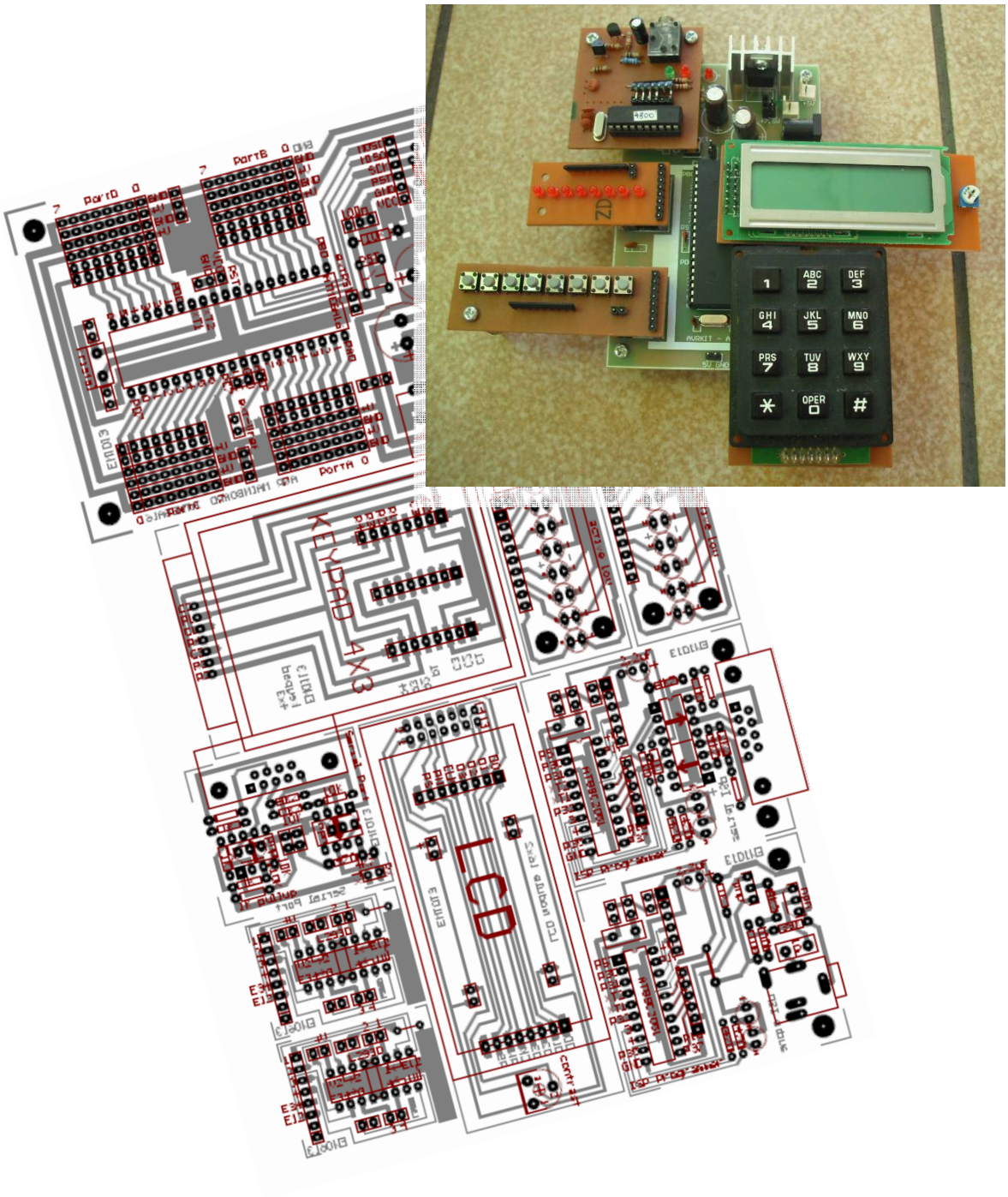
PETUNJUK PENGGUNAAN
KIT AVR

Eko Widiatmoko (e_ko_w@yahoo.com)
2015



DAFTAR ISI

BAB I
TENTANG ELEKTRONIKA DIGITAL DAN MIKROKONTROLER

Pendahuluan

Tulisan ini adalah petunjuk penggunaan kit AVR. Hal-hal yang dibahas mencakup sedikit tentang elektronika digital, "bagian dalam" mikrokontroler, cara kerja komponen-komponen dalam kit, dan cara menyusun program. Pembahasan dalam tulisan ini bertujuan untuk membangun pemahaman tentang cara bekerja dengan mikrokontroler AVR dan komponen-komponen pendukungnya.

Elektronika Digital

Sinyal digital memiliki dua keadaan, yaitu high (1) dan low (0) yang dinyatakan dengan tegangan listrik. Setiap keadaan mempunyai jangkauan tegangan tertentu, misalnya untuk keadaan low berkisar antara 0-0,9V dan untuk keadaan high berkisar 2 - 5V. Berikut ini adalah istilah-istilah yang sering dipakai dalam elektronika digital.

Clock	: Pulsa-pulsa yang waktunya teratur. Atau, bagian sistem yang mengatur kecepatan "langkah" jalannya program. Dalam kit ini digunakan clock 11,0592 MHz.
Transisi naik	: perubahan dari keadaan low menjadi high.
Transisi turun	: perubahan dari keadaan high menjadi low.
Aktif high	: fungsi bekerja ketika keadaan logika high.
Aktif low	: fungsi bekerja ketika keadaan logika low.
High Impedance (HI-Z)	: keadaan mengambang, seolah-olah tidak terhubung.
Tri-state buffer	: Bisa memiliki tiga keadaan yaitu high, low, HI-Z.

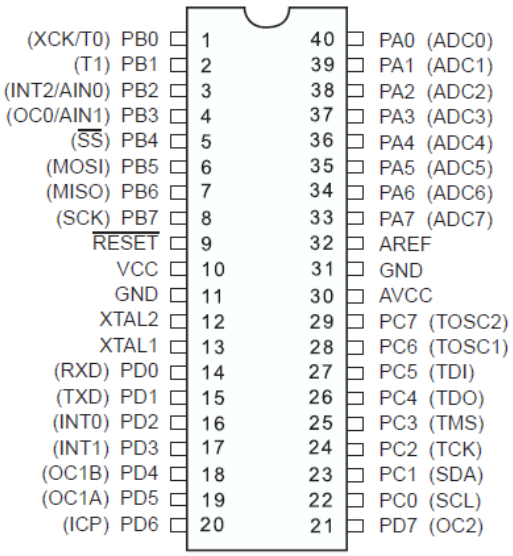
Mikrokontroler

Mikrokontroler adalah sebuah komputer lengkap yang terdapat dalam satu chip. Terdapat komponen-komponen dasar yang menyusun sebuah komputer, yaitu prosesor, memori (RAM dan ROM), jalur masukan dan keluaran, serta unit-unit penunjang lainnya. Mikrokontroler biasa digunakan dalam sistem-sistem kecil yang memerlukan kemampuan pengolahan data yang sederhana dan ringkas.

Mikrokontroler yang digunakan dalam kit AVR adalah ATMega16 yang dibuat oleh Atmel. Ini adalah satu dari sekian banyak jenis mikrokontroler dalam keluarga AVR. ATMega16 dipilih karena cukup umum digunakan dan memiliki fasilitas yang memadai untuk belajar menggunakan mikrokontroler.

Fasilitas yang dimiliki ATMega16 antara lain:

- Memori program (flash) sebesar 16 KB
- EEPROM (memori tetap) sebesar 512 byte
- SRAM (memori kerja) sebesar 1 KB
- Tegangan operasi 4,5V – 5,5V
- Kecepatan *clock* dari 0 – 16 MHz
- Clock internal dengan osilator RC 1-8 MHz
- 4 Port, masing-masing 8 bit, susunannya terlihat dalam Gambar 1.2.
- 8 jalur ADC 10 bit
- 8 sumber interrupt
- Antarmuka ISP untuk memasukkan program
- Modul USART dan SPI
- Mode hemat daya *idle* dan *power down*
- *Watchdog Timer* yang dapat diprogram



Pin dan Port

Mikrokontroler ATMega16 memiliki 32 pin untuk port yang dikelompokkan menjadi port A, B, C, dan D. Port yang berfungsi sebagai jalur masukan dan keluaran atau bagian dari sistem pengalamatan dan jalur data. Semua pin pada port dapat menghasilkan keadaan high atau low dengan kemampuan mengalirkan arus maksimal 20 mA dan keadaan *high impedance* ketika berfungsi sebagai masukan.

Setiap port memiliki tiga tempat di memori. Satu menyatakan nilai yang dikeluarkan pada port, yang kedua menyatakan nilai yang terbaca pada pin port, dan yang ketiga adalah kode yang menandakan fungsi pin-pin pada port sebagai masukan atau keluaran.

Port A

Port A terletak di kanan atas PCB (dengan arah sedemikian sehingga tulisan ATMEGA16 terbaca). Port A adalah jalur masukan untuk *Analog to Digital Converter* (ADC).

Port B

Port B terletak di kiri atas PCB. Port B digunakan dalam pengisian program melalui sistem ISP (*In System Programming*), modul SPI (*Serial Peripheral Interface*), dan pin masukan Timer 0 dan Timer 1.

Port C

Port C terletak di kanan bawah PCB. Urutan bit-bit port C terbalik jika dibandingkan dengan port lainnya. Port C digunakan untuk jalur JTAG dan TWI (*Two-Wire Interface*).

Port D

Port D terletak di kiri bawah PCB. Dalam port D terdapat jalur USART dan dua jalur interrupt.

RST (Reset)

Jika pin ini diberi sinyal *low*, mikrokontroler akan memulai program dari awal dan mengisi semua register dengan nilai awal. Pin ini terhubung dengan tombol kecil di PCB.

Pengiriman Program

Program dalam mikrokontroler disimpan dalam memori flash. Program ditulis di komputer dalam bahasa assembly atau C, kemudian dilakukan *compile* untuk mendapatkan file dalam bahasa mesin yang berisi angka-angka heksadesimal. File ini, yang memiliki nama dengan *extension* .HEX, dikirim ke dalam mikrokontroler dengan programmer.

Tersedia dua macam programmer dalam kit. Jenis pertama menggunakan komunikasi serial (RS-232) dengan kecepatan 19200 bps. Untuk komputer yang tidak memiliki jalur serial (COM1) diperlukan pengubah USB ke serial. Programmer jenis kedua menggunakan jalur *line out* atau headphone komputer. Dibandingkan dengan programmer serial, metode ini lebih lambat dan tidak bisa mengirim balik data dari mikrokontroler ke komputer, tetapi dapat digunakan di sebagian besar komputer tanpa komponen tambahan.

BAB II
BAHASA C

Pendahuluan

Contoh-contoh program di sini diberikan dalam Bahasa C. Bahasa ini dipilih karena umum digunakan dalam segala macam pemrograman. Bahasa assembly, yang telah digunakan selama puluhan tahun untuk memprogram mikrokontroler dan komputer, lebih sulit dicerna dan dirunut jalannya walaupun paling ringkas dan cepat.

Bahasa C terdiri dari kalimat-kalimat perintah menyerupai operasi matematika. Pada umumnya setiap perintah bertujuan mengatur nilai suatu objek. Objek ini bisa merupakan variabel yang tersimpan dalam RAM atau register khusus dalam mikrokontroler.

Tatacara Penulisan Program

Dalam bahasa C, setiap kalimat adalah operasi matematika atau perintah. Kalimat selalu diakhiri dengan tanda `;` (titik koma). Perintah biasanya diikuti dengan tanda kurung biasa `( )` yang berisi pernyataan yang menjelaskan perintah itu. Kumpulan kalimat biasanya ditulis di antara tanda kurung kurawal `{ }`. Penggunaan huruf besar dan huruf kecil harus diperhatikan karena artinya berbeda. Untuk menuliskan kata-kata tambahan yang tidak berhubungan dengan jalannya program, tulisan diawali dengan `//` atau ditaruh di antara `/*  */`

Sebuah program dalam bahasa C mungkin terlihat seperti ini:

```
/*
Program menyalakan deret led
Nyalakan led di port B sesuai angka yang terus bertambah.
*/
int main()
{
	char a,b;	//angka untuk hasil perhitungan
	a = 0;
	while(1)	//ulangi terus-menerus
	{
		PORTB = a;	//keluarkan nilai a di port B
		a = a + 1;
		for(c = 0; c<100; c++); //tidak melakukan apa-apa
	}
}
```

Matematika, Bilangan Biner dan Heksadesimal

Dalam mikrokontroler, karena keterbatasan memori dan kecepatan, biasanya digunakan bilangan 8 bit (`char`) atau 16 bit (`integer`). Bilangan 8 bit memiliki jangkauan sebesar 256, dan bilangan 16 bit memiliki jangkauan sebesar 65536.

Satu byte, dalam komputer, setara dengan gabungan 8 bit bilangan biner. Setiap bit bernilai sama dengan suatu pangkat dari 2, misalnya:

Bilangan biner: 1001 0011

Bit	1	0	0	1	0	0	1	1
digit	7	6	5	4	3	2	1	0
nilai	128	64	32	16	8	4	2	1

Dalam desimal, bilangan ini sama dengan 128 + 16 + 2 + 1 = 147.

Bilangan juga bisa dinyatakan dalam sistem heksadesimal. Dalam sistem bilangan ini ada 16 simbol untuk menyatakan angka, yaitu:

0 1 2 3 4 5 6 7 8 9 A B C D E F

Karena jumlah semua kemungkinan dari susunan 4 digit biner juga sama dengan 16, jadi empat bit bisa dinyatakan dengan satu angka heksadesimal, dan satu byte dengan 2 angka heksadesimal.

Tabel 2.1: Bilangan biner, heksadesimal, dan desimal untuk 0-15.

Biner	Heksadesimal	Desimal	Biner	Heksadesimal	Desimal
0000	**0**	0	1000	**8**	8
0001	**1**	1	1001	**9**	9
0010	**2**	2	1010	**A**	10
0011	**3**	3	1011	**B**	11
0100	**4**	4	1100	**C**	12
0101	**5**	5	1101	**D**	13
0110	**6**	6	1110	**E**	14
0111	**7**	7	1111	**F**	15

Misalnya:
1001 0011 = 0x93
1110 1000 = 0xE8

Operasi matematika antara 2 bilangan dalam bahasa C adalah:
+ tambah
- kurang
* kali
/ bagi
% modulo (sisa pembagian)
& bitwise AND
| bitwise OR
^ bitwise XOR
~ invers bit
<< geser kiri / left shift
>> geser kanan / right shift

Ketika menggunakan bilangan bulat, pembagian selalu menghasilkan bilangan bulat hasil pembulatan ke bawah. Jika operasi matematika pada bilangan menyebabkan angkanya lebih dari batas maksimum, bilangan akan kembali mulai dari nol.

Tips:
- Perhatikan lagi jenis bilangan sebelum mulai menghitung sampai 50000.
- `7 / 3 = 2`
- `(char) 255 + 2 = 1`
- `a + a = a * 2 = a << 1`

Semua perintah di atas biasanya ditulis dalam bentuk seperti ini:

```
a = b + 5;
a = d & 21;
c = c >> 2;
c = c / d;
```

Dalam bahasa C tidak dikenal perintah *bit set* dan *bit clear* seperti pada bahasa assembly.
Misalnya:
a = 00010001
set bit 7 pada a :
a = 10010001
clear bit 0 dari a :
a = 10010000

Untuk mencapai hal ini, perintah itu harus dibuat sendiri dengan operasi AND dan OR, misalnya:

```
a = a | 0x80; //set bit tertinggi milik a
a = a & 0xfe; //clear bit 0 milik a
```

Operasi yang bekerja pada satu bilangan saja adalah:

++	bertambah 1 (increment)
--	berkurang 1 (decrement)
-	negatif

Operasi perbandingan antara dua keadaan adalah:

==	periksa apakah dua angka sama.
>, >=, <, <=	periksa apakah angka yang satu lebih besar, lebih besar/sama, dsb.
!=	periksa apakah dua angka tidak sama.
&&	logic AND (untuk menggabungkan dua perbandingan)
||	logic OR
^^	logic XOR
!	logic NOT

Misalnya:

```
if( !(a & 1) || (b == 2) )
//jika bit terendah dari a tidak bernilai 1, atau b sama dengan 2...
```

Bentuk Program

Dalam berbagai C compiler, pertama kali kita perlu menuliskan kalimat #include. Ini diperlukan supaya compiler mengenali nama-nama register khusus yang berbeda-beda tergantung jenis mikrokontroler, misalnya:

menggunakan WinAVR:

```
#include<avr/io.h>
```

Selanjutnya, program harus mengandung sebuah fungsi utama yang bernama **`main`**. Program akan mulai berjalan dari sini.

```
int main()
{
	PORTA = 255; //set semua pin pada port A menjadi high
	…
}
```

Register

Memori mikrokontroler terdiri dari sejumlah byte yang memiliki alamat masing-masing. Di antara semua byte itu, sebagian dapat digunakan dengan bebas. Bagian inilah yang digunakan untuk menyimpan variabel-variabel yang kita definisikan. Tetapi, ada bagian khusus yang memiliki arti yang sudah ditetapkan. Bagian ini disebut *special function register* (SFR).

Setiap byte, yang disebut *register*, dalam bagian ini berhubungan dengan fungsi tertentu dalam mikrokontroler. Misalnya, untuk menyetel pin-pin pada port A, kita menuliskan angkanya pada register PORTA. Untuk membaca nilai masukan yang terbuhung dengan port A, kita membaca nilai register PINA. Untuk mengatur fungsi port A sebagai masukan atau keluaran, kita mengatur register DDRA. Juga ada register-register lain yang menandai apa yang terjadi selama program berjalan dan mungkin saja nilainya berubah sendiri tergantung keadaan.

Dalam program, kita menyebutkan nama register yang dimaksud untuk memperoleh nilainya atau mengisinya dengan angka. Beberapa bit dalam register tertentu bahkan memiliki nama tersendiri yang bisa disebutkan dalam program. Beberapa register yang sering dipakai akan dibahas satu persatu dalam bab terakhir.

Variabel

Semua variabel harus dideklarasikan sebelum digunakan. Tempat penulisan deklarasi menentukan tempat berlaku variabel itu. Jika variabel dibuat dalam fungsi utama (`main`), maka fungsi lain tidak bisa membaca variabel itu. Untuk membuat variabel yang berlaku di mana-mana, tulis deklarasinya di luar fungsi, di awal program.

Jenis variabel ditulis pada deklarasi. Yang biasa digunakan adalah:

`char`	: -127 sampai 127
`unsigned char`	: 0 sampai 255
`int`	: -32767 sampai 32767
`unsigned int`	: 0 sampai 65535

Misal:

```
char lama; //deklarasi variabel global

void tunggu()
{
	unsigned char c; //deklarasi variabel lokal
	for(c = 0; c < lama; c ++); //tidak melakukan apa-apa
}

int main()
{
	unsigned char c; //c yang ini beda dengan c milik fungsi tunggu.
	PORTA = 0xff;
	lama = 200; //variabel ini bisa dipakai di mana-mana karena global.
	tunggu();
	…
```

Percabangan dan Perulangan

Seringkali kita ingin program melakukan hal yang berbeda tergantung nilai suatu variabel atau keadaan. Maka, dibuat percabangan. Dalam bahasa C ada dua macam percabangan yaitu `if` dan `switch`. Percabangan dengan `if` bisa digunakan untuk segala macam keadaan, tetapi `switch` hanya bisa memilih berdasarkan satu angka. Perintah yang dilaksanakan untuk tiap keadaan dikelompokkan dengan `{ }`. Misalnya:

```
if(a < 3 || b == 10) PORTA = 0xff; //perintah satu baris bisa langsung, tidak perlu { }
else PORTA = 0;

if(a<10)
{
	PORTA = 0xff;
}
else if(a<50)
{
	PORTA = 0x0f;
}
else
{
	PORTA = 0;
}

switch(b)
{
	case 0:
	PORTA = 0x10;
	break;
	case 1:
	PORTA = 0x20;
```

```
        break;
        case 2:
        PORTA = 0x30;
}
```

Untuk melakukan perintah berulang-ulang, digunakan perulangan `while` atau `for`. Perulangan dengan `while` memerlukan syarat keluar, yaitu keadaan yang menyebabkan program keluar dari perulangan. Sedangkan, perulangan dengan `for` bisa menggunakan variabel yang selalu bertambah atau berkurang setiap kali ulangan, dan syarat mengulang.

```
a = 1;
while(a != 0)
{
        PORTA = a; //a akan bernilai 1,2,4,8,16,32,64,128,0. Mengapa?
        a = a + a;
}

for(c = 0; c < 50; c++) PORTA = c;      //nilai c adalah dari 0 - 49
```

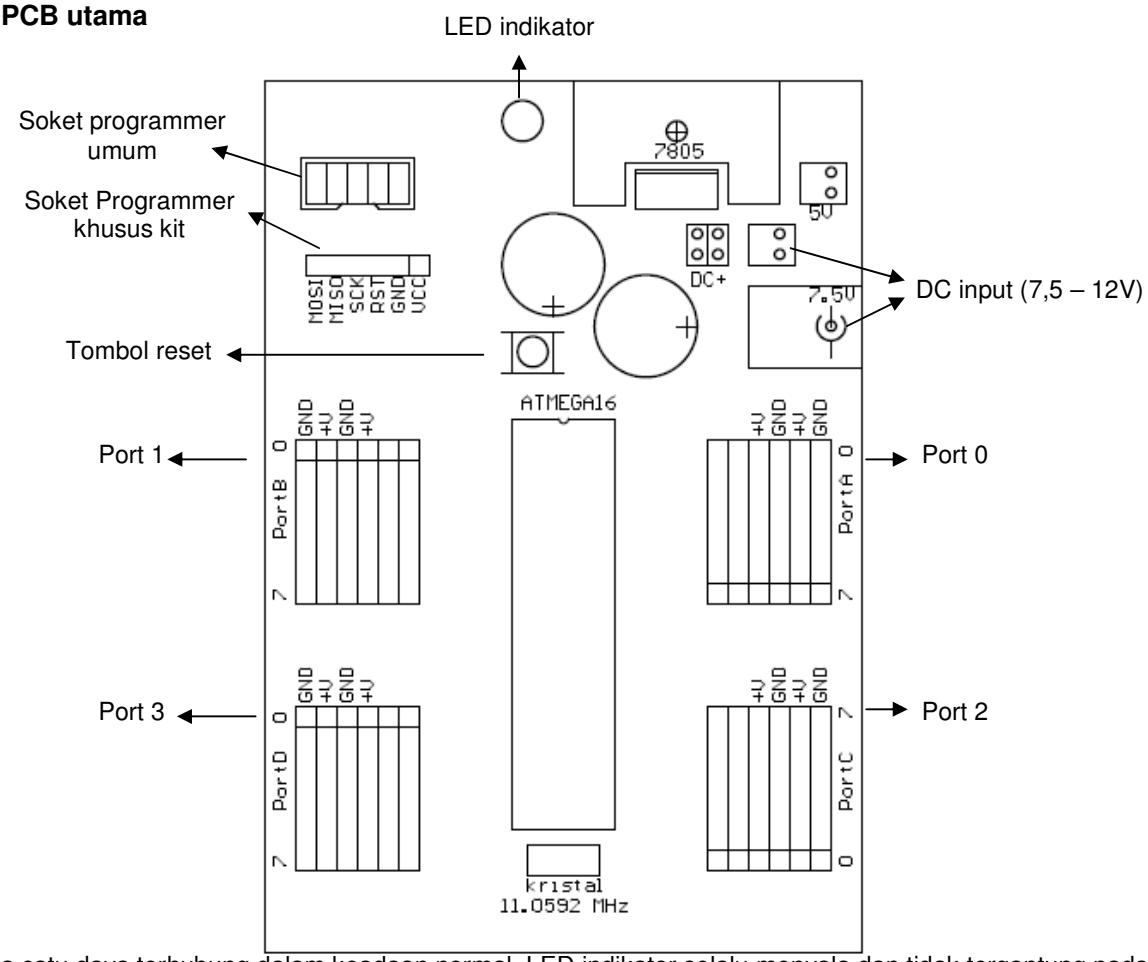
Keterangan tambahan

Ketika menggunakan compiler seperti WinAVR, mungkin perlu disetel optimisasi –O0. Optimisasi artinya kita membiarkan compiler mengubah struktur program yang dibuat menjadi lebih pendek atau ringkas dengan mempertahankan fungsi program. Misalnya, jika compiler mengetahui bahwa hasil yang sama dapat dicapai dengan perintah yang lebih sedikit, maka proses optimisasi akan mempersingkat program. Setelan –O0 mengatur supaya optimisasi tidak dilakukan. Hal ini berdampak pada kode hasil compile yang agak panjang, tetapi perintah seperti contoh berikut ini, yang fungsinya hanya membuat program menunggu beberapa saat, bisa dipakai.

```
for(c=0;c<30000;c++); //tidak melakukan apa-apa selama 30000 hitungan.
                      //Jika menggunakan optimisasi, bagian ini akan lenyap.
```

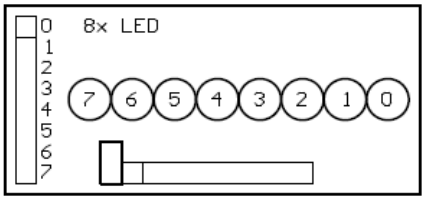
BAB III
RANGKAIAN KIT AVR

1. PCB utama



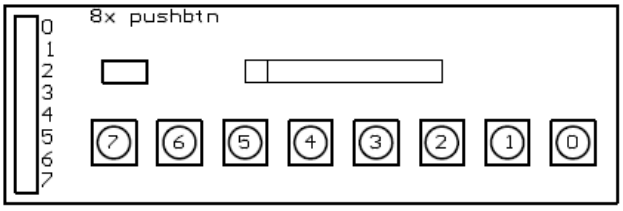
Jika catu daya terhubung dalam keadaan normal, LED indikator selalu menyala dan tidak tergantung pada program mikrokontroler.

2. Modul deret LED



LED menyala jika pin diberi logika low. Setiap LED terhubung dengan resistor seri.

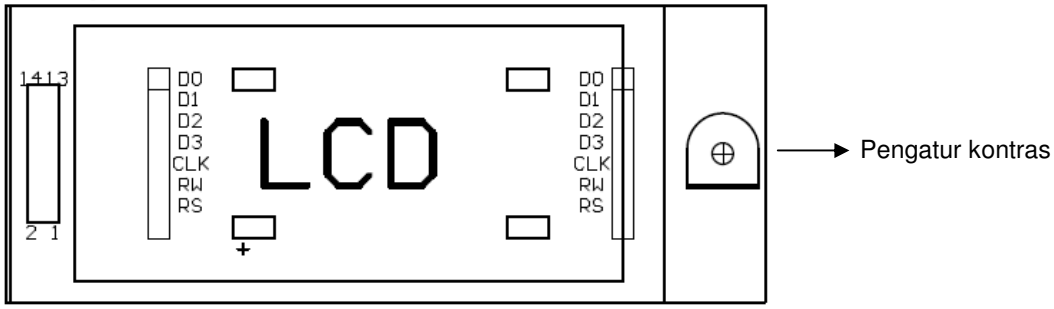
3. Modul deret tombol



Jika tombol tidak ditekan maka pin dalam keadaan high dengan resistor pull-up 1 KΩ, jika ditekan maka pin langsung terhubung dengan ground.

4. Modul LCD 16x2

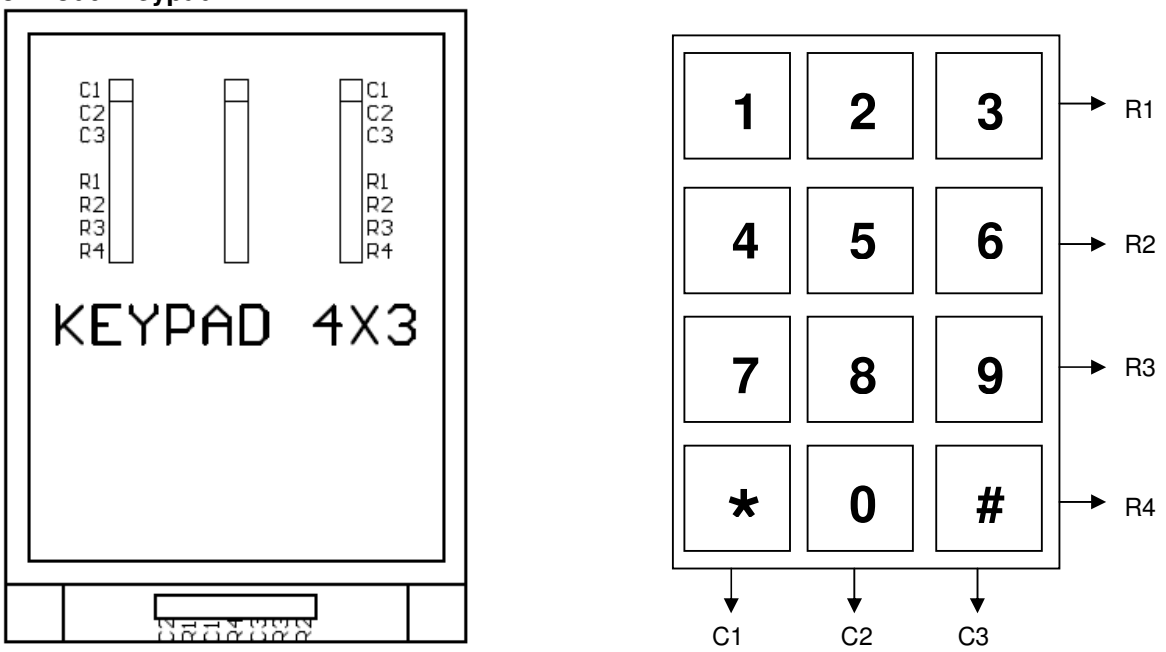
Modul LCD yang termasuk dalam kit AVR adalah modul alfanumerik 16 karakter x 2 baris tanpa lampu belakang.



Modul LCD bisa dipasang di sisi kiri atau sisi kanan PCB utama tanpa perlu dibalik. Digunakan protokol 4 bit.

D0-D3 : Data. Kirim data bit tinggi, kemudian data bit rendah.
CLK : Clock / Enable. Data diterima oleh modul saat transisi high-low.
RW : Read / Write. 0 = write, mengirim data ke modul
RS : Register Select. 0 = perintah, 1 = data

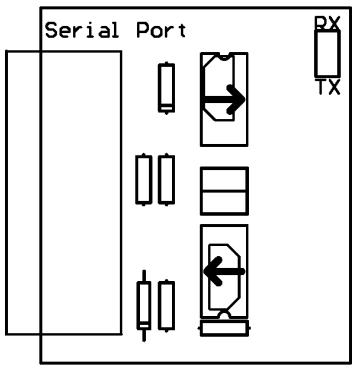
5. Modul keypad



Pin-pin modul ini tidak terhubung dengan ground atau +5V tetapi hanya terhubung dengan port. Deretan header tengah hanya berfungsi sebagai penyangga supaya kedudukan modul pada PCB utama lebih kokoh.

C1-C3 : *Column* / Kolom 1-3
R1-R4 : *Row* / Baris 1-4

6. Modul serial port

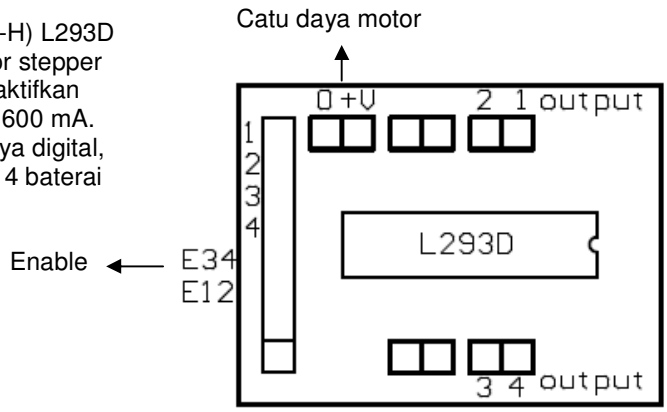


Modul ini hanya bisa dipasang di Port D0 dan D1 karena di situlah terletak pin yang berhubungan dengan modul USART pada AVR. Secara teknis modul ini bisa saja dipasang di mana saja asalkan mendapatkan catu daya, tetapi jika menggunakan pin lain, pengguna harus menulis program khusus untuk mengendalikan komunikasi serial tidak dengan modul USART.

Terdapa dua versi modul ini, satu dengan transistor dan lainnya dengan optocoupler. Jika menggunakan optocoupler, komputer (atau sisi seberang) harus mengaktifkan jalur RTS (pin 7 pada DB-9)

7. Modul DC motor driver

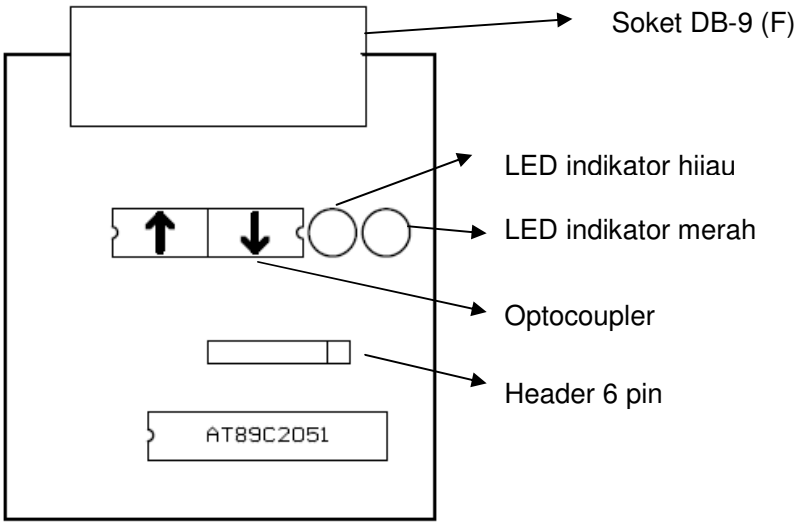
Modul ini menggunakan dual H-bridge (quad half-H) L293D untuk mengendalikan 2 motor DC atau satu motor stepper bipolar. Bit Enable diberi logika high untuk mengaktifkan pasangan dua jalur. Kemampuan arus maksimal 600 mA. Catu daya motor diperlukan terpisah dari catu daya digital, dengan tegangan lebih dari 5V, misalnya 6V dari 4 baterai AA.



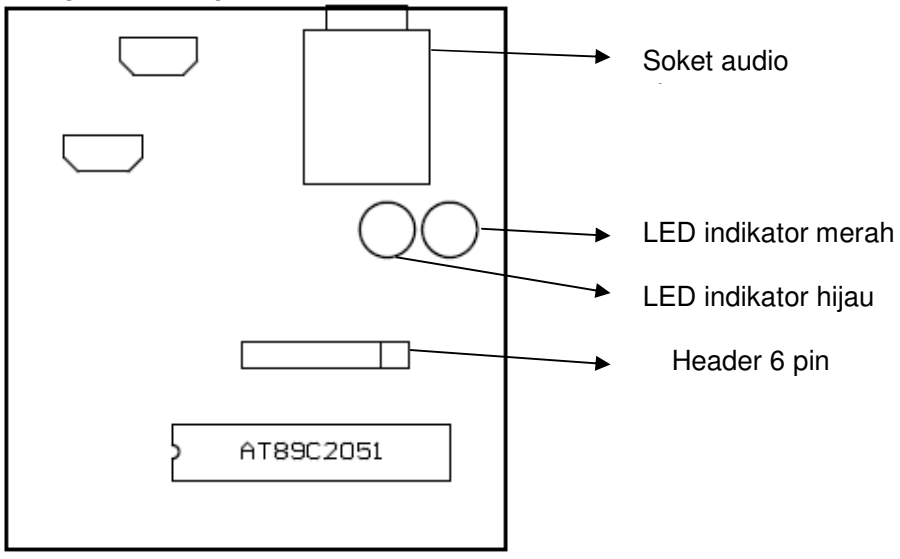
Pin 1-4 : Sinyal kontrol. Satu motor DC menggunakan 2 jalur. 01 = maju, 10 = mundur, 00/11 = rem.
E12, E34 : Sinyal Enable untuk menjadikan dua pin output "high impedance" untuk mematikan motor tanpa rem (floating).

8. Programmer

1. Programmer dengan komunikasi serial



2. Programmer dengan kabel audio

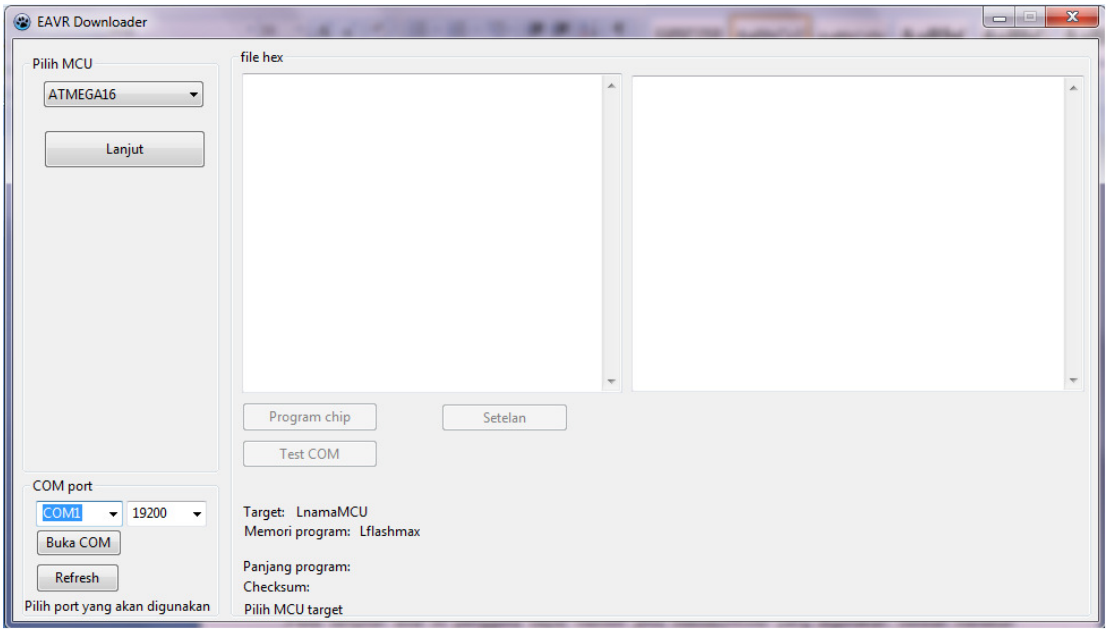


Modul programmer berfungsi memprogram memori flash dalam mikrokontroler. Tersedia dua macam programmer, dibedakan oleh metode yang digunakan untuk berkomunikasi dengan komputer. Jenis pertama menggunakan komunikasi serial (RS232) dengan kecepatan 19200 baud melalui kabel serial menuju port serial komputer (COM1 atau COM2), atau menggunakan penghubung USB ke serial menuju port USB komputer. Jenis kedua menggunakan komunikasi serial yang lebih lambat (4800 baud) melalui kabel audio.

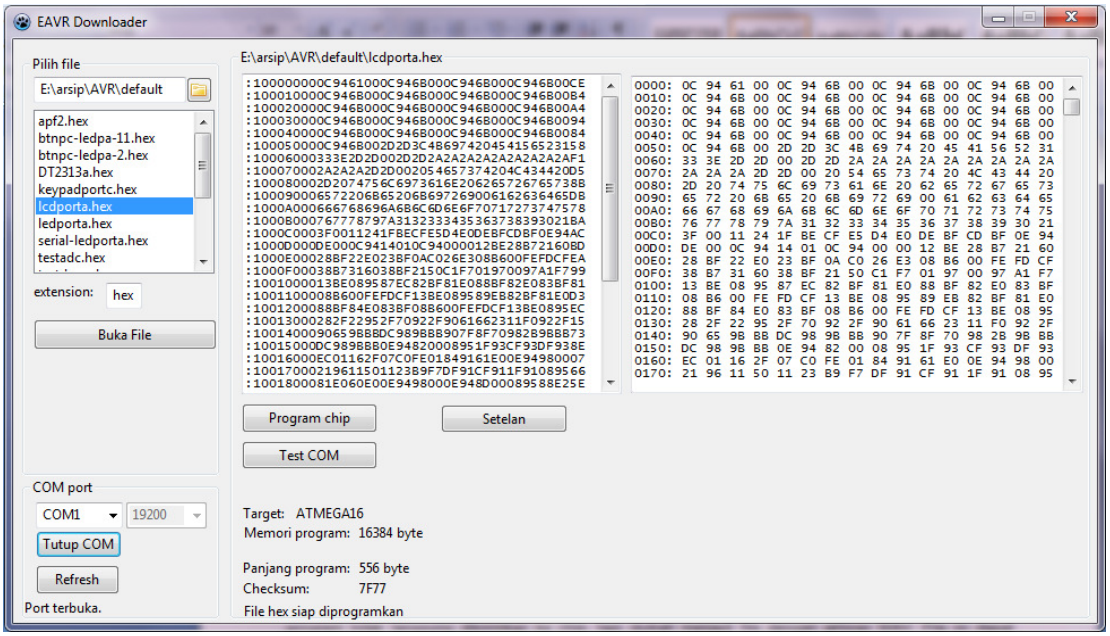
Ketika modul programmer dinyalakan dengan cara menghubungkan ke PCB utama dan menghubungkan catu daya, LED hijau akan berkedip dua kali lalu menyala. Ketika program dikirimkan, LED hijau akan berkedip sekali di awal, lalu LED merah berkedip beberapa kali, lalu LED hijau berkedip sekali di akhir komunikasi.

Program untuk memasukkan program

Untuk mengirim program yang sudah dibuat ke dalam chip, tersedia program EAVRProg yang dijalankan di komputer. Tampilan program ini tampak pada gambar berikut.



Pada tampilan awal ini pengguna dapat memilih jenis mikrokontroler yang digunakan. Setelah menekan tombol **OK**, fungsi lainnya akan terbuka. Tampilan program berikutnya tampak seperti gambar berikut.

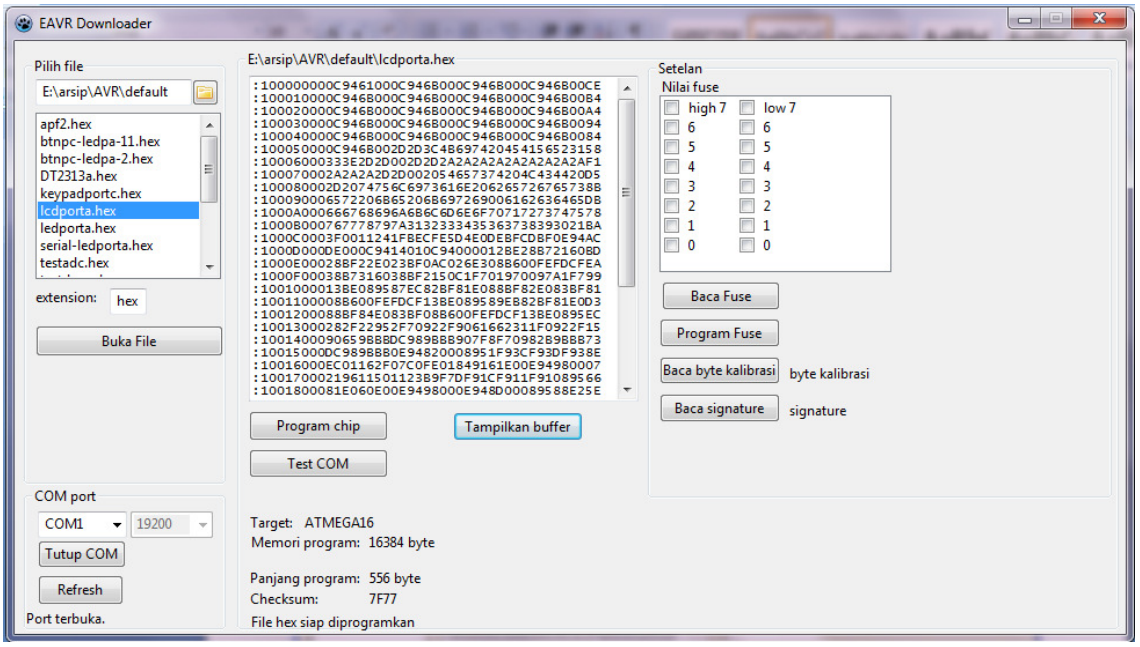


Pada tampilan ini pengguna dapat memilih file hex yang akan diprogramkan pada chip. Tekan **Buka File** untuk memilih file. Untuk mengirimkan program dapat digunakan dua cara, yaitu dengan serial port dan audio port.

Serial Port
Dalam menggunakan serial port, perlu diperhatikan nomor COM yang dipakai. Serial port asli dari komputer biasanya memiliki nomor 1. Serial port virtual dari USB-RS232 converter memiliki nomor 3 atau lebih. Untuk mengaktifkan serial port tekan tombol **Buka COM**. Baud rate yang digunakan adalah 19200 bps. Gunakan tombol **Test COM** untuk memeriksa kelancaran komunikasi dengan alat.

Audio Port
Jika tidak tersedia serial port, program dapat dikirimkan dalam bentuk gelombang listrik melalui kanal Line Out atau Headphone yang tersedia di sebagian besar komputer. Ketika tombol **Program Chip** ditekan, program tidak langsung dikirimkan ke chip, tapi diubah menjadi file dengan akhiran WAV. File ini dapat dibunyikan dengan program seperti Windows Media Player. Ketika dibunyikan dengan kabel tepasang, program akan terkirim ke alat. Perlu diperhatikan bahwa pengaturan efek suara dalam komputer bisa memengaruhi kelancaran pengiriman data. Pengaturan kerasnya suara (volume) juga perlu diperhatikan.

Fungsi Sampingan
Terdapat beberapa fungsi tambahan yang disediakan dalam program EAVRProg. Fungsi-fungsi ini dapat dimunculkan dengan menekan tombol **Setelan**. Fungsi sampingan hanya bisa dilakukan ketika menggunakan programmer dengan serial port. Tampilan program akan berubah seperti gambar berikut ini.



Fuse pada AVR
Fuse mirip dengan saklar-saklar yang terlihat pada berbagai pesawat luar angkasa dalam film, yang diaktifkan satu persatu menjelang peluncuran. Dalam datasheet setiap mikrokontroler AVR terdapat keterangan lengkap mengenai arti setiap bit fuse dan hal-hal yang perlu diperhatikan dalam mengatur fuse. Untuk penggunaan biasa pada kit AVR dapat digunakan setelan fuse berikut ini:

- Fuse high : 1100 1001 (0xC9)
- Fuse low : 1110 1111 (0xEF)

Jika program tampaknya tidak terkirim dengan baik, dapat dilakukan langkah-langkah berikut ini.
1. Kirimkan ulang program.
2. Reset programmer dengan cara mematikan catu daya lalu menyalakannya lagi.

BAB IV
MENJALANKAN MODUL-MODUL

1. Modul deret LED

Setiap LED tersambung dengan satu pin pada port, jadi LED yang menyala mewakili nilai port. LED akan menyala jika diberi logika low.

Contoh program berikut ini akan membuat LED menyala berkedip.

```
//led berkedip di port A
#include<avr\io.h>
int main()
{
	unsigned int c;
	DDRA = 0xff; //set output
	awal:
	PORTA = 0xff;
	for(c=0;c<30000;c++); //tidak melakukan apa-apa supaya program menunggu sebentar
	PORTA = 0;
	for(c=0;c<30000;c++);
	goto awal;
}
```

Pembaca dianjurkan untuk mencoba berbagai variasi dari program-program yang diberikan di sini dan mengamati hasilnya. Misalnya, dapat dibuat variasi dari program di atas:

1. Mengubah lamanya kedipan.
2. Memilih LED yang berkedip.
3. Mengatur 4 LED menyala bergantian dengan 4 LED lainnya.
4. Mengatur supaya lamanya LED menyala berbeda dengan lamanya LED mati.
5. Mengatur supaya periode kedipan tidak konstan.
6. Membuat kombinasi nyala LED lebih dari dua langkah.
7. Membuat efek “LED berjalan”.

Contoh program berikut akan membuat LED yang dipasang di port A menyala bergantian satu demi satu.

```
//led jalan di port A
#include<avr\io.h>
int main()
{
	unsigned char a;
	unsigned int c;
	DDRA = 0xff; //set output
	a = 0;
	while(1) //syarat selalu bernilai tidak nol, jadi selalu mengulang tanpa akhir.
	{
		if(a==0) a=1;
		else a = a << 1;
		PORTA = ~a; //balikkan bit-bit supaya 1 --> nyala.
		for(c=0;c<30000;c++); //tunggu agar tidak terlalu cepat
	}
}
```

Supaya “jalannya” nyala LED bisa berbalik arah, coba program berikut ini.

```
//led jalan bolak balik
#include<avr\io.h>
int main()
{
	unsigned char a, arah;
	unsigned int c;
	DDRA = 0xff; //set output
	a = 1;
	arah = 1;
	while(1)
	{
		if(arah==1) //geser kiri
		{
			a = a << 1;
			if(a==0x80) arah=0;
```

```
            }
            else //geser kanan
            {
                  a = a >> 1;
                  if(a==0x01) arah=1;
            }
            PORTA = ~a;
            for(c=0;c<30000;c++); //tunggu agar tidak terlalu cepat
      }
}
```

Program berikutnya yang dapat dicoba adalah menyalakan LED sehingga terlihat redup. Untuk ini dapat digunakan teknik *pulse width modulation* (PWM). Dengan cara ini, LED dinyalakan sebentar-sebentar dengan sangat cepat sehingga tidak terlihat berkedip. Lamanya nyala dan mati menentukan seberapa terang kelihatannya LED secara keseluruhan. Misalnya, jika LED menyala selama 1 milidetik dan mati selama 4 milidetik secara berulang-ulang, kelihatannya LED itu menyala redup 1/5 kali dari biasanya. Pembaca dipersilakan mencoba sendiri membuat program semacam ini.

2. Modul deret tombol

Setiap tombol berhubungan dengan satu pin pada port. Pin akan bernilai high jika tombol tidak ditekan, dan low jika tombol ditekan. Berikut ini adalah program untuk menyalakan LED sesuai dengan tombol yang ditekan.

```
//modul deret LED di port A
//modul deret tombol di port C
#include<avr\io.h>
int main()
{
      unsigned char a;
      DDRA = 0xff;               //set output
      DDRC = 0; //set input
      while(1)
      {
            a = PINC;            //baca tombol yang ditekan
            PORTA = a;           //tampilkan
      }
}
```

Contoh berikut ini adalah program untuk mengubah nyala deret LED satu langkah setiap kali tombol ditekan.

```
//modul deret LED di port A
//modul deret tombol di port C
#include<avr\io.h>
int main()
{
      unsigned char a, b;
      DDRA = 0xff;               //set output
      DDRC = 0;                  //set input
      b = 0x01;                  //angka yang dikeluarkan ke deret led
      while(1)
      {
            a = PINC;            //baca tombol yang ditekan
            if(a!=0xff)          //kalau ada yang ditekan akan jadi low
            {
                  b = b << 1;    //geser posisi led yang menyala
                  if(b == 0) b = 0x01;
            }
            PORTA = ~b;          //tampilkan
      }
}
```

Jika program di atas dijalankan, maka nyala LED akan bergeser dengan sangat cepat sampai-sampai tidak terlihat, setiap kali tombol ditahan pada posisi tertekan. Untuk membuat nyala LED bergeser satu langkah setiap kali tombol ditekan, program diubah supaya menunggu tombol dilepas sebelum menerima penekanan tombol berikutnya.

```
//modul deret LED di port A
//modul deret tombol di port C
#include<avr\io.h>
int main()
{
      unsigned char a, b, tekan;
```

```
	DDRA = 0xff;		//set output
	DDRC = 0;		//set input
	tekan = 0;		//0: belum ditekan, 1: sedang ditekan
	b = 0x01;		//angka yang dikeluarkan ke deret led
	while(1)
	{
		a = PINC;		//baca tombol yang ditekan
		if(tekan==0)
		{
			if(a!=0xff)
			{
				tekan = 1;
				b = b << 1;  //geser posisi led yang menyala
				if(b == 0) b = 0x01;
			}
		}
		else
		{
			if(a==0xff) tekan = 0;
		}
		PORTA = ~b;		//tampilkan
	}
}
```

Dengan program seperti ini pun kadang-kadang nyala LED bergeser lebih dari satu langkah setiap kali tombol ditekan. Peristiwa ini disebabkan oleh *contant bouncing* atau pantulan, yaitu ketidaksempurnaan kontak listrik pada fisik tombol, mungkin saja karena karat atau kotoran, sehingga perubahan keadaan dari low ke high atau sebaliknya berlangsung tidak tegas. Ketika tombol ditekan, perubahan keadaan low-high terjadi beberapa kali dengan sangat cepat sehingga sistem mendeteksi beberapa kali penekanan tombol.

Untuk mengatasi masalah ini digunakan teknik *debouncing*. Pada dasarnya, jika ditemukan penekanan tombol maka harus dipastikan bahwa sinyal-sinyal gangguan yang berlangsung sangat singkat diabaikan oleh sistem. Suatu sinyal dapat diterima jika lamanya lebih dari batas tertentu. Pembaca dianjurkan untuk mencoba sendiri membuat program seperti itu.

Beberapa program yang dapat dibuat dengan menggunakan modul deret LED dan tombol:

1. LED berjalan yang mulai atau berhenti setiap kali tombol ditekan.
2. LED berjalan dengan kecepatan berbeda tergantung tombol mana yang ditekan.
3. LED berjalan yang semakin cepat jika tombol yang ditekan semakin banyak.
4. LED berjalan satu langkah ke kanan jika ditekan tombol tertentu, dan satu langkah ke kiri jika ditekan tombol lain.
5. LED berkedip yang sesuai dengan tombol yang ditekan. Misalnya ditekan tombol 1, 3, 7, maka LED yang berkedip juga pada posisi 1, 3, 7.
6. LED yang berkedip sesuai dengan tombol yang ditekan, dan terus berkedip setelah tombol dilepas. Pola baru berubah ketika tombol ditekan lagi.
7. Simulasi mesin judi. Ketika tombol ditekan, LED mulai berjalan dengan cepat lalu perlahan-lahan melambat dan berhenti pada satu posisi yang dihitung secara acak atau dengan algoritma tertentu.
8. Permainan “mengikuti pola”. Ditampilkan LED yang menyala bergantian sesuai pola tertentu, lalu orang harus menekan tombol dengan urutan yang sama. Jika salah maka LED menyala dengan pola tertentu. Jika benar maka pola urutan nyala LED diperpanjang (“level berikutnya”)

3. Tampilan LCD

Modul LCD yang terdapat dalam kit sebenarnya merupakan sistem tersendiri yang juga memiliki mikrokontroler untuk menerima perintah dan mengatur huruf-huruf pada layar. Dalam modul ini digunakan tatacara komunikasi 4 bit. Untuk mengirim data 8 bit, setengah data dikirim lebih dulu kemudian setengah data berikutnya menyusul.
Langkah-langkah untuk mengirim data adalah sebagai berikut.

- Bit *read/write* (RW) diberi logika low supaya modul LCD aktif.
- Atur bit *register select* (RS) low untuk perintah dan high untuk data.
- Setengah data (bit 7-4) yang akan dikirim dimunculkan di D3-D0.
- Bit *enable* (CLK) diberi pulsa high.
- Setengah data (bit 3-0) dimunculkan di D3-D0.
- Bit CLK diberi pulsa high.
- Untuk mengirim data berikutnya, tunggu sekitar 40 mikrodetik, kecuali untuk perintah *clear* 1,6 milidetik.

Perintah-perintah yang dapat dikirim ke LCD ditampilkan dalam Tabel 4.1.

Tabel 4.1: Daftar perintah yang bisa dikirim ke modul LCD

Kode Biner	Arti
0000 0001	clear (hapus tampilan dan kembalikan kursor ke posisi awal)
0000 001x	home (kembalikan kursor ke posisi awal)
0000 01ab	character entry mode. a: 1 = increment, 0 = decrement b: 1 = display shift on, 0 = display shift off
0000 1abc	display on/off, set cursor a: 1 = display on b: 1 = cursor underline on c: 1 = cursor blink on
0001 abxx	display/cursor shift a: 1 = display, 0 = cursor, yang bergeser. b: 1 = geser kanan, 0 =kiri
001a bcxx	function set (perintah ini harus dikirim pertama kali) a: 1 = pengiriman data 1x8 bit, 0 = 2x4 bit b: 1 = 2 baris, 0 = 1 baris c: 1 = ukuran karakter 5x10 pixel, 0 = 5x7 pixel
01AA AAAA	set CGRAM address (CG: Character Generator)
1AAA AAAA	set display address (0x00 = baris 1, 0x40 = baris 2) perintah ini sama dengan memindahkan posisi kursor untuk penulisan berikutnya.

Urutan perintah yang biasa digunakan adalah:
0x38 = function set: 8 bit, 2 baris, 5x7 pixel
0x0c = tanpa kursor
0x06 = increment, display tidak bergeser ketika karakter masuk
0x14 = kursor bergerak ke kanan
0x01 = clear

Contoh program untuk menulis di LCD adalah sebagai berikut.

```
//Tulisan berjalan LCD di port A
#include<avr\io.h>
#include<avr\pgmspace.h>

void delay10ms(unsigned int t)
{
	//clock=kristal/8
	//overflow setiap 5400 Hz
	unsigned char c54;
	TCNT0=0;
	TIFR|=0x01;
	TCCR0=0x02; //start timer
	for(;t;t--)
	{
		for(c54=54;c54;c54--)
		{
			while(!(TIFR & 0x01)); //tunggu sampai overflow
			//sesudah overflow:
			TIFR |= 0x01; //clear flag dengan set bit
		}
	}
	TCCR0=0;
}


void lcddelay() //40 mikrosekon
{
	//clock=kristal/8 = 1,38 MHz
	//hitung 56 kali = 40 mikrosekon
	TCNT0 = 255-56;
	TIFR = 0x01; //clear flag dengan set bit
	TCCR0 = 0x02;
	while(!(TIFR & 0x01));
	TCCR0 = 0; //stop
}
```

```
void lcddelaylong() //1.5 ms
{
	//clock=kristal/256 = 43,2 kHz
	//hitung 70 kali = 1,5 ms
	TCNT0=255-70;
	TIFR=0x01;
	TCCR0=0x04;
	while(!(TIFR & 0x01));
	TCCR0=0;
}

void lcdsend(unsigned char d, unsigned char rs)
{
	//port A:
	//0 1 2 3 = data, 4 bit high dulu
	//4 = ck: pulsa high-low-high
	//5 = rw: low
	//6 = rs: 0 = perintah, 1 = data
	//7 = backlight (kalau ada)

	//4 bit tinggi
	unsigned char a;
	a = (d >> 4) | 0x10;
	if(rs) a |= 0x40;
	PORTA = a;
	PORTA &= 0xef;
	PORTA = a;
	//4 bit rendah
	a &= 0xf0;
	a |= (d & 0x0f);
	PORTA = a;
	PORTA &= 0xef;
	PORTA = a;
	lcddelay();
}

void lcdstringconst(unsigned char *s, unsigned char len)
{	//menampilkan string dari program memory sepanjang len
	for(;len;len--)
	{
		lcdsend(pgm_read_byte(s),1);
		s++;
	}
}

void lcdclear()
{	//menghapus tampilan lcd
	lcdsend(0x01,0);
	lcddelaylong();
}

void lcdinit()
{
	lcdsend(0x28,0); //protokol 4 bit
	lcdsend(0x28,0); //perintah ini perlu dikirim 2 kali
	lcdsend(0x0c,0);
	lcdsend(0x06,0);
	lcdsend(0x14,0);
	lcdclear();
}

#define lcdbaris2()		lcdsend(0xc0,0)
#define lcdbaris1()		lcdsend(0x80,0)
#define lcdgeserkiri()	lcdsend(0x18,0)
#define lcdgeserkanan()	lcdsend(0x1c,0)

unsigned char strtest1[] PROGMEM = "--<Kit EAVR13>--"; //16 huruf
unsigned char strtest2[] PROGMEM = "--************--";
unsigned char strtest3[] PROGMEM = " Test LCD - tulisan bergeser ke kiri"; //36 huruf
unsigned char strtest4[] PROGMEM = "abcdefghijklmnopqrstuvwxyz1234567890!?"; //38 huruf
int main()
{
	DDRA=0xff;
	PORTA=0xff;
	DDRC=0xff;
	PORTC=0xff;
	delay10ms(10);
	lcdinit();
	lcdstringconst(strtest1,16);
	lcdbaris2();
```

```
        lcdstringconst(strtest2,16);
        delay10ms(200); //2 detik
        lcdclear();
        lcdstringconst(strtest3,36);
        lcdbaris2();
        lcdstringconst(strtest4,38);
        while(1)
        {
                delay10ms(20);
                //geser kiri:
                lcdgeserkiri();
        }
}
```

Pada program ini ada beberapa hal yang perlu diperhatikan. Pertama, digunakan timer untuk menunggu selama waktu tertentu. Untuk keterangan lebih jelas tentang fungsi timer, lihat bagian tentang **Fungsi Dalam Mikrokontroler**. Kedua, tulisan yang ditampilkan adalah konstanta yang disimpan dalam memori program. Hal ini dilakukan dengan cara-cara berikut ini:

File header yang digunakan:

```
#include<avr\pgmspace.h>
```

Deklarasi string dengan kata kunci PROGMEM:

```
unsigned char nama_string[] PROGMEM = "[kata-kata...]";
```

Membaca byte dari memori program:

```
a = pgm_read_byte(alamat);
```

Pernyataan yang digunakan sebagai alamat bisa saja:
1. Pointer

```
unsigned char *c;
c = 200;
a = pgm_read_byte(c);      //menghasilkan apapun yang tersimpan di alamat 200
```

2. Alamat dari array

```
unsigned char s[] PROGMEM = "abcd";
a = pgm_read_byte(s);      //menghasilkan huruf a
a = pgm_read_byte(s+1);    //menghasilkan huruf b
```

Selain cara ini, untuk menuliskan string dari memori program dengan fungsi `lcdstringconst` bisa juga dengan cara berikut ini:

```
lcdstringconst(PSTR("Hello"),5);
```

Dengan menggunakan fungsi `PSTR()`, string yang akan ditulis tidak perlu dideklarasikan di awal program.

5. Keypad

Setiap tombol dalam keypad adalah saklar yang menghubungkan baris tertentu dengan kolom tertentu. Misalnya, ketika tombol 4 ditekan, kolom 1 (paling kiri) terhubung dengan baris 2, seperti terlihat dalam Gambar 4.3.

Kolom 1	Kolom 2	Kolom 3	
1	2	3	Baris 1
4	5	6	Baris 2
7	8	9	Baris 3
*	0	#	Baris 4

Gambar 4.3: Baris dan kolom pada keypad.

Untuk mengetahui apakah ada tombol yang ditekan, kita memberi sinyal low pada satu kolom. Kemudian diperiksa, apakah ada baris yang ikut menjadi low? Jika ada, maka sistem tahu bahwa ada tombol yang ditekan. Selanjutnya kolom berikutnya diperiksa. Perhatikan contoh berikut.

```
#include<avr\io.h>

//test keypad di port C
//output deret LED di port A

unsigned char bacakeypad()
{
	//kolom  1 2 3
	//port C.7 6 5

	//baris  1 2 3 4
	//port C.3 2 1 0

	unsigned char k;
	DDRC = 0xe0;

	//kolom 1
	PORTC = 0b01111111;
	//perlu menunggu beberapa saat sampai tegangannya “meresap”
	asm("nop");  asm("nop"); asm("nop"); //tidak melakukan apa-apa selama 3 clock
	k = PINC & 0x0f;
	switch(k)
	{
		case 0b00000111: return 1;
		case 0b00001011: return 4;
		case 0b00001101: return 7;
		case 0b00001110: return 10; //tanda *
	}
	//kolom 2
	PORTC = 0b10111111;
	asm("nop");  asm("nop"); asm("nop");
	k = PINC & 0x0f;
	switch(k)
	{
		case 0b00000111: return 2;
		case 0b00001011: return 5;
		case 0b00001101: return 8;
		case 0b00001110: return 0;
	}
	//kolom 3
	PORTC = 0b11011111;
	asm("nop");  asm("nop"); asm("nop");
	k = PINC & 0x0f;
	switch(k)
	{
		case 0b00000111: return 3;
		case 0b00001011: return 6;
		case 0b00001101: return 9;
		case 0b00001110: return 11; //tanda #
	}
	return 255;
}

int main()
{
	unsigned char k;
	DDRA=0xff;
	PORTA=0xff;
	while(1)
	{
		k = bacakeypad();
		if(k!=255) PORTA = ~k;
	}
}
```

Sering kali dalam menggunakan keypad kita perlu menunggu sampai tombol dilepas atau menggunakan debouncing. Program yang lebih andal mungkin saja perlu menggunakan kedua hal ini.

Dengan menggunakan tampilan LCD dan keypad, dapat dibuat berbagai macam program, misalnya:

- Kunci kombinasi digital.
- Mengetik alfabet dengan keypad seperti pada telepon genggam.

- Kalkulator.
- Permainan tebak angka.

6. Modul Serial Port

Modul ini dapat digunakan untuk berkomunikasi dengan komputer. Sementara sebagian komputer tidak lagi memiliki port COM, masih tersedia pengubah USB ke RS232 yang bisa dipakai untuk berhubungan dengan alat (Sebenarnya bisa saja berkomunikasi langsung dengan USB tetapi tatacaranya amat rumit).

Untuk menggunakan modul ini, pengguna perlu menghidupkan modul USART yang terdapat dalam AVR. Penjelasan tentang ini terdapat di bagian tentang **Fungsi Dalam Mikrokontroler**. Dalam contoh program berikut ini, digunakan fungsi-fungsi yang dibahas di bagian tersebut.

```
#include<avr\io.h>
void initserial(); //lihat definisi fungsi ini di bagian selanjutnya
void kirim(unsigned char k);
#define adaserial() (UCSRA & 0x80)

int main()
{
	unsigned char d;
	initserial(); //untuk keterangan fungsi ini lihat bagian selanjutnya
	while(1)
	{
		if(adaserial())
		{
			d = UDR;
			kirim(d); //mengirim balik byte yang diterima
		}
	}
}
```

Untuk melihat kerja program ini, kita perlu menjalankan program di komputer yang bisa mengirim dan menerima data melalui jalur serial (COM). Dalam Windows XP program ini disebut Hyperterminal, tetapi dalam versi Windows selanjutnya tidak ditemui lagi program ini. Pembaca bisa membuat program sendiri atau memakai program yang tersedia di internet, misalnya RealTerm, yang bisa didapatkan di tautan berikut. (http://sourceforge.net/projects/realterm/)

Jika kedua program, satu dalam AVR dan satu di komputer, bekerja, maka apapun yang dikirimkan dari komputer akan diulangi atau dikirim kembali oleh AVR.

Tentu saja penggunaan modul ini dapat digabungkan dengan modul-modul lainnya dalam kit, misalnya modul deret LED. Bisa dibuat sebuah program supaya byte yang diterima dari komputer digunakan sebagai kode untuk menyalakan deret LED, contohnya seperti program berikut:

```
#include<avr\io.h>

//modul deret LED di port A

void initserial();
#define adaserial() (UCSRA & 0x80)

int main()
{
	unsigned char d;
	DDRA=0xff;
	initserial();
	while(1)
	{
		if(adaserial())
		{
			d = UDR;
			PORTA = ~d;
		}
	}
}
```

Contoh-contoh program lain yang dapat dibuat misalnya:

- Mengirim kode huruf yang ditekan dari komputer ke kit dan menampilkannya di modul LCD.
- Mengirim kode huruf dari komputer untuk mengatur pola LED berjalan.
- Mengirim kode dari keypad ke komputer untuk mengaktifkan fungsi tertentu pada program komputer.

7. Modul Driver Motor DC

Penggunaan modul ini misalnya untuk mengendalikan satu atau dua motor DC (dinamo) 6V. Untuk modul ini diperlukan kabel khusus untuk menghubungkan catu daya motor yang bisa saja berbeda (sama atau lebih tinggi) dengan catu daya kit. Untuk mengambil catu daya dari kit, terdapat header di dekat regulator 7805 pada PCB utama.

Motor DC memiliki dua terminal. Jika diberikan beda tegangan di kedua terminal ini maka arus akan mengalir dan motor akan bergerak. Polaritas tegangan mempengaruhi arah putaran motor. Jika ujung positif dan negatif dibalik, motor akan berputar ke arah sebaliknya. Ini bermanfaat misalnya untuk menjalankan robot dengan dua roda bertenaga. Jika satu roda maju dan roda lainnya mundur maka robot akan berputar di tempat. Jika kedua roda berputar searah, robot berjalan maju atau mundur (tergantung definisi "depan" dan "belakang").

Modul driver motor DC berfungsi mengatur tegangan pada terminal motor sesuai dengan sinyal kontrol. Mengapa motor tidak langsung digerakkan dari port AVR? Jawabannya adalah karena kemampuan pin-pin port AVR untuk mengalirkan arus sangat kecil (10 mA) dibandingkan dengan arus yang diperlukan motor untuk bekerja normal (100-1000 mA).

Contoh program berikut ini mengatur supaya motor yang terhubung pada satu pasang keluaran modul bergerak maju dan mundur bergantian. Digunakan dua pin untuk mengendalikan keluaran dan satu pin lainnya untuk mengatur keluaran supaya aktif (enable).

```
#include<avr\io.h>

//Program menjalankan motor DC bolak-balik
//modul driver motor DC di port A, motor DC di keluaran 1 dan 2
//Port A0-A1 keluaran, A5 enable

void delay10ms(unsigned int t)
{
	unsigned char c54;
	TCNT0=0;	//mulai menghitung dari 0
	TIFR|=0x01;	//clear flag
	TCCR0=0x02;	//start timer
	for(;t;t--)	//hitung mundur sebanyak t kali
	{
		for(c54=54;c54;c54--)	//overflow 54 kali = 100 Hz
		{
			while(!(TIFR & 0x01)); //tunggu sampai overflow
			TIFR |= 0x01; //clear flag dengan set bit
		}
	}
	TCCR0=0;
}

int main()
{
	DDRA=0xff;
	PORTA = 0x20; //enable 1 dan 2
	while(1)
	{
		PORTA = 0x21; //maju
		delay10ms(200); //2 detik
		PORTA = 0x22; //mundur
		delay10ms(400); //4 detik
	}
}
```

Dengan menggabungkan keluaran nomor 1, 2, 3, dan 4, modul ini dapat mengendalikan satu motor stepper bipolar atau unipolar. Untuk motor unipolar, diperlukan dua kabel ground yang bisa disambung ke deretan jalur ground yang tersedia di PCB utama.

BAB V
FUNGSI DALAM MIKROKONTROLER

Pendahuluan

Selain modul-modul yang terdapat dalam kit, mikrokontroler AVR juga memiliki fungsi-fungsi dalam. Fungsi yang dibahas di sini adalah Timer, Interrupt, USART (komunikasi serial), EEPROM, dan ADC. Untuk penjelasan lengkap (dan sulit dicerna) tentang modul, register-register, dan cara menyetel, lihat datasheet ATMega16.

1. Timer/Counter

Ada tiga buah timer yang tersedia yaitu Timer0, Timer1, dan Timer2. Pada umumnya cara kerja timer adalah menghitung jumlah terjadinya peristiwa tertentu. Peristiwa yang dihitung bisa transisi pada pin masukan, atau clock pada sistem itu sendiri. Penghitungan ini dilakukan secara otomatis, sehingga timer sering digunakan untuk mengatur waktu. Timer dapat digunakan untuk:

- Menghitung clock sistem sebanyak nilai tertentu, sehingga menghasilkan sinyal secara periodik
- Menghitung jumlah pulsa pada pin.
- Menghitung lebar pulsa pada pin.
- Menghasilkan sinyal Pulse Width Modulation (PWM).

Inti dari timer adalah suatu angka hitungan yang bertambah (bisa juga berkurang) setiap kali satu pulsa masuk. Jika hitungan mencapai batas atas, yaitu 255 untuk timer 8 bit dan 65535 untuk timer 16 bit, maka berikutnya hitungan akan kembali ke 0. Peristiwa ini disebut *overflow*. Karena ini, timer disebut juga *counter* (penghitung).

Timer0 dan Timer2 menggunakan hitungan 8 bit, sedangkan Timer2 16 bit. Perbedaan timer2 dengan timer0 salah satunya adalah pemilihan frekuensi sumber pulsa yang berbeda dari clock sistem. Clock sistem sendiri bisa berarti frekuensi kristal, yang dalam kit AVR adalah 11,0592 MHz, atau frekuensi osilator RC internal. Kecepatan clock sistem sama dengan kecepatan mikrokontroler menjalankan program.

Register yang berhubungan dengan timer adalah:

TCCR0

Register pengendali fungsi Timer0.

Bit	Nama	fungsi	Nilai		
7	FOC0	Force output compare	0 *		
6	WGM00	Waveform generator mode	0 *		
5	COM01	Compare match output mode	0 *		
4	COM00		0 *		
3	WGM01	Waveform generator mode	0 *		
2	CS02	Sumber pulsa hitungan untuk Timer0.	000: nonaktif	011: clock sistem dibagi 64	110: pulsa dari pin T0
1	CS01		001:clock sistem	100: clock sistem dibagi 256	111: pulsa dari pin T0
0	CS00		010: clock sistem dibagi 8	101: clock sistem dibagi 1024	

*) Tidak digunakan untuk fungsi timer biasa

TCNT0

Register ini adalah angka hitungan Timer0. Dalam penggunaan biasa, angka ini terus bertambah sampai 255 kemudian kembali ke 0 sementara sinyal status (flag) Timer0 Overflow menjadi 1 (high).

TCCR1A

Register pengendali fungsi Timer1.

Bit	Nama	fungsi	Nilai
7	COM1A1	Compare output mode channel A	0 *
6	COM1A0		0 *
5	COM1B1	Compare output mode channel B	0 *
4	COM1B0		0 *
3	FOC1A	Force output compare channel A	0 *

2	FOC1B	Force output compare channel B	0 *
1	WGM11	Waveform generation mode	0 *
0	WGM10		0 *

TCCR1B

Register pengendali fungsi Timer1.

Bit	Nama	fungsi	Nilai		
7	ICNC1	Input capture noise canceler	0 *		
6	ICES1	Waveform generator mode	0 *		
5	-		0 *		
4	WGM13	Waveform generator mode	0 *		
3	WGM12		0 *		
2	CS02	Sumber pulsa hitungan untuk Timer1.	000: nonaktif	011: clock sistem dibagi 64	110: pulsa dari pin T1
1	CS01		001:clock sistem	100: clock sistem dibagi 256	111: pulsa dari pin T1
0	CS00		010: clock sistem dibagi 8	101: clock sistem dibagi 1024	

TCNT1

Register 16 bit yang berisi angka hitungan Timer1. Jika angka hitungan ini mencapai 65535 maka seterusnya akan kembali ke 0 sementara flag Timer1 Overflow menjadi 1 (high)

TCCR2

Register pengendali fungsi Timer2.

Bit	Nama	fungsi	Nilai		
7	FOC2	Force output compare	0 *		
6	WGM20	Waveform generator mode	0 *		
5	COM21	Compare match output mode	0 *		
4	COM20		0 *		
3	WGM21	Waveform generator mode	0 *		
2	CS22	Sumber pulsa hitungan untuk Timer2.	000: nonaktif	011: clock sistem dibagi 32	110: clock sistem dibagi 256
1	CS21		001:clock sistem	100: clock sistem dibagi 64	111: clock sistem dibagi 1024
0	CS20		010: clock sistem dibagi 8	101: clock sistem dibagi 128	

TCNT2

Register 8 bit yang berisi angka hitungan Timer2.

TIMSK

Register yang mengatur sinyal interrupt dari ketiga timer. Jika bit disetel 1 (high) maka interrupt akan terjadi ketika peristiwa yang berhubungan dengan bit itu terjadi. Untuk keterangan lebih lanjut, lihat bagian tentang Interrupt.

Bit	Nama	Sinyal interrupt
7	OCIE2	Timer2 Output Compare Match
6	TOIE2	Timer2 Overflow
5	TICIE1	Timer1 Input Capture
4	OCIE1A	Timer1 Output Compare Match A
3	OCIE1B	Timer1 Output Compare Match B
2	TOIE1	Timer1 Overflow
1	OCIE1	Timer0 Output Compare Match
0	TOIE0	Timer0 Overflow

TIFR

Register yang berisi sinyal status ketiga timer. Bit bernilai 1 (high) ketika peristiwa yang berhubungan dengan bit itu terjadi. Untuk mengembalikan bit ke keadaan biasa, tulis nilai 1 (high) pada bit itu.

Bit	Nama	Sinyal status (flag)
7	OCF2	Timer2 Output Compare Match
6	TOV2	Timer2 Overflow

5	TICF1	Timer1 Input Capture
4	OCF1A	Timer1 Output Compare Match A
3	OCF1B	Timer1 Output Compare Match B
2	TOV1	Timer1 Overflow
1	OCF1	Timer0 Output Compare Match
0	TOV0	Timer0 Overflow

Contoh: Menggunakan Timer0 untuk mengatur waktu

```
void delay10ms(unsigned int t)
{
	//clock=kristal/8 = 11059200/8 = 1382400 Hz
	//overflow setiap 1382400/256 = 5400 Hz
	unsigned char c54;
	TCNT0=0;	//mulai menghitung dari 0
	TIFR|=0x01;	//clear flag
	TCCR0=0x02;	//start timer
	for(;t;t--)	//hitung mundur sebanyak t kali
	{
		for(c54=54;c54;c54--)	//overflow 54 kali = 100 Hz
		{
			while(!(TIFR & 0x01)); //tunggu sampai overflow
			//sesudah overflow:
			TIFR |= 0x01; //clear flag dengan set bit
		}
	}
	TCCR0=0;
}
```

2. Interrupt

Dalam semua program sebelumnya, setiap perintah dikerjakan secara runut. Perintah berikutnya yang dijalankan sudah jelas. Namun, kadang-kadang kita ingin program mengerjakan perintah tertentu pada saat yang tidak diduga (oleh program, tapi sudah diperkirakan oleh *pembuat* program). Untuk ini digunakan interrupt.

Interrupt adalah suatu peristiwa yang menyebabkan program meloncat ke suatu bagian yang sudah disiapkan sebelumnya, tidak peduli program itu sedang mengerjakan apa sebelumnya. Bagian program yang dituju ketika interrupt terjadi disebut *Interrupt Service Routine* (ISR). Dalam bahasa assembly, kita menaruh perintah `jump` di posisi alamat yang sudah baku, yaitu *interrupt vector*. Dalam compiler WinAVR, kita membuat fungsi dengan kata kunci khusus:

```
ISR(nama_signal) //<-- isi nama_signal dengan daftar pada tabel
{
	//tulis isi fungsi di sini.

}
```

Untuk menggunakan fungsi ini kita perlu menambah satu file header:

```
#include<avr\interrupt.h>
```

Untuk menyiapkan program menerima interrupt atau untuk menolak interrupt, digunakan fungsi khusus:

```
sei(); //enable interrupt
cli(); //disable interrupt
```

Ada sejumlah peristiwa yang bisa menghasilkan interrupt. Berbagai jenis mikrokontroler dirancang dengan menyertakan daftar tertentu yang berisi sumber-sumber interrupt yang tersedia dalam datasheet. Untuk ATMega16, daftar sumber interrupt (tidak lengkap) adalah sebagai berikut.

Nama signal	Peristiwa
ADC_vect	ADC selesai mengukur tegangan
ANA_COMP_vect	Analog Comparator
EE_RDY_vect	EEPROM siap
INT0_vect	Interrupt luar
INT1_vect	Interrupt luar

INT2_vect	Interrupt luar
SPI_STC_vect	SPI selesai mengirim data
TIMER0_COMP_vect	Timer0 Compare Match
TIMER0_OVF_vect	Timer0 overflow
TIMER1_OVF_vect	Timer1 overflow
TIMER2_OVF_vect	Timer2 overflow
USART_RXC_vect	USART selesai menerima data
USART_TXC_vect	USART selesai mengirim data
USART_UDRE_vect	Register data USART kosong

3. Serial Port

Komunikasi serial adalah satu cara untuk mengirimkan data atau perintah dari satu komputer ke komputer lainnya. Dalam komunikasi serial asinkron, bit-bit dalam satu byte dikirim bergantian, mulai dari bit tertinggi sampai bit terendah dengan selang waktu yang konstan, yang disebut **baud rate**.

AVR memiliki modul yang menangani komunikasi serial, yang disebut USART (Universal Synchronous & Asynchronous serial Receiver and Transmitter). Program hanya perlu mengatur bit-bit register, kemudian mengirim byte kepada register tertentu untuk mengirim data, atau membaca register untuk membaca data yang sudah diterima. Pengiriman dan penerimaan bit-bit data dan pengaturan waktunya sudah diurus oleh modul USART.

Berikut ini adalah register-register yang berhubungan dengan modul serial.

UDR

Register ini berisi data yang sudah diterima, atau data yang akan dikirim.

UCSRA

Register yang mengatur kerja modul USART.

Bit	Nama	Fungsi	Nilai
7	RXC	Selesai menerima byte	1 = ada data yang siap dibaca
6	TXC	Selesai mengirim byte	1 = selesai mengirim byte
5	UDRE	Register data kosong	1 = register data kosong
4	FE	Frame error	? *
3	DOR	Data Overrun	? *
2	PE	Parity error	? *
1	U2X	Kecepatan 2 kali	0 = normal 1 * = kecepatan dua kali
0	MPCM	Komunikasi multiprosesor	0 *

*) Tidak digunakan dalam pembahasan di sini.

UCSRB

Register yang mengatur kerja modul USART.

Bit	Nama	Fungsi	Nilai
7	RXCIE	Interrupt enable untuk penerimaan	1 = enable
6	TXCIE	Interrupt enable untuk pengiriman	1 = enable
5	UDRIE	Interrupt enable untuk register data kosong	1 = enable
4	RXEN	Aktifkan penerimaan	1 = aktif
3	TXEN	Aktifkan pengiriman	1 = aktif
2	UCSZ2	Pengaturan jumlah bit data	0 = 8 bit
1	RXB8	Bit kesembilan yang diterima	0 *
0	TXB8	Bit kesembilan yang dikirimkan	0 *

UCSRC

Register yang mengatur kerja modul USART.

Bit	Nama	Fungsi	Nilai
7	URSEL	Memilih register ini	1
6	UMSEL	Pilihan mode	0 = asinkron
5	UPM1	Memilih mode parity	0
4	UPM0		0 = tidak menggunakan parity
3	USBS	Jumlah stop bit	0 = 1 bit, 1 = 2 bit

2	UCSZ1	Pengaturan jumlah bit data	1
1	UCSZ0		1 = 8 bit
0	UCPOL	Polaritas clock data	0 *

UBRRH dan **UBRRL**
Kedua register ini menentukan kecepatan pengiriman data atau panjangnya bit-bit. Untuk kecepatan normal (U2X = 0), nilai baud rate yang dihasilkan untuk kristal 11,0592 MHz ditampilkan dalam tabel berikut.

Baud rate	UBRRH	UBRRL
2400	0x01	0x1F
4800	0x00	0x8F
9600	0x00	0x47
19200	0x00	0x23
38400	0x00	0x11
57600	0x00	0x0B

Biasanya, laju pengiriman rendah (2400 atau 4800) digunakan ketika data yang dikirim sedikit atau melewati kabel yang panjang. Jika kabel yang digunakan lebih pendek daripada 30 cm, bisa saja digunakan baud rate hingga 57600. Perlu diperhatikan bahwa penerima dan pengirim harus menggunakan kecepatan yang sama.

Fungsi-fungsi berikut ini dapat digunakan untuk mengatur modul USART dan mengirim data:

```
void initserial() //mengaktifkan modul USART
{
	//19200 baud
	//UBRRH = 0;
	UBRRL = 0x23;
	UCSRB = 0x18;
}
```

```
void kirim(unsigned char k) //mengirim satu byte
{
	while((UCSRA & 0x20) == 0); //tunggu sampai 1 (= siap kirim)
	UDR = k;
}
```

```
void kirimint(unsigned int i) //mengirim nilai integer sebagai tulisan
{
	unsigned char d, bufangka[5]; //0 = satuan, 1 = puluhan dst
	bufangka[0] = i%10+'0'; i=i/10;
	if(i){
		bufangka[1] = i%10+'0'; i=i/10;
		if(i){
			bufangka[2] = i%10+'0'; i=i/10;
			if(i){
				bufangka[3] = i%10+'0'; i=i/10;
				if(i) bufangka[4] = i%10+'0';
				else bufangka[4] = ' ';
			}else{
				bufangka[3] = ' ';
				bufangka[4] = ' ';
			}
		}else{
			bufangka[2] = ' ';
			bufangka[3] = ' ';
			bufangka[4] = ' ';
		}
	}else{
		bufangka[1] = ' ';
		bufangka[2] = ' ';
		bufangka[3] = ' ';
		bufangka[4] = ' ';
	}
	for(d=4;d-255;d--) kirim(bufangka[d]);
}
```

```
//kondisi ini bisa digunakan untuk memeriksa adanya byte yang diterima dan siap dibaca.
#define adaserial() (UCSRA & 0x80)
```

Salah satu penggunaan komunikasi serial adalah mengirimkan data dari alat ke komputer. Misalkan alat berfungsi sebagai sensor yang mengamati lingkungan sekitar, misalnya suhu, terangnya cahaya, atau detektor maling. Alat hanya mengerjakan proses yang tidak bisa dikerjakan komputer biasa, yaitu mengukur

suhu dan cahaya, kemudian mengirimkan data ke komputer. Selanjutnya, komputer melakukan proses yang lebih rumit dan tidak bisa dikerjakan mikrokontroler, misalnya mengirim pesan lewat internet.

4. Memori tidak-mudah-hilang (EEPROM)

Dalam beberapa mikrokontroler AVR (termasuk ATMega16) terdapat memori EEPROM (Electrically Erasable Programmable Read Only Memory). Memori ini adalah penyimpanan data serbaguna yang tidak hilang jika catu daya dimatikan. Sebagai perbandingan, data dalam memori RAM yang biasa digunakan dalam penyimpanan variabel dan berbagai perhitungan akan hilang jika catu daya dimatikan. Data pada memori program juga termasuk memori yang tidak mudah hilang dan biasanya berukuran cukup besar (16 kilobyte untuk ATMega16). Memori program cocok untuk menyimpan data yang tetap dan tidak pernah atau hampir tidak pernah berubah, misalnya kata-kata pembuka yang ditampilkan dalam LCD. Memori EEPROM cocok digunakan untuk menyimpan data atau setelan yang perlu bertahan ketika sistem di-reset atau dimatikan. Contoh data seperti ini misalnya password.

Untuk menulis dan membaca EEPROM, kita perlu memperhatikan beberapa hal. Pertama, optimisasi compiler perlu diaktifkan menjadi –Os. Kedua, selama penulisan EEPROM program akan berhenti sejenak dan interrupt perlu dimatikan supaya tidak mengganggu proses ini.

Berikut ini adalah register-register yang mengatur lalu lintas data pada EEPROM.

EECR

Register pengendali sinyal baca and tulis pada EEPROM.

Bit	Nama	Fungsi
7-4	-	Tidak dipakai.
3	EERIE	Mengatur aktifnya interrupt ketika EEPROM siap bekerja.
2	EEMWE	Bit pertama yang disetel high ketika akan menulis.
1	EEWE	Bit kedua yang disetel high ketika akan menulis.
0	EERE	Bit yang disetel high ketika akan membaca.

EEAR

Register 16 bit yang menentukan alamat data EEPROM yang akan dibaca atau ditulis. ATMega16 memiliki 512 byte EEPROM, mulai dari alamat 0x0000 sampai 0x01FF. Register ini terdiri dari dua register 8 bit, yaitu **EEARH** dan **EEARL.**

EEDR

Register yang berisi data yang akan dituliskan ke EEPROM atau hasil pembacaan EEPROM.

Fungsi berikut ini dapat digunakan untuk membaca dan menulis data dari dan ke EEPROM.

```
unsigned char eeread(unsigned int adr)
{
    //tunggu operasi sebelumnya selesai:
    while(EECR & 0x02); //tunggu EEWE = 0
    EEAR = adr;     //set address
    EECR |= 0x01;   //perintah baca
    return EEDR;
}

void eewrite(unsigned char d, unsigned int adr)
{
    //tunggu operasi sebelumnya selesai:
    while(EECR & 0x02);
    EEAR = adr;     //set address
    EEDR = d;       //tulis
    //set bit perintah tulis:
    EECR |= 0x04;   //EEMWE
    EECR |= 0x02;   //EEWE
}
```

5. Analog to Digital Converter (ADC)

Dalam sinyal digital, hanya ada dua keadaan yaitu high dan low. Padahal, di antara high dan low sebenarnya terdapat banyak keadaan yang menyatakan nilai tegangan listrik sebenarnya. Supaya peralatan digital bisa mendapatkan nilai tegangan analog ini, digunakan Analog to Digital Converter (ADC).

Sebuah ADC melihat nilai tegangan listrik lalu mengeluarkan angka digital (dalam bilangan biner) yang sesuai. Misalnya, bisa saja dibuat supaya tegangan 0-5 Volt dinyatakan dengan angka 0-255. Berarti, satu angka digital bernilai 5/256 Volt atau 0,0196 Volt. Angka 255 menyatakan ketelitian 8 bit, karena untuk menulis 255 dalam bilangan biner diperlukan 8 angka. Jika jumlah bit semakin besar, ADC semakin teliti karena jangkauan tegangan dibagi-bagi menjadi lebih banyak, yaitu 0-1023 untuk 10 bit (seperti pada ATMega16), 0-4095 untuk 12 bit, dan 0-65535 untuk 16 bit. Nilai 5 Volt sendiri dalam kasus ini disebut *tegangan referensi*. Tegangan ini bisa bernilai berapa saja tergantung rancangan. Dalam kit, tegangan referensi pada pin AREF bisa diatur sama dengan catu daya (sekitar 4,98 – 5 V) dengan menyolder sambungan terbuka **S1** pada sisi bawah PCB atau menggunakan referensi internal ATMega16 yaitu 2,56 V dengan memutus sambungan **S1.**

Register-register yang berhubungan dengan ADC adalah sebagai berikut.

ADMUX

Register yang menentukan pin yang diukur tegangannya dengan ADC, dan pemilihan tegangan referensi.

Bit	Nama	Fungsi	
7	REFS1	00: menggunakan referensi luar (pin AREF)	10: tidak berarti
6	REFS0	01: menggunakan tegangan catu daya (AVCC)	11: Referensi internal 2,56 V
5	ADLAR	1 = Hasil bacaan ADC ditampilkan rata kiri dalam register hasil.	
4	MUX4	Pemilihan pin yang diukur. Gunakan 00000 untuk pin ADC0, 00001 untuk pin ADC1, dan seterusnya sampai 00111 untuk pin ADC7. Nilai lain digunakan untuk masukan diferensial (lihat datasheet)	
3	MUX3		
2	MUX2		
1	MUX1		
0	MUX0		

ADCSRA

Register yang mengendalikan fungsi ADC.

Bit	Nama	Fungsi
7	ADEN	1 = ADC dihidupkan
6	ADSC	1 = Mulai mengukur
5	ADATE	1 = Auto trigger enable
4	ADIF	Sinyal (flag) interrupt
3	ADIE	1 = ADC interrupt enable
2	ADPS2	Menentukan frekuensi clock ADC.
1	ADPS1	
0	ADPS0	

Nilai	Frekuensi
000	Clock / 2
001	Clock / 2
010	Clock / 4
011	Clock / 8
100	Clock / 16
101	Clock / 32
110	Clock / 64
111	Clock / 128

Untuk penggunaan biasa dalam kit, bisa digunakan nilai 110, yang menghasilkan clock ADC 172,8 kHz.

ADCH dan **ADCL**

Dua register ini menyimpan data hasil bacaan ADC. ADCL harus dibaca lebih dahulu sebelum ADCH. Jika hasilnya disetel rata kanan maka isi register ini tampak seperti ini:

Register ADCH - ADCL

Biner: 0000 00aa aaaa aaaa

Jika hasil bacaan disetel rata kiri maka hasilnya terlihat seperti ini:

Biner: aaaa aaaa aa00 0000

Berikut ini adalah contoh program yang menggunakan ADC untuk mengatur pola nyala modul LED. Karena ADC menggunakan Port A maka pada contoh program ini, modul LED dipasang di Port C.

```
#include<avr\io.h>
#define adcstart() ADCSRA |= 0x40
#define tungguadc() (ADCSRA & 0xbf)

int main()
{
```

```
	//test ADC channel 0
	unsigned int a;
	unsigned char d1, d2;
	DDRA = 0; //semua input
	PORTA = 0; //high impedance
	DDRC=0xff;
	PORTC=0xff;
	adcinit();
	ADCSRA = 0x86; //aktifkan modul ADC dan memilih clock untuk ADC
	ADMUX  = 0x20; //data ADC rata kiri, tegangan referensi AREF, channel 0
	while(1)
	{
		for(a=0;a<20000;a++);
		adcstart();		//set bit high untuk memulai konversi
		while(tungguadc()); //tunggu sampai bit ini low
		d1 = ADCL;		//ADCL harus dibaca lebih dulu
		d2 = ADCH;
		PORTC = ~d2;		//karena LED hanya ada 8, jadi ambil 8 bit tertinggi
	}
}
```