**MATERI 1**

**PENGENALAN MIKROKONTROLER**

**Mikrokontroler AVR ATmega**

AVR (Alf and Vegard’s Risc Processor) merupakan salah satu jenis mikrokontroler yang di dalamnya terdapat berbagai macam fungsi. AVR memiliki keunggulan dibandingkan dengan mikrokontroler jenis lain, keunggulannya yaitu AVR memiliki kecepatan eksekusi program yang lebih cepat karena sebagian besar instruksi dieksekusi dalam 1 siklus clock, lebih cepat bila dibandingkan dengan mikrokontroler jenis MCS51 yang memiliki arsitektur CISC (Complex Instruction Set Compute) dimana mikrokontoller MCS51 membutuhkan 12 siklus clock untuk mengeksekusi 1 instruksi (Heri Andrinto, 2008:2). Selain itu kelebihan mikrokontroler AVR memiliki POS (Power On Reset), yaitu tidak perlu adanya tombol reset dari luar karena cukup hanya dengan mematikan supply, maka secara otomatis AVR akan melakukan reset. Antar seri mikrokontroler AVR memiliki beragam tipe dan fasilitas, namun kesemuanya memiliki arsitektur yang sama, dan juga set instruksi yang relatif tidak berbeda. Berikut tabel perbandingan beberapa seri mikrokontroler AVR buatan Atmel.

Tabel 1. perbandingan beberapa seri mikrokontroler AVR buatan Atmel

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Seri** | **Flash (KBytes)** | **RAM (Bytes)** | **EEPROM (KBytes)** | **Pin I/O** | **Timer 16-bit** | **Timer 8-bit** | **UART** | **PWM** | **ADC 10-bit** | **SPI** | **ISP** |
| ATmega8 | 8 | 1024 | 0.5 | 23 | 1 | 1 | 1 | 3 | 6/8 | 1 | Ya |
| ATmega8535 | 8 | 512 | 0.5 | 32 | 2 | 2 | 1 | 4 | 8 | 1 | Ya |
| ATmega16 | 16 | 1024 | 0.5 | 32 | 1 | 2 | 1 | 4 | 8 | 1 | Ya |
| ATmega162 | 16 | 1024 | 0.5 | 35 | 2 | 2 | 2 | 6 | 8 | 1 | Ya |
| ATmega32 | 32 | 2048 | 1 | 32 | 1 | 2 | 1 | 4 | 8 | 1 | Ya |
| ATmega128 | 128 | 4096 | 4 | 53 | 2 | 2 | 2 | 8 | 8 | 1 | Ya |
| ATtiny12 | 1 | - | 0.0625 | 6 | - | 1 | - | - | - | - | Ya |
| ATtiny2313 | 2 | 128 | 0.125 | 18 | 1 | 1 | 1 | 4 | - | 1 | Ya |
| ATtiny44 | 4 | 256 | 0.25 | 12 | 1 | 1 | - | 4 | 8 | 1 | Ya |
| ATtiny84 | 8 | 512 | 0.5 | 12 | 1 | 1 | - | 4 | 8 | 1 | Ya |

**Keterangan:**

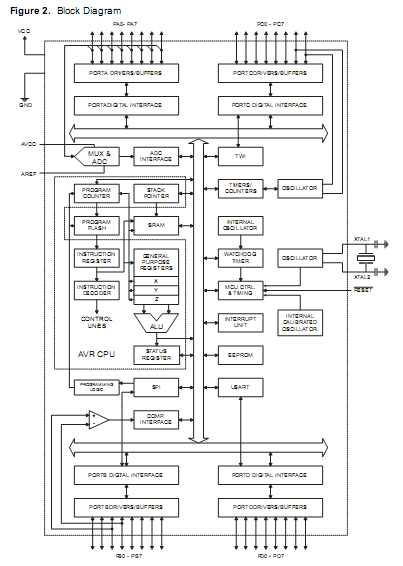
* Flash adalah suatu jenis *Read Only Memory* yang biasanya diisi dengan program hasil buatan manusia yang harus dijalankan oleh mikrokontroler
* RAM (*Random Acces Memory*) merupakan memori yang membantu CPU untuk penyimpanan data sementara dan pengolahan data ketika program sedang running
* EEPROM  (*Electrically Erasable Programmable Read Only Memory*) adalah memori untuk penyimpanan data secara permanen oleh program yang sedang running
* Port I/O adalah kaki untuk jalur keluar atau masuk sinyal sebagai hasil keluaran ataupun masukan bagi program
* Timer adalah modul dalam hardware yang bekerja untuk menghitung waktu/pulsa
* UART (*Universal Asynchronous Receive Transmit*) adalah jalur komunikasi data khusus secara serial asynchronous
* PWM (*Pulse Width Modulation*) adalah fasilitas untuk membuat modulasi pulsa
* ADC (*Analog to Digital Converter*) adalah fasilitas untuk dapat menerima sinyal analog dalam range tertentu untuk kemudian dikonversi menjadi suatu nilai digital dalam range tertentu
* SPI (*Serial Peripheral Interface*) adalah jalur komunikasi data khusus secara serial secara serial synchronous
* ISP (*In System Programming*) adalah kemampuan khusus mikrokontroler untuk dapat diprogram langsung dalam sistem rangkaiannya dengan membutuhkan jumlah pin yang minimal

**Mengenal ATmega16**

Mikrokontroler ini menggunakan arsitektur Harvard yang memisahkan memori program dari memori data, baik bus alamat maupun bus data, sehingga pengaksesan program dan data dapat dilakukan secara bersamaan (concurrent), adapun blog diagram arsitektur ATMega16.  Secara garis besar mikrokontroler ATMega16 terdiri dari :

1. Arsitektur RISC dengan throughput mencapai 16 MIPS pada frekuensi 16Mhz.
2. Memiliki kapasitas Flash memori 16Kbyte, EEPROM 512 Byte, dan SRAM 1Kbyte
3. Saluran I/O 32 buah, yaitu Port A, Port B, Port C, dan Port D.
4. CPU yang terdiri dari 32 buah register.
5. User interupsi internal dan eksternal
6. Port antarmuka SPI dan Port USART sebagai komunikasi serial
7. Fitur Peripheral

* Dua buah 8-bit timer/counter dengan prescaler terpisah dan mode compare
* Satu buah 16-bit timer/counter dengan prescaler terpisah, mode compare, dan mode capture
* Real time counter dengan osilator tersendiri
* Empat kanal PWM dan Antarmuka komparator analog
* 8 kanal, 10 bit ADC
* Byte-oriented Two-wire Serial Interface
* Watchdog timer dengan osilator internal



|  |
| --- |
|  |
| Gambar 1. Blok Diagram ATMega16 |

**Konfigurasi Pin ATMega16**

|  |
| --- |
| [E:\mikro\Baskara Blog  Dasar Teori ATMega16_files\KonfigurasiPINATMega16PDIP.png](http://baskarapunya.blogspot.com/2012/09/dasar-teori-atmega16.html) |
| Gambar 2. Konfigurasi PIN ATMega16 |

Dengan deskripsi Pin sebagai berikut :

* VCC : Sumber Tegangan
* Ground : Ground
* Port A (PA0..PA7) : Pin ini berfungsi sebagai port masukan ke A/D Converter. Port ini juga bertindak sebagai Port I/O 8-bit dua arah, jika A/D Converter itu tidak digunakan. Pin - pin Port dapat menyediakan resistor internal pull-up (yang dipilih untuk masing-masing bit). Keluaran Port A memiliki karakteristik-karakteristik pengarah simetris dengan sitem dua arah dan sumber yang tinggi. Port A bersifat tri-stated yaitu ketika kondisi reset akan aktif, sekali pun clock tidak menjalankan.
* Port B (PB0..PB7) : Port B adalah satu Port I/O 8-bit dua arah dengan resistor-resistor pull-up internal (yang terpilih untuk masing-masing bit). Keluaran buffer Port B mempunyai karakteristik-karakteristik pengarah simetris dengan kedua kemampuan sumber yang tinggi. Pada input, Port B menggunakan sumber arus rendah jika resistor-resistor pull-up diaktifkan. Port B bersifat tri-stated, yaitu reset akan aktif walaupun clock tidak dijalankan.
* Port C (PC0..PC7) : Port C adalah satu Port I/O 8-bit dua arah dengan resistor-resistor pull-up internal (yang terpilih untuk masing-masing bit). Keluaran Port C mempunyai karakteristik-karakteristik pengarah simetris dengan kedua kemampuan sumber tinggi. Pada input, Port C menggunakan sumber arus rendah jika resistor-resistor pull-up diaktifkan. Port C bersifat tri-stated ketika reset aktif, walaupun clock tidak aktif. Jika antar muka JTAG adalah dimungkinkan, resistor-resistor pull-up pin PC5(TDI), PC3(TMS) dan PC2(TCK) akan diaktifkan walaupun reset aktif.
* Port D (PD0..PD7) : Port D adalah satu Port I/O 8-bit dua arah dengan resistor-resistor pull-up internal (yang terpilih untuk masing-masing bit). Pin - pin output Port D mempunyai karakteristik-karakteristik pengarah simetris dengan kedua kemampuan sumber tinggi. Pada input, Port D secara eksternal menggunakan sumber arus rendah yang dengan mengaktifkan resistor-resistor pull-up. Port D bersifat tri-stated, yaitu reset menjadi aktif walaupun clock tidak diaktifkan.
* RESET : Pada Input Reset, besarnya amplitude yang dibutuhkan untuk mengaktifkan reset adalah lebih besar dari panjang pulse minimum untuk mengatur ulang (sesuai datasheet), sekali pun clock itu tidak diaktifkan. Pulsa yang lebih pendek belum tentu dapat mengaktifkan reset.
* XTAL1 : Input pembalik / pembangkit Oscillator penguat dan input rangkaian operasi clock internal.
* XTAL2 : Output dari pembalik / pembangkit Oscillator penguat.
* AVCC : AVCC adalah pin sumber tegangan untuk Port A dan A/D Converter. Pin harus disambungkan secara eksternal ke VCC, walaupun konverter analog-digital tidak digunakan. Jika konverter analog-digital digunakan, Pin harus dihubungkan ke VCC melalui suatu Low-Pass Filter.
* AREF : AREF berfungsi sebagai pin referensi analog untuk A/D Converter.

**Port sebagai input/output digital**

ATMega16 mempunyai empat buah port yang bernama *PortA, PortB, PortC, dan PortD*. Keempat port tersebut merupakan jalur *bidirectional* dengan pilihan *internal pull-up*. Tiap port mempunyai tigabuah register bit, yaitu DDxn, PORTxn, dan PINxn. Huruf ‘x’ mewakilinama huruf dari port sedangkan huruf ‘n’ mewakili nomor bit. BitDDxn terdapat pada I/O address DDRx, bit PORTxn terdapat padaI/O address PORTx, dan bit PINxn terdapat pada I/O address PINx.

Bit DDxn dalam register DDRx (*Data Direction Register*) menentukan arah pin. Bila DDxn diset 1 maka Px berfungsi sebagai pin output. Bila DDxn diset 0 maka Px berfungsi sebagai pin input.Bila PORTxn diset 1 pada saat pin terkonfigurasi sebagai pin input, maka resistor *pull-up* akan diaktifkan. Untuk mematikan resistor *pull-up*, PORTxn harus diset 0 atau pin dikonfigurasi sebagai pin output. Pin port adalah *tri-state* setelah kondisi reset. Bila PORTxn diset 1 pada saat pin terkonfigurasi sebagai pin output maka pin port akan berlogika 1. Dan bila PORTxn diset 0 pada saat pin terkonfigurasi sebagai pin output maka pin port akan berlogika 0. Saat mengubah kondisi port dari kondisi *tri-state* (DDxn=0, PORTxn=0) ke kondisi *output high* (DDxn=1, PORTxn=1) maka harus ada kondisi peralihan apakah itu kondisi *pull-up enabled* (DDxn=0, PORTxn=1) atau kondisi *output low* (DDxn=1, PORTxn=0).

Biasanya, kondisi pull-up enabled dapat diterima sepenuhnya, selama lingkungan impedansi tinggi tidak memperhatikan perbedaan antara sebuah *strong high driver* dengan sebuah *pull-up*. Jika ini bukan suatu masalah, maka bit PUD pada register SFIOR dapat diset 1 untuk mematikan semua *pull-up* dalam semua port. Peralihan dari kondisi *input dengan pull-up* ke kondisi *output low* juga menimbulkan masalah yang sama. Kita harus menggunakan kondisi *tri-state* (DDxn=0, PORTxn=0) atau kondisi output high (DDxn=1, PORTxn=0) sebagai kondisi transisi.

Tabel 2. Konfigurasi pin port



Bit 2 – PUD : *Pull-up Disable*

Bila bit diset bernilai 1 maka *pull-up* pada port I/O akan dimatikan walaupun *register* DDxn dan PORTxn dikonfigurasikan untuk menyalakan *pull-up* (DDxn=0, PORTxn=1).

**MATERI 2**

**PENGENALAN *CODEVISON* AVR**

**Bahasa C**

Bahasa C merupakan salah satu bahasa pemrograman yang paling populer untuk pengembangan program-program aplikasi pada sistem mikroprosesor. Penggunaan bahasa C akan sangat efisien terutama untuk program mikrokontroler yang berukuran relatif besar. Bahasa C dikatakan sebagai bahasa pemrograman terstruktur karena strukturnya menggunakan fungsi-fungsi sebagai program-program bagiannya (subroutine). Beberapa hal yang perlu diketahui pada bahasa C diantaranya adalah:

* 1. Preprocessor #

Preprocessor (#) digunakan untuk memasukkan text dari file lain, mendefinisikan macro yang dapat mengurangi beban kerja pemrograman dan meningkatkan legibity source code (mudah dibaca). #include dipakai untuk membaca file yang diantaranya berisi deklarasi fungsi dan definisi konstanta.

* 1. Fungsi main()

Fungsi pertama yang harus ada di program C sudah ditentukan namanya, yaitu bernama main(). Suatu fungsi di program C dibuka dengan kurung kurawal ({) dan ditutup dengan kurung kurawal tertutup (}). Diantara kurung kurawal dapat dituliskan statement – statement program C.

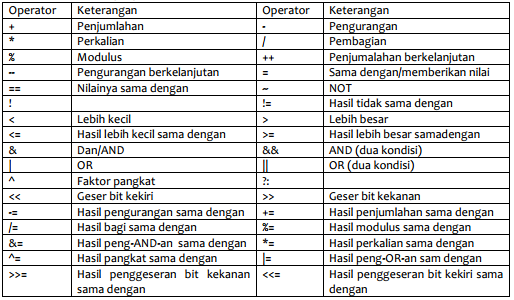
* 1. Komentar

Dalam bahasa C, suatu komentar ditulis dengan diawali tanda /\* dan diakhiri dengan tanda \*/, apabila komentar untuk multiple lines. Sedangkan untuk komentar single line, komentar diawali dengan //.

* 1. Operator

Operator atau tanda operasi adalah suatu tanda atau simbol yang digunakan untuk suatu operasi tertentu. Berikut ini operator bahasa C.

Tabel 3. Operator Bahasa C



* 1. Tipe Data

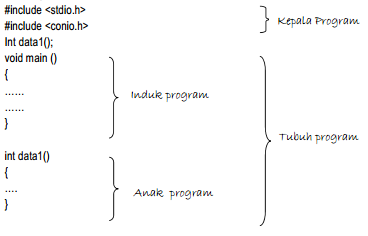
Tabel dibawah ini memberikan informasi mengenai ukuran memori yang diperlukan dan kawasan dari masing-masing tipe data dasar.

Tabel 4. Ukuran Memori Tipe Data

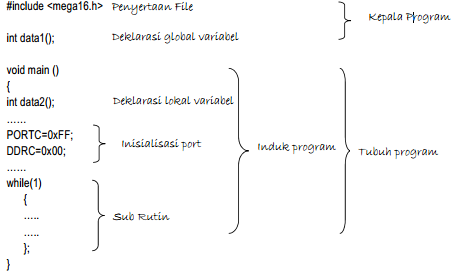
|  |  |  |
| --- | --- | --- |
| Tipe | Ukuran | Range |
| Bit | 1 bit | 0,1(tipe data bit hanya dapat digunakan untuk variable global) |
| Char | 8 bit | -128 to 127 |
| Unsigned char | 8 bit | 0 to 255 |
| Signed char | 8 bit | -128 to 127 |
| Int | 16 bit | -32768 to 32767 |
| Short int | 16 bit | -32768 to 32767 |
| Unsigned int | 16 bit | 0 to 65535 |
| Signed int | 16 bit | -32768 to 32767 |
| Long int | 32 bit | -2147483648 to 2147483647 |
| Unsigned long int | 32 bit | 0 to 4294967295 |
| Signed long int | 32 bit | -2147483648 to 2147483647 |
| Float | 32 bit | ± 1.175e-38 to ±3.402e38 |
| Double | 32 bit | ± 1.175e-38 to ±3.402e38 |

* 1. Struktur Program

Instruksi-instruksi bahasa pemrograman yang ada pada bahasa C tidak semuanya digunakan dalam pemrograman mikrokontroler. Struktur dan urutan penulisan program hampir sama untuk keduanya. Struktur bahasa C memiliki kepala program dan tubuh program, sedangkan tubuh program bisa terdiri dari induk program dan anak program. Berikut struktur sederhana dari pemrograman bahasa C.



Penulisan struktur bahasa C di dalam CV AVR dapat dilihat seperti di bawah ini:



Deklarasi sebuah variabel dapat digolongkan menjadi 2, yaitu lokal variabel dan global variabel. Lokal variabel dipakai dan hanya dapat diakses pada sub program tempat mendeklarasikannya, sedangkan global variabel dipakai dan dapat diakses seluruh bagian program. IInisialisasi PORT digunakan untuk memfungsikan PORT yang dituju sebagai masukan/keluaran serta nilai defaultnya. Sedangkan bagian sub rutin adalah blok program yang akan selalu dikerjakan terus-menerus oleh mikroprosesor selama mikrokontroler hidup.

Beberapa Instruksi-instruksi dalam bahasa C yang sering digunakan dapat ditulis sebagai berikut:

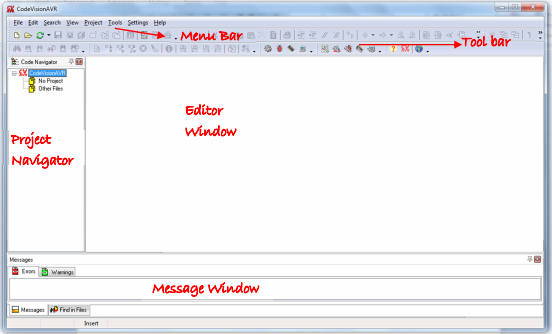
Tabel 5. Instruksi Bahasa C

|  |  |  |
| --- | --- | --- |
| No | Fungsi | Bahasa Pemrograman |
| 1. | Syarat | if (kondisi )  {  ………(aksi yang dikerjakan)  }; |
| 2. | Percabangan | if (kodisi)  {  ……(aksi yang dikerjakan)  }  else if (kondisi)  {  …..(aksi yang dikerjakan)  }  …..  …..  else  {  …..(aksi yang dikerjakan)  }; |
| 3. | Percabangan | switch (variable)  {  case nilai\_variabel\_ke-1:  {  ….. (aksi yang dikerjakan)  }  case nilai\_variabel\_ke-2:  {  ….. (aksi yang dikerjakan)  }  …………  ………..  default:  {  …. (aksi yang dikerjakan)  }  } |
| 4. | Melompat | goto alamat\_tujuan;  …………….  …………….  alamat\_tujuan:  …………… |
| 5. | Melompat keluar dari  perulangan | Break; |
| 6. | perulangan | while (kondisi)  {  ……(aksi yang dikerjakan)  } |
| 7. | perulangan | do  {  …..(aksi yang dikerjakan)  }  while (syarat); |
| 8. | perulangan | for (nilai\_awal,syarat,operasi++/ --)  {  …..(aksi yang dikerjakan)  }; |

**Perangkat Lunak CodeVisionAVR (CVAVR)**

CodeVisionAVR pada dasarnya merupakan perangkat lunak pemrograman mikrokontroler keluarga AVR berbasis bahasa C. Ada tiga komponen penting yang telah diintegrasikan dalam perangkat lunak ini: compiler C, IDE dan Program generator. Berdasarkan spesifikasi yang dikeluarkan oleh perusahaan pengembangnya, compiler C yang digunakan hampir mengimplementasikan semua komponen standar yang ada pada bahasa C standar ANSI (seperti struktur program, jenis tipe data, jenis operator, dan pustaka fungsi standar-berikut penamaannya). Tetapi walaupun demikian, dibandingkan bahasa C untuk aplikasi komputer, compiler C untuk mikrokontroler ini memiliki sedikit perbedaan yang disesuaikan dengan arsitektur AVR tempat program C tersebut ditanamkan (embedded).

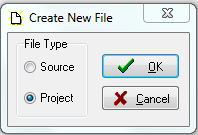
Khusus untuk pustaka fungsi, disamping pustaka standar (seperti fungsi-fungsi matematik, manipulasi string, pengaksesan memori dan sebagainya), CodeVisionAVR juga menyediakan fungsi-fungsi tambahan yang sangat bermanfaat dalam pemrograman antarmuka AVR dengan perangkat luar yang umum digunakan dalam aplikasi kontrol. Beberapa fungsi pustaka yang penting diantaranya adalah fungsi-fungsi untuk pengaksesan LCD, komunikasi I2C, IC RTC (Real time Clock), sensor suhu LM35, SPI (Serial Peripheral Interface) dan lain sebagainya. Untuk memudahkan pengembangan program aplikasi, CodeVisionAVR juga dilengkapi IDE yang sangat user friendly (lihat gambar 19). Selain menu-menu pilihan yang umum dijumpai pada setiap perangkat lunak berbasis Windows, CodeVisionAVR ini telah mengintegrasikan perangkat lunak downloader (in system programmer) yang dapat digunakan untuk mentransfer kode mesin hasil kompilasi kedalam sistem memori mikrokontroler AVR yang sedang diprogram. Secara garis besar bagian-bagian CVAVR dapat diuraikan seperti gambar berikut ini;



Gambar 3. IDE perangkat lunak CodeVisionAVR

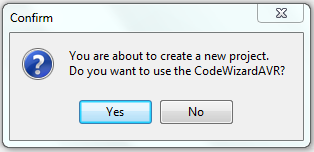
Untuk memulai menulis program didalam software CVAVR terlebih dahulu melakukan langkah-langkah sebagai berikut:

1. **File** → **New** → Pilih **Project**



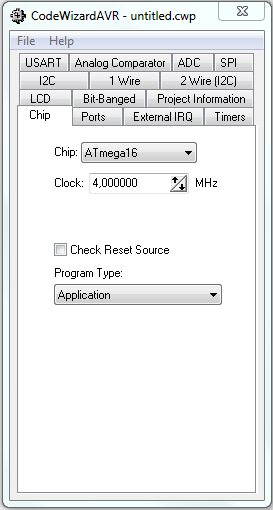
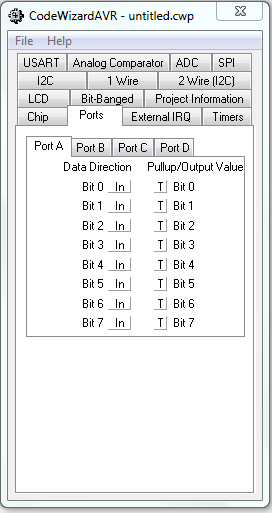
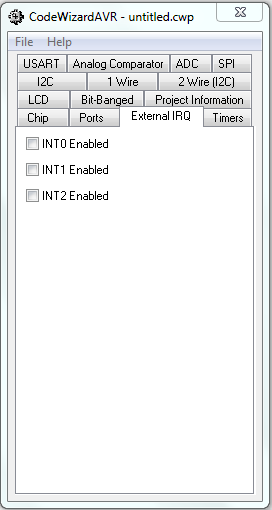
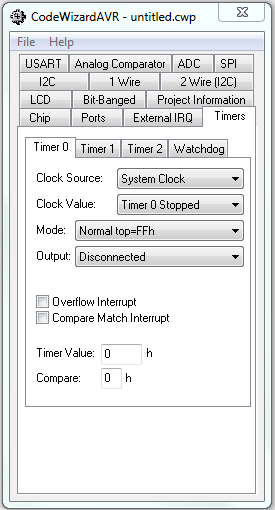
Gambar 4. Membuat File baru

1. Selanjutnya akan muncul window konfirmasi menggunakan AGP CodeWizardAVR → Yes



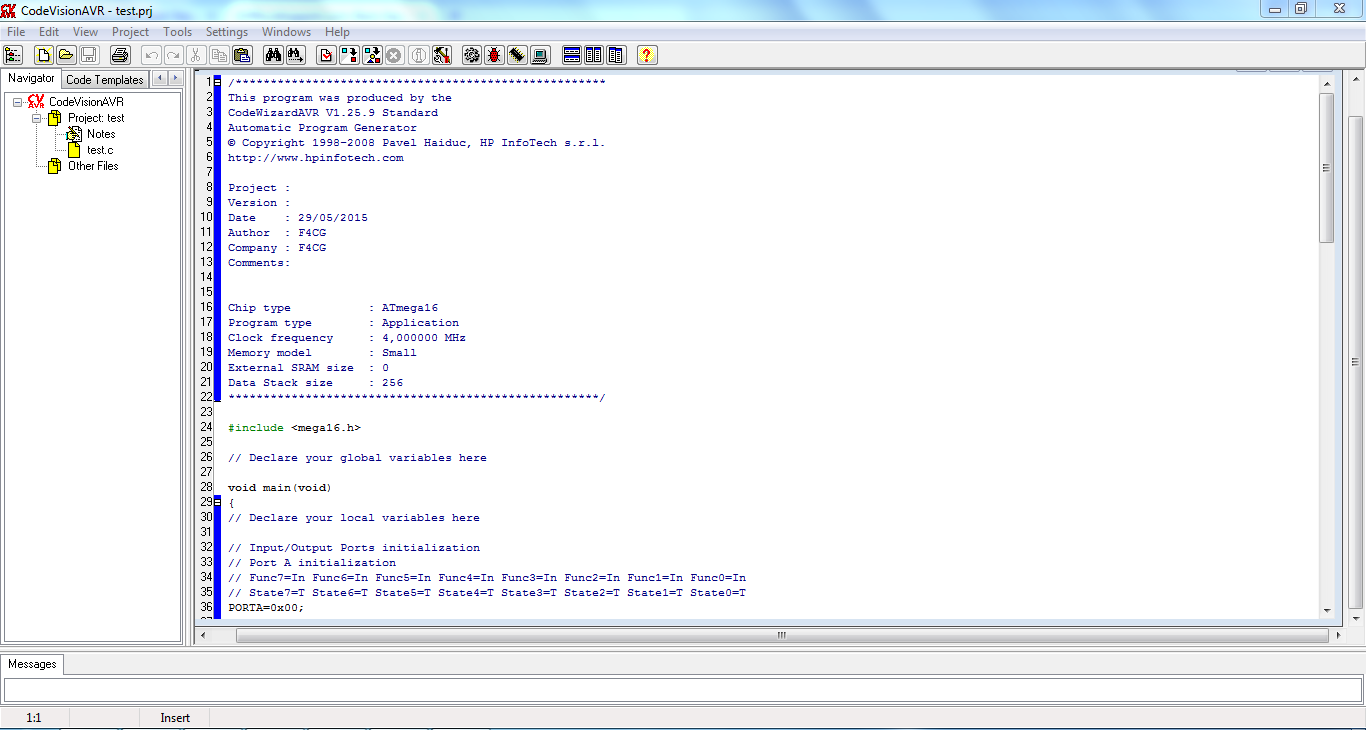
Gambar 5. Project baru menggunakan CodeWizardAVR

1. Window CodeWizardAVR digunakan untuk pengaturan PORT, USART, ADC, LCD, timer dan fasilitas yang lainnya sesuai dengan fungsi yang diinginkan.

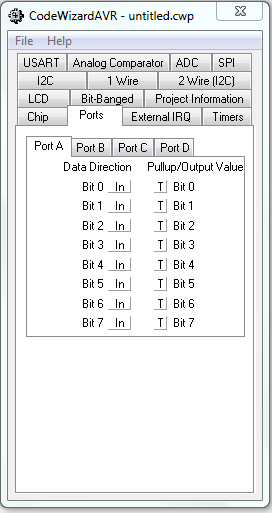


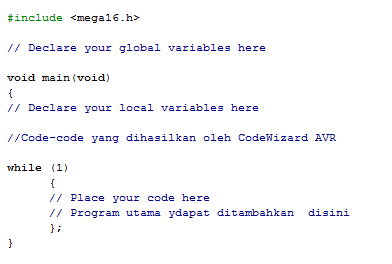
Gambar 6. Konfigurasi program melalui CodewizardAVR

1. Setelah selesai dengan pengaturan pada CodeWizardAVR pilih **File** → **Generate, Save and exit** (*catatan: pemeberian nama file sebanyak 3x; dengan nama file yang sama; hindari kalimat yang panjang, capital dan spasi*)
2. Selesai pemberian nama file, akan muncul window utama editor program seperti berikut:



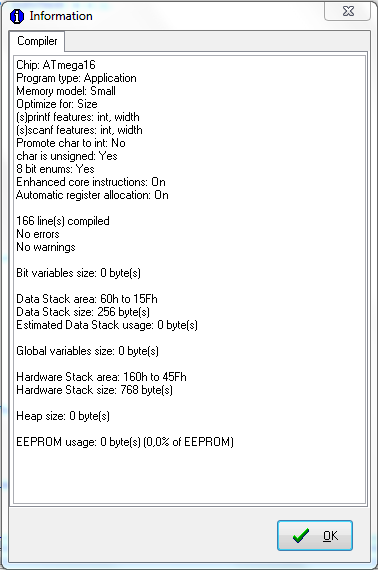
Gambar 7. Jendela Code program

1. Setelah berhasil membuat program menggunakan CodeWizardAVR, tambahkan variabel dan instruksi-instruksi tambahan ke dalam program. Kode program yang dihasilkan codewizard sebagai berikut:



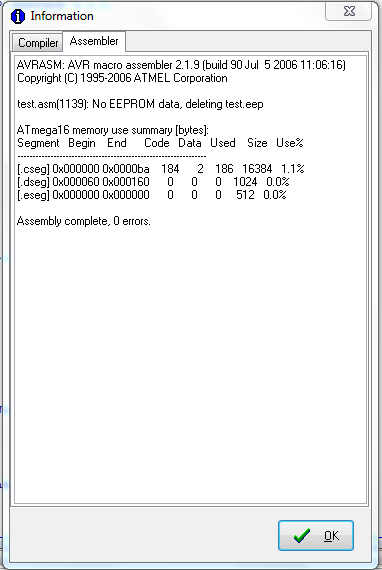
Gambar 8. Code Program yang dihasilkan

1. Jika sudah selesai membuat program, compile program. Pilih **Project** → **Compile**



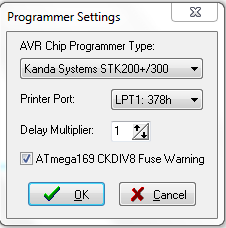
Gambar 9. Hasil Proses Kompilasi

1. Jika ada kesalahan, klik keterangan **error** atau **warning** yang terdapat pada bagian messages, kemudian letak kesalahan akan ditampilkan, perbaiki kesalahan tersebut dan compile kembali. Jika sudah tidak error, pilih **Project** → **Make**



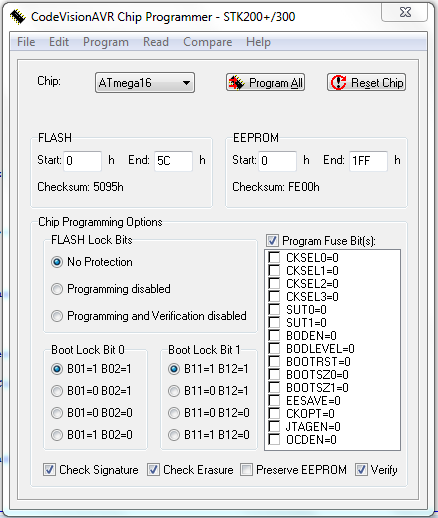
Gambar 10. Hasil Proses Make

1. Untuk memasukkan program yang sudah kita buat ke IC mikrokontroler AVR Atmega 16, lakukan terlebih dahulu setting programer, pilih **Setting → Programmer**, pilih Kanda System STK200+/300 untuk AVR Chip Programmer Type. Pilih printer port=LPT1:378h, biarkan Delay Multiplier=1 dan pilihan untuk Atmega16. Lalu tekan tombol **OK.**

****

Gambar 11. Setting Programmer

1. Untuk memasukkan program ke Chip Mikrokontroler Atmega16, Pilih **Tools → Chips Programmer → Program All**



Gambar 12. Mengisi Program ke dalam Chip Mikrokontroler

**MATERI 3**

**Praktik Mikrokontroler**

**Topik : AKSES I/O**

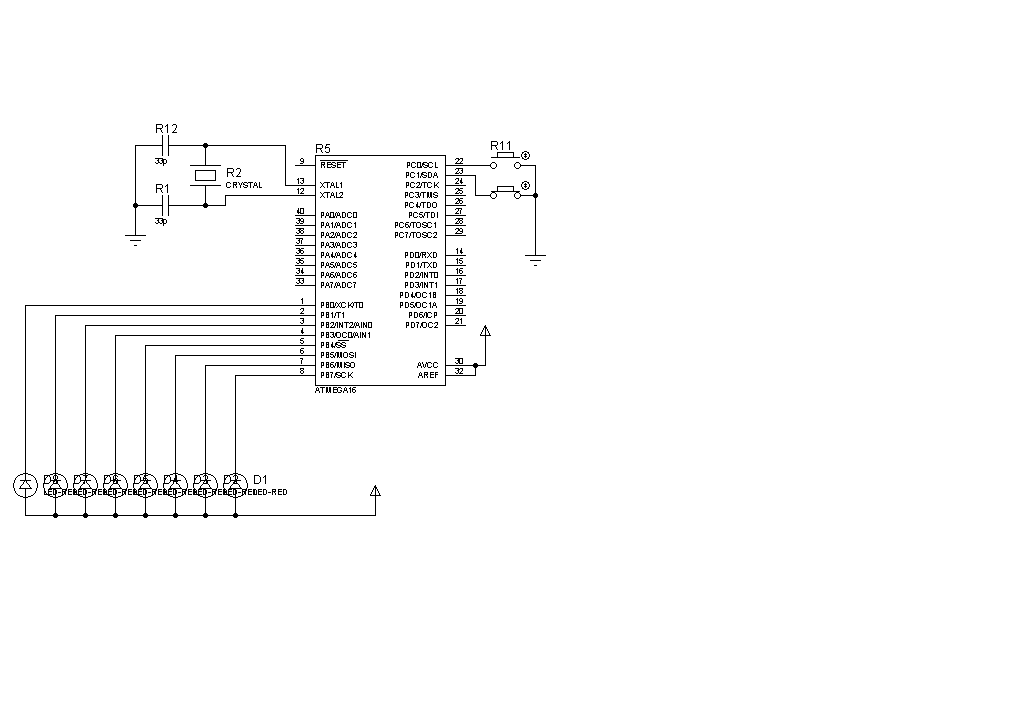
**Kajian Teori**

Pheriperal mikrokontroler keluarga AVR (ATMega16/8535) memungkinkan untuk diset sebagai keluaran dan masukan. Pengaturan tersebut dapat dilakukan dengan bantuan Code Wizard AVR pada salah satu port yang diinginkan. Penggunaan program secara langsung juga dapat dilakukan untuk mengatur fungsi dari pada setiap port pada mikrokontroler.

Untuk mengakses port pada mikrokontoler AVR maka anda harus mengenal register yang ada pada I/O port. Ada tiga **register bit** pada I/O port yaitu DDRx, PORTx dan PINx. Register DDRx digunakan untuk menentukan apakah port tersebut akan dijadikan sebagai input atau output. Sedangkan register PORTx dipakai untuk mengirim data keluar dari port ketika DDRx diset sebagai output. Dan register PINx dipakai untuk membaca data pada port ketika DDRx diset sebagai input.

Karena ketiga register di atas adalah register bit maka masing-masing pin pada port bisa diset secara bebas. Misalkan, kita menginginkan sebagian dari PORTA dijadikan sebagai input dan sebagian yang lain dijadikan sebagai output. Hal ini mudah dilakukan dengan mengatur register DDRA, sebagian sebagai input dan sebagian yang lain sebagai output. Agar pin pada port berfungsi sebagai input maka bit pada register DDRx diset 0. Sedangkan pin pada port akan berfungsi sebagai output ketika bit pada register DDRx diset1.

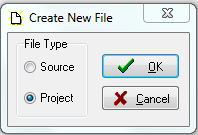
**Gambar Rangkaian Simulasi**



Gambar 13. Rangkaian Mikrokontroler

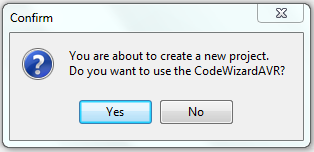
**Langkah Percobaan**

1. Buatlah rangkaian diatas ke dalam program Proteus 7 Profesional.
2. Buat Program ke dalam CV AVR. Setelah membuka Program CodeVision AVR Pilih **File** → **New** → Pilih **Project**



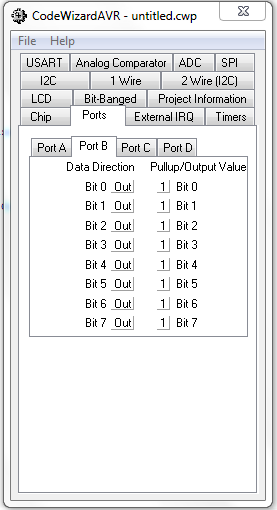
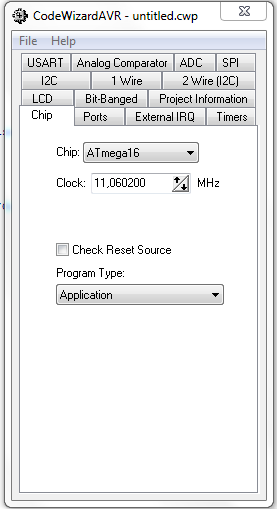
Gambar 14. Membuat File baru

1. Selanjutnya akan muncul window konfirmasi menggunakan AGP CodeWizardAVR → Yes



Gambar 15. Project baru menggunakan CodeWizardAVR

1. Atur Port sesuai dengan gambar rangkain dimana Port C.O digunakan sebagai input dan Port B.0 – B.7 digunakan sebagai output.



Gambar 16. Pengaturan Port dan Chip

1. Setelah selesai dengan pengaturan pada CodeWizardAVR pilih **File** → **Generate, Save and exit**
2. Masukkan program di bawah ini ke dalam bagian program utama

While

{

if(PINC.0==0)

{

PORTB.0=0; //LED bit 0 ON

}

else

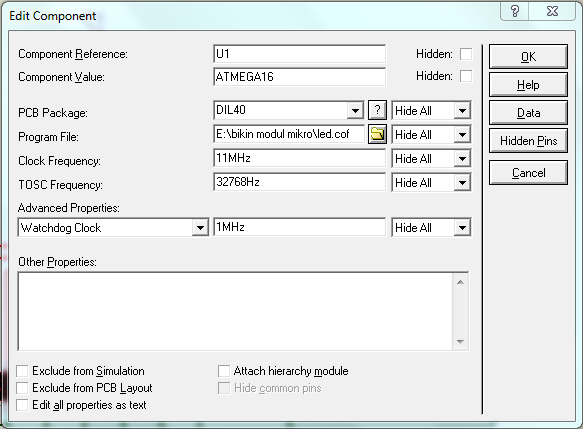
{

PORTB.0=1; //LED bit 0 OFF

};

};

1. Jika sudah selesai membuat program, compile program. Pilih **Project** → **Compile** sampai tidak ada error kemudian pilih **Project** →**Make**
2. Selanjutnya buka rangkaian di Proteus kemudian klik 2x di dalam komponen Mikrokontroler ATMega 16 sehingga akan muncul tampilan edit komponen. Selanjutnya pilih program file dengan file program CV AVR yang sudah dibuat dan pilih **OK.**



Pemilihan File CV AVR

1. Jalankan rangkaian di program ISIS Profesional (Proteus).
2. Dengan langkah yang sama ubah program tersebut dengan program seperti yang ada dibawah ini. Program LED Led-1 On, Led-2 Off, Led-3 On, Led-4 Off, Led-5 On, Led-6 Off, Led-7 On, Led-8 Off

|  |
| --- |
| while (1)  {  PORTB.0=0;  PORTB.1=1;  PORTB.2=0;  PORTB.3=1;  PORTB.4=0;  PORTB.5=1;  PORTB.6=0;  PORTB.7=1;  };  } |

1. Selanjutnya dengan cara yang sama coba program berikut dimana led berkedip bersamaan

|  |
| --- |
| #include <mega16.h>  #include <delay.h>  void main(void)  {  ……..  ……..  while (1)  {  PORTB=0xFF;  delay\_ms(1000);  PORTB=0x00;  delay\_ms(1000);  };  } |

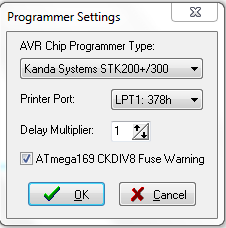
1. Selanjutnya dengan cara yang sama buat program di bawah ini dimana LED geser bergantian ke –kanan.

|  |
| --- |
| while (1)  {  PORTB=0b11111111;  delay\_ms(1000);  PORTB=0b11111110;  delay\_ms(1000);  PORTB=0b11111101;  delay\_ms(1000);  PORTB=0b11111011;  delay\_ms(1000);  PORTB=0b11110111;  delay\_ms(1000);  PORTB=0b11101111;  delay\_ms(1000);  PORTB=0b11011111;  delay\_ms(1000);  PORTB=0b10111111;  delay\_ms(1000);  PORTB=0b01111111;  delay\_ms(1000);  }; |

1. Untuk menggeser LED bergantian seperti program di atas, dapat dilakukan juga dengan program di bawah ini:

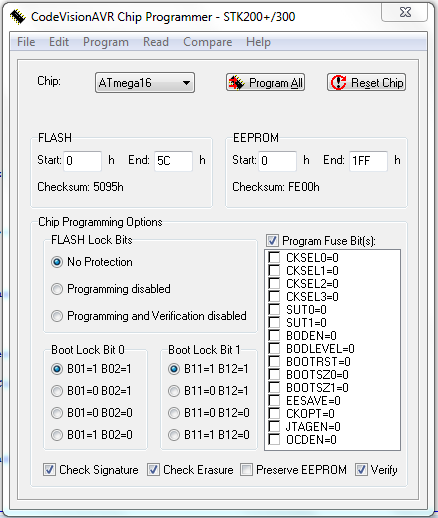
|  |
| --- |
| #include <mega16.h>  #include <delay.h>  int temp; // Declare your global variables here  void main(void)  {  PORTB=0xFF;  DDRB=0xFF; //inisialisasi portD sebagai pengganti output  temp=1<<8; // bit ke-8 di set berlogika 1  PORTD= ~ temp; //invert variable temp supaya hanya satu led yang menyala  delay\_ms(1000);  while (1)  {  delay\_ms(1000);  temp>>=1; //geser satu kali  PORTB= ~ temp; //invert variable temp supaya hanya satu led yang nyala  if (temp==0)  {  temp=1<<8; //bit ke 8 di set logika 1 kembali  }  };  } |

1. Masukkan program yang sudah kita buat ke IC mikrokontroler AVR Atmega 16, lakukan terlebih dahulu setting programer, pilih **Setting → Programmer**, pilih Kanda System STK200+/300 untuk AVR Chip Programmer Type. Pilih printer port=LPT1:378h, biarkan Delay Multiplier=1 dan pilihan untuk Atmega169. Lalu tekan tombol **OK.**

****

Gambar 18. Setting Programmer

1. Pilih **Tools → Chips Programmer → Program All**



Gambar 19. Mengisi Program ke dalam Chip Mikrokontroler

1. Cobalah di dalam Trainer Mikrokontroler

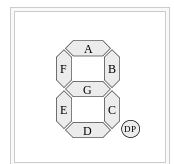
**Latihan Mandiri**

1. Buatlah program apabila ditekan Sw1 ditekan nyala LED berjalan bergantian ke kiri, apabila ditekan Sw2 nyala LED berjalan bergantian ke kanan.

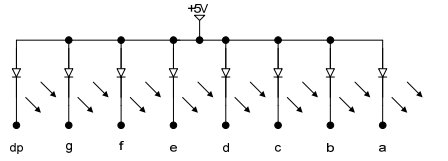
**TOPIK: SEVEN SEGMENT**

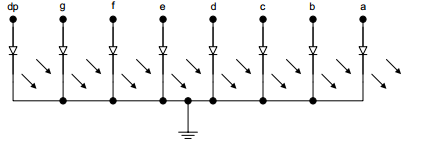
**Kajian Teori**

Mengakses 7 segment padaAVR dapat dilakukan dengan mode direct (8 bit), BCD (4 bit), maupun 8 bit multiplex. 7 segment adalah salah satu [perangkat layar](http://id.wikipedia.org/w/index.php?title=Perangkat_layar&action=edit&redlink=1) untuk menampilkan [sistem angka desimal y](http://id.wikipedia.org/wiki/Sistem_angka)ang merupakan alternatif dari layar *dot-matrix*. 7 segmen ini seringkali digunakan pada [jam digital,](http://id.wikipedia.org/wiki/Jam_digital) meteran elektronik, dan perangkat elektronik lainnya yang menampilkan informasi numerik.



Gambar 20. Konstruksi 7 segment display

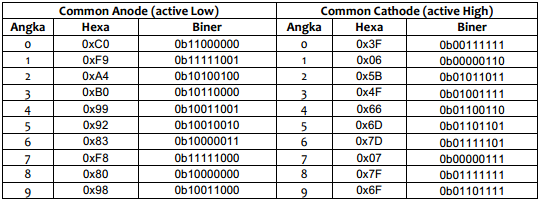
7 segment terdiri dari 2 jenis atau type yang beredar dipasaran, yaitu Common Anode dan Common Cathode. Common memiliki terjemahan “bersama”, artinya salah satu kutup pada 7 segment dijadikan menjadi satu, atau dapat dikatakan satu kaki 7 segment dipakai bersama dengan jenis kutup yang sejenis. Pengetahuan akan common pada setiap penggunaan 7 segment sangatlah penting, dikarena berkaitan dengan cara untuk menghidupkannya apakah active high atau active low. Secara skematik dua jenis tersebut dapat digambarkan sebagai berikut:



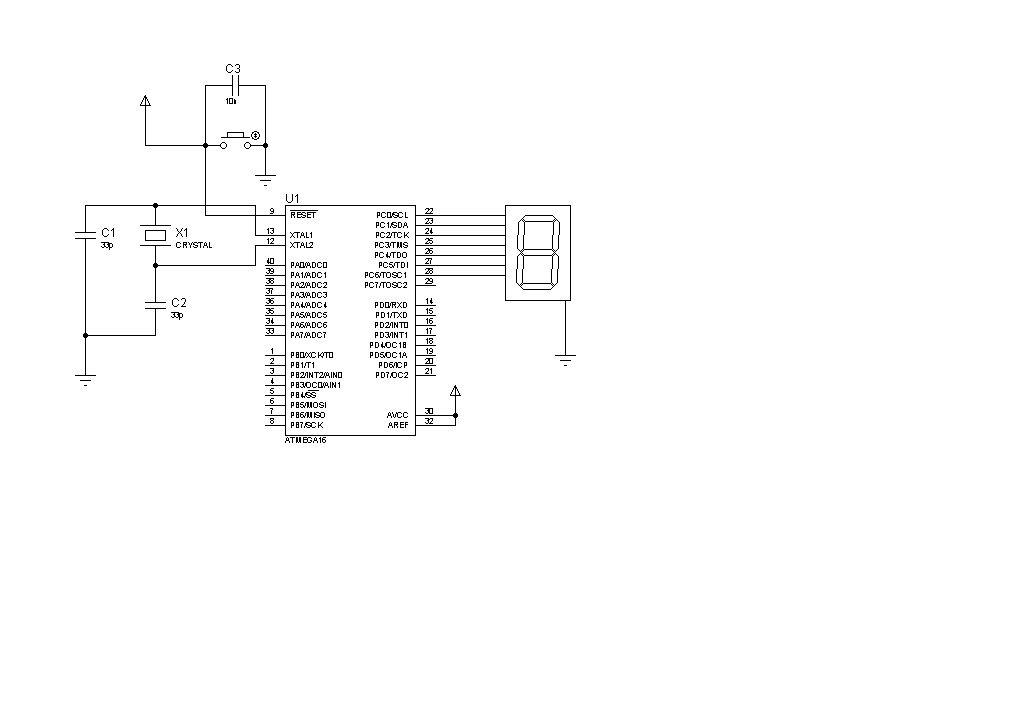
Gambar 21. Skematik Common Anoda dan Common Kathoda

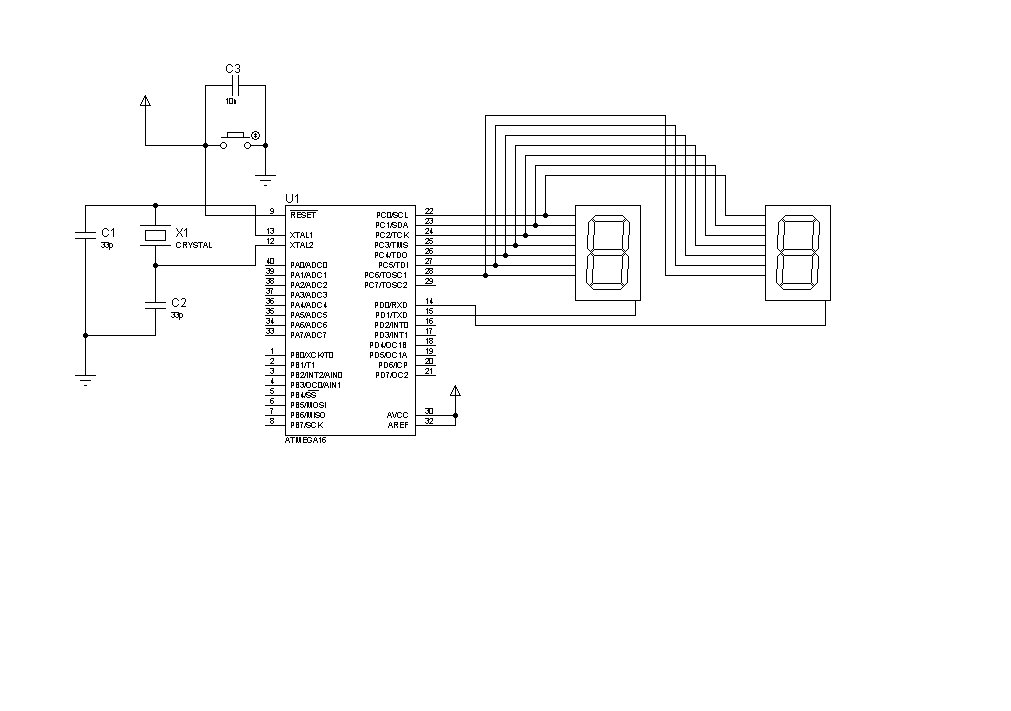
Cara kerja dari seven segmen common anode akan aktif pada kondisi low "0" dan akan off pada kondisi high "1". Sedangkan kode hexa common anode adalah kebalikan atau selisih dari common katode . Berikut tabel daftar data keluaran untuk menghidupkan 7 segment.

Tabel 6. daftar data keluaran untuk menghidupkan 7 segment



**Gambar Rangkaian**

****

****

Gambar 21. Rangkaian Seven Segment

**Langkah Percobaaan**

1. Rangkailah rangkaian diatas ke dalam simulasi ISIS Profesional (Proteus Profesional)
2. Buatlah Program ke dalam CV AVR seperti langkah pada percobaan sebelumnya dimana pada pengaturan Port sesuai dengan rangkaian.

|  |
| --- |
| #include <mega16.h>  #include <delay.h>  void main(void)  {  PORTC=0xFF;  DDRC=0xFF; //set port c sebagai output 7segment  PORTD=0x00;  DDRD=0x00;  #define seg\_a 0x01  #define seg\_b 0x02  #define seg\_c 0x04  #define seg\_d 0x08  #define seg\_e 0x10  #define seg\_f 0x20  #define seg\_g 0x40  #define seg\_h 0x80    while (1)  {  PORTC=~(seg\_g|seg\_h); //angka 0  delay\_ms(100);  PORTC=(seg\_b|seg\_c); // angka 1  delay\_ms(100);  PORTC=~(seg\_f|seg\_c|seg\_h); //angka 2  delay\_ms(100);  PORTC=~(seg\_f|seg\_e|seg\_h); // angka 3  delay\_ms(100);  PORTC=(seg\_f|seg\_g|seg\_b|seg\_c); //angka 4  delay\_ms(100);  PORTC=~(seg\_b|seg\_e|seg\_h); // angka 5  delay\_ms(100);  PORTC=~(seg\_a|seg\_b|seg\_h); //angka 6  delay\_ms(100);  PORTC=(seg\_a|seg\_b|seg\_c); // angka 7  delay\_ms(100);  PORTC=(~seg\_h); //angka 8  delay\_ms(100);  PORTC=~(seg\_e|seg\_d|seg\_h); // angka 9  delay\_ms(100);  };  } |

1. Compile dan Make program tersebut dengan langkah seperti materi sebelumnya dan masukkan program ke dalam simulasi Proteus Profesional. Setelah simulasi berjalan cobakan pada hardware nyata.
2. Selanjutnya dengan cara yang sama buatlah program dibawah ini untuk menampilkan angka 0 – 9.

|  |
| --- |
| #include <mega16.h>  #include <delay.h>  void main(void)  {  PORTC=0xFF;  DDRC=0xFF; //set port c sebagai output 7segment  PORTD=0x00;  DDRD=0x00;  while (1)  {  PORTC=0b00111111; //angka 0  delay\_ms(100);  PORTC=0b00000110; // angka 1  delay\_ms(100);  PORTC=0b01011011; //angka 2  delay\_ms(100);  PORTC=0b01001111; // angka 3  delay\_ms(100);  PORTC=0b01100110; //angka 4  delay\_ms(100);  PORTC=0b01101101; // angka 5  delay\_ms(100);  PORTC=0b01111101; //angka 6  delay\_ms(100);  PORTC=0b00000111; // angka 7  delay\_ms(100);  PORTC=0b01111111; //angka 8  delay\_ms(100);  PORTC=0b01101111; // angka 9  delay\_ms(100);  };  } |

1. Dengan cara yang sama, tambahkan digit seven segment (digit puluhan dan digit satuan) kemudian buat program di bawah ini:

|  |
| --- |
| #include <mega16.h>  #include <delay.h>  #include <stdio.h>  Unsigned char digit[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};  int satuan,puluhan,data,data\_temp; //temp=temporary data  void ambil\_data()  {  data\_temp=data;  satuan=data\_temp%10;  puluhan=(data\_temp/10)%10;  }  void tampilkan\_7\_seg()  {  PORTD.0=0;  PORTD.1=1;  PORTC=digit[satuan];  delay\_ms(500);  PORTD.0=1;  PORTD.1=0;  PORTC=digit[puluhan];  delay\_ms(1000);  }  void main(void)  {  data=11;  PORTA=0x00;  DDRA=0xFF;  PORTB=0xFF;  DDRB=0x00;  PORTC=0xFF;  DDRC=0xFF;  PORTD=0xFF;  DDRD=0xFF;  while (1)  {  ambil\_data();  tampilkan\_7\_seg();  };  } |

**Latihan Mandiri**

Buatlah program menghitung mundur dari 999 hingga 0

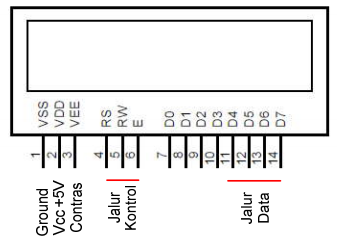
**TOPIK: AKSES LCD**

**KAJIAN TEORI**

Mengakses LCD padaAVR dapat dilakukan dengan mode 4 bit. Untuk menggunakan mode 4 bit, kita dapat menggunakan library built in pada Code Vision. Library ini telah terdapat berbagai fungsi untuk menampilkan karakter ke LCD. Beberapa perintah dasar akses LCD antara lain;

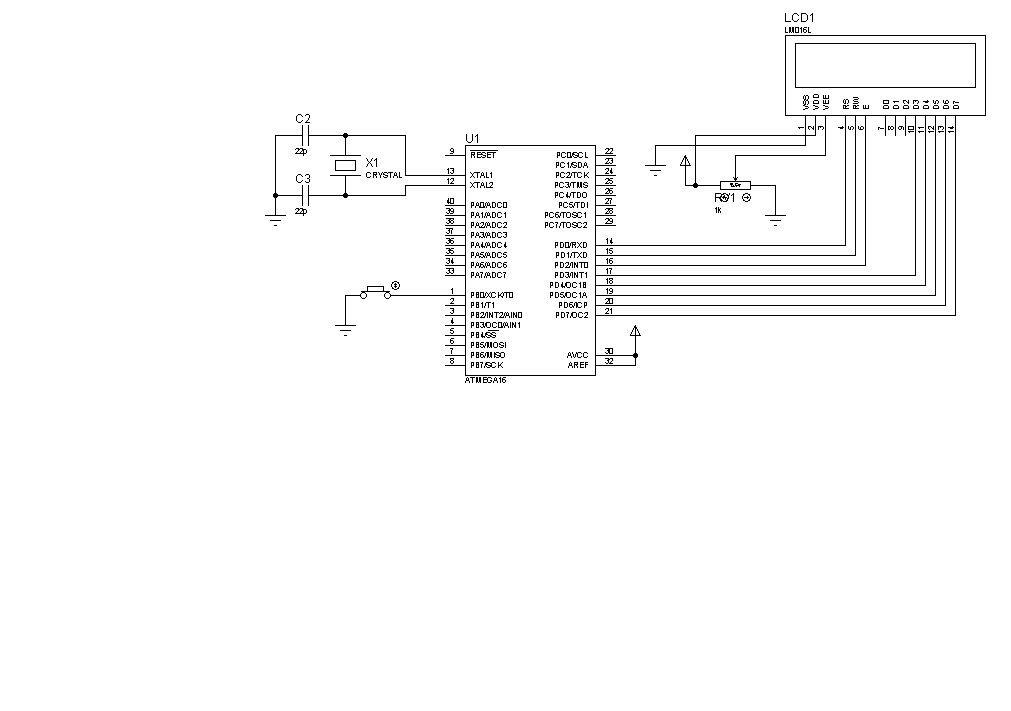
1. Membersihkan layar **LCDClear(**
2. Set cursor pada posisi tertentu **LCDGotoXY(x,y);** x merupakan colom pada LCD, y adalah barisnya. Pada LCD 16x2, terdapat 16 yakni 0-15 dan 0-1.
3. Inisialisasi LCD **lcd\_init(void)**
4. Menuliskan karakter ke LCD **lcd\_putchar(char c).** Misalnya **lcd\_putchar(”C”)** akan menuliskan karakter C ke LCD.
5. Menuliskan string ke LCD **lcd\_putsf(char\*str)**. Misalnya, lcd\_putsf(“TEMPERATURE”) akan menuliskan string TEMPERATURE pada LCD. Sebelum dapat menggunakan berbagai fungsi tersebut, pada bagian header dari program kita harus menginclude librarynya.

Berikut ilustrasi konfigurasi fungsi masing-masing pin:



Gambar 22. Fungsi tiap PIN

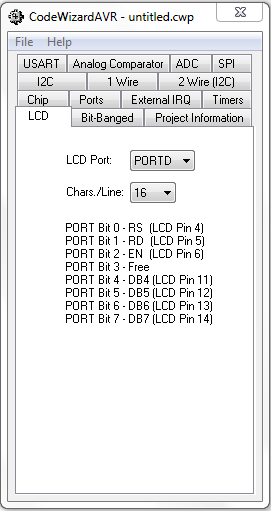
**Gambar Rangkaian**



Gambar 23. Rangkaian

**Langkah Percobaan**

1. Buatlah rangkaian diatas ke dalam program Proteus 7 Profesional.
2. Buat Program ke dalam CV AVR. Setelah membuka Program CodeVision AVR Pilih **File** → **New** → Pilih **Project**
3. Selanjutnya akan muncul window konfirmasi menggunakan AGP CodeWizardAVR → Yes (Seperti percobaan sebelumnya)
4. Untuk mendefinisikan port yang terhunung ke LCD pilih LCD → Pilih PORTD

  
Gambar 24. Konfigurasi LCD Port

1. Jika sudah mengonfigurasi project, pilih file → generate, save and exit, sehingga akan tampil source code dalam bahasa C. Setelah itu, tambahkan instrksi-instruksi tambahan ke dalam program yang sudah ada.

|  |
| --- |
| #include <mega16.h>  #include <delay.h>  // Alphanumeric LCD Module functions  #asm  .equ \_\_lcd\_port=0x12 ;PORTD  #endasm  #include <lcd.h>  // Declare your global variables here  void main(void)  {  //..... program lain yang diperoleh dari CodeWizard AVR  // LCD module initialization  lcd\_init(16);  while (1)  {  // Place your code here  lcd\_gotoxy(0,0); //masuk ke i digit pertama  lcd\_putsf("PELATIHAN"); //  };  } |

1. Compile dan Make program tersebut dengan langkah seperti materi sebelumnya dan masukkan program ke dalam simulasi Proteus Profesional. Setelah simulasi berjalan cobakan pada hardware nyata.
2. Dengan cara yang sama buatlah program untuk menampilkan tulisan berkedip.

|  |
| --- |
| #include <mega16.h>  #include <delay.h>  // Alphanumeric LCD Module functions  #asm  .equ \_\_lcd\_port=0x12 ;PORTD  #endasm  #include <lcd.h>  // Declare your global variables here  void main(void)  {  //..... program lain yang diperoleh dari CodeWizard AVR  // LCD module initialization  lcd\_init(16);  while (1)  {  // Place your code here  lcd\_gotoxy(0,0); //masuk ke i digit pertama  lcd\_putsf("PELATIHAN");  delay\_ms(500);  lcd\_clear();  delay\_ms(100);  };  } |

**Praktik Mandiri**

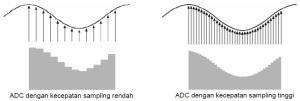
1. Buatlah program jika tombol ditekan, LCD menampilkan “tombol1 di tekan” dan 8 led pada PortA menyala. Jika tombol tidak ditekan, LCD akan menampilkan “silakan tekan!” dan 8 led di portA padam.
2. buatlah program dengan tulisan “LOADING…” berjalan dari kiri ke kanan setelah itu muncul “PELATIHAN CV AVR” tulisan berjalan dari kanan kekiri.

**TOPIK : ADC (*ANALOG TO DIGITAL CONVERTION*)**

**Kajian Teori**

ADC **(***Analog To Digital Converter***)** adalah perangkat elektronika yang berfungsi untuk mengubah sinyal analog (kontinyu) menjadi sinyal digital (deskret). Perangkat ADC dapat berbentuk suatu modul atau rangkaian elektronika maupun suatu chip IC. ADC berfungsi untuk menjembatani pemrosesan sinyal analog oleh sistem digital.

Kecepatan sampling suatu ADC menyatakan “seberapa sering sinyal analog dikonversikan ke bentuk sinyal digital pada selang waktu tertentu”. Kecepatan sampling biasanya dinyatakan dalam *sample per second* (SPS).



Gambar 25. Kecepatan sampling ADC dalam ketelitian

Resolusi ADC menentukan “ketelitian nilai hasil konversi ADC”. Sebagai contoh: ADC 8 bit akan memiliki output 8 bit data digital, ini berarti sinyal input dapat dinyatakan dalam 255 (2n –1) nilai diskrit, ADC 10 bit memiliki 1023 nilai deskret. ADC 12 bit memiliki 12 bit output data digital, ini berarti sinyal input dapat dinyatakan dalam 4096 nilai diskrit. Dari contoh diatas ADC 12 bit akan memberikan ketelitian nilai hasil konversi yang jauh lebih baik daripada ADC 8 bit.

Resolusi ADC = Vref/(nilai bit-1)

Contoh

Jika diketahui Vref = 5 V

Prosesor = ATmega16 (8bit)

Resolusi ADC = 5/(256-1)

= 0,0196 V

Artinya setiap kenaikan 0,0196 V maka nilai ADC akan bertambah 1 nilai deskret ADC. Dengan cara sebaliknya akan diperoleh nilai tegangan input. Resulosi dapat ditingkatkan dengan memperkecil nilai referensi, misalnya:

Jika diketahui Vref = 2,5 V

Prosesor = ATmega16 (8bit)

Resolusi ADC = 2,5/(256-1)

= 0,0098 V

Dengan demikian dapat diartikan bahwa setiap kenaikan 0,0098 V maka nilai deskret ADC akan naik 1 poin. Prinsip kerja ADC mengkonversi sinyal analog ke dalam bentuk besaran yang merupakan rasio perbandingan sinyal input dan tegangan referensi. Sebagai contoh, bila tegangan referensi 5 volt, tegangan input 3 volt, rasio input terhadap referensi adalah 60%. Jadi, jika menggunakan ADC 8 bit dengan skala maksimum 255, akan didapatkan sinyal digital sebesar 60% x255 = 153 (bentuk decimal) atau 10011001 (bentuk biner).

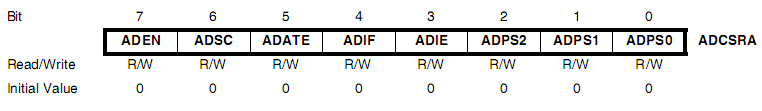
Sinyal Analog = (sample/max\_value) \* *reference voltage*

= (153/255) \* 5

= 3 Volts

AVR ATMega16 memiliki fitur ADC sebanyak 8 channel dengan resolusi 10 bit register yang digunakan untuk setting ADC ini adalah:

1. **ADCSRA – ADC Control and Status Register**



Gambar 26. Register ADCSRA

* Bit 7 – ADEN : ADC Enable

Diisi 1 untuk mengaktifkan ADC, diisi 0 untuk mematikan ADC sekaligus memberhentikan konversi yang sedang berlangsung.

* Bit 6 – ADSC : ADC Start Conversion

Pada mode single-conversion, set bit ini untuk memulai tiap konversi. Pada mode free-running, set bit ini untuk konversi pertama kalinya. Bit ADSC bila dibaca akan bernilai 1 selama proses konversi, dan bernilai 0 bila konversi selesai. Mengisi bit ini dengan nilai 0 tidak akan mempunyai dampak.

* Bit 5 – ADATE : ADC Auto Trigger Enable

Bila bit ini diisi 1 maka auto trigger ADC akan diaktifkan. ADC akan memulai konversi pada saat tepi positif dari sumber sinyal trigger yang dipilih. Sumber sinyal trigger ditentukan dengan menseting bit ADTS pada register SFIOR.

* Bit 4 – ADIF : ADC Interrupt Flag

Bit ini akan bernilai 1 pada saat ADC selesai mengkonversi dan Data register telah diupdate. ADC Conversion Complete Interrupt akan dijalankan bila bit ADIE dan bit-I pada register SREG diset 1. ADIF akan di-clear secara hardware bila mengerjakan penanganan vektor interrupt yang bersesuaian.

Alternatifnya, ADIF dapat di-clear dengan menuliskan 1. Hati- hati bila bekerja dengan Read-Modify-Write pada ADCSRA, interrupt yang tertunda dapat dinonaktifkan/batal. Hal ini juga berakibat sama bila instruksi SBI dan CBI digunakan.

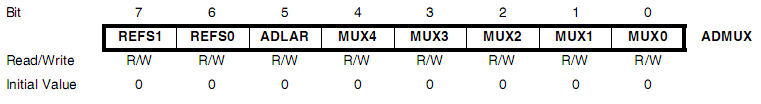
* Bit 3 – ADIE : ADC Interrupt Enable

Mengisi bit ini dan bit-I pada register SREG menjadi 1 akan mengaktifkan ADC Conversion Complete Interrupt.

* Bit 2:0 – ADPS2:0 – Bit pemilih ADC Prescaler

Menentukan bilangan pembagi antara sumber clock XTAL ke clock ADC.

1. **ADMUX – ADC Multiplexer Selection Register**



Gambar 27. Register ADMUX

* Bit 7:6 – REFS1:0 : Bit Pemilih tegangan referensi

00: vref = AREF

01: vref = AVCC dengan ekstrenal capasitor pada AREF

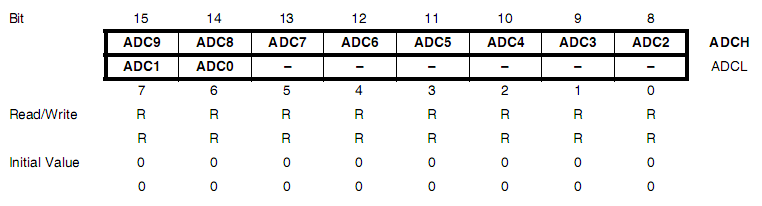
10: vref= reserved

11: vref=internal 2,56 volt dengan eksternal kapasitor pada AREF

* Bit 5 – ADLAR : ADC Left Adjust Result
* Bit 4:0 – MUX4:0 : Bit pemilih Analog Channel dan Gain

1. **ADCH – ADC data register**

Bila ADCLAR = 1

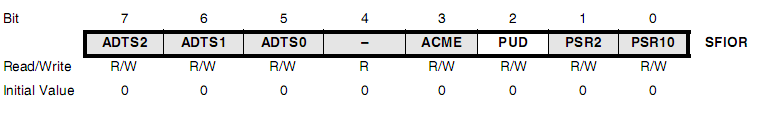


Gambar 28. ADC data register

Setelah ADC selesai melakukan konversi kedua register ini berisi hasil konversi. Bila channel differensial dipilih maka hasilnya dalam format 2’s complement. Saat ADCL dibaca, data register tidak akan meng-update data sampai ADCH dibaca. Jika hasilnya dirata kiri (left adjust) dan hanya butuh 8-bit maka cukuplah dengan membaca ADCH. Jika butuh 10-bit, baca ADCL dahulu kemudian ADCH.

1. **SFIOR – Special Function I/O Register**

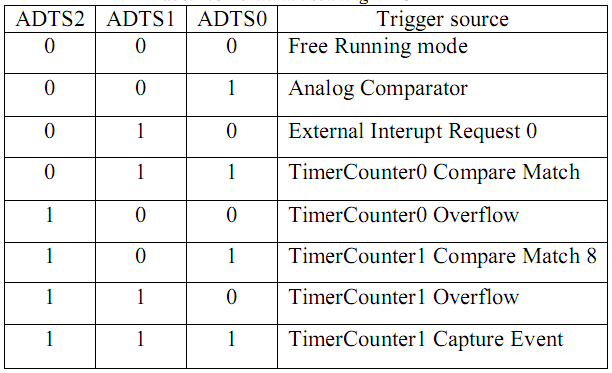
Register SFIOR berfungsi untuk sumber auto triger. Dimana kita dapat memilih beberapa mode untuk konversi.



Gambar 29. Register SFIOR

Dengan konfigurasi seperti dibawah maka dapat memilih mode start ADC, ADC akan konversi ketika berdasarkan mode yang dipilih.

Tabel 7. Pemilihan scaning ADC



* Bit 7:5 – ADTS2:0 : ADC Auto Trigger Source

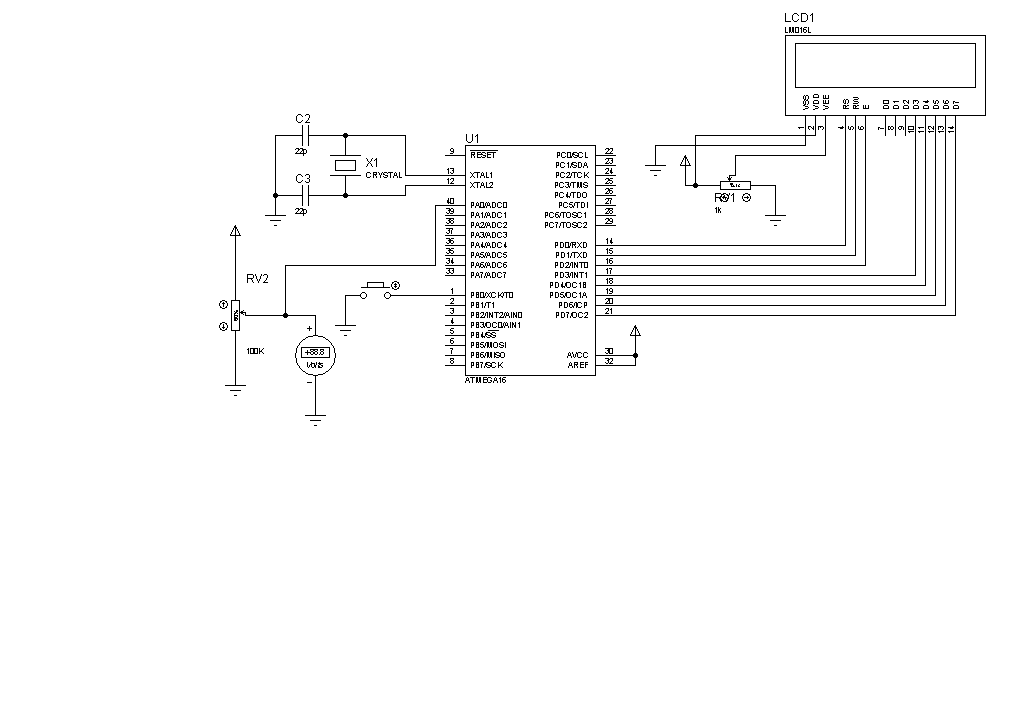
Bila ADATE dalam register ADCSRA diset 1, maka nilai dalam bit-bit ini akan menentukan sumber mana yang akan mentrigger konversi ADC. Bila bit ADATE bernilai 0, maka bit-bit ini tidak akan mempunyai efek. Sebuah konversi ditrigger oleh sinyal rising-edge dari interrupt flag yang dipilih. Perlu diingat bahwa memindah sumber trigger yang di-clear ke sumber trigger lain yang di-set akan menyebabkan positive-edge pada sinyal trigger.

Bila ADEN dalam register ADCSRA diset, juga akan memulai konversi. Memindah mode ke mode freerunning tidak akan menyebabkan pulsa trigger, meskipun bila flag interrupt ADC diset.

* Bit 4 – RES : Reserved bit

Bit cadangan, bila dibaca hasilnya nol.

**Gambar Rangkaian**

****

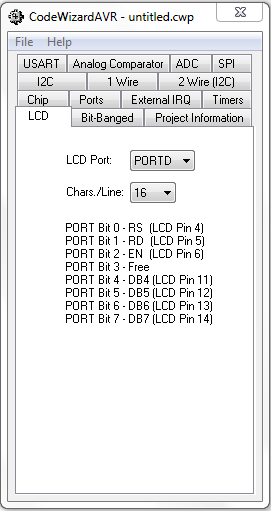
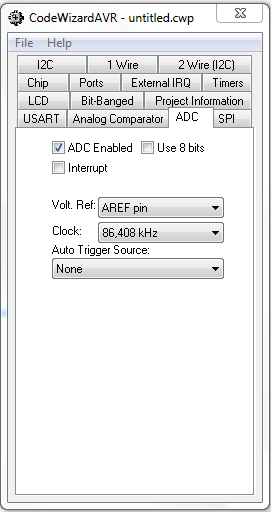
Aref = analog referensi diset 5V

Dapat dimodifikasi nilai ref <=5V

Gambar 30. Rangkaian Mikrokontroler

**Langkah Percobaan**

1. Buatlah rangkaian diatas dengan menggunakan program Proteus 7 Profesional.
2. Buat Program ke dalam CV AVR. Setelah membuka Program CodeVision AVR Pilih **File** → **New** → Pilih **Project**
3. Selanjutnya akan muncul window konfirmasi menggunakan AGP CodeWizardAVR → Yes (Seperti percobaan sebelumnya)
4. Untuk mendefinisikan port yang terhubung ke LCD pilih LCD → Pilih PORTD dan pengaturan ADC.

   
Gambar 24. Konfigurasi LCD Port dan ADC

1. Jika sudah mengonfigurasi project, pilih file → generate, save and exit, sehingga akan tampil source code dalam bahasa C. Setelah itu, tambahkan instrksi-instruksi tambahan ke dalam program yang sudah ada.

|  |
| --- |
| #include <mega16.h>  #include <stdio.h>  // Alphanumeric LCD Module functions  ...... dari CodeWizard AVR  // Read the AD conversion result  ....... dari CodeWizard AVR  // Declare your global variables here  unsigned char data=0;  char kata[16];  void main(void)  {  ...... dari CodeWizard AVR  // LCD module initialization  lcd\_init(16);  while (1)  {  data=read\_adc(0); //baca data yang masuk pada ADC0 (PINA0)  sprintf(kata,"suhu : %d",data); //tampilkan data pada LCD  lcd\_gotoxy(0,0); //posisi pada layar lcd  lcd\_puts(kata);  delay\_ms(100); //waktu tunda biar nggak kecepetan  lcd\_clear(); //clear layar LCD  };  } |

1. Compile dan Make program tersebut dengan langkah seperti materi sebelumnya dan masukkan program ke dalam simulasi Proteus Profesional. Setelah simulasi berjalan cobakan pada hardware nyata.

**Praktik Mandiri**

Dengan cara yang sama, buatlah program dengan untuk membuat termometer dengan sensor LM35 DZ.

**TOPIK : KENDALI MOTOR DC MENGGUNAKAN PWM (Pulse Width Modulation)**

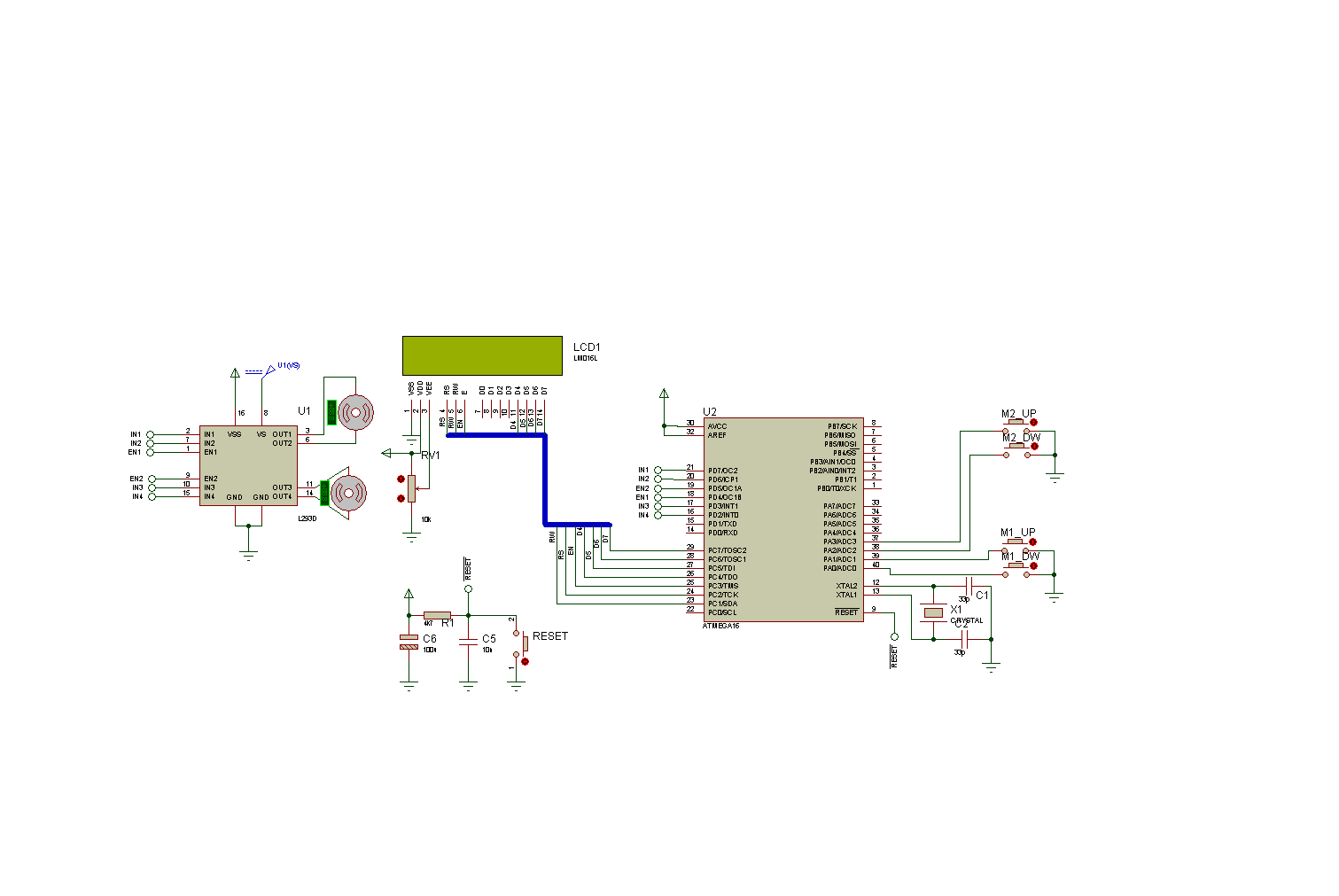
**Kajian Teori**

PWM (Pulse Width Modulation) adalah salah satu teknik modulasi dengan mengubah lebar pulsa (duty cylce) dengan nilai amplitudo dan frekuensi yang tetap. Satu siklus pulsa merupakan kondisi high kemudian berada di zona transisi ke kondisi low. Lebar pulsa PWM berbanding lurus dengan amplitudo sinyal asli yang belum termodulasi. Duty Cycle merupakan representasi dari kondisi logika high dalam suatu periode sinyal dan di nyatakan dalam bentuk (%) dengan range 0% sampai 100%, sebagai contoh jika sinyal berada dalam kondisi high terus menerus artinya memiliki duty cycle sebesar 100%. Jika waktu sinyal keadaan high sama dengan keadaan low maka sinyal mempunyai duty cycle sebesar 50%.

Aplikasi penggunaan PWM biasanya ditemui untuk pengaturan kecepatan motor dc, pengaturan cerah/redup LED, dan pengendalian sudut pada motor servo. Contoh penggunaan PWM pada pengaturan kecepatan motor dc semakin besar nilai duty cycle yang diberikan maka akan berpengaruh terhadap cepatnya putaran motor. Apabila nilai duty cylce-nya kecil maka motor akan bergerak lambat.

Fitur PWM pada ATMEGA 16 memiliki resolusi sebesar 8 bit jadi bernilai (2^8) = 256, dengan range 0-255. Resolusi yang dimaksud yaitu rentang data (range) yang mampu dibaca oleh mikrokontroler terhadap nilai PWM-nya.

Gambar Rangkaian



Langkah Percobaan

1. Buatlah rangkaian diatas dengan menggunakan program proteus 7 profesional
2. Buat Program ke dalam CV AVR. Setelah membuka Program CodeVision AVR Pilih **File** → **New** → Pilih **Project**
3. Selanjutnya akan muncul window konfirmasi menggunakan AGP CodeWizardAVR → Yes (Seperti percobaan sebelumnya)
4. Untuk mendefinisikan port yang terhubung ke LCD pilih LCD → Pilih PORTC dan pengaturan input tombol
5. Pengaturan Timer