Makalah Seminar Tugas Akhir

EMBEDDED C PADA MIKROKONTROLER AVR AT90S8515

Fajar Mahadmadi
e-mail: jafar98@plasa.com
L2F 098 613
Jurusan Teknik Elektro Fakultas Teknik Universitas Diponegoro

Abstrak

Atmel merupakan perusahaan pembuat chip yang terkenal dalam teknologi pembuatan Flash memory dan EEPROM. Saat Atmel memasukkan Flash PEROM ke mikrokontroler seri AT89Cxx sebagai anggota baru keluarga mikrokontroler MCS51, keluarga mikrokontroler MCS51 bagaikan mendapat darah segar dan bangkit dari kerentaannya. Meskipun umurnya lebih dari 25 tahun kini MCS51 makin banyak digunakan oleh para pemula. Menyusul keberhasilannya tersebut, Atmel merancang sendiri mikrokontroler baru yang dinamakan sebagai keluarga mikrokontroler AVR.

AT90S8515 adalah salah satu jenis AVR tipe klasik yang mempunyai unjuk kerja hampir sama dengan AT89C52. Keunggulan dari AT90S8515 ini adalah kemampuan ISP (In System Programming) mempunyai internal EEPROM sebesar 512 bytes, 512 RAM, PWM, kecepatan clock sampai dengan 8 MHZ dengan frekuensi kerja sama dengan frekuensi kristal osilator, interupsi 11 jalan, analog komparator, serta 32 register serbaguna yang langsung terhubung dengan ALU sehingga menyerupai akumulator.

Bahasa pemrograman C merupakan bahasa level atas yang paling banyak dipakai untuk mirokontroler. Sebagai mikrokontroler modern, AVR dirancang dengan mempertimbangkan sifat-sifat pengkodean bahasa C sehingga program yang dihasilkan kompiler bisa sekecil mungkin dan secepat mungkin.

I. PENDAHULUAN

1.1 Latar Belakang Masalah.

AT90S8515 adalah salah satu jenis AVR tipe klasik yang mempunyai banyak fitur. Fitur tersebut antara lain adalah kemampuan ISP (*In System Programming*), Flash memory sebesar 4 *Kwords* mempunyai EEPROM internal sebesar 512 bytes, RAM sebesar 512 bytes, kecepatan clock sampai dengan 8 MHz dengan frekuensi kerja sama dengan frekuensi kristal osilator, 11 interupsi, analog komparator, 32 register yang langsung terhubung dengan ALU sehingga menyerupai akumulator, serta dirancang dengan mempertimbangkan sifat-sifat pengkodean bahasa C.

Pemilihan AT90S8515 dalam Tugas Akhir (TA) ini adalah karena kemiripan sifatnya dengan AT89C52 dalam hal perangkat keras, kemudahannya diperoleh di pasaran (Surabaya), serta harganya yang murah, sehingga terjangkau oleh mahasiswa yang ingin bereksperimen. Dengan memahami satu jenis IC ini diharapkan akan membuka gerbang bagi pemahaman tipe AVR yang lain.

Dalam umurnya yang baru 7 tahun, AVR masih tergolong baru dan belum banyak digunakan di Indonesia. Dengan pertimbangan harga yang murah, ketersediaan di pasaran, serta banyaknya fitur yang ada pada AVR, maka diperkirakan mikrokontroler ini akan banyak dipergunakan pada masa yang akan datang.

1.1 Maksud dan Tujuan

Tujuan dari tugas akhir ini adalah:

Tujuan dan manfaat dari pengambilan judul "Embedded C pada Mikrokontroler AVR AT90S8515" memberikan gambaran mengenai bagaimana konfigurasi perangkat keras dan bagaimana pemrograman chip dengan menggunakan bahasa C sehingga membuka gerbang pemahaman mengenai mikrokontroler AVR dan menambah khasanah penggunaan mikrokontroler.

1.2 Pembatasan Masalah

Pembahasan masalah dalam tugas akhir ini dibatasi pada hal-hal pokok yang sering digunakan dalam penggunaan mikrokontroler baik mengenai perangkat lunak maupun perangkat keras

II. PERANGKAT KERAS

2.1 Keluarga AVR

AVR adalah sebuah keluarga mikrokontroler yang mempunyai anggota dengan tipe tertentu seperti AT90S1200, ATTinty11, dan ATMega8L. Secara garis besar mempunyai mempunyai sub keluarga yaitu Tiny yaitu dengan fitur sederhana, Klasik dengan fitur menengah, dan Mega dengan fitur lengkap. Instruksi yang dipergunakan pada tiap tipe adalah sama, hanya jumlah instruksi pada masing-masing tipe berbeda

2.2 Mikrokontroler AT90S8515

Keistimewaan AT90S8515 adalah sebagai berikut:

a. Arsitektur RISC
b. 118 instruksi sebagian besar satu siklus instruksi.
c. 32x8 register kerja serbaguna
d. 8 MIPS (*Mega Instructions per Second*) pada 8 MHZ.
e. 8 Kbytes *In-System Programmable Flash* (1000 siklus hapus/tulis)
f. 512 bytes RAM
g. 512 bytes *In-System Programmable* EEPROM (100.000 siklus hapus/tulis)
h. Pemrograman terkunci untuk program *Flash* dan keamanan data pada EEPROM.
i. Satu 8-bit *timer/counter* dengan Prescaler terpisah.
j. Satu 16-bit *timer/counter* dengan Prescaler terpisah yang dapat digunakan untuk mode *Compare*, Mode *Capture* dan dual 8-,9-,atau 10 bit PWM
k. Analog *comparator* dalam chip.
l. Pewaktu *Watchdog* terprogram dengan Osilator dalam chip.
m. Serial UART terprogram.
n. Antarmuka serial SPI master/*slave*
o. Mode power *down* dan catu rendah senggang.

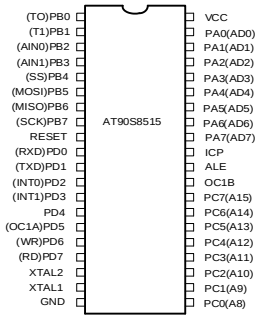
p. Sumber interupsi internal dan eksternal.
q. 32 jalur I/O terprogram.

Konsep RISC (*Reduced Instruction Set Computer*) muncul setelah konsep sebelumnya yaitu CISC (*Complex Instruction Set Computer*).

Sistem CISC terkenal dengan banyaknya *instruction set*, mode pengalamatan yang banyak, format instruksi dan ukuran yang banyak, instruksi yang berbeda dieksekusi dalam jumlah siklus yang berbeda.

Sistem dengan RISC pada AVR mengurangi hampir semuanya, yaitu meliputi jumlah instruksi, mode pengalamatan, dan format. Hampir semua instruksi mempunyai ukuran yang sama yaitu 16 bit. Sebagian besar instruksi dieksekusi dalam satu siklus CPU.

2.3 Susunan Kaki Mikrokontroler AT90S8515



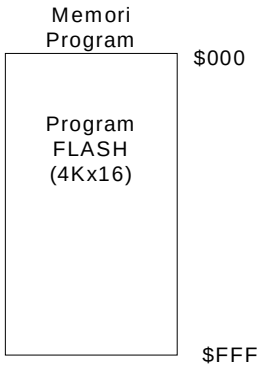
Gambar 2.1 Susunan kaki mikrokontroler AT90S8515

2.4 Organisasi Memori

AVR menggunakan arsitektur Harvard sehingga memisahkan memori serta bus data dengan program. Program ditempatkan di ISP (*In-System Programmable*/terprogram dalam sistem) *Flash Memory*. Memori data terdiri dari 32 x8 bit register serbaguna, 64x8 bit register I/O, 512x8 bit internal RAM, dan 64 Kx8 bit RAM eksternal.

2.4.1 Memori Program

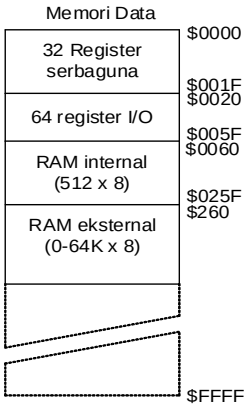
AT90S8515 mempunyai kapasitas memori program sebesar 8 K*byte*s. Karena semua format instruksi berupa kata (*word*) 16-32 bit maka format memori program ini adalah 4Kx16 bit. Memori *Flash* ini dirancang untuk dapat di hapus dan tulis sebanyak seribu kali. Program *Counter* (PC)-nya sepanjang 12 bit sehingga mampu mengakses hingga 4096 lokasi memori.



Gambar 2.2 Memori Program

2.4.2 Memori Data

Memori data yang terdiri dari 32 register serbaguna, 64 register I/O dan RAM perlihatkan pada Gambar 2.3



Gambar 2.3 Memori data

2.4.2.1 32 Register Serbaguna

32 *byte*s *general purpose working register* atau register serbaguna mendukung adanya konsep register akses cepat. Ini berarti waktu akses dari register adalah satu detak atau satu operasi ALU (*Arithmetic Logic Unit*).

2.4.2.2 Ruang *Memory* I/O

Ruang memori I/O berisi 64 alamat untuk fungsi periferal CPU seperti register kontrol, *timer/counter*, dan fungsi I/O yang lain.

.4.2.3 RAM Internal dan Eksternal

Kapasitas dari RAM internal adalah sebesar 512 *byte*s. Ini menempati ruang alamat setelah ruang 64 register I/O.

Jika RAM eksternal digunakan, ruang alamat yang dipakai akan mengikuti ruang alamat RAM internal sampai dengan maksimum 64K, tergantung ukuran RAM eksternal.

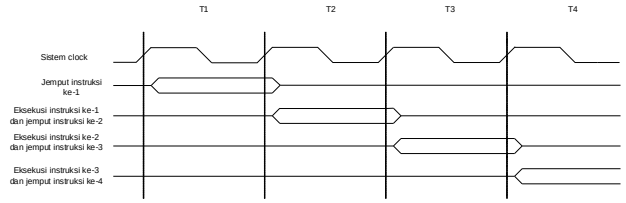
Ketika alamat yang digunakan untuk mengakses melebihi alamat dari ruang memori internal RAM, maka RAM eksternal akan terakses dengan instruksi yang sama dengan instruksi pada RAM internal. Jalur pengontrolan RAM untuk fungsi baca dan tulis masing-masing menggunakan kaki $\overline{RD}$ dan $\overline{WR}$. Ketika mengakses RAM internal, maka fungsi $\overline{RD}$ dan $\overline{WR}$ tidak diaktifkan dan kaki tersebut berfungsi seperti biasa. Operasi RAM eksternal di-*enable* oleh seting bit SRE di register MCUCR.

2.5 Pewaktuan CPU

Mikrokontroler AT90S8515 memiliki osilator internal (*on chip* osilator) yang dapat digunakan sebagai sumber detak bagi CPU.

2.6 Siklus Mesin

Gambar 2.10 menunjukkan siklus jemput dan siklus eksekusi dari sebuah instruksi paralel yang menggunakan prinsip arsitektur Harvard dan konsep register akses cepat. Konsep dasar ini adalah untuk memperoleh 1 MIPS (*Mega Instruction Per Second*).



Gambar 2.4 Penjemputan dan eksekusi instruksi paralel

2.7 Interupsi

Mikrokontroler AVR AT90S8515 memiliki 12 saluran interupsi.

Tabel 2.3 Vektor *Reset* dan Interupsi

No vekt	Alamat program	Sumber	Definisi interupsi
1	$000	*RESET*	*Reset* eksternal, *reset power on*, *reset watchdog*
2	$001	INT0	Interupsi eksternal 0
3	$002	INT1	Interupsi eksternal 1
4	$003	*TIMER*1CAPT	*Timer/Counter*1 *Capture Event*
5	$004	*TIMER*1COMPA	*Timer/Counter*1 *Compare Match*A
6	$005	*TIMER*1COMPB	*Timer/Counter*1 *Compare Match*B
7	$006	*TIMER*1OVF	*Timer/Counter*1 *Overflow*
8	$007	*TIMER*0OVF	*Timer/Counter*0 *Overflow*
9	$008	SPI, STC	*Serial* Transfer *Complete*
10	$009	UART, RX	UART, Rx *Complete*
11	$00A	UART, UDRE	UART, Data Register *Empty*
12	$00B	UART, TX	UART, Tx *Complete*
13	$00C	ANA_COMP	*Analog comparator*

2.7.1 Sumber Interupsi *Reset*

1. *Power on Reset*. Mikrokontroler di-*reset* ketika tegangan catu daya berada di bawah ambang dari ambang *Power on Reset*.
2. Eksternal *Reset*. Mikrokontroler di*reset* ketika kondisi rendah diberikan pada kaki *RESET* lebih dari 50 ns.
3. *Reset Watchdog*. Mikrokontroler di*reset* ketika periode waktu *watchdog* berakhir dan *watchdog* diaktifkan

2.7.2 Sumber Interupsi INT0 dan INT1

Sumber interupsi INT0 dan INT1 adalah sumber interupsi eksternal yang terjadi ketika picuan interupsi pada kaki INT0 (12) atau INT1(13). Pengontrolan dari interupsi ini adalah oleh register GIMSK dan MCUCR. Register bendera yang terpengaruh oleh interupsi ini adalah register GIFR.

2.7.3 Sumber Interupsi *Timer*1 *Capture, Timer*1 CompA, *Timer*1 CompB, *Timer*1 OVF, *Timer*0 OVF.

Sumber interupsi *Timer*1 *Capture* akan menghasilkan suatu tangkapan (*capture*) dari isi *Timer/Counter*1 untuk diisikan dalam *Input Capture* Register (ICR1) yang dipicu oleh kejadian pada kaki ICP (kaki 31). Pengaturan dari ICP ini ditentukan oleh register *Timer/Counter*1 *Control* Register (TCCR1B). *Analog* komparator juga dapat digunakan untuk memicu *input capture*. Bit peng-*enable Input Capture* ini adalah bit 3 (TICIE1) pada register TIMSK.

Sumber interupsi dari *Timer*1 CompA (*compare* A) akan menghasilkan interupsi ketika *Timer/Counter*1 menghitung sampai pada nilai yang sama dengan nilai pada *Timer/counter*1 *Output Compare* register (OCR1AH dan OCR1AL). Bit peng-*enable Timer*1 CompA ini adalah bit OCE1A pada register TIMSK.

Sumber interupsi dari *Timer*1 CompB (*compare* B) akan menghasilkan interupsi ketika *Timer/Counter*1 menghitung sampai pada nilai yang sama dengan nilai pada *Timer/Counter*1 *Output Compare* Register (OCR1BH dan OCR1BL). Bit peng-*enable Timer*1 CompB ini adalah bit OCE1B pada register TIMSK.

Sumber interupsi *Timer*1 OVF (*overflow*) akan menghasilkan interupsi ketika terjadi *overflow* pada *Timer*1. Bit peng-*enable*-nya adalah bit TOIE1 pada register TIMSK.

Sumber interupsi *Timer*0 OVF (*overflow*) akan menghasilkan interupsi ketika terjadi *overflow* pada *Timer*0. Bit peng-*enable*-nya adalah bit TOIE0 pada register TIMSK.

2.7.4 Sumber Interupsi SPI (*Serial Peripheral Interface*), STC (*Serial* Transfer *Complete*)

Sumber interupsi STC (*Serial* Transfer *Complete*) akan menghasilkan interupsi ketika data yang dikirim secara *serial* menggunakan SPI telah selesai dikirim. *Flag* yang menunjukkan STC ini adalah *flag* SPIF (SPI *Interrupt Flag*) pada register status SPSR (SPI Status Register). Peng-*enable* fungsi interupsi ini adalah bit SPIE (SPI *Interrupt Enable*) pada register SPCR (SPI *Control* Register).

2.7.5 Sumber Interupsi UART (*Universal Asynchronous Receiver Transmintter*)

Sumber interupsi yang disediakan oleh UART ini meliputi Rx *Complete*, Data Register *Empty*, dan Tx *Complete*.

Sumber interupsi Rx (*Receive*) *Complete* akan menghasilkan interupsi pada vektor interupsi dengan alamat vektor $009 yang disebabkan karakter yang diterima dikirim dari *Receiver Shift* Register ke UDR.

Sumber interupsi UDRE (UART Data Register *Empty*) akan menghasilkan interupsi pada vektor interupsi dengan alamat vektor $00A yang disebabkan karakter yang ada di UDR (UART Data Register) dikirim ke *Transmit Shift* Register.

Sumber interupsi TX (*Transmit*) *Complete* akan menghasilkan interupsi pada vektor interupsi dengan alamat vektor $00B yang disebabkan seluruh karakter (termasuk bit *stop*) dalam *Transmit Shift* Register telah digeser keluar dan tak ada data baru yang dituliskan pada UDR.

2.7.6 Sumber Interupsi *Analog comparator*

Sumber *Analog comparator* (Pembanding *Analog*) akan menghasilkan interupsi pada vektor interupsi dengan alamat vektor $00B yang disebabkan kejadian pada keluaran komparator memicu interupsi.

2.8 Timer/counter

AT90S8515 mempunyai dua buah *timer/counter*, masing-masing adalah *Timer/Counter*0 (8 bit) dan *Timer/Counter*1 (16 bit). Sebagai *timer*, kedua *timer/counter* dapat mengambil detak dari sumber detak internal, dan sebagai *counter*, dengan

mengambil detak eksternal sebagai sumber yang memicu penghitungan (*counting*).

2.8.1 *Timer/Counter*0

Register TIMSK berfungsi meng-*enable* interupsi *overflow* dari *Timer/Counter*0. *Overflow* berarti *Timer/Counter*0 selesai menghitung yaitu pada hitungan 255.

Waktu hitung dari *Timer/Counter*0 sebelum terjadi *overflow* adalah sesuai dengan rumus:

$$T = (256 - n).\frac{1}{f_{CK}}.F_b \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.1)$$

dimana : T = waktu hitung
n = nilai awal pada TCNT0
f_{CK} =frekuensi detak osilator kristal
F_b = faktor bagi pada penggunaan *prescaler*

2.8.2 *Timer/Counter*1

*Timer/Counter*1 ini adalah 16 bit, dengan fungsi yang lebih kompleks dari *Timer/Counter*0. Interupsi yang dapat dihasilkan dari *Timer/Counter*1 ini adalah T/C1 *Overflow*, T/C1 *Compare Match*A, T/C *Compare Match*B dan T/C *Input Capture*.

2.8.2.1 *Timer/Counter*1 sebagai sumber interupsi

Register peng-*enable* interupsi *Timer/Counter*1 adalah TIMSK. *Flag* interupsi ada pada register TIFR.

2.8.2.1.1 Input Capture

ACIC (*Analog comparator Input Capture Enable*) adalah salah satu bit dari register ACSR (*Analog comparator Control and Status Register*). Bit ini berfungsi untuk meng-enable fungsi Input *Capture* pada *Timer/Counter1* untuk dipicu oleh Analog *comparator*. ACO (Analog *comparator* Output) adalah salah satu bit dari ACSR yang terhubung langsung dengan keluaran analog *comparator*.

2.8.2.1.2 Compare Match A/B

Compare Match A/B merupakan dua fungsi interupsi masing-masing adalah interupsi *Compare Match* A dan *Compare Match* B. Nilai dari isi *Timer*/*Counter*1 akan selalu dibandingkan dengan isi suatu register yaitu register OCR1(A/B) (*Timer*/*Counter*1 Output *Compare* Register A/B). Interupsi akan terjadi ketika *Timer*/*Counter* menghitung sampai pada nilai yang sama dengan nilai pada OCR1(A/B). Vektor interupsi dari *compare match* A adalah alamat $004, sedangkan untuk *Compare Match* B pada alamat $005.

2.8.2.1.3 Overflow

Yang dimaksud dengan *overflow* adalah *Timer*/*Counter*1 selesai menghitung pada nilai terbesar yaitu $65535 = 2^{16}$. Bit peng-enable interupsi ini adalah bit TOIE1 yang terdapat pada register TIMSK. Lama waktu hitung sebelum terjadi *overflow* dapat diperoleh dengan rumus:

$$T = (65535 - n).\frac{1}{f_{CK}}.F_b \quad \ldots\ldots\ldots\ldots\ldots\ldots (2.2)$$

dimana:
T= Waktu hitung
n = nilai awal pada TCNT1 dalam desimal
f_{CK} = frekuensi detak (osilator kristal) (Hz)
F_b = faktor bagi pada penggunaan prescaler.

2.8.2.2 *Timer*/*Counter1 sebagai* Output *Compare*

Fungsi output *compare* berarti fungsi pembandingan isi *Timer*/*Counter*1 dengan suatu register yaitu register OCR1(A/B). Ketika terjadi *compare match* (persamaan nilai) antara isi *Timer*/*Counter* (pada register TCNT1) dengan isi register OCR1(A/B) maka akan memberikan efek pada kaki OC1(A/B) sesuai dengan Tabel 2.10.

Tabel 2.10 Compare 1 Mode Select

COM1X1	*COM1X0*	*Deskripsi*
0	*0*	Timer/Counter*1 di*-disconect *dari kaki keluaran OC1X*
0	*1*	Toggle *jalur keluaran OC1X*
0	*0*	Clear *jalur keluaran OC1X*
0	*1*	*Mengeset jalur keluaran OC1X*

Catatan: X=A atau B

Pada mode PWM, bit-bit ini mempunyai fungsi yang berbeda.

2.8.2.3 *Timer*/*Counter1* untuk fungsi PWM

Untuk mengoperasikan *Timer*/*Counter*1 menjadi fungsi PWM, maka kombinasi bit PWM11 dan PWM10 pada register TCCR1A harus dipilih pada mode PWM. Kombinasi bit-bit tersebut sesuai dengan Tabel 2.11

Tabel 2.11 Pemilihan mode PWM

PWM11	*PWM10*	*Deskripsi*
0	*0*	*Operasi PWM* Timer/Counter*1 di*-disable
0	*1*	Timer/Counter*1 adalah PWM 8 bit*
1	*0*	Timer/Counter*1 adalah PWM 9 bit*
1	*1*	Timer/Counter*1 adalah PWM 10 bit*

2.8.3 Prescaler

AT90S8515 dilengkapi dengan fasilitas *prescaling* untuk *Timer*/*Counter*. Prescaling berarti membagi frekuensi detak (CK) dengan bilangan pembagi 8, 64, 256, 1024. Ini akan sangat berguna pada penggunaan waktu tunda yang akan bertambah lama dibandingkan dengan tidak adanya faktor bagi pada sumber detaknya.

2.9 SPI (*Serial Peripheral Interface*)

SPI (Serial *Peripheral Interface*) atau antarmuka periferal serial adalah sarana yang diberikan AT90S8515 untuk transfer data sinkron berkecepatan tinggi yaitu antara MCU dengan divais periferal atau antara beberapa divais AVR atau antara divais master dengan divais *slave*

2.10 Baud *Rate*

Pembangkit *baud rate* adalah pembagi frekuensi yang membangkitkan *baud rate* berdasarkan rumus berikut:

$$BAUD = \frac{f_{CK}}{16(UBRR + 1)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.3)$$

dimana:
Baud = Baud rate
f_{CK} =Frekuensi detak kristal
UBRR= Isi dari UBRR (UART *Baud Rate* Register)(0-255)

2.11 Watchdog Timer

Watchdog *Timer* ini adalah salah satu fasilitas yang ada pada AT90S8515. Fasilitas ini digunakan untuk mereset MCU, ketika periode pewaktuan Watchdog habis.

2.12 UART (*Universal Asynchronous Receiver Transmitter*)

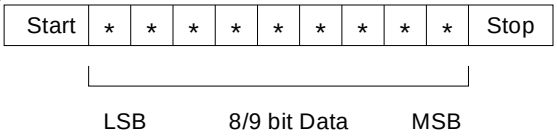
UART berfungsi untuk komunikasi data secara serial melalui kaki TXD dan kaki RXD. Kaki TXD (PD1) untuk jalur data yang dikirim, sedangkan kaki RXD (PD0) untuk jalur data yang diterima

2.12.1 Pengiriman Data

Transmisi data diinisialisasi dengan menuliskan data yang akan dikirimkan ke dalam register UDR (UART I/O Data Register). Data ini kemudian akan diberikan ke register penggeser (*shift* register) sebelum akhirnya dikirim dalam format *serial*.

2.12.2 Penerimaan Data

Data yang dapat dikirim atau terima adalah sepanjang 8 atau 9 bit. Ini tidak termasuk bit start ataupun bit *stop*.



Gambar 2.5 Format data 8/9 bit

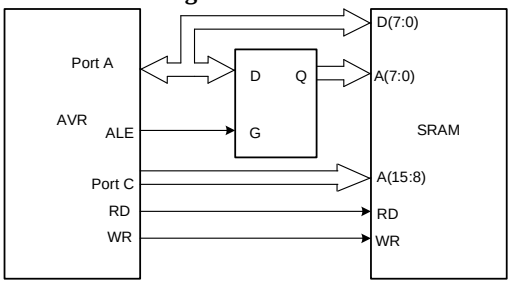
2.13 Analog comparator

Prinsip kerja dari perangkat keras di atas adalah membandingkan nilai tegangan *analog* pada masukan PB2 (AIN0) dan masukan negatif PB3 (AIN1). Ketika tegangan pada masukan positif lebih tinggi daripada masukan negatif, maka keluaran ACO menjadi set. ACO (*Analog comparator Output*) merupakan bit 5 dari ACSR (*Analog comparator* and Status Register).

2.14 EEPROM

AT90S8515 memiliki EEPROM (*Electrical Erasable Programmable Read Only Memory*) sebesar 512x8 bit. EEPROM ini dapat ditulis dan dibaca. Register yang mengatur proses penulisan dan pembacaan ini terdiri dari regiser alamat, register data, dan register kontrol.

2.15 Antarmuka dengan RAM Eksternal.



Gambar 2.6 Hubungan AT90S8515 dengan RAM eksternal

2.16 Mode *Sleep*

Mode sleep terbagi menjadi dua macam, yaitu mode *Idle* dan mode *power down*.

Mode *idle* berarti menghentikan CPU tapi tetap mengijinkan *Timer/Counter*, *Watchdog*, dan sistem interupsi untuk tetap meneruskan operasi.

2.17 SREG

SREG (Status Register) atau register status berisi *flag* yang menandakan keadaan mikrokontroler terutama setelah operasi aritmatika. Register tersebut telah perlihatkan pada gambar

Bit	7	6	5	4	3	2	1	0	
$3F($5F)	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	
Nilai inisial	0	0	0	0	0	0	0	0	

Gambar 2.7 Status Register (SREG)

2.18 *Stack Pointer*

Stack Pointer (SP) ini adalah sepanjang 16 bit yang merupakan gabungan dari 2 buah register pada ruang memori I/O yaitu SPL dan SPH. Stack Pointer ini perlihatkan pada Gambar 2.53:

Bit	15	14	13	12	11	10	9	8	
$3E($5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
$3D($5D)	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	
Nilai inisial	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Gambar 2.8 Stack Pointer

III.PERANGKAT LUNAK

3.1 Pendahuluan

Mikrokontroler AVR dirancang dengan mempertimbangkan sifat-sifat pengkodean bahasa C, sehingga bahasa tingkat tinggi inilah yang kemudian cenderung digunakan daripada bahasa tingkat tinggi lainnya seperti bahasa basic ataupun pascal. Bahasa C yang digunakan pada AVR ini adalah ANSI (*American National Standard Institute*) C. Alasan utama pemilihan bahasa C ini adalah karena bahasa C merupakan gabungan dari bahasa tingkat tinggi dan juga tingkat rendah yang menyediakan kemampuan operasi-operasi bit, byte, alamat-alamat memori, dan register. Bahasa C yang digunakan untuk memprogram mikrokontroler ini kemudian disebut sebagai *embedded* C, atau bahasa C yang tertanam pada mikrokontroler.

Program *compiler* C yang digunakan pada Tugas Akhir ini adalah CodeVisionAVR versi 1.23 yang dapat diperoleh secara cuma-cuma di website www.hpinfotech.ro. Program C yang dapat di-*compile* menggunakan *compiler* ini terbatas tapi mencukupi untuk program-program kecil untuk keperluan eksperimen pribadi dan belajar para mahasiswa. Program hasil kompilasi yaitu *file object* COFF dapat didebug dalam level C menggunakan AVR studio versi 3.55 yang juga dapat diperoleh secara cuma-cuma di website www.atmel.com.

3.2 BAHASA C

3.2.1 Komentar

Komentar memudahkan *programmer* untuk memberikan keterangan tambahan mengenai listing program. Komentar ini tidak diikut sertakan dalam proses kompilasi.

Contoh:

```
//ini adalah format komentar satu baris
/*ini format komentar lebih dari satu baris*/
```

3.2.2 Kata Kunci

Kata kunci merupakan kata yang sudah disediakan oleh compiler dan tidak boleh digunakan oleh *programmer* sebagai identifier.

3.2.3 Identifier

Identifier atau pengenal adalah nama yang diberikan ke sebuah variabel, fungsi, label atau obyek yang lain. *Identifier* dapat berisi huruf (A…Z, a…z) dan angka (0…9) serta karakter *underscore* (_). *Identifier* hanya dapat diawali dengan huruf atau *underscore*. Huruf besar dan huruf kecil dibedakan, sehingga `variabel1` berbeda dengan `VARIABEL1`. Panjang *identifier* maksimum adalah 32 karakter.

3.2.4 Tipe Data

Tabel berikut mendaftar semua tipe data yang di-dukung oleh CodeVisionAVR C *compiler*.

Tipe	Ukuran (bit)	Kisaran
Bit	1	0 , 1
`Char`	8	-128 sampai 127
`Unsigned char`	8	0 sampai 255
Signed `char`	8	-128 sampai 127
Int	16	-32768 sampai 32767
Short int	16	-32768 sampai 32767
Unsigned int	16	0 sampai 65535
Signed int	16	-32768 sampai 32767
Long int	32	-2147483648 sampai 2147483647
Unsigned long int	32	0 sampai 4294967295
Signed long int	32	-2147483648 sampai 2147483647
`Float`	32	±1.175e-38 sampai ±3.402e38
Double	32	±1.175e-38 sampai ±3.402e38

3.2.5 Konstanta

Konstanta integer dan long integer dapat ditulis dalam bentuk desimal misal 34, biner dengan awalan 0b (misal 0b00010001), hexadesimal dengan awalan 0x (misal 0x5F), atau oktal dengan awalan 0 (misal 054).

Konstanta *unsigned* integer dapat ditambah akhiran U (misal 10000U)

Konstanta *long integer* dapat ditambah akhiran L (misal 99L)

Konstanta *unsigned long integer* dapat ditambah akhiran UL (misal 99UL)

Konstanta *floating point* dapat ditambah akhiran F (misal 1,234F)

Konstanta karakter harus diapit oleh tanda kutip tunggal (misal ‘a’)

Konstanta string harus diapit oleh tanda kutip ganda (misal “`halo`”)

Jika string yang diletakkan diantara tanda kutip sebagai parameter fungsi, string ini akan secara otomatis dianggap sebagai konstanta dan ditempatkan di memori *flash.*

3.2.6 Variabel

Variabel program dapat berupa global (terakses pada semua fungsi program) atau lokal (hanya terakses di dalam fungsi yang mendeklarasikan). Jika tidak diinisialisasi secara khusus, variabel global secara otomatis diset pada 0 pada *startup* program

3.2.7 Menentukan Alamat Penempatan Ram untuk variabel global

Variabel global dapat diisikan pada lokasi RAM tertentu menggunakan operator @.

Contoh:

```
   /* variabel integer “a” diisikan di
   RAM dengan alamat 80h */
int a @0x80;
```

3.2.8 Variabel Bit

Variabel bit adalah variabel global khusus yang ditempatkan pada ruang memori R2 sampai dengan R14. Variabel ini dideklarasikan menggunakan kata kunci `bit`. Sintaksnya adalah:

```
bit <identifier>;
```

3.2.9 Alokasi Variabel ke Register

Dalam rangka memperoleh manfaat penuh dari arsitektur AVR dan instruction set, *compiler* mengalokasikan beberapa variabel program ke register chip. Register dari R2 sampai dengan R14 dapat dialokasikan untuk variabel bit. Jumlah register yang akan digunakan dapat ditentukan. Nilai ini harus sama dengan terendah yang dibutuhkan oleh program. Jika direktif *compiler* `#pragma regalloc+` digunakan, sisa register dalam kisaran R2 ke R14 yang tidak digunakan untuk variabel bit, dialokasikan untuk global variabel `char` dan int. Jika pengalokasian register otomatis di-*disable*, kata kunci register dapat dogunakan untuk menentukan variabel global mana yang dialokasikan ke register.

Contoh:

```
/* disable alokasi register otomatis */
#pragma regalloc-
/*  pengalokasian  variabel  ‘alfa’  ke
sebuah register */
register int alfa;
/*  Pengalokasian  variabel  ‘beta’  ke
pasangan register R10, R12 */
register int beta @10;
```

Variabel lokal int dan `char` secara otomatis dialokasikan ke register R16 sampai dengan R20.

3.2.10 Struktur

Struktur adalah kumpulan dari anggota nama yang didefinisikan oleh pengguna. Anggota struktur dapat berupa tipe data yang disediakan *compiler*, *array* dari tipe data ini, atau *pointer* yang menunjuk mereka.

Struktur didefinisikan menggunakan kata kunci `struct`.

Sintaksnya adalah:

```
[<storage modifier>] struct [<structure tag-name>] {
          [<type>   <variable-name[,   variable-name, ...]>];
          [<type>   <variable-name[,   variable-name, ...]>];
           ...
          } [<structure variables>];
```

3.2.11 Union

Unions adalah kumpulan anggota yang diberi nama yang didefinisikan oleh *user* yang berbagi ruang memori yang sama. Anggota dapat berupa sebarang tipe data yang di-*support*, *array* dari tipe data ini atau *pointer* ke mereka. Union didefinisikan menggunakan kata kunci `union`.
Sintaksnya adalah:

```
[<storage modifier>] union [<union tag-name>] {
      [<tipe> <nama-variabel[, nama-variabel, ...]>];
      [<tipe> <nama-variabel[, nama-variabel, ...]>];
      ...
      } [<variabel union>];
```

3.2.12 Enumerasi

Tipe data enumerasi dapat digunakan dalam rangka menghasilkan *identifier* mnemonic untuk sejumlah nilai int. Kata kunci `enum` digunakan untuk tujuan ini.
Sintaksnya adalah:

```
[<storage modifier>] enum [<enum tag-name>] {
     [<nama-konstanta[[=penginisial-konstanta],
      nama-konstanta, ...]>]}
     [<variabel enum>];
```

3.2.13 Mendefinisikan Tipe Data

Tipe data yang didefinisikan user dideklarasikan menggunakan kata kunci typedef. Sintaksnya adalah:

```
typedef [<storage modifier>] <tipe> <identifier>;
```

Nama simbol (*identifier*) mengacu pada (tipe)

3.2.14 Global Variabel Memori Map File

Selama proses kompilasi, *compiler* C menghasilkan *file* peta memori variabel global, yang di dalamnya tertera lokasi alamat RAM, alokasi register, dan ukuran dari variabel global yang digunakan oleh program. *File* ini memiliki ekstensi .map.

Anggota struktur dan union tertera tersendiri sesuai dengan alamat yang bersesuaian dan ukuran mereka.

3.2.15 Konversi Tipe

Dalam sebuah ekspresi, jika dua operan dari operator biner adalah dari tipe yang berbeda, kemudia *compiler* akan mengkonversi satu dari dua operan menjadi tipe operan yang lain.

Compiler menggunakan aturan sebagai berikut:

1. jika salah satu operan adalah bertipe `float` kemudian operan yang lain dikonversi menjadi tipe yang sama.
2. Jika salah satu operan bertipe `long int` atau `unsigned long int` kemudian operan yang lain dikonversi ke dalam tipe yang sama.
3. Jika salah satu operan dari tipe `int` atau `unsigned int` kemudian operan yang lain dikonversi menjadi tipe yang sama.

   Dengan demikian tipe `char` atau tipe `unsigned char` mendapat prioritas yang paling rendah.

3.2.16 Operator

Compiler mengijinkan operator berikut:

```
 +        -        ^        >>
 *        /        <<       +=
 %        ++       -=       %=
 --       =        /=       *=
 ==       ~        &=       |=
 !        !=       ^=       <<=
 <        >        >>=      ||
 <=       >=       |
 &        &&       ?
```

3.2.17 Fungsi

Prototipe fungsi dapat digunakan untuk mendeklarasikan sebuah fungsi. Deklarasi ini mengikutsertakan informasi tentang parameter fungsi.
Contoh:

```
int alfa(char par1, int par2, long par3);
```

Definisi fungsi aktual dapat ditulis di tempat lain sebagai:

```
int alfa(char par1, int par2, long par3) {
/* berisi statement-statement */
}
```

Parameter fungsi dilewatkan pada stack data. Nilai fungsi dikembalikan pada register R30, R31, R22, dan R23 (dari LSB ke MSB).

3.2.18 Pointer

Pointer (variabel penunjuk) adalah suatu variabel yang berisi alamat suatu memori tertentu. Jadi, *pointer* bukan berisi suatu nilai data, tetapi berisi dengan suatu alamat.

Mengacu pada arsitektur Harvard dari mikrokontroler AVR, yang memisahkan ruang alamat untuk memori data (RAM), program (*flash*) dan EEPROM, *compiler* menerapkan tiga tipe *pointer*.

Variabel yang ditempatkan di RAM diakses menggunakan *pointer* normal. Untuk mengakses konstanta yang ditempatkan di memori *flash*, kata kunci `flash` harus digunakan. Untuk mengakses variabel yang ditempatkan di EEPROM, kata kunci `eeprom` harus digunakan. Walaupun *pointer* dapat menunjuk lokasi memori yang berbeda, secara default adalah ditempatkan pada RAM.

3.2.19 Mengakses register I/O

Compiler menggunakan kata kunci `sfrb` dan `sfrw` untuk mengakses register I/O mikrokontroler AVR, menggunakan instruksi *assembly* IN dan OUT.
Contoh:

```
/* Definisi  SFR */
sfrb PINA=0x19;  /* akses 8 bit ke SFR */
sfrw TCNT1=0x2c; /* akses 16 bit SFR */
void main(void) {
unsigned char a;
a=PINA;       /* Membaca pin input PORTA */
TCNT1=0x1111; /* Menulis ke register TCNT1L & TCNT1H */
}
```

Alamat dari register I/O dipredefinisikan pada header *file* 90s8515.h yang berada pada subdirektori \INC. *File* tersebut dapat diikutsertakan menggunakan preprosesor `#include` pada awal program.

3.2.20 Akses level Bit Register I/O

Akses bit level ke register I/O dapat dikerjakan menggunakan pemilih bit yang dilampirkan setelah nama register I/O. Karena akses bit level ke register I/O dilakukan menggunakan instruksi CBI, SBI, SBIC dan SBIS, alamat register harus berada dalam kisaran 0 sampai dengan 1Fh untuk `sfrb` dan dalam kisaran 0 sampai dengan 1Eh untuk sfrw.

3.2.21 Mengakses EEPROM

Mengakses EEPROM internal AVR diselesaikan menggunakan variabel global, yang didahului oleh kata kunci `eeprom`.

3.2.22 Menggunakan interupsi

Akses ke sistem interupsi AVR diterapkan menggunakan kata kunci `interrupt`.
Contoh:

```
/* Terpanggil secara otomatis ketika terjadi
interupsi eksternal */
interrupt [2] void int0-eskternal(void) {
/* Kode program ditulis di sini */
}
/* Terpanggil secara otomatis pada overflow
TIMER0 */
interrupt [8] void overflow_timer0(void) {
/* Kode program ditulis di sini */
}
```

Nomor vektor interupsi dimulai dari 1. *Compiler* akan secara otomatis menyimpan semua register yang digunakan ketika memanggil fungsi interupsi dan dan mengisikan kembali setelah selesai.

3.2.23 Preprosesor

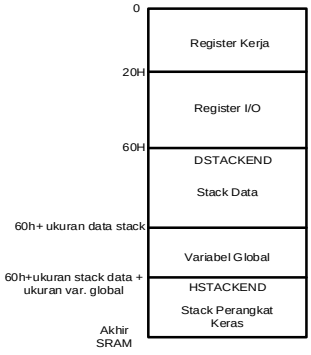
Direktiv preprosesor memungkinkan untuk:

1. Mengikutsertakan teks dari *file* yang lain,seperti *file header* yang berisi prototipe *library* dan fungsi *user*
2. Mendefinisikan macros yang mengurangi upaya pemrograman dan memperbaiki kemudahterbacaan dari *source* code.
3. Mengeset kompilasi kondisional untuk tujuan pen-*debug*-an dan memperbaiki portabilitas program.
4. Mengeluarkan direktiv khusus *compiler*.

3.2.24 Organsisasi Memori RAM

Sebuah program terkompilasi memiliki peta memori seperti yang diperlihatkan pada gambar 3.1.

Register kerja berisi 32x8 bit register kerja serbaguna. Compiler menggunakan register berikut: R0, R1, R15, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31.



Gambar 3.1 Organisasi Memori RAM

Juga beberapa register dari R2 sampai dengan R14 dapat ditempati untuk variabel bit global oleh register. Sisa register yang tidak digunakan, dalam kisaran ini, ditempati untuk variabel global `char` dan int. Register R16 sampai dengan R21 ditempati untuk variabel local `char` dan int .

3.2.25 Menggunakan File Start-up Eksternal

Pada setiap program, *compiler* CodeVisionAVR secara otomatis menghasilkan sebuah urutan kode untuk membuat inisialisasi berikut dengan segera setelah chip AVR reset:

1. tabel lompatan vektor interupsi
2. disable interupsi global
3. disable akses EEPROM
4. disable Watdhdog *Timer*
5. akses RAM eksternal dan enable wait state jika diperlukan.
6. Meng-clear register R2…R14
7. Meng-clear RAM
8. Inisialisasi variabel global yang ditempatkan di RAM
9. Inisialisasi register *stack pointer* data Y
10. Inisialisasi register *stack pointer* SP
11. Inisialisasi register UBRR jika diperlukan.

3.2.26 Mengikutsertaka Bahasa Assembly dalam Program

Bahasa *assembly* dapat diikutsertakan dimanapun dalam program menggunakan direktiv `#asm` dan `#endasm`.

3.2.27 Memanggil Fungsi Assembly dari C

Contoh berikut menunjukkan bagaimana mengakses fungsi yang ditulis dalam bahasa *assembly* dari program C:

```
// Fungsi dalam deklarasi assembly
// Fungsi ini akan mengembalikan a+b+c
#pragma warn- /* Ini akan
menghindarkan adanya peringatan pada
saat kompilasi*/
int jml_abc(int a, int b, unsigned
char c) {
#asm
    ldd   r30,y+3 ;R30=LSB a
    ldd   r31,y+4 ;R31=MSB a
    ldd   r26,y+1 ;R26=LSB b
    ldd   r27,y+2 ;R27=MSB b
    add   r30,r26 ;(R31,R30)=a+b
    adc   r31,r27
    ld    r26,y   ;R26=c
    clr   r27     ;promosi unsigned
char c ke int
    add   r30,r26
;(R31,R30)=(R31,R30)+c
    adc   r31,r27
#endasm
}
#pragma warn+ // mengenable peringatan
pada kompilasi
void main(void) {
int r;
// Pemanggilan fungsi dan menempatkan
hasilnya di r
r=jml_abc(2,4,6);
}
```

Compiler melewatkan parameter fungsi menggunakan *stack* data. Pertama *compiler*

mendorong (*push*) parameter integer a, kemudian b, dan akhirnya parameter `unsigned char` c.

3.2.28 Menggunakan AVR Studio Debugger

CodeVisionAVR didesain untuk dapat dibantu oleh Atmel AVR Studio *debugger* versi 3 dan 4.

Dalam rangka memungkinkan pen-*debug*-an level *source* C menggunakan AVR Studio, format *file* keluaran COFF harus dipilih pada pilihan menu Project|Configure|Assembler.

AVR Studio Debugger di-akses menggunakan perintah menu Tools|debugger atau tombol toolbar Debugger.

3.2.29 Batasan

Compiler codeVissionAVR versi 1.23 mempunyai pembatasan:

- *Pointer* ke *pointer* tidak diijinkan
- *Array* dari struktur atau union dapat hanya memiliki satu dimensi.

Bitfield dalam struktur tidak diterapkan. Untuk pengisian level bit gunakan variabel bit.

IV.DEVELOPER TOOLS

4.1 STK200 ISP Dongle

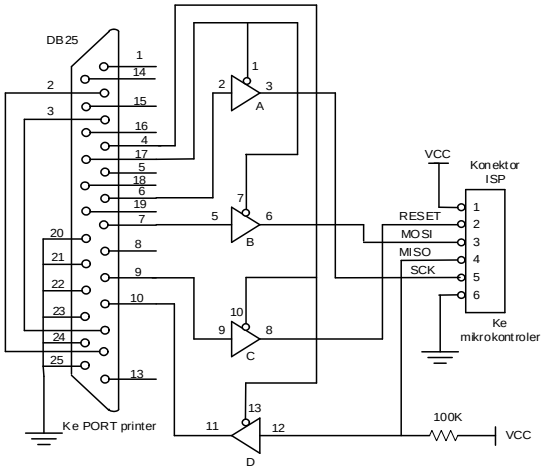
Kemampuan *In-System Programming* yang dimiliki mikrokontroler AVR membuat mikrokontroler ini mudah untuk digunakan, karena selama ini pengisian program ke dalam sistem dengan mikrokontroler yang sedang dirancang bangun merupakan masalah cukup pelik, yang sangat merepotkan pemakai pemula mikrokontroler.

Rangkaian ini dihubungkan ke sistem target melalui enam koneksi, 2 kabel untuk sumber daya rangkaian ini yaitu VCC dan ground, 4 kabel sebagai sinyal kontrol untuk mengisi memori-program AVR, yang pertama adalah RESET dipakai untuk me-reset AVR, tiga lainnya adalah sinyal MOSI (*Master Out Slave In*) untuk mengirim data dari PC ke AVR, sinyal MISO (*Master In Slave Out*) untuk mengirim data dari AVR ke PC, dan SCK (*Serial Clock*) untuk mendorong data seri. Ketiga sinyal ini dikenal sebagai sinyal kontrol SPI (*Serial Peripheral Interface*) ciptaan Motorola.

IC 74LS125 merupakan *Quad* 3-*state Buffers*, dipakai untuk memutus hubungan antara PC dan AVR pada sistem target pada saat tidak dilakukan *In-System Programming*, sehingga meskipun rangkaian ini masih terpasang sistem target masih dapat bekerja seperti biasa.

Kaki 1 dan 4 dihubungkan menjadi satu, demikian pula dengan kaki 10 dan 13. Kaki-kaki IC 74LS125 tersebut merupakan *tri-state* kontrol. Jika kaki ini bernilai ‘1’ maka output kaki 3,6,8, dan 11 dalam keadaan mengambang (tidak ‘0’ dan tidak ‘1’) sehingga IC 74LS125 tidak berpengaruh apa pun terhadap AVR.

Proses pengisian memori-program AVR terjadi sebagai berikut, dalam keadaan catu daya (Vcc)terpasang, kaki RESET dan kaki SCK pada AVR di-nol-kan, sekitar 20 mili-detik kemudian melalui MOSI dikirim ke AVR kode 0ACh, 053h, 0xxh dan 0xxh (0xxh artinya apapun boleh) untuk mengaktifkan fasilitas *In-System Programming*, AVR akan menjawab dengan kode 053h yang dikirim melalui MISO. Setelah itu AVR siap menerima kiriman perintah-perintah berikutnya untuk melakukan pengisian memori-program maupun EEPROM di dalam chip AVR. Tata cara pengisian ini dibahas secara rinci dalam *data sheet* AVR, tidak dibahas lebih lanjut di sini karena semuanya sudah ditangani oleh program CodeVisionAVR.



Gambar 4.1 Paralel port STK200 ISP DONGLE

Kaki 3 pada konektror DB25 dihubungkan ke kaki 11, demikian pula kaki 2 dihubungkan ke kaki 12. Hubungan ini untuk menandai bahwa port printer terhubung ke rangkaian yang dapat digunakan untuk *In-System Programming*. Program AVR ISP akan memeriksa dulu adanya hubungan tersebut, sebelum melakukan proses *In-System Programming*.

MOSI, MISO, SCK dan RESET adalah nama-nama kaki pada mikrokontroler AVR, karena mikrokontroler AVR terdiri dari berbagai jenis dengan jumlah kaki IC yang berlainan, maka nomor kaki untuk keempat sinyal tersebut tidak sama untuk berbagai jenis.

Untuk IC AVR AT90S8515 MOSI terletak di kaki nomor 6, MISO nomor 7, SCK nomor 8, dan RESET kaki nomor 9.

BAB V APLIKASI

5.1 Pengaturan LCD

Program:

```
1  /*
2    LCD Demo
3    LCD dihubungkan dengan port A AVR
4   4 RS - 1  PC0
5   5 RD - 2  PC1
6   6 EN - 3  PC2
7  11 D4 - 5  PC4
8  12 D5 - 6  PC5
9  13 D6 - 7  PC6
10  14 D7 - 8  PC7
11  */
12  //LCD dihubungkan dengan Port A
13  #asm
14  .equ __lcd_port=0x1b ;PORTA
15  #endasm
16  // include rutin LCD driver
17  #include <lcd.h>
18  void main(void)
19  {
20 // inisialisasi LCD 2 baris 16 kolom
21 lcd_init(16);
22 // posisi awal penulisan karakter
23 lcd_gotoxy(0,0);
24 // tampilkan pada display
25 lcd_putsf("Teknik Elektro");
```

```
26 // berhenti di sini
27 while (1);
28 }
```

Mode pengiriman data yang digunakan adalah mode 4 bit, sedang jalur kontrol yang digunakan adalah jalur pin RS, RD, EN. Ketujuh jalur I/O ini diatur menggunakan Port A dari AT90S8515.

Baris ke-14 merupakan pendefinisian atau deklarasi dari Port A. Baris ke-17 berarti mengikutkan file header yang bernama lcd.h. File ini berisi prototype dari fungsi-fungsi yang akan digunakan oleh program. File ini juga memberitahu kompiler tentang file pustaka (.lib) yang berisi fungsi-fungsi yang digunakan oleh program. Fungsi `lcd_init()`, `lcd_gotoxy()`, dan `lcd_putsf()` terletak pada file lcd.lib yang sudah diinformasikan oleh file lcd.h.

Baris ke-21 adalah pemanggilan fungsi `lcd_init()` dengan argumen 16 yang merupakan jumlah kolom dari lcd. `Lcd_init()` berfungsi menginisialisasi lcd agar siap diberikan karakter sejumlah 16 kolom. Secara default fungsi ini memerintahkan lcd untuk menulis karakter dari baris pertama kolom pertama, dari kiri ke kanan, tanpa ada pergeseran karakter, dan tanpa blink.

Fungsi `lcd_gotoxy()` adalah untuk menempatkan posisi awal dari penulisan karakter. `Lcd_gotoxy(0,0)` berarti penulisan karakter dimulai dari baris pertama dan kolom pertama.

`lcd_putsf("Teknik Elektro")` merupakan baris pemanggil fungsi `lcd_putsf` dengan argumen string `Teknik Elektro`. Tulisan inilah yang akan tertulis pada lcd.

5.2 MENAMPILKAN HASIL

Format data yang ditampilkan pada LCD dapat diubah-ubah sesuai dengan kebutuhan. Fungsi-fungsi menampilkan hasil berada pada file header stdio.h. dan lcd.h. Berikut ini akan diberikan program untuk menampilkan data dalam format tertentu.

```
#asm
.equ __lcd_port=0x1B
#endasm
#include <lcd.h>
#include <stdio.h>
unsigned char N,buf[32];
void main (void)
{lcd_init(16);
lcd_gotoxy(0,0);
N=255;
sprintf(buf,"%dd=%Xh",N,N);
lcd_puts(buf);
}
```

Jika dijalankan, di LCD akan tertampil:

`255d=FFh`

Program di atas format data dalam notasi desimal dan notasi heksadesimal. Fungsi `sprintf()`mengkonvesi masukan berupa variabel N ke keluaran dalam format sesuai dengan format specifier. Format *specifier* pada contoh di atas adalah `%d` dan `%X`. `%d` untuk menampilkan notasi desimal sedang `%X` untuk notasi heksadesimal dalam notasi kapital. Hasil konversi dimasukkan dalam anggota larik berdimensi satu, bertipe `unsigned char` dengan nama `buf.` Anggota dari larik ini kemudian dikirimkan ke LCD oleh fungsi `lcd_puts()` yang berada pada file lcd.h

5.3 KONVERSI DATA

Konversi data diperlukan untuk menyesuaikan dengan fungsi-fungsi tertentu. Sebagai contoh, program berikut akan mengkonversikan data dengan tipe `float` menjadi data dengan tipe `char` yang ditempatkan dalam anggota-anggota array.

Program:

```
#asm
   .equ __lcd_port=0x1B
#endasm
#include <lcd.h>
#include <stdlib.h>
#include <math.h>
float A;
char buf[6];
void main (void)
{lcd_init(16);
lcd_gotoxy(0,0);
A=27.592;
ftoa(A,5,buf);
lcd_puts(buf);
}
```

Jika program dijalankan pada mikrokontroler AVR maka pada LCD akan tertampil:

`27.59200`

Variabel A berisikan suatu konstanta pecahan atau bertipe `float`. Fungsi `lcd_puts()` hanya dapat menampilkan karakter dari masukan array bertipe `char`. Oleh karena itu, tipe variabel A harus dikonversikan ke dalam bilangan-bilangan dengan tipe `char` yang termasuk anggota dari array dengan nama `buf` dengan 5 angka di belakang koma. Konversi ini dilakukan oleh fungsi `ftoa()` yaitu *`float to char`*. Array ini kemudian ditampilkan di LCD oleh fungsi `lcd_puts().`

5.4 OPERASI ARITMATIKA

Bahasa C sangat mendukung operasi aritmatika karena banyaknya operator aritmatika dan juga fungsi-fungsi yang ada pada library matematis math.lib. Prototype dari fungsi-fungsi pada math.lib berada pada file header math.h. Fungsi-fungsi yang dapat dipergunakan antara lain fungsi sinus, kosinus, tangen, arc sin, arc cos, arc tan, akar, dan logarima .Berikut ini akan diberikan contoh program untuk menghitung akar dari suatu bilangan. Karena program yang akan dikompilasi terbatas untuk compiler evaluation version maka program dibuat sederhana yaitu dengan tidak melibatkan masukan dari keypad.

```
#asm
   .equ __lcd_port=0x1B
#endasm
#include <lcd.h>
#include <stdlib.h>
#include <math.h>
char buf[7];
float B;
void main (void)
{lcd_init(16);
lcd_gotoxy(0,0);
B=sqrt(11);
ftoa(B,3,buf);
lcd_puts(buf);
}
```

Variabel B mempunyai tipe data `float` atau pecahan. Variabel B ini adalah hasil operasi matematika dari fungsi `sqrt().` Fungsi ini memberikan hasil akar kuadrat dari argumen fungsi konstanta 11.Hasil ini kemudian diberikan ke variabel

B dengan tipe data `float`. Tipe ini kemudian dikonversi ke dalam tipe `char` baru kemudian ditampilkan LCD.

5.5 RUNNING LED

Program:

```
1 /* Program untuk menyalakan Led bergantian dari
LSB ke MSB*/
2
3 // Pendefinisian register I/O untuk AT90s8515
4 #include <90s8515.h>
5 // Led LSB akan menyala
6 unsigned char logika_led=0x01;
7
8 // Fungsi Interupsi dari timer1, terjadi setiap
0,5 detik
9 interrupt [TIM1_OVF] void overflow_timer1(void)
10 {
11    // Pengisian nilai awal dari Timer 1
12      TCNT1=0;
13      // Geser kiri
14      logika_led<<=1;
15      if (logika_led==0x00) logika_led=0x01;
16      // Kirim ke Port D
17      PORTB=logika_led;
18 }
19
20 //******Program Utama******
21 void main(void)
22 {
23      // Inisialisasi Port B
24      DDRB=0xff;
25      // Nyalakan Led LSB
26      PORTB=logika_led;
27      // Inisialisasi Timer 1
28      TCCR1A=0;
29      // Prescale untuk Timer 1 dengan detak =
xtal/256
30      TCCR1B=4;
31      // Isi nilai awal dari Timer 1
32      TCNT1=0;
33      // clear flag interupsi Timer 1
34      TIFR=0;
35      // enable interupsi overflow Timer 1
36      TIMSK=0x80;
37      // Mengenable interupsi global
38      #asm
39      sei
40      #endasm
41         // menunggu adanya interupsi dari Timer 1

42      while (1);
43 }
```

Led 8 bit merupakan susunan 8 buah led yang disusun secara paralel pada port keluaran Port B dari mikrokontroler AT90S8515.

Program di atas menyalakan led LSB atau bit terkecil yaitu paling kanan kemudian menggeser nyala led setiap kira-kira 2 detik. Baris ke-4 merupakan pendefinisian register I/O dari mikrokontoler, sehingga penulisan nama register I/O langsung dapat dikenali oleh kompiler. Baris ke-6 merupakan deklarasi variabel dengan nama identifier (pengenal) `logika_led`. Penulisan identifier adalah sesuai dengan aturan pada Turbo C.

Program utama terletak pada baris ke-21. Port D dikonfigurasikan sebagai keluaran sehingga harus diinisialisasi dengan mengisikan konstanta 0xFF pada DDRD. Led paling kanan mulai dinyalakan pada baris ke-26. *Timer* 1 dikonfigurasikan sebagai penghasil tundaan sekitar 2 detik. Untuk ini maka *Timer*/*Counter*1 dipisahkan dari pin keluaran OC1X dan dalam mode tak menggunakan PWM. Konfigurasi ini diperoleh dengan menginisialisasi *Timer*1 dengan cara mengisi konstanta 0 pada register TCCR1A. Karena frekuensi kerja dari mikrokontroler AVR sama dengan frekuensi kristal maka fungsi prescaler digunakan sebagai pengali waktu tiap detak pada *Timer*1. Hal ini dilakukan dengan mengisikan konstanta 4 pada register TCCR1B sehingga frekuensi tiap detak *Timer* adalah frekuensi kristal dibagi dengan 256. Karena frekuensi kristal yang dipergunakan adalah 7372800 Hz maka frekuensi tiap detak *Timer*1 adalah 28800 Hz. Perioda satu detak *timer* berarti $3{,}47 \times 10^{-5}$. Untuk menghasilkan *overflow*, maka *Timer*1 harus berdetak sebanyak (65535 – nilai awal pada TCNT1) kali = (65535-0) kali = 65535 kali . Jadi waktu tunda yang dihasilkan adalah 2,28 detik. TIMSK = 0x80 berarti mengeset bit TOIE1 yang berarti mengeneble *Timer*1 *overflow*. Instruksi "sei" adalah instruksi dalam bahasa assembly yang disisipkan dalam bahasa C yang bertujuan mengeset bit I pada register status SREG yang bertujuan mengenable interupsi global. Instruksi `while(1)` pada baris 42 membuat program *counter* selalu berputar pada baris tersebut. Instruksi ini bertujuan menunggu interupsi yang berasal dari *overflow* *Timer*1.

Ketika terjadi *overflow* pada *Timer*1, yang berarti bit TOV1 pada register TIFR menjadi berlogika tinggi, program akan melompat ke baris ke 9 yang berisikan rutin pelayanan interupsi yang berasal dari *overflow* *Timer*1. Rutin ini berfungsi menggeser nyala dari led ke arah kiri setiap terjadi *overflow* *Timer*1 yang terjadi setiap kira-kira 2 detik.

5.6 Komunikasi dengan Personal Computer

Program:

```
1  /*program untuk aplikasi UART. Mikro akan
menerima data dari PC
2    dan dengan segera mengembalikan data
tersebut ke PC*/
3  #include <90s8515.h>
4    //definisi dari bit-bit pada register
kontrol UART
5  #define RXCIE 7
6  #define TXCIE 6
7  #define UDRIE 5
8  #define RXEN       4
9  #define TXEN  3
10 #define CHR9  2
11 #define RXB8  1
12 #define TXB8  0
13  //definisi dari bit-bit pada register
status UART
14 #define RXC 7
15 #define TXC 6
16 #define UDRE 5
17 #define FE   4
18 #define OVR  3
19 //prototipe fungsi
20 void InisialisasiUART(    unsigned    char
baudrate );
21 unsigned char TerimaByte( void );
22 void KirimByte( unsigned char data );
23 //program utama
24 void main( void )
25 {
26      InisialisasiUART(  3  );//inisialisasi
untuk baudrate 115kbit
27   while ( 1 )//selamanya
28   {
29     KirimByte( TerimaByte() );
30   }
31 }
32 //bagaian fungsi
```

```
33 void InisialisasiUART( unsigned char baudrate )
34 {
35   UBRR = baudrate;
36      UCR  =  (  (1<<RXEN)  |  (1<<TXEN));//enable
penerima dan pengirim
37 }                                  //UART
38 unsigned char TerimaByte( void )
39 {
40    while ( !(USR & (1<<RXC)) )/*tunggu data yang
datang*/
41    ;
42   return UDR;
43 }
44 void KirimByte( unsigned char data )
45 {
46   while ( !(USR & (1<<UDRE)) )
47   ;
48   UDR = data;
49 }
```

Baris ke-23 memanggil fungsi inisialisasi UART dengan argumen 3. Fungsi inisialisasiUART menentukan baud*rate* yang digunakan untuk komunikasi dengan PC. Argumen 3 merupakan konstanta yang akan diisikan pada register UBRR. Dengan menggunakan frekuensi kristal 7372800 Hz maka baud*rate* yang dihasilkan adalah 115200, dengan error 0,0.

Baris ke-29 memanggil fungsi `kirimByte` dengan argumen adalah data yang diberikan oleh fungsi `terimaByte`. Fungsi ini selalu menunggu data yang diberikan oleh fungsi terimaByte. Jika fungsi terimaByte tidak memberikan suatu nilai keluaran yang berarti tidak ada data yang dikirim oleh PC maka fungsi kirimByte tidak akan memberikan suatu keluaran.

Fungsi TerimaByte berada pada baris ke-38 dengan argumen void atau tanpa masukan. Baris ke-40 adalah baris untuk menunggu adanya data yang datang pada pin RXD pada mikrokontroler. Jika ada data serial pada pin RXD maka bit tujuh pada register USR akan berlogika tinggi sehingga program akan diteruskan pada baris ke-42 yaitu return UDR yaitu memberikan data kepada fungsi main. Jika belum ada data yang masuk ke pin RXD maka bit ke-7 dari register USR akan berlogika rendah sehingga pengkondisi while akan bernilai benar dan program akan berputar kembali ke baris 40.

Data yang dihasilkan oleh fungsi terimaByte selanjutnya diperlakukan sebagai masukan oleh fungsi kirimByte. Masukan pada fungsi ini harus bertipe `unsigned char` atau data 8 bit tak bertanda. Data ini yang kemudian akan dikirimkan kembali ke PC melalui serial port.

Program komunikasi yang digunakan untuk menerima dan mengirim data dapat menggunakan program terminal yang sudah tersedia satu paket dengan Codevision AVR compiler. Agar komunikasi dapat terjadi, maka baud*rate* yang digunakan harus diset sama yaitu 115200 bit per detik.

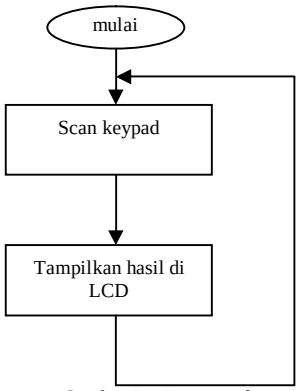
5.7 KEYPAD 1

Program:

```
/*Konfigurasi hardware keypad
  1 PC0 -----0----1----2----3
            |    |    |    |
  2 PC1 -----4----5----6----7
            |    |    |    |
  3 PC2 -----8----9----10---11
            |    |    |    |
  4 PC3 -----12---13---14---15
            |    |    |    |
  5 PC4 -----     |    |    |
                  |    |    |
  6 PC5 ----------     |    |
                       |    |
  7 PC6 ---------------     |
                            |
  8 PC7 --------------------
*/
#asm
    .equ __lcd_port=0x1B
#endasm
#include <lcd.h>
#include <stdio.h>
#include <90s8515.h>
#include <delay.h>
unsigned char dataA,dataB;
unsigned char E,D;
char buf[33];
void scan_baris (void);
void scan_kolom (void);
void tampil (unsigned char masukan);
unsigned char scan_keypad(void);
unsigned char scan_keypad(void)
{
unsigned char C;
    scan_baris();
    scan_kolom();
    C=dataA+dataB;
    return C;
}
void scan_kolom (void)
{
    DDRC=0x0f;
    PORTC=0xf0;
    delay_us(50);
    if (~PINC&16)
            dataB=0;
    else if (~PINC&32)
            dataB=1;
    else if (~PINC&64)
            dataB=2;
    else if (~PINC&128)
            dataB=3;
    else
            dataB=4;
}
void scan_baris(void)
{
    DDRC=0xf0;
    PORTC=0x0f;
    delay_us(50);
    if (~PINC&1)
            dataA=0;
    else if (~PINC&2)
            dataA=4;
    else if (~PINC&4)
            dataA=8;
    else if (~PINC&8)
            dataA=12;
    else
            dataA=13;
}
void tampil (unsigned char masukan)
{
sprintf(buf,"%X",masukan);
         lcd_puts(buf);
}
main()
```

```
78  { lcd_init(16);
79  lcd_gotoxy(0,0);
80  tampil(0);
81  lcd_gotoxy(0,0);
82  while (1)
83     {
84  D=scan_keypad();
85  delay_ms(100);
86  E=scan_keypad();
87  if (E!=17)
88  if (E!=D)
89  tampil(E);
90  }
91  }
```

Secara garis besar prinsip kerja dari pengaturan keypad adalah seperti pada Gambar 5.1.

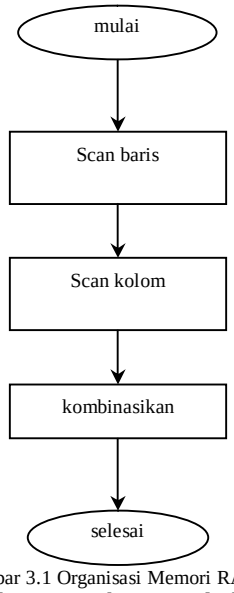


Gambar 5.1 Diagram alir scan keypad

Program akan mendeteksi adanya penekanan keypad kemudian menampilkan hasilnya pada LCD. Sedang diagram alir dari blok scan keypad dijelaskan pada gambar 5.2.

Blok dari scan baris diimplementasikan dalam program dimulai dari baris 56 sampai dengan baris 71. Baris 58 dan 59 merupakan inisialisasi dari Port C yang akan digunakan sebagai port masukan sekaligus keluaran dari keypad. DDRC yaitu register pengontrol arah data pada port C diisi dengan konstanta 0xF0. Hal ini berarti nibble atas digunakan sebagai keluaran dan nibble bawah digunakan sebagai masukan. Dengan mengirimkan konstanta 0x0F pada PORTC maka internal *pull-up* pada *nibble* bawah diaktifkan. Kondisi pada nibble bawah inilah yang mengindikasikan adanya penekanan pada baris tertentu. Sesuai dengan konfigurasi dari perangkat keras, PC0, PC1, PC2, PC3 masing-masing dihubungkan dengan baris pertama, kedua, ketiga, dan keempat. Jika ada penekanan pada baris pertama maka kondisi dari PC0 akan menjadi *low*, begitu jika ada penekanan pada baris kedua, PC1 akan menjadi *low* dan seterusnya. Jika ada penekanan pada baris pertama, uC akan memberikan nilai 0 pada variabel dataA. Untuk baris kedua, ketiga, dan keempat masing-masing akan memberikan nilai 4, 8 dan 12. Jika tidak ada penekanan pada keempat baris maka uC akan memberikan nilai 13 pada variabel dataA.

Blok scan kolom diimplementasikan dalam program pada baris 40 sampai dengan 55. Baris ke-42 dan ke-43 merupakan baris inisialisasi dari PORTC. DDRC diisi dengan konstanta 0x0F sehingga *nibble* atas berfungsi sebagai keluaran dan nibble bawah difungsikan sebagai keluaran. Dengan mengisikan konstanta 0xF0 pada register PORTC maka *pull-up* internal pada *nibble* atas dari Port C akan diaktifkan. Program akan selalu mengecek kondisi dari nibble atas ini. Pengecekan dilakukan pada register PINC karena register ini menunjukkan kondisi logika yang sesungguhnya



Gambar 3.1 Organisasi Memori RAM

pada PortC. Adanya penekanan pada kolom pertama ditandai dengan berubahnya kondisi logika pada kaki Port C bit 0 dari kondisi tinggi ke kondisi rendah. Pada program, kondisi tersebut dikenali dengan berubahnya kondisi logika dari tinggi menjadi rendah pada register PINC bit 0. Adanya penekanan pada kolom kedua, ketiga, dan keempat dapat diketahui dengan mengecek kondisi logika masing-masing pada PINC bit 1, bit 2 dan bit ke-3. Ketika terjadi penekanan pada kolom pertama, yang berarti kondisi logika dari PINC.0 rendah, uC akan memberikan konstanta 0 pada variabel dataB. Untuk penekanan pada kolom kedua, ketiga, keempat, uC masing-masing akan memberikan konstanta 1, 2, 3. Ketika tidak terjadi penekanan sama sekali pada kolom, uC akan memberikan konstanta 4.

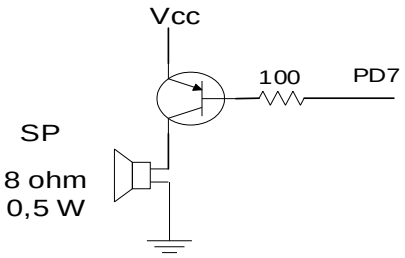
Blok kombinasikan pada diagram alir diimplementasikan pada baris ke-27 pada program. Baris ini adalah baris operasi aritmatika yaitu operasi penjumlahan. Bilangan yang dijumlahkan adalah konstanta yang diperoleh pada rutin scan baris dan scan kolom. Hasil penjumlahan dari baris dan kolom dapat digunakan untuk mengetahui pada baris dan kolom berapa penekanan. Sebagai contoh, jika hasil penjumlahan adalah 0 maka keypad yang ditekan adalah keypad baris pertama kolom pertama. Hasil penjumlahan ini kemudian akan dikirimkan ke LCD.

Blok tampilkan hasil di LCD diimplementasikan pada baris 72 hingga 76. Prototype dari fungsi `sprintf()`berada pada file stdio.h, sedang fungsi yang sesungguhnya berada di file stdio.lib. Fungsi `sprintf()` pada baris 74 berguna untuk memindahkan data variabel `masukan` dengan tipe data integer heksadesimal tak bertanda yang ditampilkan dalam huruf kapital ke suatu larik dengan nama buf berelemen 33.

Prototype dari fungsi `lcd_puts()` berada pada file lcd.h. Fungsi `lcd_puts()` yang sesungguhnya berada pada file lcd.lib. Fungsi `lcd_puts()` berguna untuk menampilkan karakter yang ada pada buf.

5.8 MEMBUNYIKAN SPEAKER

Speaker akan menghasilkan suara jika terdapat sinyal audio yang diberikan padanya. Sinyal ini dapat berupa sinyal kotak maupun sinyal sinus. Pada tugas akhir ini dipilih speaker dengan spesifikasi daya 0,5 W dan impedansi 8 ohm sehingga sudah mampu menghasilkan suara hanya dengan arus kecil. Rangkaian yang digunakan adalah sesuai dengan gambar:



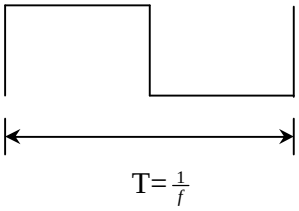
Gambar 5.3 Skema rangkaian speaker

Tinggi rendah nada tergantung pada frekuensi yang digunakan.

Program:

```
#include <90s8515.h>
#include <delay.h>
#include <nada.h>
main()
{
DDRD=0xFF;
#asm
cli
#endasm
while(1)
{
mi_1per2();
mi_1per2();
do_1();
do_1();
do_1();
diam_2();
do_1per2();
do_1per2();
ssi_1per2();
do_1per2();
re_1per2();
diam_2();
la_1per2();
la_1per2();
re_1();
re_1();
re_1();
diam_2();
re_1per2();
re_1per2();
do_1per2();
re_1per2();
mi_1per2();
diam_2();
do_1per2();
mi_1per2();
sol_1();
sol_1();
sol_1();
diam_2();
la_1per2();
sol_1per2();
la_1per2();
si_1per2();
doo_1per2();
diam_4();
diam_4();
}
}
```

Program di atas berisikan urutan-urutan nada dengan tinggi rendah yang berbeda-beda. Fungsi-fungsi tersebut berupa nada si rendah sampai dengan mi tinggi. Masing-masing nada memiliki variasi setengah ketukan, satu ketukan, dua ketukan, dan empat ketukan. Tinggi rendah nada tergantung dari frekuensi yang diberikan. Untuk menghasilkan frekuensi seperti yang dikehendaki dilakukan dengan memberikan sinyal kotak dengan frekuensi tertentu pada PortB. Proses ini dijelaskan pada Gambar 5.7



Gambar 5.4 Gambar satu gelombang keluaran penghasil nada

Tugas dari mikrokontroler adalah mengirimkan logika tinggi dengan durasi waktu $\frac{1}{2}$ T, dan mengirim logika rendah dengan durasi waktu $\frac{1}{2}$ T. Sebagai contoh diberikan potongan listing program dari file library nada.lib

```
void re_1(void)
{
for(I=1;I<=(108);I++)//nada re
{
PORTD=0xFF;
delay_us(1743);
PORTD=0x00;
delay_us(1743);
}
diam_1per8();
}
```

Fungsi di atas akan menghasilkan nada re dengan 1 ketukan. Nada re dihasilkan dengan memberikan logika tinggi ke PORTB dengan durasi waktu 1743 mikro detik, kemudian dengan memberikan logika rendah dengan durasi waktu 1743 mikro detik. Gelombang kotak ini diulangi sebanyak 108 sehingga menghasilkan 1 ketukan nada. Gelombang kotak ini diberikan ke basis transistor A733 untuk mengendalikan speaker.

V. PENUTUP

5.1 Kesimpulan

Setelah menganalisa perangkat keras, perangkat lunak, program bantu, developer tool, dan contoh aplikasi dari mikrokontroler AT90S8515, maka penulis dapat menarik kesimpulan sebagai berikut:

6.1.1 AT90S8515 memiliki fitur yang handal untuk perancangan suatu sitem, sehingga pada masa yang akan datang mempunyai kesempatan yang besar untuk diterapkan pada banyak aplikasi.

6.1.2 Dengan harga yang terjangkau, maka mikrokontroler ini bagus untuk proyek-proyek eksperimen para mahasiswa.

6.1.3 AT90S8515 dapat diprogram menggunakan bahasa pemrograman dengan standard ANSI-C

6.1.4 Embedded C atau program bahasa C untuk mikrontroler AVR 90S8515 mempunyai kemampuan yang handal karena prinsip logikanya menggunakan bahasa tinggat tinggi sehingga mendekati ke logika manusia. Selain itu

bahasa C mampu mengakses suatu memori atau register dalam level bit.

6.1.5 Program hasil kompilasi dari bahasa tingkat tinggi akan lebih besar daripada hasil kompilasi dari bahasa tingkat menengah dalam hal ini assembly.

6.1.6 File-file fungsi yang terdapat pada file library sangat membantu dalam perancangan program.

6.1.7 Mempelajari sintaks dalam bahasa C lebih sulit dibandingkan dengan bahasa assembly tetapi pada proses selanjutnya perancangan program akan sangat lebih mudah menggunakan bahasa C.

5.2 **Saran**

Agar sistem yang dibahas oleh penulis lebih bermanfaat, penulis menyampaikan saran-saran sebagai berikut:

6.2.1 Karena compiler yang digunakan oleh penulis menggunakan CodeVisionAVR dengan versi *evaluation* yang berarti program yang dapat dikompilasi terbatas, maka penulis menyarankan untuk menggunakan compiler yang sifatnya cuma-cuma seperti AVR GNU GCC.

6.2.2 Contoh-contoh program yang diberikan oleh penulis sifatnya masih sangat sederhana, tetapi merupakan aplikasi mendasar. Oleh karena itu penulis menyarankan untuk mengkombinasikan aplikasi-aplikasi tersebut pada program yang kompleks.

6.2.3 *Developer-tool* dari port *printer* lebih murah dan mudah untuk dibuat. Selain itu tidak akan mengganggu ketika aplikasi yang digunakan adalah untuk komunikasi serial dengan menggunakan port serial COM1/COM2.

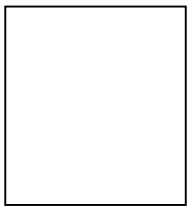
6.2.5 Karena kemampuan ISP (*In System Programming*) yang ada pada AVR 90S8515, maka akan sangat bagus untuk dijadikan modul-modul praktikum.

6.2.6 Karena program hasil kompilasi bahasa C lebih besar daripada program hasil kompilasi bahasa assembly maka alokasi memori adalah hal yang penting untuk dipertimbangkan.

6.2.7 Pengetahuan tentang bahasa assembly tetap diperlukan karena bahasa C yang digunakan sering mengikutsertakan baris-baris dalam bahasa assembly.

DAFTAR PUSTAKA

1. Daniel Tabak; Advanced Microprocessors, Second Edition; McGRAW-HILL INTERNATIONAL EDITIONS Electrical and Electronic Technology Series; 1995.
2. Jogiyanto Hartono, MBA, Ph.D; Konsep Dasar Pemrograman Bahasa C, Edisi Kedua; ANDI; 2000.
3. Abdul Kadir; Pemrograman Dasar Turbo C untuk IBM PC; ANDI; 1997.
4. www.atmel.com
5. www.hpinfotech.ro



Fajar Mahadmadi, lahir di Magelang pada tanggal 25 Juni 1980 pada saat ini sedang menyelesaikan pendidikan S-1 di Jurusan Teknik Elektro Universitas Diponegoro dengan konsentrasi Elektronika.

Menyetujui/Mengesahkan,

Pembimbing I	Pembimbing II
Ir. Sudjadi, MT	Trias Andromeda, ST, MT
NIP 131 558 567	NIP 132 283 185