AVR

Programming in C

Suprapto

Mengapa Program AVR dalam C?

1. Bahasa C Lebih Mudah dan membutuhkan Waktu cepat dibandingkan assembly
2. C lebih mudah di modifikasi dan diupdate.
3. Anda dapat menggunakan code yang tersedia dalam fungsi pustaka.
4. Code C lebih portable
5. Pada microcontroller dengan modifikasi sedikit atau tanpa sama sekali
6. Walaupun ukuran file HEX bahasa Assembly yang dihasilkan lebih kecil dibanding C tapi Pemrograman pada Assembly language lebih membosankan (*tedious*) dan membutuhkan waktu lama.

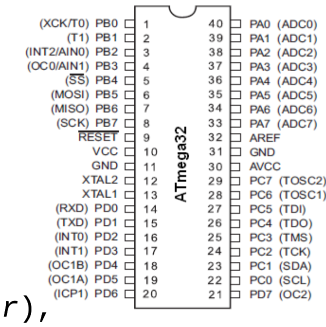
Type data dalam C

Tipe data	Size	Range data
unsigned char	8-bit	0 sampai 255
char	8-bit	-128 sampai +127
unsigned int	16-bit	0 sampai 65,535
int	16-bit	-32,768 sampai +32,767
unsigned long	32-bit	0 sampai 4,294,967,295
long	32-bit	-2,147,483,648 sampai +2,147,483,648
float	32-bit	±1.175e-38 sampai ±3.402e38
double	32-bit	±1.175e-38 sampai ±3.402e38

Program 1

```
// Program dibawah ini mengirim data 00-FF ke Port B.

#include <avr/io.h>         //standard AVR header

int main(void)
{
   unsigned char z;

   DDRB = 0xFF;              //PORTB sebagai Out

   for(z = 0; z <= 255; z++)
       PORT B = z;
   return 0;
}
```

I/O Ports in AVR



- ATmega32/16:IC 40 pin
- dibagi menjadi 4 port
  PORTA, PORTB, PORTC, PORTD.
- Tiap port mempunyai 3 register I/O
- Ketiga register tersebut
  DDRx (***D****ata* ***D****irection* ***R****egister*),
  PORTx(*Data Register*)
  PINx(***P****ort* ***IN****put pins*)
- Misalnya untuk PortB mempunyai register **PORTB**, **DDRB**, **PINB**.
- Tiap reister I/O registers mempunyai lebar data 8 bit, dan tiap port mempunyai maksimum 8 pin.

I/O Ports in AVR

Port	alamat	digunakan	Port	alamat	digunakan
PORTA	**$3B**	**Output**	**PORTC**	**$35**	**Output**
DDRA	**$3A**	**Direction**	**DDRC**	**$34**	**Direction**
PINA	**$39**	**Input**	**PINC**	**$33**	**Input**
PORTB	**$38**	**Output**	**PORTD**	**$32**	**Output**
DDRB	**$37**	**Direction**	**DDRD**	**$31**	**Direction**
PINB	**$36**	**Input**	**PIND**	**$30**	**Input**

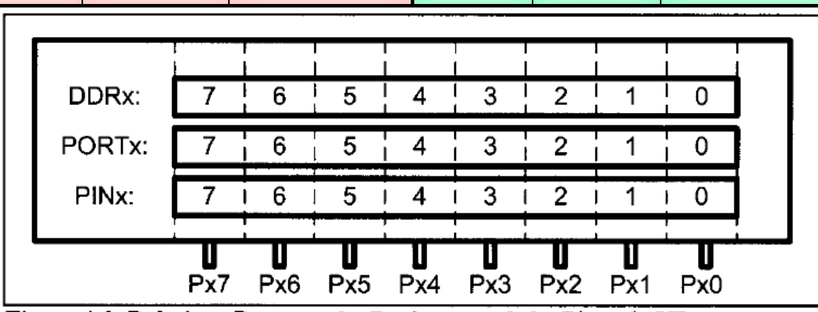
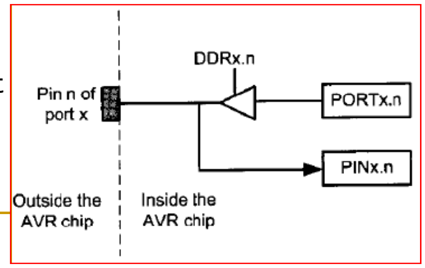


Figure 4-2. Relations Between the Registers and the Pins of AVR

Data Direction Register (DDRx)

- Register **DDRx** digunakan untuk tujuan membuat port ***input*** atau ***output***.
- Jika diisi data **1** pada reg **DDRx** maka **PORTx** sebagai **Output**.
- Jika diisi data **0** pada reg **DDRx** maka **PORTx** sebagai **Input**

```
DDRC = 0xFF;
//konfig PORTC sebagai output

DDRA = 0x00;
//konfig PORTC sebagai input
```



Port Input Pin Register (PINx)

- Untuk **read** data pada pin mikrokontroller, harus membaca pada register **PINx**.
- Untuk mengirim **data out** pada pin, harus menggunakan register **PORTx**.
- Pada saat sebagai masukan tersedia resistor pull-up internal pada tiap pin.

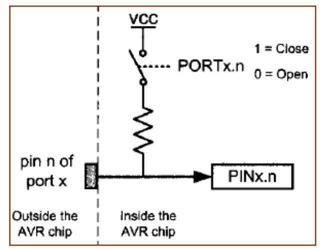
Perbedaan kondisi pada Pin Mikrokontroller AVR

PORTx	DDRx	
	0 (sbg input)	1 (sbg Output)
0	Input & high impedance	Out 0
1	Input & Pull-up	Out 1

Data Register (PORTx)

- Register PORTx sebagai kendali pull-up, aktif atau tidak
- Tulis data 1 ke register PORTx akan mengaktifkan resistor pull-up internal
- Tulis data 0 ke register PORTx akan deactivate atau mematikan resistor pull-up internal

```
DDRA = 0x00;
//konfigurasi PORTA sbg input

PORTA = 0xFF;
//aktifkan resistor pull-up
```



Program 2

```
// program untuk mengirim data HEX dengan nilai ASCII
// karakter of 0,1,2,3,4,5,A,B,C,D ke Port B.
#include <avr/io.h>        //standard AVR header
int main(void){            //the code starts from here
   unsigned char myList[] = "012345ABCD";
   unsigned char z;
   DDRB = 0xFF;            //PORTB is output
   for(z=0; z<10; z++)     //ulangi 10 kali dan increment z
       PORTB = myList[z] ; //keluarkan caracter ke PORTB
   while(1);               //needed if running on a trainer
   return 0;
}
```

Program 3

```
// Program ini mengeluarkan data toggle pada semua bit
   Port B sebanyak 200 kali.
#include <avr/io.h>         // standard AVR header
int main(void){             // code start from here
   DDRB = 0xFF;             // PORTB is output
   PORTB = 0xAA;            // PORTB is 10101010
   unsigned char z;
   for(z=0; z < 200; z++)  // jalankan sebanyak 200 kali
       PORTB = ~ PORTB;     // toggle PORTB
   while(1);                // stay here forever
   return 0;
}
```

Program 4

```
// Program mengirim nilai -4 sampai +4 ke Port B.

#include <avr/io.h>         //standard AVR header

int  main(void){
   char mynum[] = {-4,-3,-2,-1,0,+1,+2,+3,+4} ;
   unsigned char z;
   DDRB = 0xFF;             // PORTB   sebagai   output
   for( z=0 ; z<=8 ; z++)
       PORTB  =  mynum[z];
   while(1);                // stay  here   forever
   return 0;
}
```

Program 5

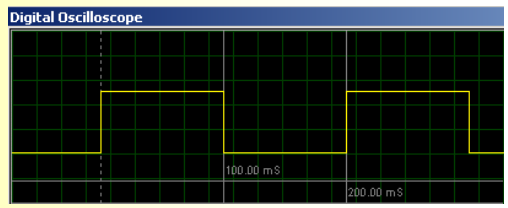
```
// program toggle semua bit pada Port B 50,000 kali.
#include <avr/io.h>        //standard AVR header

int main(void){
   unsigned int z;
   DDRB = 0xFF;             //PORTB   sebagai  output

   for( z=0 ; z<50000 ; z++){
        PORTB = 0x55;
        PORTB = 0xAA;
   }
   while(1);                //stay here  forever
   return 0;
}
// jalankan dan amati program diatas menggunakan simulator
```

Program 6

```
// program toggle semua bit pada Port B 100,000 kali.

#include <avr/io.h> // standard AVR header
int main(void)
  {
   unsigned long z;  // tipe data unsigned :(65535)
   DDRB = 0xFF;      // PORTB sebagai output

   for( z=0 ; z<100000 ; z++){
        PORTB = 0x55;
        PORTB = 0xAA;
   }
   while(1);         //stay here forever
   return 0;
}
```

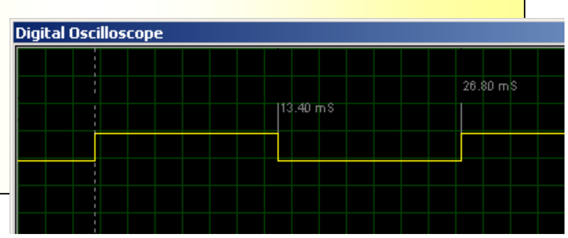
Program 7

```
// Program toggle semua bit pada Port B secara kontinu
// dengan delay 100 ms,dimana sistem uC diberi XTAL=8MHz.

#include <avr/io.h>         // header AVR standard

void delay100ms(void){      // coba beri nilai angka beda
   unsigned int i;          // compiler dan uji hasilnya
   for(i=0; i<42150; i++);
}

int main(void){
   DDRB = 0xFF;             // PORTB sebagai output
   while(1){
        PORTB = 0xAA;
        delay100ms() ;

        PORTB = 0x55;
        delay100ms();
   }
   return 0;
}
```



Program 8

```
// program toggle secara terus-menerus melalui Port B
// secara kontinu dengan waktu delay 10 ms.
// gunakan predefined fungsi delay Win AVR.

#include <util/delay.h>     //fungsi delay
#include  <avr/io.h>        //header AVR standard

int  main(void){

   DDRB   =   0xFF;         //PORTB  sebagai  output

   while(1)
   {
        PORTB = 0xAA;
        _delay_ms(10);

        PORTB = 0x55;
        _delay_ms(10);
   }
   return   0;
}
```



Program 9 : I/O PROGRAMMING

```
// LED disambung pada pin Port B. Tulis program C pada AVR
// program akan menunjukan hitungan dari 0 sampai FFH
// (0000 0000 sampai 1111 1111 dalam biner) pada LED.

#include <avr/io.h>

int main(void)
{
   DDRB = 0xFF;
   while (1)
   {
        PORTB = PORTB + 1;
   }
   return 0;
}
```

Program 10 : I/O PROGRAMMING

```
// Tulis program  C pada AVR untuk mendapatkan data byte
   dari Port B dan kemudian kirim ke Port C.

#include <avr/io.h>        // standard AVR header
int main(void){
   unsigned char temp;
   DDRB = 0x00;            // Port B sebagai input
   DDRC = 0xFF;            // Port C sebagai output

   while(1){
        temp = PINB;
        PORTC = temp;
   }
   return 0;
}
```

Program 11 : I/O PROGRAMMING

```
// data dibaca dari Port C dan dimasukan ke variabel temp.
   jika datanya kurang dari 100 selanjutnya di keluarkan
   melalui Port B, jika lebih keluarkan melalui Port D
#include <avr/io.h>         //standard AVR header
int main(void){
   DDRC  = 0x00;            //Port  C  sebagai input
   DDRB  = 0xFF;            //Port  B  sebagai output
   DDRD  = 0xFF;            //Port  D  sebagai output
   unsigned  char  temp;
   while(1){
       temp = PINC;         //baca dari PINB
       if(temp < 100 )
               PORTB = temp;
       else
               PORTD = temp;
   }
   return   0;
}
```

Program 12 : BITWISE OPERATIONS

```
// tulis dan jalankan program pada simulator.
// Amati hasilnya

#include <avr/io.h>         //standard AVR      header
int main(void) {
   DDRA   =   0xFF;         // Port  A output
   DDRB   =   0xFF;         // Port  B output
   DDRC   =   0xFF;         // Port  C output
   DDRD   =   0xFF;         // Port  D output
   PORTA  =   0x35 & 0x0F;  // bitwise AND
   PORTB  =   0x04 | 0x68;  // bitwise OR
   PORTC  =   0x54 ^ 0xF0;  // bitwise XOR
   PORTD  = ~ 0x55;         // bitwise NOT
   while(1);
   return 0;
}
```

Program 13 : BITWISE OPERATIONS

```
// program operasi toggle hanya pada bit 4 Port B

#include <avr/io.h>      //standard AVR header

int main(void)
{
  DDRB = 0xFF;           //PORTB  sebagai output
  while(1)
  {
      PORTB = PORTB ^ 0b00010000;
      //set bit   4 (bit ke-5) PORTB
  }
  return 0;
 }
```

Program 14: BITWISE OPERATIONS

```
// program untuk memonitor bit 5 port C. jika bernilai
   tinggi, kirim data 55H ke Port B; sebaliknya kirim AAH
   Port B.

#include <avr/io.h> // standard AVR header

int main(void){
   DDRB = 0xFF;      // PORTB sebagai output
   DDRC = 0x00;      // PORTC sebagai input
   DDRD = 0xFF;      // PORTB sebagai output
   while(1){
   if (PINC & 0b00100000)   // cek bit 5 PINC
       PORTB = 0x55;
   else
       PORTB = 0xAA;
   }
   return 0;
}
```

Program 15: BITWISE OPERATIONS

```
// misal rangkaian sensor pintu disambung 1 Port B, dan
   LED disambung  ke bit 7 Port C. tulis program untuk
   memonitor sensor, ketika pintunya dibuka LED menyala.

#include <avr/io.h>             //standard AVR header
int main(void){
   DDRB = DDRB & 0b11111101;    //pin 1 Port B sbg input
   DDRC = DDRC | 0b10000000;    //pin 7 Port C sbg output
   while(1){
        if (PINB & 0b00000010)  //cek pin 1 PINB
                PORTC = PORTC | 0b10000000;
                //set pin 7 PORTC
        else
                PORTC = PORTC & 0b01111111;
                //clear pin 7 PORTC
   }
   return 0;
}
```

Program 16: BITWISE OPERATIONS

```
// Tulis program untuk membaca pin 1 dan 0 Port dan
   keluarkan kode ASCII ke Port D
#include <avr/io.h>         //standard AVR header
int main(void){
 unsigned char z;
 DDRB = 0;               // Port B sbg input
 DDRD = 0xFF;            // Port D sbg output
 while(1){               // ulangi
   z = PINB;             // baca PORTB
   z = z & 0b00000011;// disable bit yang tidak digunakan
   switch(z){            // make decision
        case(0): PORTD = '0'; break; // ASCII 0
        case(1): PORTD = '1'; break; // ASCII 1
        case(2): PORTD = '2'; break; // ASCII 2
        case(3): PORTD = '3'; break; // ASCII 3
   }
 }
 return 0;
}
```

Program 17: BITWISE OPERATIONS

```
// program untuk monitor bit 7 Port B. jika berisi 1, buat
   bit 4 Port B sebagai input, sebaliknya, ubah pin 4 Port
   B sebagai output.

#include <avr/io.h>           //standard AVR header
int main(void){
   DDRB = DDRB & 0b01111111;  //bit 7 Port B sbg input
   // DDRB &= 0b01111111;
   while (1){
        if(PINB & 10000000)
                //bit 4 Port B sbg input
                DDRB = DDRB & 0b11101111;
                // DDRB &= 0b11101111;
        else
                //bit 4 Port B sbg output
                DDRB = DDRB | 0b00010000;
                // DDRB |= 0b00010000;
   }
   return 0;
}
```

Program 18: BITWISE OPERATIONS

```
// program untuk mendapatkan status bit 5 Port B dan kirim
   bit 7 port C secara terus-menerus.

#include <avr/io.h>           //standard AVR header

int main(void){
   DDRB = DDRB & 0b11011111;  // bit 5 Port B sbg input
   DDRC = DDRC | 0b10000000;  // bit 7 Port C sbg output

   while (1){
        if(PINB & 0b00100000) //set bit 7 Port C dgn 1
                PORTC = PORTC | 0b10000000;
                PORTC |= 0b10000000;
        else                  //clear bit 7 Port C dgn 0
                PORTC = PORTC & 0b01111111;
                PORTC &= 0b01111111;
   }
   return 0;
}
```

Program 19 : BITWISE OPERATIONS

```
// Tulis program toggle semua pins Port B secara terus-
   menerus.
#include <avr/io.h>          // standard AVR header
int main(void){
   DDRB  = 0xFF;             // Port B sbg output
   PORTB = 0xAA;
   while(1)
   { PORTB = ~ PORTB; }      // toggle pada PORTB
   return 0;
}
```

```
#include <avr/io.h>            // standard AVR header
int main(void){
   DDRB = 0xFF; PORTB = 0xAA; // Port B  sbg output
   while(1)
        PORTB = PORTB ^ 0xFF;
   return 0;
}
```

AVR Fuse Bits

- There are some features of the AVR that we can choose by programming the bits of fuse bytes. These features will reduce system cost by eliminating any need for external components.
- ATmega16 has two fuse bytes. Tables 8-6 and 8-7 give a short description of the fuse bytes.
- The Atmel website (http://www.atmel.com) provides the complete description of fuse bits for the AVR microcontrollers.
- If a fuse bit is incorrectly programmed, it can cause the system to fail. An example of this is changing the SPIEN bit to 0, which disables SPI programming mode. In this case you will not be able to program the chip any more!
- The fuse bits are '0' if they are programmed and '1' when they are not programmed.

AVR Fuse Bits

Table 8-6: Fuse Byte (High)

Fuse High Byte	Bit No.	Description	Default Value
OCDEN	7	Enable OCD	1 (unprogrammed)
JTAGEN	6	Enable JTAG	0 (programmed)
SPIEN	5	Enable SPI serial program and data downloading	0 (programmed)
CKOPT	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the chip erase	1 (unprogrammed)
BOOTSZ1	2	Select boot size	0 (programmed)
BOOTSZ0	1	Select boot size	0 (programmed)
BOOTRST	0	Select reset vector	1 (unprogrammed)

Table 8-7: Fuse Byte (Low)

Fuse High Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown-out detector trigger level	1 (unprogrammed)
BODEN	6	Brown-out detector enable	1 (unprogrammed)
SUT1	5	Select start-up time	1 (unprogrammed)
SUT0	4	Select start-up time	0 (programmed)
CKSEL3	3	Select clock source	0 (programmed)
CKSEL2	2	Select clock source	0 (programmed)
CKSEL1	1	Select clock source	0 (programmed)
CKSEL0	0	Select clock source	1 (unprogrammed)

- In addition to the fuse bytes in the AVR, there are 4 lock bits to restrict access to the Flash memory.
- These allow you to protect your code from being copied by others.
- In the development process it is not recommended to program lock bits because you may decide to read or verify the contents of Flash memory.
- Lock bits are set when the final product is ready to be delivered to market.

Fuse Bits and Oscillator Clock Source

- There are different clock sources in AVR. You can choose one by setting or clearing any of the bits CKSEL0 to CKSEL3.
- The four bits of CKSEL3, CKSEL2, CKSEL1, and CKSEL0 are used to select the clock source to the CPU.

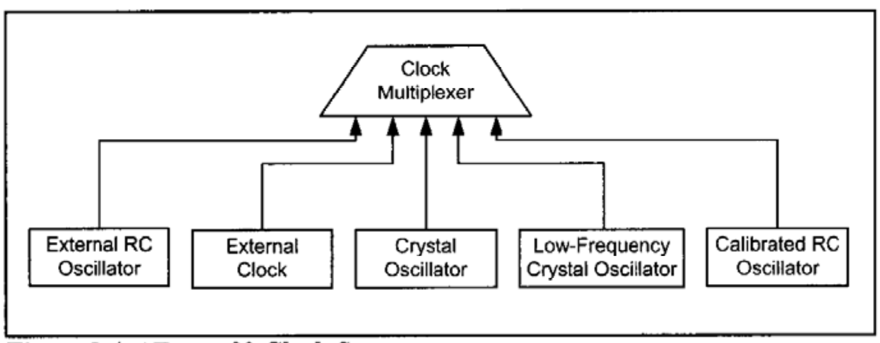


Figure 8-4. ATmega32 Clock Sources

Fuse Bits and Oscillator Clock Source

- The default value the four bits is (0001), which uses the 1MHz internal RC oscillator. In this option there is no need to connect an external crystal and capacitors to the chip.
- This default setting ensures that all users can make their desired clock source setting using an In-System or Parallel Programmer.

Table Device Clocking Options Select[1]

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

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

Fuse Bits and Oscillator Clock Source

- As you see in Table 8-8, by changing the values of CKSEL0-CKSEL3 we can choose among 1, 2,4, or 8 MHz internal RC frequencies; but it must be noted that using an internal RC oscillator can cause about 3% inaccuracy and is not recommended in applications that need precise timing.
- The external RC oscillator is another source to the CPU. As you see in Figure 8-5, to use the external RC oscillator, you have to connect an external resistor and capacitors to the XTAL1 pin.

Table 8-8: Internal RC Oscillator Operation Modes

CKSEL3...0	Frequency
0001	1 MHz
0010	2 MHz
0011	4 MHz
0100	8 MHz

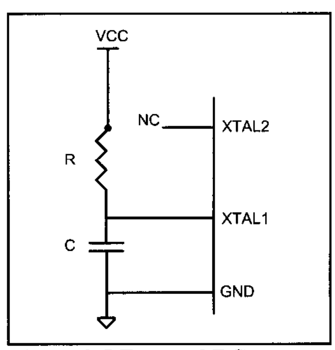


Figure 8-5 External RC

Fuse Bits and Oscillator Clock Source

- The values of R and C determine the clock speed.
- The frequency of the RC oscillator circuit is estimated by the equation

  $$f = 1/(3RC)$$

Table 8-9: External RC Oscillator Operation Modes

CKSEL3...0	Frequency (MHz)
0101	<0.9
0110	0.9–3.0
0111	3.0–8.0
1000	8.0–12.0

- When you need a variable clock source you can use the external RC and replace the resistor with a potentiometer.
- By turning the potentiometer you will be able to change the frequency. Notice that the capacitor value should be at least 22 pF.
- By programming the CKOPT fuse, you can enable an internal 36 pF capacitor between XTAL1 and GND, and remove the external capacitor. As you see in Table 8-9, by changing the values of CKSEL0-CKSEL3, we can choose different frequency ranges

Fuse Bits and Oscillator Clock Source

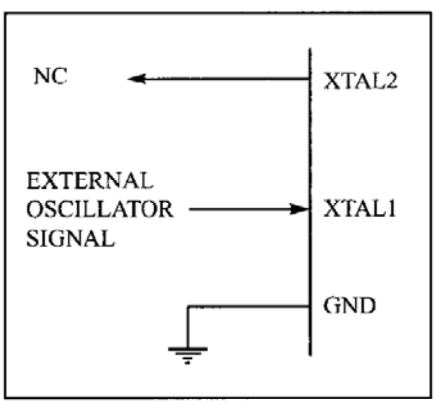


Figure 8-6a. XTAL1 Connection to an External Clock Source

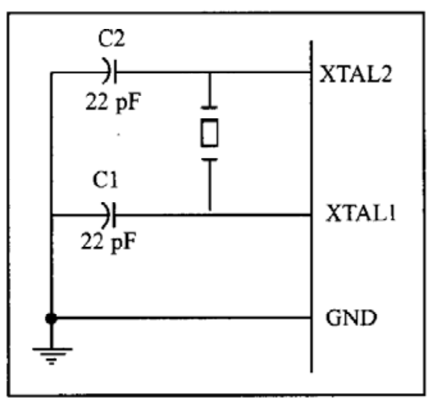


Figure 8-6b. XTAL1–XTAL2 Connection to Crystal Oscillator

- By setting CKSEL0...3 bits to 0000, we can use an external clock source for the CPU. In Figure 8-6a you see the connection to an external clock source

Fuse Bits and Oscillator Clock Source

- The most widely used option is to connect the XTAL1 and XTAL2 pins to a crystal (or ceramic) oscillator, as shown in Figure 8-6b.
- In this mode, when CKOPT is programmed, the oscillator output will oscillate with a full rail-to-rail swing on the output, causing a more powerful clock signal.
- This is suitable when the chip drives a second clock buffer or operates in a very noisy environment.

Table: ATmega32 Crystal Oscillator Frequency Choices and Capacitor Range

CKOPT	CKSEL3...1	Frequency (MHz)	C1 and C2 (pF)
1	101	0.4–0.9	Not for crystals
1	110	0.9–3.0	12–22
1	111	3.0–8.0	12–22
0	101, 110, 111	More than 1.0	12–22

Fuse Bits and Oscillator Clock Source

- As you see in Table, this mode has a wide frequency range. When CKOPT is not programmed, the oscillator has a smaller output swing and a limited frequency range. This mode cannot be used to drive other clock buffers, but it does reduce power consumption considerably.
- There are four choices for the crystal oscillator option. The Table shows all of these choices.
- Mode 101 cannot be used with crystals, and only ceramic resonators can be used.

Example 8-1

Find the instruction cycle time for the ATmega32 chip with the following crystal oscillators connected to the XTAL1 and XTAL2 pins.
(a) 4 MHz (b) 8 MHz (c) 10 MHz

Solution:
(a) Instruction cycle time is 1/(4 MHz) = 250 ns
(b) Instruction cycle time is 1/(8 MHz) = 125 ns
(c) Instruction cycle time is 1/(10 MHz) = 100 ns

Fuse Bits and Reset Delay

- The most difficult time for a system is during power-up. The CPU needs both a stable clock source and a stable voltage level to function properly.
- In AVRs, after all reset sources have gone inactive, a delay counter is activated to make the reset longer.
- This short delay allows the power to become stable before normal operation starts.
- You can choose the delay time through the SUT1, SUT0, and CKSEL0 fuses.
- Table 8-11 shows start-up times for the different values of SUT1, SUT0, and CKSEL fuse bits and also the recommended usage of each combination.
- Notice that the third column of Table 8-11 shows start-up time from power-down mode.

Fuse Bits and Reset Delay

Table 8-11: Startup Time for Crystal Oscillator and Recommended Usage

CKSEL0	SUT1...0	Start-Up Time from Power-Down	Delay from Reset (VCC = 5)	Recommended Usage
0	00	258 CK	4.1	Ceramic resonator, fast rising power
0	01	258 CK	65	Ceramic resonator, slowly rising power
0	10	1K CK	-	Ceramic resonator, BOD enabled
0	11	1K CK	4.1	Ceramic resonator, fast rising power
1	00	1K CK	65	Ceramic resonator, slowly rising power
1	01	16K CK	-	Crystal oscillator, BOD enabled
1	10	16K CK	4.1	Crystal oscillator, fast rising power
1	11	16K CK	65	Crystal oscillator, slowly rising power

Brown-out detector

- Occasionally, the power source provided to the V_{cc} pin fluctuates, causing the CPU to malfunction.
- The ATmega family has a provision for this, called ***brown-out detection***. The BOD circuit compares VCC with BOD-Level and resets the chip if VCC falls below the BOD-Level.
- The BOD-Level can be either 2.7 V when the **BODLEVEL fuse bit** is one (not programmed) or 4.0 V when the BODLEVEL fuse is zero (programmed).
- You can enable the BOD circuit by programming the BODEN fuse bit.
- When VCC increases above the trigger level, the BOD circuit releases the reset, and the MCU starts working after the time-out period has expired.
- If you are using an external crystal with a frequency of more than 1 MHz you can set the CKSEL3, CKSEL2, CKSEL1, SUT1, and SUTO bits to 1 (not programmed) and clear CKOPT to 0 (programmed)

Explaining the HEX file for AVR

- In the AVR Studio environment, the object file is fed into the linker program to produce the Intel hex file.
- The hex file is used by a programmer such as the AVRISP to transfer (load) the file into the Flash memory.
- The AVR Studio assembler can produce three types of hex files. They are
  - Intel Intellec 8/MDS (Intel Hex),
  - Motorola S-record,
  - Generic.

Table 8-12: Intel Hex File Formats Produced by AVR Studio

Format Name	File Extension	Max. ROM Address
Extended Intel Hex file	.hex	20-bit address
Motorola S-record	.mot	32-bit address
Generic	.gen	24-bit address

Brown-out detector

- The AVR Studio creates Extended Intel Hex File. which supports 1M address space.