pagers. It is common to nd embedded MMC controllers in some high-end icrocontrollers, but it is not necessary to use a hard-,are MMC controller to interface with the MMC card. he SPI peripheral in SHARP's LH79520 microcontrol-r can easily handle this task.

This application note describes how to implement ie interface between SHARP's LH79520 System-on-hip and a MultiMediaCard in both hardware and soft-are. It will discuss using the LH79520 in MultiMedi-Card applications, connecting the MultiMediaCard to ie LH79520's built-in SPI controller, and how to read ie FAT16 master boot block in the MultiMediaCard.

H79520 Processor Bandwidth

The LH79520 is a 32-bit general-purpose microcon-oller, using the ARM720T core in a 176-pin QFP pack-ge. The core can operate at up to 77.4 MHz and the us can operate at up to 51.6 MHz. This is a System-on-hip with many peripherals including MMU, CACHE, SP, UART, SDRAM Controller, PWM, VIC, GPIO, and 64 k-color LCD controller. Of these, the MultiMedi-Card can best be connected through the SPI interface, nich is supported by the SSP. The SPI controller in the H79520 can operate at up to half of the bus clock eed, (approximately 25 Mbit/s) so this makes the SPI ntroller a good fit to drive the MultiMediaCard to hieve its maximum throughput of 20 Mbit/s.

Connecting the MultiMediaCard to the SHARP LH79520 Microcontroller

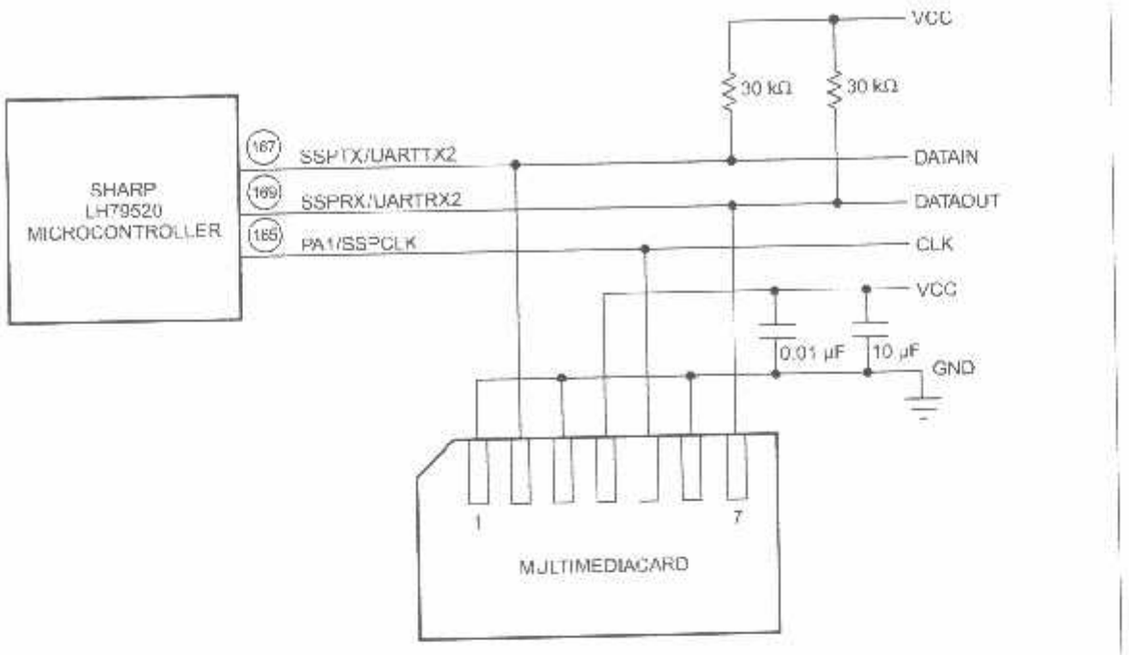
The MultiMediaCard is based on an advanced 7-pin serial bus mode known as 'MultiMediaCard mode'. Most MultiMediaCards have a communication voltage from 2.0 V to 3.6 V, a memory access voltage of 2.7 V to 3.6 V, and the capacity can be anywhere from 4MB into the gigabyte range. The MultiMediaCard has two modes of operation, one called 'MultiMedia mode' and one called 'SPI mode'. A similar arrangement could be used for SD (Secure Digital) cards; however, this Application Note will only cover MMC cards in SPI mode; for MultiMedia mode, please refer to the MMC card specifications.

Table 1 shows the MMC pin assignments for SPI mode. Figure 1 shows a method of connection for SPI mode from the LH79520 to the MMC card. Note that pin 1 of the MMC card is tied to ground if there is only one MMC card in the system. If there are multiple MMC cards in the system, use a GPIO to control pin 1 of each card. When pin 1 is goes LOW, the corresponding MMC card is enabled.

A pull-up resistor on the DataIn pin and DataOut pin is necessary because the MMC card drives pins in 'Open Drain' mode. Caps between ground and power are important for noise reduction on the clock and data lines for the MMC.

Table 1. MMC Pin Assignment in SPI Mode

PIN	NAME	TYPE	SPI DESCRIPTION
1	nCS	Input	Chip Select (Active LOW)
2	DataIn	Input	Host-to-Card Commands and Data
3	VSS1	Power	Supply Voltage Ground
4	VDD	VCC	Supply Voltage
5	CLK	Input	Clock
6	VSS2	Power	Supply Voltage Ground
7	DataOut	Output	Card-to-Host Data and Status



OTE: ◯ Circled numbers are LH79520 pin numbers.

Figure 1. LH79520-to-MultiMediaCard Connection

SPI Commands

Communications between the microcontroller and the MMC are initiated by different commands sent from the microcontroller to the MMC. The most common of these commands are listed in Table 2; for the complete command set, refer to the MMC specifications.

Table 2. SPI Commands

CMD INDEX	ARGUMENT	RESPONSE	ABBREVIATION	COMMAND DESCRIPTION
CMD0	None	R1	GO_IDLE_STATE	Resets the MultiMediaCard
CMD1	None	R1	SEND_OP_COND	Activates the card initialization process
CMD13	None	R2	SEND_STATUS	Asks the selected card to send its status register
CMD16	[31:0]block length	R1	SET_BLOCKLEN	Selects a block length (in bytes) for all following block commands (read and write).
CMD17	[31:0]data address	R1	READ_SINGLE_BLOCK	Reads a block of size selected by the SET_BLOCKLEN command
CMD24	[31:0]data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command
CMD32	[31:0]data address	R1	TAG_SECTOR_START	Sets the address of the first sector of the erase group
CMD33	[31:0]data address	R1	TAG_SECTOR_END	Sets the address of the last sector in a continuous range within the selected erase group, or the address of a single sector to be selected for erase.
CMD34	[31:0]data address	R1	UNTAG_SECTOR	Removes one previously selected sector from the erase selection
CMD38	[31:0]don't care	R1b	ERASE	Erases all previously selected sectors
CMD59	[31:1]don't care [0:0]CRC option	R1	CRC_ON_OFF	Turns the CRC option on or off. A '1' in the CRC option bit will turn the option on. A '0' will turn it off

COMMAND TRANSMISSION

All commands are 6 bytes long and are transmitted MSB first.

Table 3. Command Transmissions

BYTE 1	7	6	5	4	3	2	1	0
FIELD	0	1	Command					

BYTES 2 - 5	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FIELD	Argument															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIELD	Argument															

BYTE 6	7	6	5	4	3	2	1	0
FIELD	CRC						1	x

CRC CALCULATION

The CRC bit calculation is performed:

7-bit CRC Calculation: $G(x) = x^7 + x^3 + 1$

$M(x)$ = (start bit) × x^{39} + (second bit) × x^{38}+...+(last bit before CRC) × x^0

CRC[6...0] = Remainder[$(M(x) \times x^7)/G(x)$]

RESPONSE FORMAT R1

This response token is sent by the card after every command with the exception of SEND_STATUS commands. It is 1 byte long; the MSB is always set to zero and the other bits are error indications. A '1' signals an error.

The structure of the R1 format is given in Table 4 and Table 5.

RESPONSE FORMAT R1B

This response token is identical to the R1 format with the addition of the optional BUSY signal. The Card holds the DataIn line LOW to signal BUSY; this can last for any period until the Card has finished processing the current transaction. Once the Card releases the line, it is ready for the next command.

Table 4. R1 Format Byte Structure

BIT	7	6	5	4	3	2	1	0
FIELD	0	Parameter Error	Address Error	Erase Seq Error	Com CRC Error	Illegal Command	Erase Reset	In Idle State

Table 5. R1 Format Byte Definitions

BIT	NAME	DESCRIPTION
7	0	Fixed to '0'
6	Parameter Error	The command's argument (e.g. address, block length) was out of the allowed range for this card
5	Address Error	A misaligned address, which did not match the block length, was used in the command
4	Erase Seq Error	An error in the sequence of erase commands occurred
3	Com CRC Error	The CRC check of the last command failed
2	Illegal Command	An illegal command code was detected
1	Erase Reset	An erase sequence was cleared before executing because an out of erase sequence command was received
0	In Idle State	The card is in idle state and running initializing process

RESPONSE FORMAT R2

This 2-byte-long, response token is sent by the card as a response to the SEND_STATUS command. The format of the R2 status is given in Table 6 and Table 7.

Table 6. Response Format R2 Bits

BITS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIELD	0	Parameter Error	Address Error	Erase Seq Error	Com CRC Error	Illegal Command	Erase Reset	In Idle State	Out of Range	Erase Param	WP Violation	Card ECC Failed	CC Error	Error	WP Erase Skip	0

Table 7. Response R2 Format Definitions

BIT	NAME	DEFINITION
15	0	Fixed to '0'
14	Parameter Error	The command's argument (e.g. address, block length) was out of the allowed range for this card
13	Address Error	A misaligned address, which did not match the block length was used in the command
12	Erase Seq Error	An error in the sequence of erase commands occurred
11	Com CRC Error	The CRC check of the last command failed
10	Illegal Command	An illegal command code was detected
9	Erase Reset	An erase sequence was cleared before executing because an out of erase sequence command was received
8	In Idle State	The card is in idle state and running initializing process
7	Out of Range	
6	Erase Param	An invalid selection, sectors or groups, for erase
5	WP Violation	The command tried to write a write protected block
4	Card ECC Failed	Card internal ECC was applied but failed to the corrected data
3	CC Error	Internal card controller error
2	Error	A general or an unknown error occurred during the operation
1	WP Erase Skip	Only partial address space was erased due to existing WP blocks
0	0	Fixed to '0'

DATA RESPONSE

Every data block written to the card will be acknowledged by a data response token. It is one byte long and has a format as seen in Table 8.

Table 8. Data Response Byte Structure

BIT	7	6	5	4	3	2	1	0
FIELD	0	0	0	0	Status			1

The status bits may be one of two states:

'010' = Data accepted

'101' = Data rejected due to a CRC error

DATA TOKENS

Read and write commands have data transfers associated with them. Data is being transmitted or received via data tokens. All data bytes are transmitted MSB first.

Data tokens are 4 to 515 bytes long and have a format as seen in Table 9.

Data Token bytes 2 to 513 can be any data block length, since their payload is User Data.

The last two bytes of the Data Token are a 16-bit CRC.

Table 9. Data Token Start Byte (Byte 1) Structure

BIT	7	6	5	4	3	2	1	0
FIELD	1	1	1	1	1	1	1	0

DATA ERROR TOKEN

If a read operation fails and the card can not provide the required data it will send a data error token, instead. This token is one byte long and has a format as seen in Table 10.

Table 10. Data Error Token Structure

BIT	7	6	5	4	3	2	1	0
FIELD	0	0	0	0	Out_of_Range	Card_ECC_Failed	CC_Error	Error

SPI Protocol

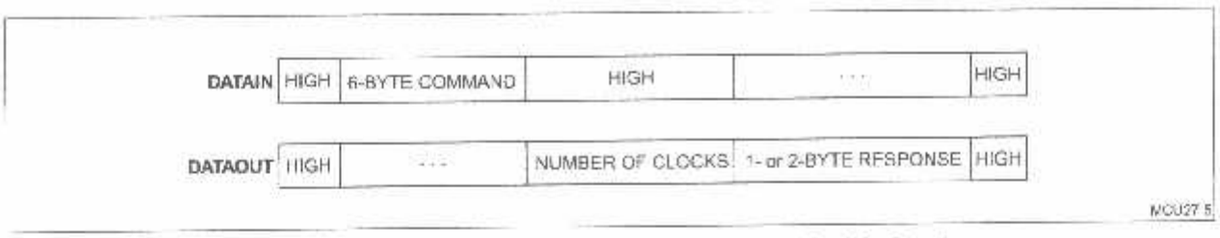


Figure 2. Host Command to Card Response — Card is Ready

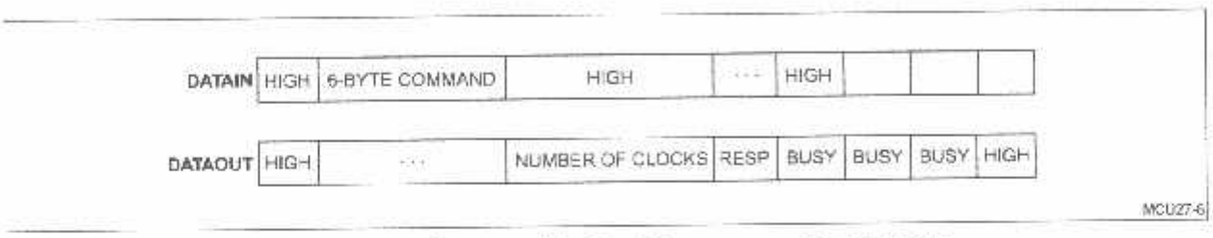


Figure 3. Host Command to Card Response — Card is Busy

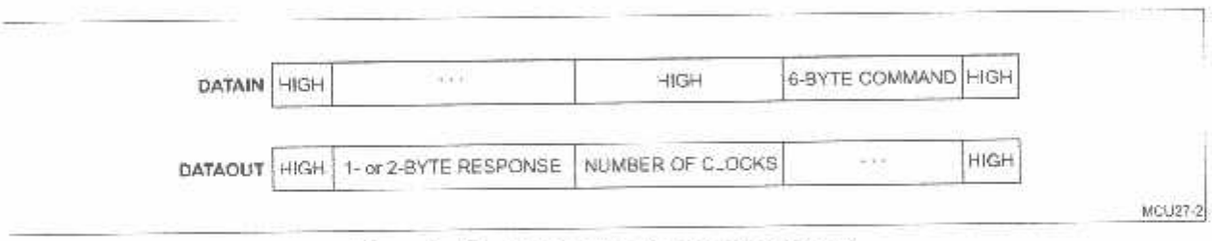


Figure 4. Card Response to Host Command

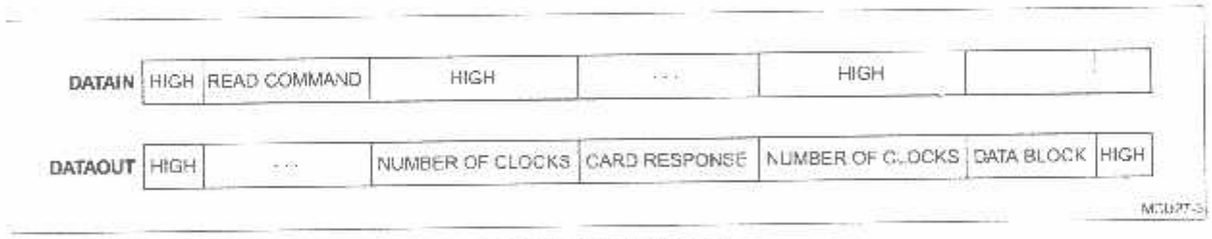


Figure 5. Data Read

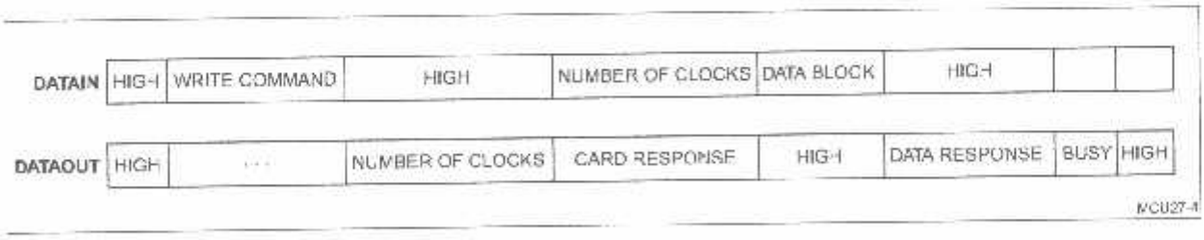


Figure 6. Data Write

Software Implementation

A source code listing for the MMC function is provided with this application note. See 'Code Listing'.

SHARP SPI DRIVER

SHARP source code for LH79520 microcontroller drivers (SPI, UART, LCD, and so on) can be downloaded from the Sharp Microelectronics of the Americas web page, at www.sharpsma.com. The tag for the source code is called the ABL BlueStreak Software Library. Within this Library, you can download the full source code for all SHARP microcontroller drivers. The MMC interface source code is based on the LH79520 drivers.

The LH79520 Synchronous Serial Port (SSP) peripheral supports devices utilizing Motorola SPI, National Semiconductor Mircrowire or Texas Instruments's Synchronous Serial interfaces.

The SSP performs parallel-to-serial conversion on data written to an internal Transmit FIFO, then transmits the data, in serial fashion, to an external slave peripheral. The SSP also receives serial data from an external slave peripheral, performs a serial-to-parallel conversion on the received data, and buffers the received data to an internal Receive FIFO. Both FIFOs are 16 bits wide × 8 storage locations deep. Data frame sizes may be programmed to be from 4 to 16 bits in length.

The LH79520 DMAC (DMA controller) can be programmed to transfer data to and from the on-chip SSP. For more information, you may wish to download and refer to the LH79520 User's Guide, available on www.sharpsma.com.

The driver for the LH79520 SSP is named 79520_ssp_driver.c. The driver for the MMC interface is named lh79520_mmc_driver.c, see the 'Code Listing' section.

LOCK CONTROL

The SPI bus clock signal can be used by the SPI host to set the cards to energy saving mode or to control data flow (to avoid under-run or over-run conditions) on the bus. The host is allowed to change the clock frequency or stop it altogether.

There are a few restrictions the SPI host must follow:

The bus frequency can be changed at any time, but only up to the maximum data transfer frequency, defined by the MultiMediaCards.

It is an obvious requirement that the clock must be running for the MultiMediaCard to output data or response tokens. After the last SPI bus transaction, the host is required to provide 8 clock cycles for the card to complete the operation before shutting down the clock. During this 8-clock period, the state of the CS signal is irrelevant. It can be asserted or de-asserted.

SPI BUS TRANSACTIONS

Here is a list of the various SPI bus transactions:

- A command/response sequence. Eight clocks must be output after the card response end bit. The CS signal can be asserted or de-asserted during these 8 clocks.
- A read data transaction. Eight clocks must be output after the end bit of the last data block.
- A write data transaction. Eight clocks must be output after the end bit of the last data block.
- A write data transaction. Eight clocks must be output after the CRC status token.
- The host is allowed to stop the clock of a BUSY card. The MultiMediaCard will complete the programming operation regardless of the host clock. However, the host must provide a clock edge for the card to turn off its BUSY signal. Without a clock edge, the MultiMediaCard (unless previously disconnected by de-asserting the CS signal) will force the DataOut line LOW and hold it there.

MODE SELECTION

The MultiMediaCard's SPI mode is the mode used for this Application Note. All transactions described in this Application Note are based on the SPI mode.

The MultiMediaCard wakes up in the MultiMediaCard mode. It will enter SPI mode if the CS signal (pin 1 of the MMC) is asserted LOW during the reception of the Reset command (CMD0). If the card is in MultiMediaCard mode, it will not respond to SPI-based commands. If SPI mode is requested, the card will switch to SPI mode and respond with the SPI mode R1 response.

To return to the MultiMediaCard mode, power cycle the card. In SPI mode, the MultiMediaCard protocol state machine is not observed. MultiMediaCard commands supported in SPI mode are always available.

Since the card defaults to MultiMediaCard mode after a power cycle, Pin 1 (CS) must be pulled LOW and CMD0 (followed by a valid CRC byte) must be sent on the CMD (DataIn, Pin 2) line for the card to enter SPI mode.

In SPI mode, CRC checking is disabled by default. However, since the card always powers up in MultiMediaCard mode, CMD0 must be followed by a valid CRC byte (even though the command is sent using the SPI structure). Once the card enters SPI mode, CRCs are disabled by default.

CMD0 is a static command and always generates the same 7-bit CRC of 4Ah. Adding the '1' end bit (bit 0) to the CRC creates a CRC byte of 95h. The following hexadecimal sequence can be used to send CMD0 in all situations for SPI mode, since the CRC byte (although required) is ignored once in SPI mode. The entire CMD0 appears as: 40 00 00 00 00 95 (hexadecimal).

This CMD0 command (0x40 0x00 0x00 0x00 0x00 0x95) is the same command to switch the MMC card from MultiMediaCard mode to SPI mode. After this command is sent, CRC checking is disabled by default unless you want to enable it. When CRC checking is off, the last byte in a 6-byte command is ignored for read/write commands.

COMMAND AND RESPONSE

In the MMC command format, a command is comprised of 6 bytes and is sent MSB first. Once the SPI mode is set for 8-bit data width, 6-byte commands can be sent continuously. See the MMC_send_cmd ( ) function in the Code Listing.

Command responses may get a little tricky. The starting bit of the response may not align with the first block of the byte. The starting bit of the response may happen anywhere in the clock stream, depending on the speed of the MMC and the clock. So there is a need for manual alignment in software. See the MMC_get_response ( ) function in the Code Listing.

RESET SEQUENCE

The initialization command is described in the following sequence:

1. Send 80 clocks to start bus communication
2. Assert nCS LOW
3. Send CMD0
4. Send 8 clocks for delay
5. Wait for a valid response
6. If there is no response, back to step 4
7. Send 8 clocks of delay
8. Send CMD1
9. Send 8 clocks of delay
10. Wait for valid response
11. Send 8 clocks of delay
12. Repeat from step 9 until the response shows READY.

It will take a large number of cycles for CMD1 to finish its sequence. After every power cycle, the MMC will be in Idle state (not active), the Idle bit in its response will be 1 if using CMD13 (SEND_STATUS) to check the status. Once the CMD1 process is finished, the Idle bit in the response is cleared. Only after MMC is fully up from Idle mode to Active, can it be read and written.

See the MMC_init ( ) function in the Code Listing.

DATA READ

The SPI mode supports single block read operations only. Upon reception of a valid Read command, the card will respond with a Response token followed by a Data token in the length defined by a previous SET_BLOCK_LENGTH command. The start address can be any byte address in the valid address range of the card. Every block however, must be contained in a single physical card sector. After the Data Read command is sent from microcontroller to the card, the microcontroller will need to monitor the data stream input and wait for Data Token 0xFE. Since the response start bit 0 can happen any time in the clock stream, it's necessary to use software to align the bytes being read.

See the MMC_start_sector_read ( ) function in the Code Listing.

DATA WRITE

Data Write operations are similar to Data Read. In SPI mode, the MMC supports single block writes only. Upon reception of a valid Write command, the card will respond with a Response token and wait for a data block to be sent from the host. The only valid block length, however, is 512 bytes. After a data block is received, the card will respond with a Data-response token and if the data block is received with no errors, it will be programmed.

The microcontroller must first send the Write command, followed by the bytes to be written. After all the bytes have been sent, the microcontroller waits for the response. Based on the response received, the microcontroller can check whether there is any error in the response. After the response is sent back from the card, the card will set DataOut LOW because it will take time to do the write.

See the MMC_start_sector_write ( ) and mmc_write_data( ) function in the Code Listing.

DATA ERASE

Data erase follows a similar sequence to Data Read and Data Write. See mmc_erase_sector ( ) function in the Code Listing.

READING FAT16 FILE SYSTEM MASTER BOOT BLOCKS

Although the MultiMediaCard memory space is byte-addressable with addresses ranging from 0 to the last byte, it is not a simple byte array but rather it is divided into several structures.

Memory bytes are grouped into 512-byte blocks called sectors. Every block can be individually read, written, and erased.

Sectors are grouped into Erase groups of 16 or 32 sectors depending on card size. Any combination of sectors within one group or any combination of Erase groups can be erased in a single Erase command. A Write command implicitly erases the memory before writing new data into it. An explicit Erase command can be used for pre-erasing memory to speed up the next Write operation.

The FAT16 file system is commonly used on PCs, and it's easy to find a card reader to format the MultiMediaCard. After the MMC is formatted in FAT16 format, byte 0 to byte 0x200 will be used for the FAT16 format. This 512 bytes in sector 0 is also called the Master Boot Record (MBR) of the card, and will be in this format as shown in Table 13.

Inside the MBR, each partition entry is defined as shown in Table 14.

Table 13. FAT16 Master Boot Record

OFFSET	SIZE	DESCRIPTION
000h	446 bytes	Executable code (Boots Computer)
1BEh	16 bytes	1st Partition Entry (See Table 14)
1CEh	16 bytes	2nd Partition Entry
1DEh	16 bytes	3rd Partition Entry
1EEh	16 bytes	4th Partition Entry
1FEh	2 bytes	Executable Marker (55h AAh)

Table 14. FAT16 Partition Entries

OFFSET	SIZE	DESCRIPTION
00h		Current State of Partition, 1 byte (00h = Inactive, 80h = Active)
01h	1 byte	Beginning of Partition – Head
02h	2 bytes	Beginning of Partition – Cylinder/Sector
04h	1 byte	Type of Partition (See Table 15)
05h	1 byte	End of Partition – Head
06h	1 word	End of Partition – Cylinder/Sector
08h	1 double word	Number of Sectors Between the MBR and First Sector in the Partition
0Ch	1 double word	Number of Sectors in the Partition

Partition Type Table

Software should first read the Master Boot Record (MBR). The address for this command is 0x0. If you can read this block properly, the rest the FAT16 file system will follow easily.

Table 15. Partition Types

VALUE	DESCRIPTION
00h	Unknown or Nothing
01h	12 bit FAT
04h	16 bit FAT (Partition Smaller than 32MB)
05h	Extended MS-DOS Partition
06h	16 bit FAT (Partition Larger than 32MB)
0Bh	32 bit FAT (Partition Up to 2048GB)
0Ch	Same as 0Bh, but uses LBA1 13h Extensions
0Eh	Same as 06h, but uses LBA1 13h Extensions
0Fh	Same as 05h, but uses LBA1 13h Extensions

CONCLUSION

Using the LH79520's SPI interface is a practical way to read and write to a MultiMediaCard. This Application Note provides you with an operational basis for using an MMC in a design with the LH79520. For a detailed software implementation, see the Code Listing.

MultiMediaCard

MMC Host Algorithm Guideline

Date : October, 2004

Samsung Electronics Co., LTD
Semiconductor Flash Memory Product Planning & Applications

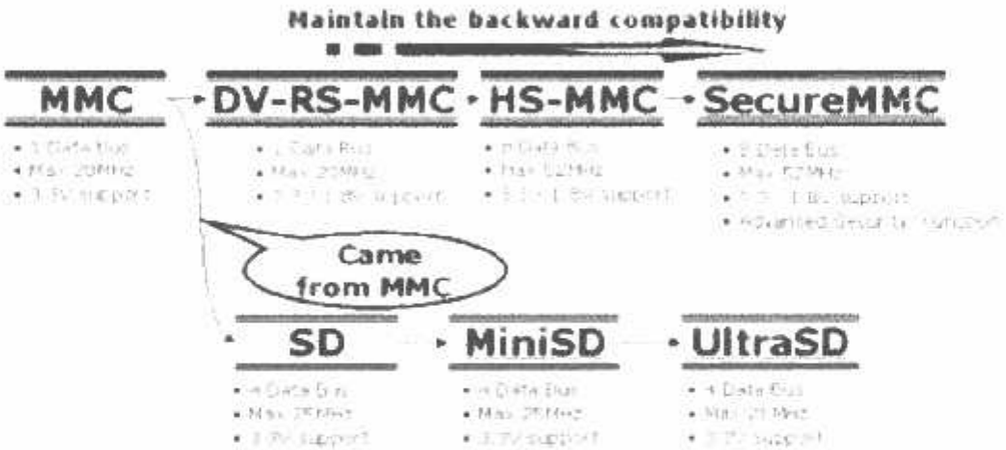
1. Introduction

1.1 Introduction

Today mobile devices' form factors are becoming smaller, and due to this fact MMC card is replacing the larger Compact Flash card rapidly. MMC card is flexible for the hosts, because of it's compatibility with the SD card hosts. Recently MMCplus and MMC mobile has been developed to provide high performance with x8 bus and 52MHz clock speed.

Although most host devices are implemented according to the MMC standard specification, the implementations are largely depend on the developer's interpretation of the MMC specification. Due to this fact some implementation shows incompatibility with certain MMC cards. With the result from extensive compatibility test with various DSC, DVC, PDA, and USB reader, this report proposes MMC standard compatible MMC host device implementation methodology.

❑ **SD cards originally came from MMC**



➢ **Almost all DSC vendors support both MMC and SD easily**

1.2 Advantage of MMC

- MMCA Open Standard
- No License Fee, No Royalty, lowest cost
- Widest choice of top card suppliers with their own controller & flash design & fabrication
- Over 100 members representing the leading manufacturers of host systems, cards, controllers, connectors, etc
- Best specs (size, speed, voltage, security, compliance, etc.)
- Already adopted by 4 out of the top 5 mobile phone suppliers

2. MMC card General Function

2.1 Comparison of MMC mode and SPI mode

MMC Card supports both MMC mode and SPI mode. MMC host device supports one of these two modes. Although the default mode for the MMC card is MMC mode SPI mode is supported for the flexible application support. But under the SPI mode not all the features and registers of MMC mode is supported, and thus cannot fully utilize MMC cards performance. This guideline explains the implementation method for the MMC mode to fully exploit the high performance of the MMC card.

- MMC mode : This mode is a main of the MultiMediaCard protocol. CMD and DAT lines are bi-directional channel.
- SPI mode : This mode is a subset of the MultiMediaCard protocol, designed to communicate with a SPI channel, commonly found in Motorola's (and lately a few other vendors') microcontrollers. The interface is selected during the first reset command after power up (CMD0) and cannot be changed once the part is power on. From the application point of view, the advantage of the SPI mode is the capability of using an off-the-shelf host, hence reducing the design-in effort to minimum. The disadvantage is the loss of performance of the SPI mode versus MultiMediaCard mode(lower data transfer rate, hardware CS, etc.).

		MMC Interface Mode			SPI Interface Mode		
Comparison of system specification	Interface	Ten-wire bus(CLK,CMD, DAT0-8)			Three-wire serial data bus(CLK,DI,DO)+CS		
	Frequency	0-20MHz, 0-26MHz, 0-52MHz			0-20MHz		
	Card selection	Card is selected by MMC bus protocol. Host sends the relative card address to select the card which has the same one			Card is selected by the CS signal		
	Access mode	Single block access, Multiple block access, Stream access			Single block access, Multi block access		
Pin Arrangement	Pin No.	Name	Type	Description	Name	Type	Description
	1	DAT3	I/O/PP	Data	CS	Input	Chip Select
	2	CMD	I/O/PP/OD	Command/ Response	DI	I/PP	Data In
	3	VSS1	-	GND	VSS		GND
	4	VDD		VCC	VDD	-	VCC
	5	CLK	Input	Clock	SCLK	Input	Clock
	6	VSS2	-	GND	VSS2		GND
	7	DAT0	I/O/PP	Data	DO	O/PP	Data Out
	8	DAT1	I/O/PP	Data	Not Used		
	9	DAT2	I/O/PP	Data	Not Used		
	10-13	DAT4-DAT7	I/O/PP	Data	Not Used		

Table 1. MultiMediaCard Interface Pin Configuration

2.2 MMC mode Bus protocol

CMD line is used for the command and response transfer and DAT line is used for the data transfer between MMC host device and the MMC card. Command is sent from

the host device to the card and response sent from the card to the host. Data is sent from the card to the host for the read operation and vice versa for the write operation.

Figure 1. MMC Bus Protocol

2.3 SPI mode Bus protocol

CMD and DAT line for the MMC mode is replaced by dataIn and dataOut line for the SPI mode. While CMD and DAT line were bidirectional buses, dataIn and dataOut lines are unidirectional buses. So the command and data from the host device are sent over dataIn line and response and data from the card are sent over the dataOut line.

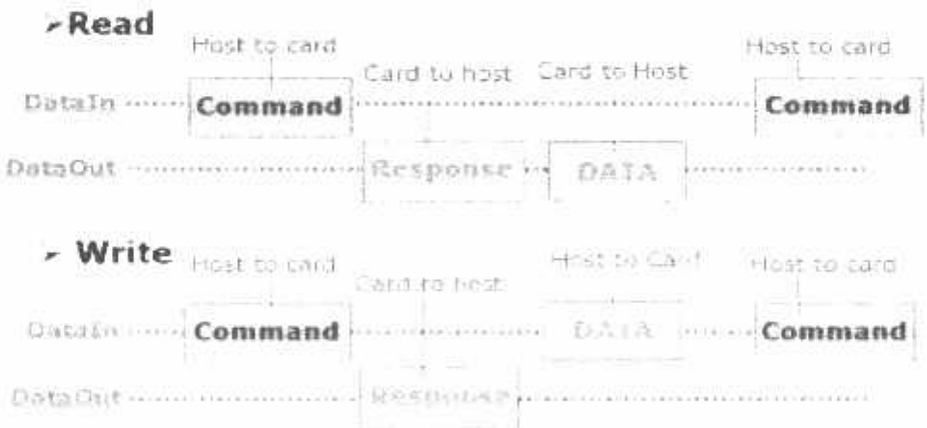


Figure2. SPI Bus Protocol

2.4 General Function Description

MMC system has various operation modes such as Inactive mode, Card identification mode, Data Transfer mode, Interrupt mode. Most of the time it's operational the system is under the Card identification mode and Data transfer mode.

- Card identification mode : The host will be in card identification mode after reset and while it is looking for new cards on the bus..
  After power-on by the host, cards is in MultiMediaCard mode and in Idle State. Command GO_IDLE_STATE(CMD0) is the software reset command and puts the card into Idle State. After the bus is activated, the host will request the cards to send its valid operation conditions (CMD1). The response to CMD1 is the 'wired and' operation on the condition restrictions of all cards in the system. Incompatible cards are sent into Inactive State. The host then issues the

broadcast command ALL_SEND_CID(CMD2), asking all cards for its unique card identification(CID) number. Since CID numbers are unique for each card, there should be only one card which successfully sends its full CID-number to the host. This card then goes into Identification State. Thereafter, the host issues CMD3(SET_RELATIVE_ADDR) to assign to this card in the future data transfer mode.

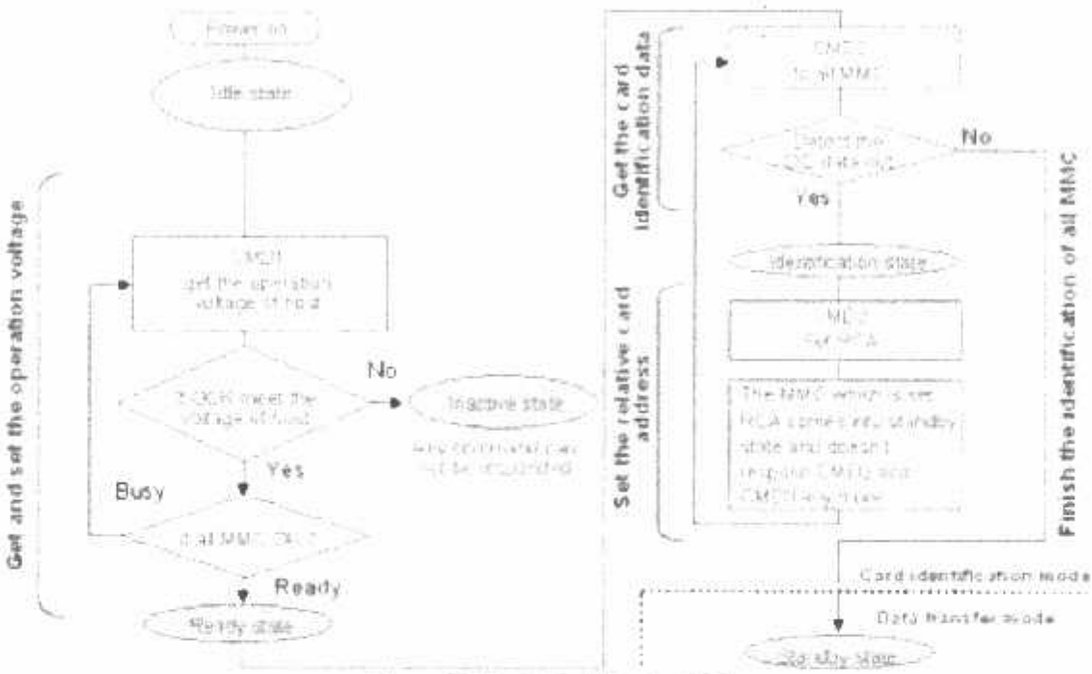


Figure 3. Card Identification Mode

- Data Transfer mode : Card will enter data transfer mode once an RCA is assigned to them. The host will enter data transfer mode after identifying all the cards on the bus. The host issues SEND_CSD(CMD9) to obtain the Card Specific Data(CSD register), e.g. block length, card storage capacity, maximum clock rate, etc.
  The broadcast command SET_DSR(CMD4) configures the driver stages of the card. It programs its DSR register corresponding to the application bus layout and the data.
  All data communication in the Data Transfer Mode is point-to-point between the host and the selected card. All addressed commands get acknowledged by response on the CMD line.

CMD Index	Abbreviation	Command Description
CMD60... CMD63	reserved for manufacturer	

Table 10. Application Specific Commands (Class 8)

4. MMC card Application & General Algorithm

4.1 MMC card Application

Today the biggest application for the MMC card is DSC, which comprises 70% of the total MMC card market. But other applications such as DVC, PDA, MP3, Mobile phone are growing rapidly. With the introduction of RS-MMC to the market and rapid multimedia feature convergence for the mobile phones, the adaptation of the MMC card by the mobile phone is expected to grow tremendously.

4.2 Efficient MMC Standard Algorithm

Although the MMC host devices implement the MMC standard specification, many times the host device results in unstable state by issuing abundant needless commands, or continuing without proper status checking.

4.2.1 Operation Sequence

In summary, the host device goes through initialization step after the Power-On. At this step the card is under the Identification mode. After this step the card is under data transfer mode. After the initialization the host device reads the FAT information from the card and performs read and write operation afterwards. Following are detailed explanations of the each step

4.2.2 Initialization

After the Power-on the host devices sends reset command (CMD0). To check the card's operation condition Host issues CMD1 and puts the card into ready state. After CMD1 host issues CMD2 to check the card's CID. Then the host issues CMD3 to put the card into Stand-by state.

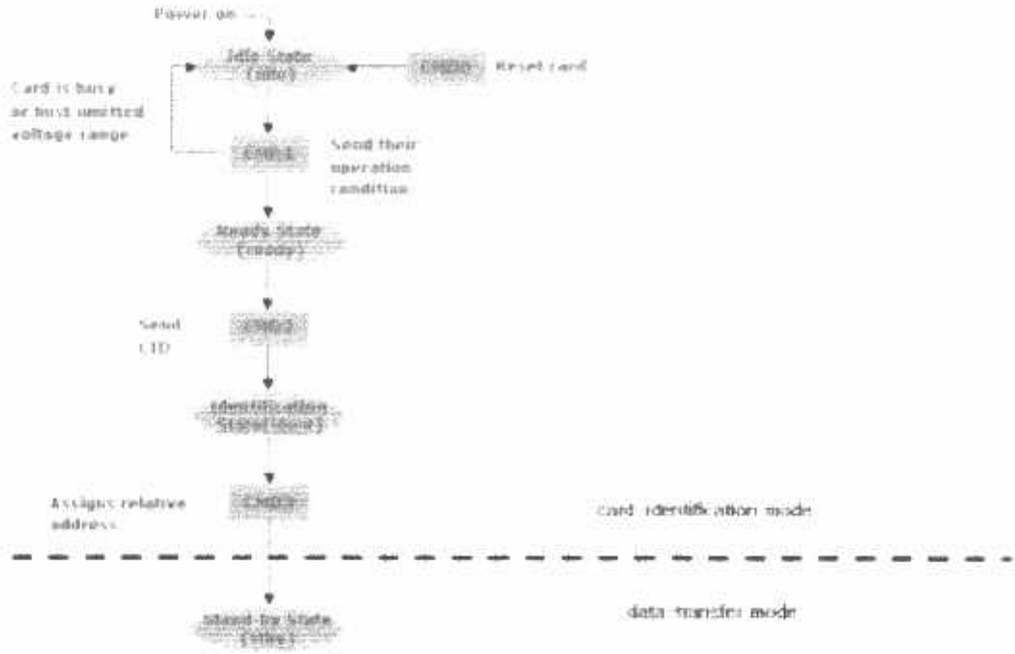


Figure 5. Initialization

4.2.3 MBR, PBR and FAT access

After the initialization the host accesses the card's MBR, PBR, and FAT region. When accessing FAT information, CMD18 should be used over CMD17 to achieve accurate and stable transfer of the information.

Under the Data transfer mode CMD13 (SEND_STATUS) should be used prior to each command to check the card's current status.

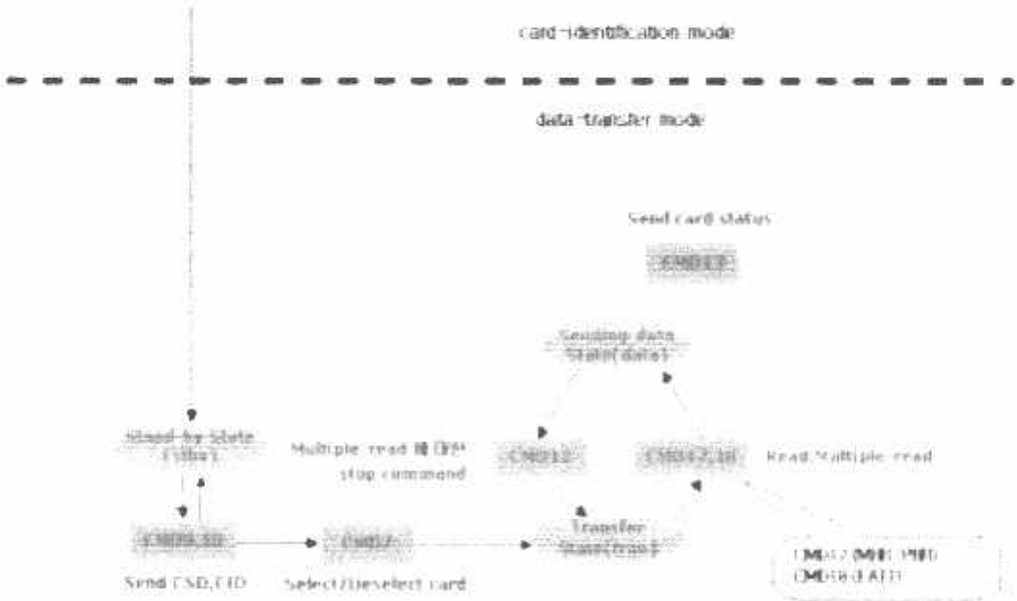


Figure 6. MBR,PBR & FAT access

4.2.4 Read, Write & Erase

■ Data read

For the read operation CMD17 (READ_SINGLE_BLOCK) and CMD18 (READ_MULTIPLE_BLOCK) are used. For the efficient read operation only the required data should be read.

Data Read(CMD17+CMD13 or CMD18+CMD12+CMD13)

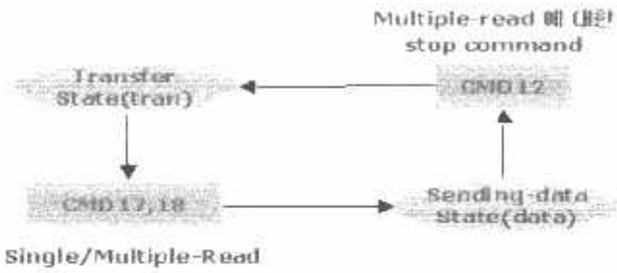


Figure 7. Read

■ Data write

For the write operation CMD24 (WRITE_SINGLE_BLOCK) and CMD25 (WRITE_MULTIPLE_BLOCK) are used. After the FAT is read, data write operation is performed followed by the FAT update.

FAT read(CMD18+CMD12) →
Data Write(CMD24+CMD13 or CMD25+CMD12+CMD13) →
FAT update(CMD24 or CMD25)

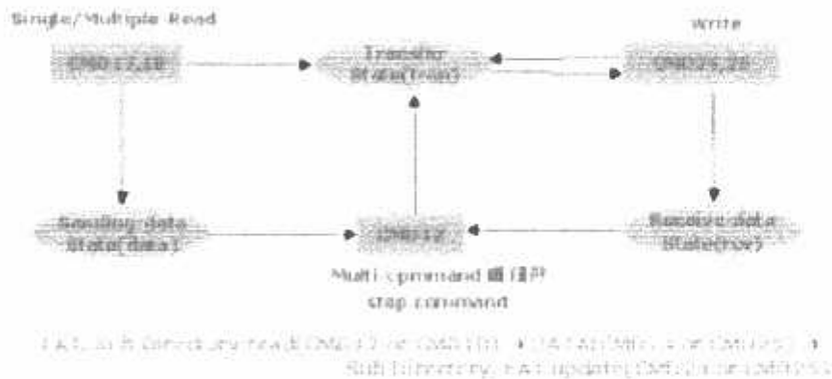


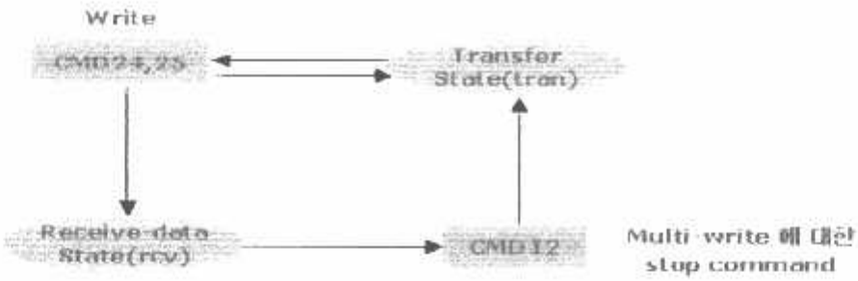
Figure 8. Write

Multiple Block Read and Write command continues until the STOP command is issued. So when the CMD18 and CMD25 are used, it has to be finished with the CMD12.

■ Data erase

The information in the FAT has to be erased and the file in the sub-directory has to be erased.

FAT update(CMD24 or CMD25+CMD12) →
Sub Directory update(CMD24 or CMD25+CMD12)



After each Read, Write, and Erase operation is performed the host has to use CMD13 to check the card's status

5. Conclusion

The implementation guideline explained in this paper is based on the MMC standard specification. On top of the standard specification, the result from the benchmarking of the various application hosts has been used to guide the most stable and efficient host implementation methodology.

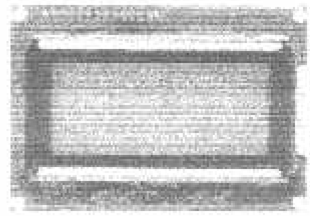
Dot Matrix Liquid Crystal Display Modules

CHARACTER TYPE

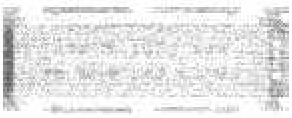
- **FEATURES :**
  - Slim, light weight and low power consumption
  - High contrast and wide viewing angle
  - Built-in controller for easy interfacing
  - LCD modules with built-in EL or LED backlight

M1641

L1642

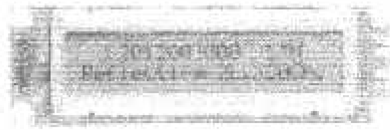


L1614



M1532

L1652



L2012

- **SPECIFICATIONS :**

Standard products　　Products of optional specification

cter Format (character x line)		16 x 1	16 x 2	16 x 2	16 x 2	16 x 4	20 x 2
		M1641	M1632	L1642	L1652	L1614	L2012
ive		M16410AS	M16320AS	L16420CJ000S	L165200J200S	L161400J000S	L201200J000S
klight		M164190WS	M163290WS	L16422 1J000S	L16522 1J200S	L161421J000S	L201221J000S
acklight		M16417DYS	M16327DYS	L16428 1J000S	L16528 1J200S	L1614B1J000S	L201281J000S
ive (wide temp)		M16410CS	M16320CS	L16420 0L000S	L165200L200S	L161400L000S	L201200L000S
acklight (wide temp)		M16417JYS	M16327JYS	L1642B1L000S	L1652B1L200S	L1614B1L000S	L201281L000S
cter font		5x7 dots + cursor	5x7 dots + cursor	5x7 dots + cursor	5x7 dots + cursor	5x7 dots + cursor	5x7 dots + cursor
e	Reflective	80,0 x 36,0 x 11,3	85,0 x 30,0 x 10,1	80,0 x 36,0 x 11,3	122,0 x 44,0 x 11,3	87,0 x 60,0 x 11,6	116,0 x 37,0 x 11,3
	EL backlight	80,0 x 36,0 x 11,3	85,0 x 30,0 x 10,1	80,0 x 36,0 x 11,3	122,0 x 44,0 x 11,3	87,0 x 60,0 x 11,6	116,0 x 37,0 x 11,3
T) mm	LED backlight	80,0 x 36,0 x 15,8	80,0 x 30,0 x 15,8	80,0 x 36,0 x 15,8	122,0 x 44,0 x 15,6	87,0 x 60,0 x 15,8	116,0 x 37,0 x 15,8
g area (HxV) mm		64,5 x 13,8	32,0 x 16,0	64,5 x 13,8	99,0 x 24,0	61,8 x 25,2	83,0 x 18,6
ter size (HxV) mm *1		3,07 x 5,73	2,78 x 4,27	2,95 x 3,80	4,84 x 8,06	2,95 x 4,15	3,20 x 4,85
e (HxV) mm		0,55 x 0,75	0,50 x 0,55	0,50 x 0,55	0,92 x 1,10	0,55 x 0,55	0,60 x 0,65
supply voltage (VDD-VSS) V		+5 V	+5 V	+5 V	+5 V	+5 V	+5 V
t consumption	IDD	1,5	2,0	1,8	2,0	2,7	2,0
a)	ILC *4	0,2	0,2	0,3	0,4	1,1	0,4
g method (duty)		1/16	1/16	1/16	1/16	1/16	1/16
r LSI		KS0066 or equivalent	KS0066 MSM5839 or equivalent	KS0066 MSM5839 or equivalent	KS0066 MSM5839 or equivalent	KS0066 KS0063 or equivalent	KS0066 KS0063 or equivalent
ing temperature (°C)	normal temp.	0 to +50	0 to +50	0 to +50	0 to +50	0 to +50	0 to +50
	wide temp. *2	-20 to +70	-20 to +70	-20 to +70	-20 to +70	-20 to +70	-20 to +70
e temperature (°C)	normal temp.	-20 to +60	-20 to +60	-20 to +60	-20 to +60	-20 to +60	-20 to +60
	wide temp.	-30 to +80	-30 to +80	-30 to +80	-30 to +80	-30 to +80	-30 to +80
	Reflective	25	25	25	50	50	40
t	EL backlight	30	30	30	55	55	45
	LED backlight	35	40	35	65	65	60
	Model	5S	5S	5S	5C	5A	5A
ns	Power supply (V)	+5,0	+5,0	+5,0	+5,0	+5,0	+5,0
	current consumption (mA) *3	10	10	10	35	45	45
	Forward current consumption (mA)	100	112	100	240	200	154
ht	Forward input voltage (V,typ.)	+4,1	+4,1	+4,1	+4,1	+4,1	+4,1

cursor
H: Horizontal　V: Vertical　T: Thickness (max)

h external temperature compensation

uding EL backlight

at normal temperature range

r policy is one of continuous improvements we reserve the right to change the specifications for the products in the catalogue without notice.

L2022

L2432

L4042

L2014

M4024

• SPECIFICATIONS :

: Standard products : Products of optional specification

Character Format (character x line)			20 x 2	20 x 4	24 x 2	40 x 2	40 x 4
Model			L2022	L2014	L2432	L4042	M4024
Reflective				L201400J000S	L243200J000S	L404200J000S	M40240AS
EL backlight				L201421J000S	L243221J000S	L404221J000S	M40249CWS
LED backlight				L2014B1J000S	L2432B1J000S	L4042B1J000S	M40247DYS
Reflective (wide temp.)			L202200P000S	L201400_000S	L243200L000S	L404200_000S	M40240CS
LED backlight (wide temp.)			L2022B1P000S	L2014B1L000S	L2432B1L000S	L4042B1L000S	M40247JYS
Character font			5x7 dots + cursor	5x7 dots + cursor	5x7 dots + cursor	5x7 dots + cursor	5x7 dots + cursor
Module size (HxWxT) mm	Reflective		180,0 x 40,0 x 10.5	98,0 x 60,0 x 11,6	118.0 x 36.0 x 11,3	182.0 x 33.5 x 11,3	198.0 x 54.0 x 10.1
	EL backlight		180,0 x 40,0 x 10.5	98,0 x 60,0 x 11,6	118.0 x 36.0 x 11,3	182.0 x 33.5 x 11,3	198.0 x 54.0 x 10.1
	LED backlight		180.0 x 40.0 x 14.5	98,0 x 60,0 x 15.6	118.0 x 36.0 x 15,8	182.0 x 33.5 x 15,3	198.0 x 54.0 x 16.3
Viewing area (HxV) mm			149.0 x 23,0	76.0 x 25.2	94.5 x 17,8	154.4 x 15.8	147,0 x 29.5
Character size (HxV) mm *1			3,0 x 9,66	2,95 x 4,15	3,20 x 4,85	3,20 x 4,85	2,78 x 4,27
Dot size (HxV) mm			1,12 x 1,12	0,55 x 0,55	0,60 x 0,65	0,60 x 0,65	0,50 x 0,55
Power supply voltage (VDD-VSS) V			+ 5 V	+ 5 V	+ 5 V	+ 5 V	+ 5 V
Current consumption (mA typ)	IDD		4,2	2,9	2,5	3.0	8,0
	ILC *4		2,6	1,2	0,5	1.0	3,0
Driving method (duty)			1/16	1/16	1/16	1/16	1/16
Built-in LSI			KS0066 KS0063 or equivalent	KS0066 MSV5839 or equivalent	KS0066 KS0063 or equivalent	KS0066 KS0063 or equivalent	KS0066 VSM5839 or equivalent
Operating temperature (°C)		normal temp.	-	0 to + 50	0 to + 50	0 to + 50	0 to + 50
		wide temp. *2	- 20 to + 70	- 20 to + 70	- 20 to + 70	- 20 to + 70	- 20 to + 70
Storage temperature (°C)		normal temp.	-	- 20 to + 60	- 20 to + 60	- 20 to + 60	- 20 to + 60
		wide temp.	- 30 to + 80	- 30 to + 80	- 30 to + 80	- 30 to + 80	- 30 to + 80
Weight (g typ.)	Reflective		80	55	40	70	90
	EL backlight			60	45	75	105
	LED backlight		110	70	60	95	140
Inverters for EL backlight	Model		-	5A	5A	5C	5D
	Power supply (V)		- 5.0	+ 5.0	+ 5.0	+ 5.0	+ 5.0
	Current consumption (mA) *3		-	45	45	23	80
LED backlight	Forward current consumption (mA)		320	240	150	260	480
	Forward input voltage (V typ.)		+ 4.1	+ 4.1	+ 4.1	+ 4.1	+ 4.1

*1 : Excluding cursor H : Horizontal V : Vertical T : Thickness (max)

*2 : With external temperature compensation

*3 : Including EL backlight

*4 : Based on normal temperature range

Dot Matrix Liquid Crystal Display Modules

GRAPHIC TYPE

- **FEATURES :**
  - •Wide viewing angle and high contrast
  - •Full dot configuration fits any application
  - •Slim, light weight and low power consumption
  - •Available in STN and FSTN

- **SPECIFICATIONS :**

rmat (HxV,dot)			97 x 32	128 x 32	128 x 64	128 x 64
ype	Reflective	built-in RAM	Y97031	G1213	G1215	G1226
mode)	Reflective wide temp.	built-in RAM	-	-	-	-
	LED backlight	built-in RAM	-	G121300N0005	G121603N0005	-
	LED backlight wide temp	built-in RAM	-	-	-	G122481N0005
Type	Transmissive	-	-	G121981N0005	G121981N0005	-
mode)	with CFL backlight	built-in controller	-	-	-	-
	Transflective	built-in RAM	-	-	-	-
a size	Reflective (no backlight)		Y97031LEBW	-	-	-
x T)	LED backlight		47,5 x 55,4 x 2,1	75,0 x 41,5 x 6,8	75,0 x 32,7 x 6,8	-
	CFL backlight		-	75,0 x 41,5 x 8,9	75,0 x 62,7 x 8,9	93,0 x 70,0 x 11,4
g area (HxV) mm			-	-	-	-
s (H x V) mm			43,5 x 23,9	60,0 x 21,3	60,0 x 32,5	70,7 x 35,9
ch (H x V) mm			0,35 x 0,48	0,40 x 0,48	0,40 x 0,40	0,44 x 0,44
supply voltage (V)		(VDD - VSS)	0,39 x 0,52	0,43 x 0,51	0,43 x 0,43	0,48 x 0,48
		(VLC - VSs)	+5,0	+5,0	+5,0	+5,0
l consumption		IDD		-3,0	-6,1	-8,2
		IDD (built-in controller)	0,10	2,0	2,0	3,0
p.)		ILC	-	-	-	-
	Driving method (duty)		1/33	1/64	1/64	1/64
LS		Driver	SED1530	HD61202	HD61202	KS0107
				HD61203	HD61203	KS0108
		Controller	or equivalent	or equivalent	or equivalent	or equivalent
ng temperature range (°C)			-	-	-	-
e temperature range (°C)			-20 to +70	-20 to +70	-20 to +70	0 to +50
	Reflective (Transflective no backlight)		-30 to +80	-30 to +80	-30 to +80	-20 to +60
)	LED backlight		10	23	35	-
	CFL backlight		-	35	40	72
cklight	Forward current consumption (mA)			-	-	-
	Forward input voltage (V, typ.)			40	90	125
	Mode			3,8	4,1	4,1
for CFL	Power supply voltage (V)		-	-	-	-
	Current consumption (mA, typ.)		-	-	-	-
				-	-	-

-in DC/DC converter (single power source)

with external temperature compensation circuit

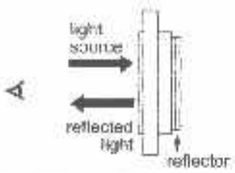
ur policy is one of continues improvements we reserve the right to change the specifications of the products in the catalogue without notice

iquid Crystal Display Modules

REFLECTIVE/TRANSFLECTIVE/TRANSMISSIVE LCD

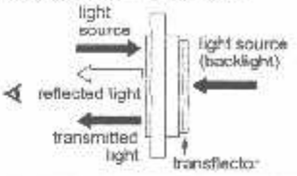
Reflective LCD

leflector bonded to the rear polarizer eflects the incoming ambient light. Low power onsumption because no backlight is required.



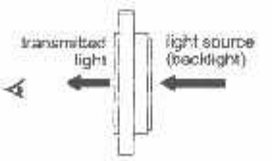
❷ Transflective LCD

Transflector bonded to the rear polarizer reflects light from the front as well as enabling lights to pass through the back. Used with backlight off in bright light and with it on in low light to reduce power consumption.

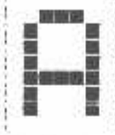
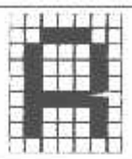


❸ Transmissive LCD

Without reflector or transflector bonded to the rear polarizer. Backlight required. Most common is transmissive negative image.



POSITIVE/NEGATIVE MODE

Positive type		Negative type		Negative type (inverse image) (when data is inverted) 

N TYPE/STN TYPE/FSTN TYPE

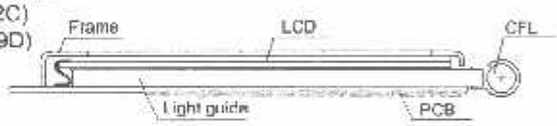
	(Background/dot color) Gray/Black	TN(Twisted Nematic) type is most conventional and economical. It is used for static drive LCD and low-duty drive LCD (watch,calculator, etc.)
'N	Yellowgreen/Dark blue Gray/Dark blue White/Blue	STN (Super Twisted Nematic) type has a higher twist angle, and thus provides clear visibility and wider viewing angle. This is suitable especially for high-duty drive LCD.
FTN	White/Black	FSTN (Film Super Twisted Nematic) type utilizes RCF (Retardation Control Film) to remove the coloring of STN LCD. Thus FSTN type provides easy-to-read black-and-white display.

TRUCTURE AND FEATURE OF LCD MODULE WITH BACKLIGHT

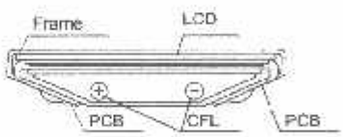
FL (Cold Cathode Fluorescent Lamp) backlight

eatures: high brightness, long service life, inverter required

Edge backlight type
(G2446,G242C)
(G321D,G649D)

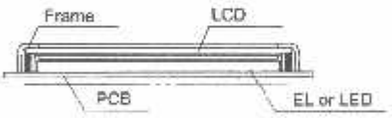


Backlight type



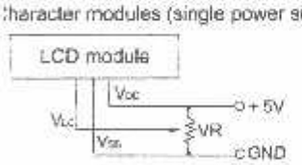
EL (Electroluminescent Lamp) backlight
LED (Light Emitting Diode) backlight

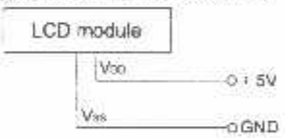
Features: EL: thin, inverter required
LED: long service life, low voltage driving, no inverter required

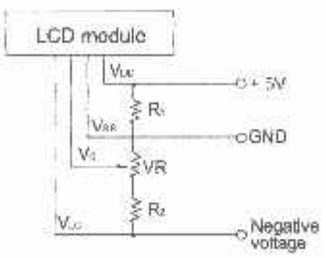


OWER SUPPLY

haracter modules (single power supply) ● G2446,G242C (Built-in DC-DC conv.) ● G321D, G324E and G649D

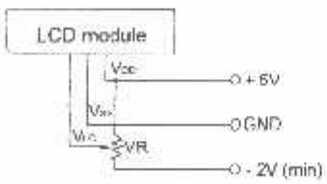


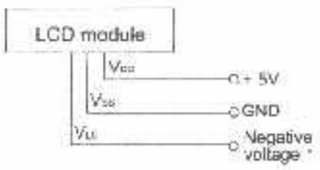




Note 1.Contrast can be adjusted by VR.
Note 2.For module with backlight, power supply for backlight is necessary.

Character Modules(Dual power supply) ● Y1206 and G1226





*Negative voltage should be variable for contrast adjustment.

TDA2003

10W CAR RADIO AUDIO AMPLIFIER

DESCRIPTION

The TDA 2003 has improved performance with the same pin configuration as the TDA 2002.

The additional features of TDA 2002, very low number of external components, ease of assembly, space and cost saving, are maintained.

The device provides a high output current capability (up to 3.5A) very low harmonic and cross-over distortion.

Completely safe operation is guararteed due to protection against DC and AC short circuit between all pins and ground, thermal over-range, load dump voltage surge up to 40V and fortuitous open ground.

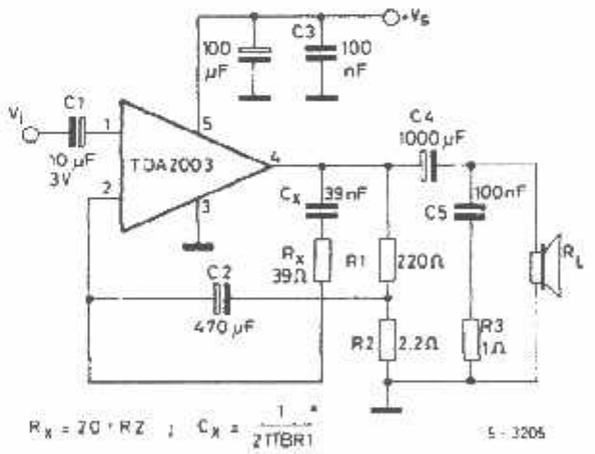
PENTAWATT

ORDERING NUMBERS : TDA 2003H
TDA 2003V

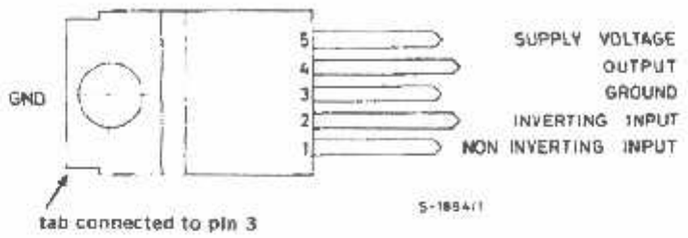
ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Peak supply voltage (50ms)	40	V
V_S	DC supply voltage	28	V
V_S	Operating supply voltage	18	V
I_O	Output peak current (repetitive)	3.5	A
I_O	Output peak current (non repetitive)	4.5	A
Ptot	Power dissipation at Tcase = 90°C	20	W
T_{stg}, T_j	Storage and junction temeperature	-40 to 150	°C

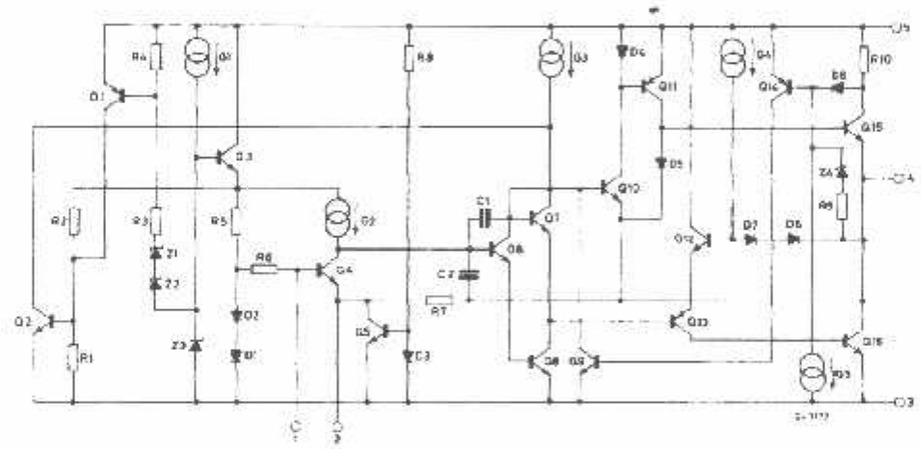
TEST CIRCUIT



PIN CONNECTION (top view)



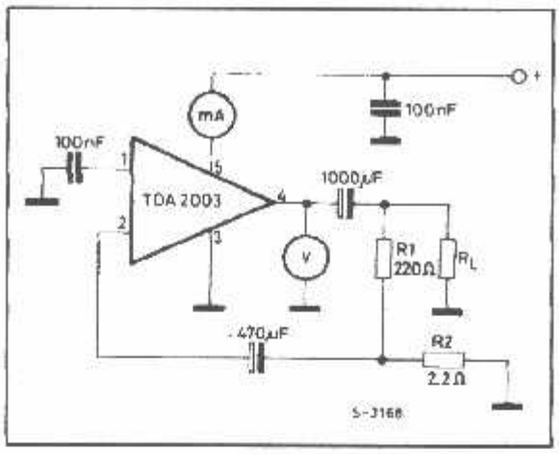
SCHEMATIC DIAGRAM



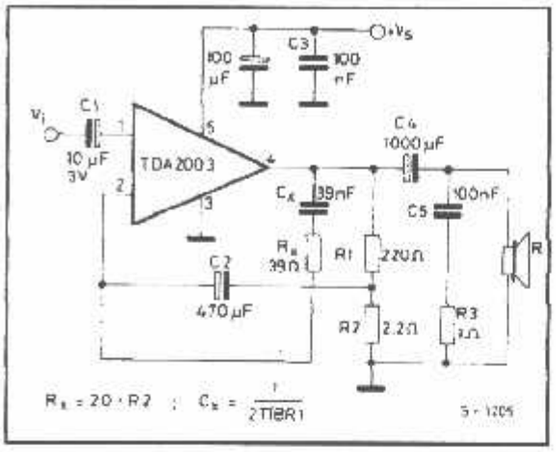
THERMAL DATA

Symbol	Parameter		Value	Unit
$R_{th\text{-}j\text{-}case}$	Thermal resistance junction-case	max	3	°C/W

DC TEST CIRCUIT



AC TEST CIRCUIT



ELECTRICAL CHARACTERISTICS (V_s = 14.4V, T_{amb} = 25 °C unless otherwise specified)

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
DC CHARACTERISTICS (Refer to DC test circuit)						
V_s	Supply voltage		8		18	V
V_o	Quiescent output voltage (pin 4)		6.1	6.9	7.7	V
I_d	Quiescent drain current (pin 5)			44	50	mA
AC CHARACTERISTICS (Refer to AC test circuit, Gv = 40 dB)						
P_o	Output power	d = 10% f = 1 kHz R_L = 4Ω R_L = 2Ω R_L = 3.2Ω R_L = 1.6Ω	5.5 9	6 10 7.5 12		W W W W
$V_{i(rms)}$	Input saturation voltage		300			mV
V_i	Input sensitivity	f = 1 kHz P_o = 0.5W R_L = 4Ω P_o = 6W R_L = 4Ω P_o = 0.5W R_L = 2Ω P_o 10W R_L = 2Ω		14 55 10 50		mV mV mV mV

ELECTRICAL CHARACTERISTICS (continued)

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
B	Frequency response (-3 dB)	P_o = 1W R_L = 4Ω		40 to 15,000		Hz
d	Distortion	f = 1 kHz P_o = 0.05 to4.5W R_L = 4Ω P_o = 0.05 to 7.5W R_L = 2Ω		0.15 0.15		% %
R_i	Input resistance (pin 1)	f = 1 kHz	70	150		kΩ
G_v	Voltage gain (open loop)	f = 1 kHz f = 10 kHz		80 60		dB dB
G_v	Voltage gain (closed loop)	f = 1 kHz R_L = 4Ω	39.3	40	40.3	dB
e_N	Input noise voltage (0)			1	5	µV
i_N	Input noise current (0)			60	200	pA
η	Efficiency	f = 1 Hz P_o = 6W R_L = 4Ω P_o = 10W R_L = 2Ω		69 65		% %
SVR	Supply voltage rejection	f = 100 Hz V_{ripple} = 0.5V R_g = 10 kΩ R_L = 4Ω	30	36		dB

(0) Filter with noise bandwidth: 22 Hz to 22 kHz

Figure 1. Quiescent output voltage vs. supply voltage

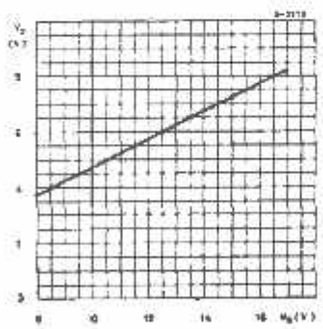


Figure 2. Quiescent drain current vs. supply voltage

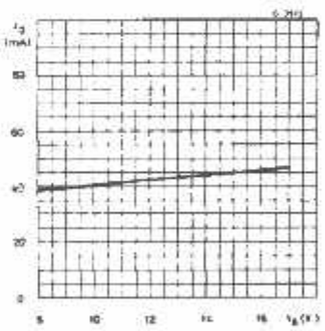


Figure 3. Output power vs. supply voltage

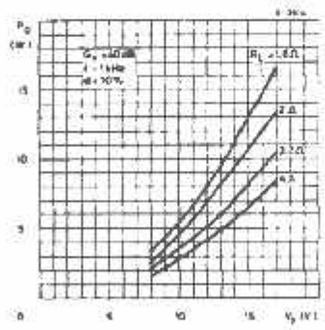


Figure 4. Output power vs. load resistance R_L

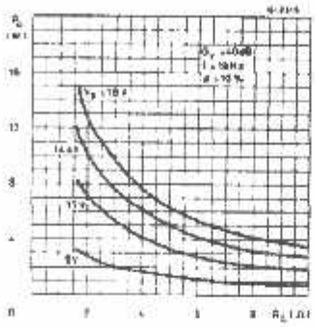


Figure 5. Gain vs. input sensivity

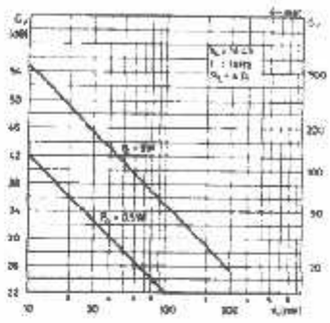


Figure 6. Gain vs. input sensivity

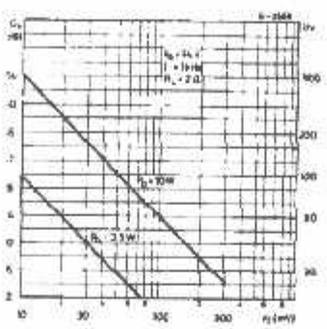


Figure 7. Distortion vs. output power

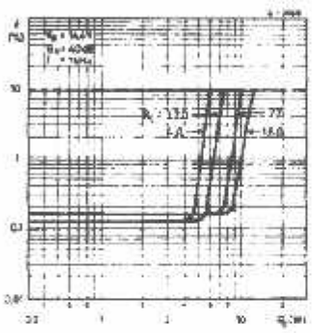


Figure 8. Distortion vs. frequency

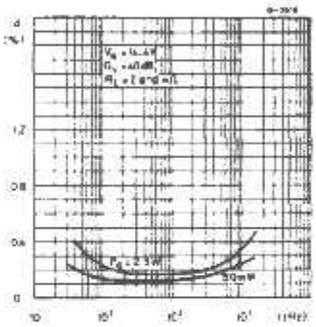


Figure 9. Supply voltage rejection vs. voltage gain

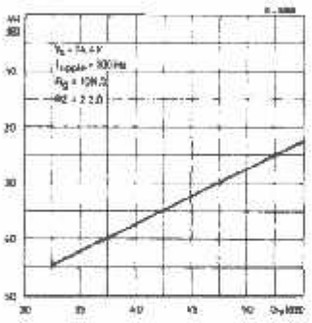


Figure 10. Supply voltage rejection vs. frequency

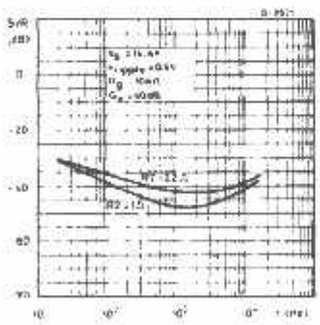


Figure 11. Power dissipation and efficiency vs. output power (R_L = 4Ω)

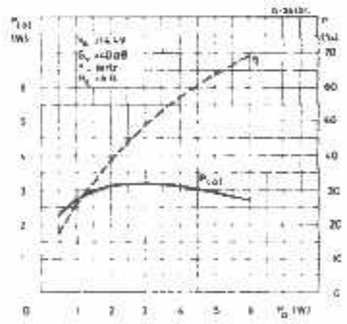
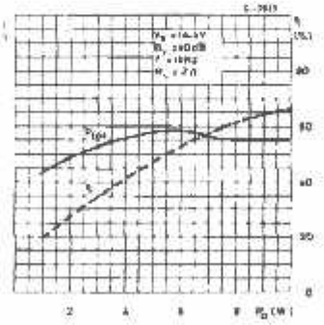


Figure 12. Power dissipation and efficiency vs. output power (R_L = 2Ω)



LISTING PROGRAM

```
                                    lcd_mega8
* ================================================
      Routine untuk menjalankan LCD 16x2 Character
      type L1602 atau M1632 compatible
      data bus          : 4 bit
wr/rd mode     : write only
author                    : MARTHEN LEUNARD
NIM                                  : 0117025
      Teknik Elektro ITN Malang
      Indonesia
 ================================================ */

nclude <delay.h>

efine     LCDDATA7       PORTD.7
efine     LCDDATA6       PORTD.6
efine     LCDDATA5       PORTD.5
efine     LCDDATA4       PORTB.7
efine     LCD_RS         PORTD.4
efine     LCD_EN         PORTD.3

id LCD_data(char c,char dat)

  LCD_RS = c;
  if ((dat & 0x80)==0x80) LCDDATA7=1; else LCDDATA7=0;
  if ((dat & 0x40)==0x40) LCDDATA6=1; else LCDDATA6=0;
  if ((dat & 0x20)==0x20) LCDDATA5=1; else LCDDATA5=0;
  if ((dat & 0x10)==0x10) LCDDATA4=1; else LCDDATA4=0;
  LCD_EN = 0;          LCD_EN = 1;
  if ((dat & 0x08)==0x08) LCDDATA7=1; else LCDDATA7=0;
  if ((dat & 0x04)==0x04) LCDDATA6=1; else LCDDATA6=0;
  if ((dat & 0x02)==0x02) LCDDATA5=1; else LCDDATA5=0;
  if ((dat & 0x01)==0x01) LCDDATA4=1; else LCDDATA4=0;
  LCD_EN = 0;          LCD_EN = 1;
  delay_ms(2);


id Tulis_LCD(char a,char flash *dat)

  char i = 0;
  LCD_data(0,a); while(dat[i] != 0) { LCD_data(1,dat[i]); i++; }


id LCD_test()

      DDRD=0xFF;
      DDRB.7=1;
 initialisasi LCD
      delay_ms(2000);
      LCD_data(0,0x3F); delay_ms(100); LCD_data(0,0x3F);delay_ms(100);
      LCD_data(0,0x3F); LCD_data(0,0x08); LCD_data(0,0x01); LCD_data(0,0x06);
      LCD_data(0,0x2F); LCD_data(0,0x0E); LCD_data(0,0x06);


      Tulis_LCD(0x80,"  EKO BAGUS S.  ");
      Tulis_LCD(0xC0,"NIM : 01.17.013 ");

      while(1);
```

```
                                        LCD.c
* ================================================
      Routine untuk menjalankan LCD 16x2 Character
      type L1602 atau M1632 compatible
      data bus        : 4 bit
wr/rd mode    : write only
author                  : MARTHEN LEUNARD
NIM                                 : 0117025
      Teknik Elektro ITN Malang
      Indonesia
 ================================================ */

nclude <delay.h>

efine    LCDDATA7      PORTD.7
efine    LCDDATA6      PORTD.6
efine    LCDDATA5      PORTD.5
efine    LCDDATA4      PORTB.7
efine    LCD_RS        PORTD.4
efine    LCD_EN        PORTD.3

id LCD_data(char c,char dat)

  LCD_RS = c;
  if ((dat & 0x80)==0x80) LCDDATA7=1; else LCDDATA7=0;
  if ((dat & 0x40)==0x40) LCDDATA6=1; else LCDDATA6=0;
  if ((dat & 0x20)==0x20) LCDDATA5=1; else LCDDATA5=0;
  if ((dat & 0x10)==0x10) LCDDATA4=1; else LCDDATA4=0;
  LCD_EN = 0;          LCD_EN = 1;
  if ((dat & 0x08)==0x08) LCDDATA7=1; else LCDDATA7=0;
  if ((dat & 0x04)==0x04) LCDDATA6=1; else LCDDATA6=0;
  if ((dat & 0x02)==0x02) LCDDATA5=1; else LCDDATA5=0;
  if ((dat & 0x01)==0x01) LCDDATA4=1; else LCDDATA4=0;
  LCD_EN = 0;          LCD_EN = 1;
  delay_ms(2);


id Tulis_LCD(char a,char flash *dat)

  char i = 0;
  LCD_data(0,a); while(dat[i] != 0) { LCD_data(1,dat[i]); i++; }
```

```
*****************************************
is program was produced by the
)dewizardAVR V1.24.0 Standard
itomatic Program Generator
Copyright 1998-2003 HP InfoTech s.r.l.
:tp://www.hpinfotech.ro
mail:office@hpinfotech.ro

'oject :
rsion :
te    : 3/19/2009
thor  : HL
mpany :
mments:


ip type           : ATmega8
ogram type        : Application
ock frequency     : 8.000000 MHz
mory model        : Small
ternal SRAM size  : 0
ta Stack size     : 256
*****************************************/

nclude <mega8.h>
nclude <LCD_mega8.c>

 Declare your global variables here

id main(void)

 Declare your local variables here

 Input/Output Ports initialization
 Port B initialization
 Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
 State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
RTB=0x00;
RB=0x00;

 Port C initialization
 Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In
 State0=T State1=T State2=T State3=T State4=T State5=T State6=T
RTC=0x00;
RC=0x00;

 Port D initialization
 Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
 State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
RTD=0x00;
RD=0x00;

 Timer/Counter 0 initialization
 Clock source: System Clock
 Clock value: Timer 0 Stopped
CR0=0x00;
NT0=0x00;

 Timer/Counter 1 initialization
 Clock source: System Clock
 Clock value: Timer 1 Stopped
 Mode: Normal top=FFFFh
 OC1A output: Discon.
 OC1B output: Discon.
 Noise Canceler: Off
 Input Capture on Falling Edge
CR1A=0x00;
```

```
CR1B=0x00;
NT1H=0x00;
NT1L=0x00;
R1AH=0x00;
R1AL=0x00;
R1BH=0x00;
R1BL=0x00;

 Timer/Counter 2 initialization
 Clock source: System Clock
 Clock value: Timer 2 Stopped
 Mode: Normal top=FFh
 OC2 output: Disconnected
SR=0x00;
CR2=0x00;
NT2=0x00;
R2=0x00;

 External Interrupt(s) initialization
 INT0: Off
 INT1: Off
UCR=0x00;

 Timer(s)/Counter(s) Interrupt(s) initialization
MSK=0x00;

 Analog Comparator initialization
 Analog Comparator: Off
 Analog Comparator Input Capture by Timer/Counter 1: Off
 Analog Comparator Output: Off
SR=0x80;
COR=0x00;

ile (1)
    {
      LCD_test();

    };
```

```
**********************************************
iis program was produced by the
)deWizardAVR V1.24.0 Standard
itomatic Program Generator
Copyright 1998-2003 HP InfoTech s.r.l.
:tp://www.hpinfotech.ro
mail:office@hpinfotech.ro

'oject :
:rsion :
:te    : 3/19/2009
thor   : HL
mpany :
mments:


ip type           : ATmega8
ogram type        : Application
ock frequency     : 8.000000 MHz
mory model        : Small
ternal SRAM size  : 0
ta Stack size     : 256
**********************************************/

nclude <mega8.h>

 SPI functions
#include <spi.h>

nclude <MARTHEN.C>

 Declare your global variables here

id main(void)

 Declare your local variables here

 Input/Output Ports initialization
 Port B initialization
 Func0=In Func1=In Func2=In Func3=In Func4=Out Func5=In Func6=In Func7=In
 State0=T State1=T State2=T State3=T State4=0 State5=T State6=T State7=T
TB=0x00;
B=0x90;

 Port C initialization
 Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In
 State0=T State1=T State2=T State3=T State4=T State5=T State6=T
TC=0x00;
C=0x00;

 Port D initialization
 Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
 State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
TD=0x00;
D=0xFF;

 Timer/Counter 0 initialization
 Clock source: System Clock
 Clock value: Timer 0 Stopped
R0=0x00;
T0=0x00;

 Timer/Counter 1 initialization
 Clock source: System Clock
 Clock value: Timer 1 Stopped
 Mode: Normal top=FFFFh
 OC1A output: Discon.
```

```
 OC1B output: Discon.
 Noise Canceler: Off
 Input Capture on Falling Edge
CR1A=0x00;
CR1B=0x00;
CNT1H=0x00;
CNT1L=0x00;
CR1AH=0x00;
CR1AL=0x00;
CR1BH=0x00;
CR1BL=0x00;

' Timer/Counter 2 initialization
 Clock source: System Clock
 Clock value: Timer 2 Stopped
 Mode: Normal top=FFh
 OC2 output: Disconnected
SR=0x00;
CR2=0x00;
NT2=0x00;
R2=0x00;

 External Interrupt(s) initialization
 INT0: Off
 INT1: Off
UCR=0x00;

 Timer(s)/Counter(s) Interrupt(s) initialization
MSK=0x00;

 Analog Comparator initialization
 Analog Comparator: Off
 Analog Comparator Input Capture by Timer/Counter 1: Off
 Analog Comparator Output: Off
SR=0x80;
COR=0x00;

 SPI initialization
 SPI Type: Slave
 SPI Clock Rate: 2000.000 kHz
 SPI Clock Phase: Cycle Half
 SPI Clock Polarity: Low
 SPI Data Order: MSB First
CR=0x40;
SR=0x00;

'le (1)
    {
              jalan();

    };
```

```
*****************************************
s program was produced by the
eWizardAVR V1.24.0 Standard
omatic Program Generator
opyright 1998-2003 HP InfoTech s.r.l.
p://www.hpinfotech.ro
ail:office@hpinfotech.ro

oject :
rsion :
te    : 3/19/2009
thor  : HL
mpany :
mments:


ip type           : ATmega8
ogram type        : Application
ock frequency     : 8.000000 MHz
mory model        : Small
ternal SRAM size  : 0
ta Stack size     : 256
*****************************************/

nclude <mega8.h>

 SPI functions
nclude <spi.h>

nclude <MARTHEN.C>
nclude <LCD.C>
 ================================================
      Routine untuk menjalankan LCD 16x2 Character
      type L1602 atau M1632 compatible
      data bus        : 4 bit
wr/rd mode    : write only
author                  : MARTHEN LEUNARD
NIM                                  : 0117025
      Teknik Elektro ITN Malang
      Indonesia
 ================================================ */

nclude <delay.h>

efine    LCDDATA7      PORTD.7
efine    LCDDATA6      PORTD.6
efine    LCDDATA5      PORTD.5
efine    LCDDATA4      PORTB.7
efine    LCD_RS        PORTD.4
efine    LCD_EN        PORTD.3

id LCD_data(char c,char dat)

  LCD_RS = c;
  if ((dat & 0x80)==0x80) LCDDATA7=1; else LCDDATA7=0;
  if ((dat & 0x40)==0x40) LCDDATA6=1; else LCDDATA6=0;
  if ((dat & 0x20)==0x20) LCDDATA5=1; else LCDDATA5=0;
  if ((dat & 0x10)==0x10) LCDDATA4=1; else LCDDATA4=0;
  LCD_EN = 0;          LCD_EN = 1;
  if ((dat & 0x08)==0x08) LCDDATA7=1; else LCDDATA7=0;
  if ((dat & 0x04)==0x04) LCDDATA6=1; else LCDDATA6=0;
  if ((dat & 0x02)==0x02) LCDDATA5=1; else LCDDATA5=0;
  if ((dat & 0x01)==0x01) LCDDATA4=1; else LCDDATA4=0;
  LCD_EN = 0;          LCD_EN = 1;
  delay_ms(2);
```

```
                                        MMC__
  Tulis_LCD(char a,char flash *dat)

  char i = 0;
  LCD_data(0,a); while(dat[i] != 0) { LCD_data(1,dat[i]); i++; }

clude <SPI.C>
*****************************************
hip type            : ATmega8
Clock frequency     : 8000000Hz
Program baca saja mmc
By                                                          : Marthen

*****************************************/
nclude <stdio.h>
nclude <delay.h>

efine Play                  PINC.0
efine Stop                  PINC.1
efine Back                  PINC.2
efine Next                  PINC.3
efine Kdown                 PINC.2
efine Kup                           PINC.3
efine Vdown                 PINC.4
efine Vup                           PINC.5

efine SPIDI   4             // Port B bit 6 (pin7): data in (data from MMC)
efine SPIDO   3             // Port B bit 5 (pin6): data out (data to MMC)
efine SPICLK 5        // Port B bit 7 (pin8): clock
efine SPICS   0             // Port B bit 4 (pin5: chip select for MMC

SPI Control Register */
efine    SPIE          7
efine    SPE           6
efine    DORD          5
efine    MSTR          4
efine    CPOL          3
efine    CPHA          2
efine    SPR1          1
efine    SPR0          0
SPI Status Register */
efine    SPIF          7
efine    WCOL          6

 salah;
igned char count,vol,speed;
r received = 0;
igned int waktu;
igned long i;
atile unsigned char pwm = 0x7F;

Timer 1 overflow interrupt service routine
errupt [TIM1_OVF] void timer1_ovf_isr(void)

      OCR1A = pwm;
      OCR1B = 0xFF-pwm;


d SPIinit(void)

     DDRB &= ~(1 << SPIDI);  // set port B SPI data input to input
     DDRB |= (1 << SPICLK);  // set port B SPI clock to output
     DDRB |= (1 << SPIDO);   // set port B SPI data out to output
     DDRB |= (1 << SPICS);   // set port B SPI chip select to output
     SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0);
     PORTB &= ~(1 << SPICS); // set chip select to low (MMC is selected)
oba ctr
     SPCR=0x50;      // clock rate 2x2000khz
```

```
      SPSR=0x01;


r SPI(char d)
// send character over SPI
      char received = 0;
      unsigned int batas=65000;
      salah=0;
      SPDR = d;
      while (batas--)
      {
            if (SPSR & (1<<SPIF)) goto OK;
      }
salah=1;
      OK:
      if(!salah) received = SPDR;// else received=0x7F;
      return (received);


ar Command1(char befF, unsigned int AdrH, unsigned int AdrL, char befH )
      // sends a command to the MMC
      SPI(0xFF);
      SPI(befF);
      SPI((unsigned char)(AdrH >> 8));
      SPI((unsigned char)AdrH);
      SPI((unsigned char)(AdrL >> 8));
      SPI((unsigned char)AdrL);
      SPI(befH);
      SPI(0xFF);
      return SPI(0xFF);       // return the last received character


ır MMC_Init(void)
/ init SPI
      char i;
      PORTB |= (1 << SPICS); // disable MMC
      // start MMC in SPI mode
      for(i=0; i < 10; i++) SPI(0xFF); // send 10*8=80 clock pulses
      PORTB &= ~(1 << SPICS); // enable MMC

      if (Command1(0x40,0,0,0x95) != 1) goto mmcerror; // reset MMC

  // if there is no MMC, prg. loops here
      if (Command1(0x41,0,0,0xFF) !=0) goto st;
      return (1);
error:
      return (0);


d sendmmc(void)
/ Read 512 byte from MMC, send to PORT D
igned int j, delay, hitung, temp;
Ulangi:
//LED = 0;
waktu = 0;
      if (MMC_Init()==0) goto Ulangi;
      i = 0;
      // baca 512 pertama utk hilangkan TOC
      Command1(0x52,0x1,0x200,0xFF);
      while(i<450)
      {
            while(SPI(0xFF) != 0xFE){ if (salah) goto Ulangi; }
            for(j=0; j < 512; j++) {SPDR = (0xFF); while(!(SPSR &
<SPIF)));}
            i++;
      }
      //LED = 1;
```

```
      while(1)          // always loop here !
      {                           // wait for 0xFE - start of any transmission
            while(SPI(0xFF) != 0xFE) { if (salah) goto Ulangi; } // ATT:
ecast (char)0xFE is a must!

            for(j=0; j < 512; j++)
            {
                  SPDR = (0xFF);  temp = vol >> 4; if (temp==0) temp=1;
                  while(!(SPSR & (1<<SPIF))); if (temp < 14) pwm =
DR/temp;
                  delay=speed;
                  while(delay--);
            }
            i++;
            if(!Kup)   speed++; else if(!Kdown) speed--;
            if (speed<20) speed=20; else if (speed>90) speed=90;
            if (!Vup) vol++; else if(!Vdown) vol--;
            if (vol==0) vol=1; else if (vol==255) vol=254;
            if (hitung>38)
            {
                  waktu++; hitung=0;
            } else hitung++;
            if (i>195000) goto Ulangi; // Baca hingga 100 Mbyte
      }


Timer 0 overflow interrupt service routine
errupt [TIM0_OVF] void timer0_ovf_isr(void)

      switch(count)
      {
       //     case 1: com4=1; PORTC=_7seg[waktu/1000]; com1=0; count++; break;
            //case 2: com1=1; PORTC=_7seg[waktu%1000/100]; com2=0; count++;
ak;
            //case 3: com2=1; PORTC=_7seg[waktu%100/10]; com3=0; count++;
ak;
            //case 4: com3=1; PORTC=_7seg[waktu%10]; com4=0; count=1; break;
            default: count=1;
      }


d jalan()

      DDRB.5=1;
      // ==== initialisasi LCD ====
      delay_ms(2000);
      LCD_data(0,0x3F); delay_ms(100); LCD_data(0,0x3F);delay_ms(100);
      LCD_data(0,0x3F); LCD_data(0,0x08); LCD_data(0,0x01); LCD_data(0,0x06);
      LCD_data(0,0x2F); LCD_data(0,0x0E); LCD_data(0,0x06);
      // === MENULIS JUDUL ALAT ====
      Tulis_LCD(0x80,"  TUGAS  AKHIR  ");
      Tulis_LCD(0xC0,"MMC AUDIO PLAYER");
      delay_ms(5000);
      Tulis_LCD(0x80,"MARTHEN  LEUNARD");
      Tulis_LCD(0xC0,"  NIM : 0117025 ");
elay_ms(5000);
/ === INITIALISASI MMC VIA SPI ===
      waktu=0;
      speed=43;
      vol=1;
      SPIinit();
      sendmmc();


Declare your global variables here

d main(void)
```

```
eclare your local variables here

nput/Output Ports initialization
Port B initialization
Func0=In Func1=In Func2=In Func3=In Func4=Out Func5=In Func6=In Func7=In
State0=T State1=T State2=T State3=T State4=0 State5=T State6=T State7=T
TB=0x00;
RB=0x90;

Port C initialization
Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In
State0=T State1=T State2=T State3=T State4=T State5=T State6=T
RTC=0x00;
RC=0x00;

Port D initialization
Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
RTD=0x00;
RD=0xFF;

Timer/Counter 0 initialization
Clock source: System Clock
Clock value: Timer 0 Stopped
CR0=0x00;
NT0=0x00;

Timer/Counter 1 initialization
Clock source: System Clock
Clock value: Timer 1 Stopped
Mode: Normal top=FFFFh
OC1A output: Discon.
OC1B output: Discon.
Noise Canceler: Off
Input Capture on Falling Edge
CR1A=0x00;
CR1B=0x00;
NT1H=0x00;
NT1L=0x00;
1AH=0x00;
1AL=0x00;
1BH=0x00;
1BL=0x00;

Timer/Counter 2 initialization
Clock source: System Clock
Clock value: Timer 2 Stopped
Mode: Normal top=FFh
OC2 output: Disconnected
R=0x00;
R2=0x00;
T2=0x00;
2=0x00;

External Interrupt(s) initialization
INT0: Off
INT1: Off
CR=0x00;

Timer(s)/Counter(s) Interrupt(s) initialization
SK=0x00;

Analog Comparator initialization
Analog Comparator: Off
Analog Comparator Input Capture by Timer/Counter 1: Off
Analog Comparator Output: Off
```

```
   0x80;
  :=0x00;

  PI initialization
 ;PI Type: Slave
 ;PI Clock Rate: 2000.000 kHz
 SPI Clock Phase: Cycle Half
 SPI Clock Polarity: Low
 SPI Data Order: MSB First
.R=0x40;
;R=0x00;

ile (1)
    {
            jalan();

    };
```

```
                                                LCD__
*****************************************
s program was produced by the
lewizardAVR V1.24.0 Standard
:omatic Program Generator
:opyright 1998-2003 HP InfoTech s.r.l.
:p://www.hpinfotech.ro
iail:office@hpinfotech.ro

)ject :
'sion :
te    : 3/19/2009
thor  : HL
npany :
mments:


ip type           : ATmega8
ogram type        : Application
ock frequency     : 8.000000 MHz
mory model        : Small
ternal SRAM size  : 0
ta Stack size     : 256
*****************************************/

nclude <mega8.h>
nclude <LCD_mega8.c>
 ================================================
      Routine untuk menjalankan LCD 16x2 Character
      type L1602 atau M1632 compatible
      data bus        : 4 bit
vr/rd mode    : write only
uthor                 : MARTHEN LEUNARD
NIM                                   : 0117025
      Teknik Elektro ITN Malang
      Indonesia
 ================================================ */

iclude <delay.h>

efine    LCDDATA7      PORTD.7
efine    LCDDATA6      PORTD.6
efine    LCDDATA5      PORTD.5
efine    LCDDATA4      PORTB.7
efine    LCD_RS        PORTD.4
efine    LCD_EN        PORTD.3

id LCD_data(char c,char dat)

  LCD_RS = c;
  if ((dat & 0x80)==0x80) LCDDATA7=1; else LCDDATA7=0;
  if ((dat & 0x40)==0x40) LCDDATA6=1; else LCDDATA6=0;
  if ((dat & 0x20)==0x20) LCDDATA5=1; else LCDDATA5=0;
  if ((dat & 0x10)==0x10) LCDDATA4=1; else LCDDATA4=0;
  LCD_EN = 0;          LCD_EN = 1;
  if ((dat & 0x08)==0x08) LCDDATA7=1; else LCDDATA7=0;
  if ((dat & 0x04)==0x04) LCDDATA6=1; else LCDDATA6=0;
  if ((dat & 0x02)==0x02) LCDDATA5=1; else LCDDATA5=0;
  if ((dat & 0x01)==0x01) LCDDATA4=1; else LCDDATA4=0;
  LCD_EN = 0;          LCD_EN = 1;
  delay_ms(2);


d Tulis_LCD(char a,char flash *dat)

  char i = 0;
  LCD_data(0,a); while(dat[i] != 0) { LCD_data(1,dat[i]); i++; }
```

```
d LCD_test()

      DDRD=0xFF;
      DDRB.7=1;
initialisasi LCD
      delay_ms(2000);
      LCD_data(0,0x3F); delay_ms(100); LCD_data(0,0x3F);delay_ms(100);
      LCD_data(0,0x3F); LCD_data(0,0x08); LCD_data(0,0x01); LCD_data(0,0x06);
      LCD_data(0,0x2F); LCD_data(0,0x0E); LCD_data(0,0x06);


      Tulis_LCD(0x80,"  EKO BAGUS S.  ");
      Tulis_LCD(0xC0,"NIM : 01.17.013 ");

      while(1);


 Declare your global variables here

id main(void)

 Declare your local variables here

 Input/Output Ports initialization
 Port B initialization
 Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
 State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
RTB=0x00;
RB=0x00;

 Port C initialization
 Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In
 State0=T State1=T State2=T State3=T State4=T State5=T State6=T
RTC=0x00;
RC=0x00;

 Port D initialization
 Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
 State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
RTD=0x00;
RD=0x00;

 Timer/Counter 0 initialization
 Clock source: System Clock
 Clock value: Timer 0 Stopped
:R0=0x00;
VT0=0x00;

 Timer/Counter 1 initialization
 Clock source: System Clock
 Clock value: Timer 1 Stopped
 Mode: Normal top=FFFFh
 OC1A output: Discon.
 OC1B output: Discon.
 Noise Canceler: Off
 Input Capture on Falling Edge
:R1A=0x00;
:R1B=0x00;
IT1H=0x00;
IT1L=0x00;
:1AH=0x00;
:1AL=0x00;
.1BH=0x00;
.1BL=0x00;

 Timer/Counter 2 initialization
```

```
 Clock source: System Clock
 Clock value: Timer 2 Stopped
 Mode: Normal top=FFh
 OC2 output: Disconnected
;R=0x00;
:R2=0x00;
NT2=0x00;
?2=0x00;

 External Interrupt(s) initialization
 INT0: Off
 INT1: Off
JCR=0x00;

 Timer(s)/Counter(s) Interrupt(s) initialization
MSK=0x00;

 Analog Comparator initialization
 Analog Comparator: Off
 Analog Comparator Input Capture by Timer/Counter 1: Off
 Analog Comparator Output: Off
SR=0x80;
IOR=0x00;

ile (1)
    {
      LCD_test();

    };
```

```
*****************************************
s program was produced by the
lewizardAVR V1.24.0 Standard
:omatic Program Generator
:opyright 1998-2003 HP InfoTech s.r.l.
:p://www.hpinfotech.ro
iail:office@hpinfotech.ro

)ject :
rsion :
te    : 3/19/2009
thor  : HL
mpany :
mments:


ip type           : ATmega8
ogram type        : Application
ock frequency     : 8.000000 MHz
mory model        : Small
ternal SRAM size  : 0
ta Stack size     : 256
*****************************************/

nclude <mega8.h>

 Declare your global variables here

id main(void)

 Declare your local variables here

 Input/Output Ports initialization
 Port B initialization
 Func0=In Func1=Out Func2=Out Func3=In Func4=In Func5=In Func6=In Func7=In
 State0=T State1=0 State2=0 State3=T State4=T State5=T State6=T State7=T
RTB=0x00;
RB=0x06;

 Port C initialization
 Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In
 State0=T State1=T State2=T State3=T State4=T State5=T State6=T
RTC=0x00;
RC=0x00;

 Port D initialization
 Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
 State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
RTD=0x00;
RD=0x00;

 Timer/Counter 0 initialization
 Clock source: System Clock
 Clock value: Timer 0 Stopped
CR0=0x00;
NT0=0x00;

 Timer/Counter 1 initialization
 Clock source: System Clock
 Clock value: 8000.000 kHz
 Mode: Fast PWM top=OCR1A
 OC1A output: Non-Inv.
 OC1B output: Inverted
 Noise Canceler: Off
 Input Capture on Falling Edge
CR1A=0xB3;
CR1B=0x19;
```

```
 T1H=0x00;
 T1L=0x00;
 1AH=0x00;
 1AL=0x00;
 1BH=0x00;
 1BL=0x00;

 Timer/Counter 2 initialization
 Clock source: System Clock
 Clock value: Timer 2 Stopped
 Mode: Normal top=FFh
 OC2 output: Disconnected
SR=0x00;
CR2=0x00;
NT2=0x00;
R2=0x00;

 External Interrupt(s) initialization
 INT0: Off
 INT1: Off
UCR=0x00;

 Timer(s)/Counter(s) Interrupt(s) initialization
MSK=0x00;

 Analog Comparator initialization
 Analog Comparator: Off
 Analog Comparator Input Capture by Timer/Counter 1: Off
 Analog Comparator Output: Off
SR=0x80;
TOR=0x00;

ile (1)
    {
    // Place your code here

    };
```

```
/*****************************************
* Chip type           : ATmega8
* Clock frequency     : 8000000Hz
* Program baca saja mmc
* By
: Marthen
*****************************************/
#include <stdio.h>
#include <delay.h>

#define Play                    PINC.0
#define Stop                    PINC.1
#define Back                    PINC.2
#define Next                    PINC.3
#define Kdown                   PINC.2
#define Kup                             PINC.3
#define Vdown                   PINC.4
#define Vup                             PINC.5

#define SPIDI   4               // Port B bit 6 (pin7): data in (data
from MMC)
#define SPIDO   3               // Port B bit 5 (pin6): data out
(data to MMC)
#define SPICLK 5        // Port B bit 7 (pin8): clock
#define SPICS   0               // Port B bit 4 (pin5: chip select
for MMC

/* SPI Control Register */
#define     SPIE            7
#define     SPE             6
#define     DORD            5
#define     MSTR            4
#define     CPOL            3
#define     CPHA            2
#define     SPR1            1
#define     SPR0            0
/* SPI Status Register */
#define     SPIF            7
#define     WCOL            6

bit salah;
unsigned char count,vol,speed;
char received = 0;
unsigned int waktu;
unsigned long i;
volatile unsigned char pwm = 0x7F;

// Timer 1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
        OCR1A = pwm;
        OCR1B = 0xFF-pwm;
}

void SPIinit(void)
{
        DDRB &= ~(1 << SPIDI);  // set port B SPI data input to input
        DDRB |= (1 << SPICLK);  // set port B SPI clock to output
        DDRB |= (1 << SPIDO);   // set port B SPI data out to output
        DDRB |= (1 << SPICS);   // set port B SPI chip select to
output
//      SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0);
        PORTB &= ~(1 << SPICS); // set chip select to low (MMC is
selected)
//coba ctr
        SPCR=0x50;      // clock rate 2x2000khz
        SPSR=0x01;
```

```
}

char SPI(char d)
{  // send character over SPI
//        char received = 0;
          unsigned int batas=65000;
          salah=0;
          SPDR = d;
          while (batas--)
          {
                  if (SPSR & (1<<SPIF)) goto OK;
          }
    salah=1;
          OK:
          if(!salah) received = SPDR;// else received=0x7F;
          return (received);
}

char Command1(char befF, unsigned int AdrH, unsigned int AdrL, char
befH )
{         // sends a command to the MMC
          SPI(0xFF);
          SPI(befF);
          SPI((unsigned char)(AdrH >> 8));
          SPI((unsigned char)AdrH);
          SPI((unsigned char)(AdrL >> 8));
          SPI((unsigned char)AdrL);
          SPI(befH);
          SPI(0xFF);
          return SPI(0xFF);       // return the last received character
}

char MMC_Init(void)
{ // init SPI
          char i;
          PORTB |= (1 << SPICS); // disable MMC
          // start MMC in SPI mode
          for(i=0; i < 10; i++) SPI(0xFF); // send 10*8=80 clock pulses
          PORTB &= ~(1 << SPICS); // enable MMC

          if (Command1(0x40,0,0,0x95) != 1) goto mmcerror; // reset MMC

st: // if there is no MMC, prg. loops here
          if (Command1(0x41,0,0,0xFF) !=0) goto st;
          return (1);
mmcerror:
          return (0);
}

void sendmmc(void)
{ // Read 512 byte from MMC, send to PORT D
unsigned int j, delay, hitung, temp;
    Ulangi:
    //LED = 0;
    waktu = 0;
          if (MMC_Init()==0) goto Ulangi;
          i = 0;
          // baca 512 pertama utk hilangkan TOC
          Command1(0x52,0x1,0x200,0xFF);
          while(i<450)
          {
                  while(SPI(0xFF) != 0xFE){ if (salah) goto Ulangi; }
                  for(j=0; j < 512; j++) {SPDR = (0xFF); while(!(SPSR &
(1<<SPIF)));}
                  i++;
          }
          //LED = 1;
```

```
        while(1)        // always loop here !
        {                       // wait for 0xFE - start of any
transmission
                while(SPI(0xFF) != 0xFE) { if (salah) goto Ulangi; }
// ATT: typecast (char)0xFE is a must!

                for(j=0; j < 512; j++)
                {
                        SPDR = (0xFF);  temp = vol >> 4; if (temp==0)
temp=1;
                        while(!(SPSR & (1<<SPIF))); if (temp < 14)
pwm = SPDR/temp;
                        delay=speed;
                        while(delay--);
                }
                i++;
                if(!Kup)   speed++; else if(!Kdown) speed--;
                if (speed<20) speed=20; else if (speed>90) speed=90;
                if (!Vup) vol++; else if(!Vdown) vol--;
                if (vol==0) vol=1; else if (vol==255) vol=254;
                if (hitung>38)
                {
                        waktu++; hitung=0;
                } else hitung++;
                if (i>195000) goto Ulangi; // Baca hingga 100 Mbyte
        }
}

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
        switch(count)
        {
         //     case 1: com4=1; PORTC=_7seg[waktu/1000]; com1=0;
count++; break;
                //case 2: com1=1; PORTC=_7seg[waktu%1000/100];
com2=0; count++; break;
                //case 3: com2=1; PORTC=_7seg[waktu%100/10]; com3=0;
count++; break;
                //case 4: com3=1; PORTC=_7seg[waktu%10]; com4=0;
count=1; break;
                default: count=1;
        }
}
```

```
#include <LCD.C>
#include <SPI.C>

void jalan()
{
        DDRB.5=1;
        // ==== initialisasi LCD ====
        delay_ms(2000);
        LCD_data(0,0x3F); delay_ms(100);
LCD_data(0,0x3F);delay_ms(100);
        LCD_data(0,0x3F); LCD_data(0,0x08); LCD_data(0,0x01);
LCD_data(0,0x06);
        LCD_data(0,0x2F); LCD_data(0,0x0E); LCD_data(0,0x06);
        // === MENULIS JUDUL ALAT ====
        Tulis_LCD(0x80,"  TUGAS  AKHIR  ");
        Tulis_LCD(0xC0,"MMC AUDIO PLAYER");
        delay_ms(500);
        Tulis_LCD(0x80,"MARTHEN  LEUNARD");
        Tulis_LCD(0xC0,"  NIM : 0117025 ");
  delay_ms(500);
  // === INITIALISASI MMC VIA SPI ===
        waktu=0;
        speed=43;
        vol=1;
        SPIinit();
        sendmmc();
}
```