AVR Microcontroller

Microprocessor Course

Chapter 8

AVR HARDWARE CONNECTION,HEX FILE AND FLASH LOADERS

Aban 1397 (Version 1.2)

AVR I/O PORT PROGRAMMING
8.1 ATMEGA 32 PIN CONNECTION

The ATmega family members come in different packages, such as

- **DIP (dual in-line package),**
- MLF (Micro Lead Frame Package), and
- QFP (quad flat package).

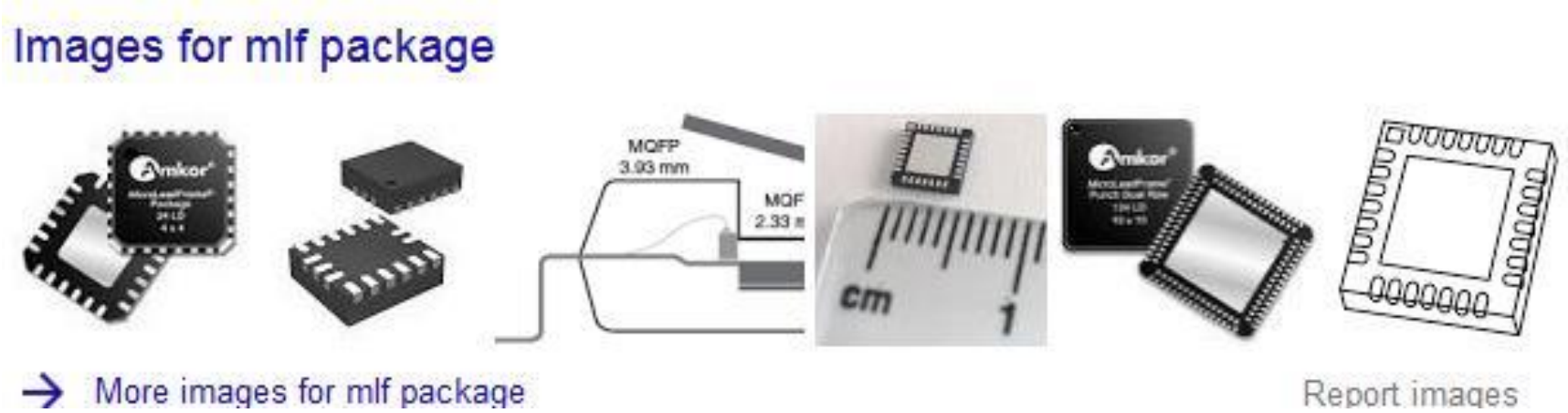
In microelectronics, a dual in-line **package** (**DIP** or DIL), or dual in-line pin **package** (DIPP) is an electronic component **package** with a rectangular housing and two parallel rows of electrical connecting pins. The **package** may be through-hole mounted to a printed circuit board (PCB) or inserted in a socket.

Dual in-line package - Wikipedia
https://en.wikipedia.org/wiki/Dual_in-line_package

AVR I/O PORT PROGRAMMING
8.1 ATMEGA 32 PIN CONNECTION

The ATmega family members come in different packages, such as

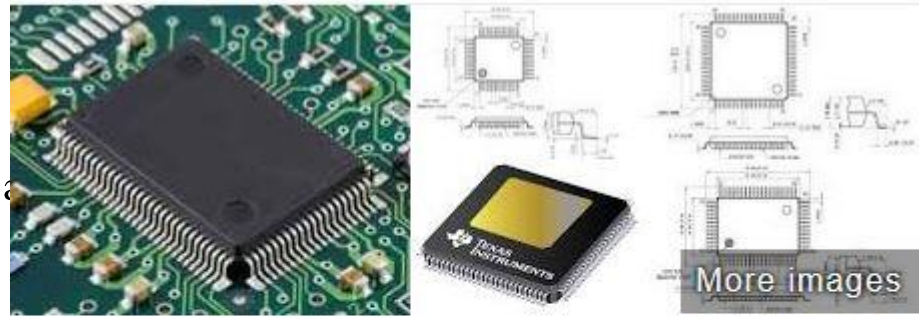
- DIP (dual in-line package),
- **MLF (Micro Lead Frame Package), and**
- QFP (quad flat package).



AVR I/O PORT PROGRAMMING
8.1 ATMEGA 32 PIN CONNECTION

The ATmega family members come in different packages, such as

- DIP (dual in-line package),
- MLF (Micro Lead Frame Package), a
- **QFP (quad flat package).**



AVR I/O PORT PROGRAMMING

8.1 ATMEGA 32 PIN CONNECTION

Figure 8-1 shows the pins for the ATmega32. DIP (dual in-line package),

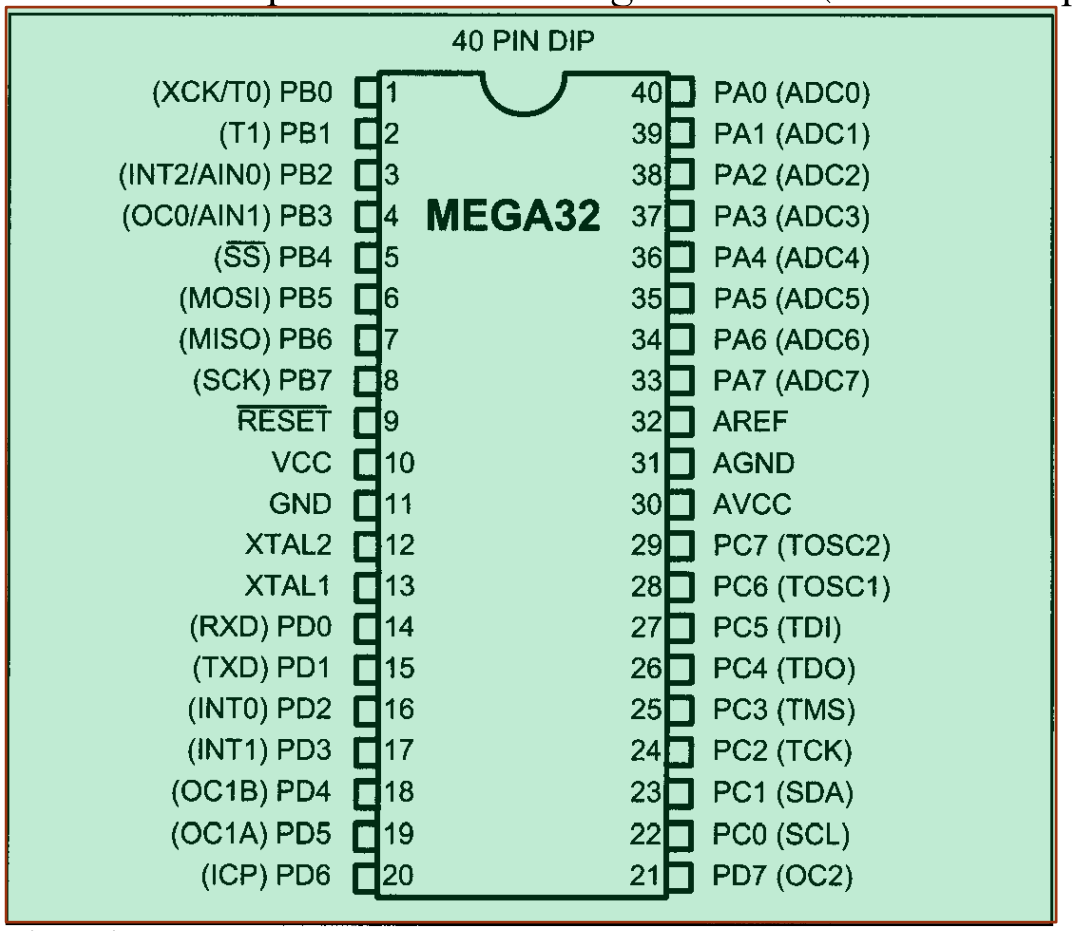


Figure 8-1. ATmega32 Pin Diagram

AVR I/O PORT PROGRAMMING

8.1 ATMEGA 32 PIN CONNECTION

VCC

This pin provides supply voltage to the chip. The typical voltage source is ***+5 V. Some AVR family members have lower voltage for VCC pins in order to*** reduce the noise and power dissipation of the AVR system.

For example, ATmega32L operation voltage is 2.7-5.5 V. We can choose other options for the operating voltage level by setting BOD fuse bits.

AVCC

AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to VCC, even if the ADC is not used. In Chapter 13 you will see how to connect this pin if you want to use ADC.

AVR I/O PORT PROGRAMMING
8.1 ATMEGA 32 PIN CONNECTION

AREF

AREF is the analog reference pin for ADC. In Chapter 13 we will discuss it further.

GND

Two pins are also used for ground. In chips with 40 pins and more, it is common to have multiple pins for VCC and GND. This will help reduce the noise (ground bounce) in high-frequency systems.

AVR I/O PORT PROGRAMMING
8.1 ATMEGA 32 PIN CONNECTION

XTALl and XTAL2

The ATmega32 has many options for the clock source. Most often a quartz crystal oscillator is connected to input pins XTALl and XTAL2. The quartz crystal oscillator connected to the XTALl and XTAL2 pins also needs two capacitors.

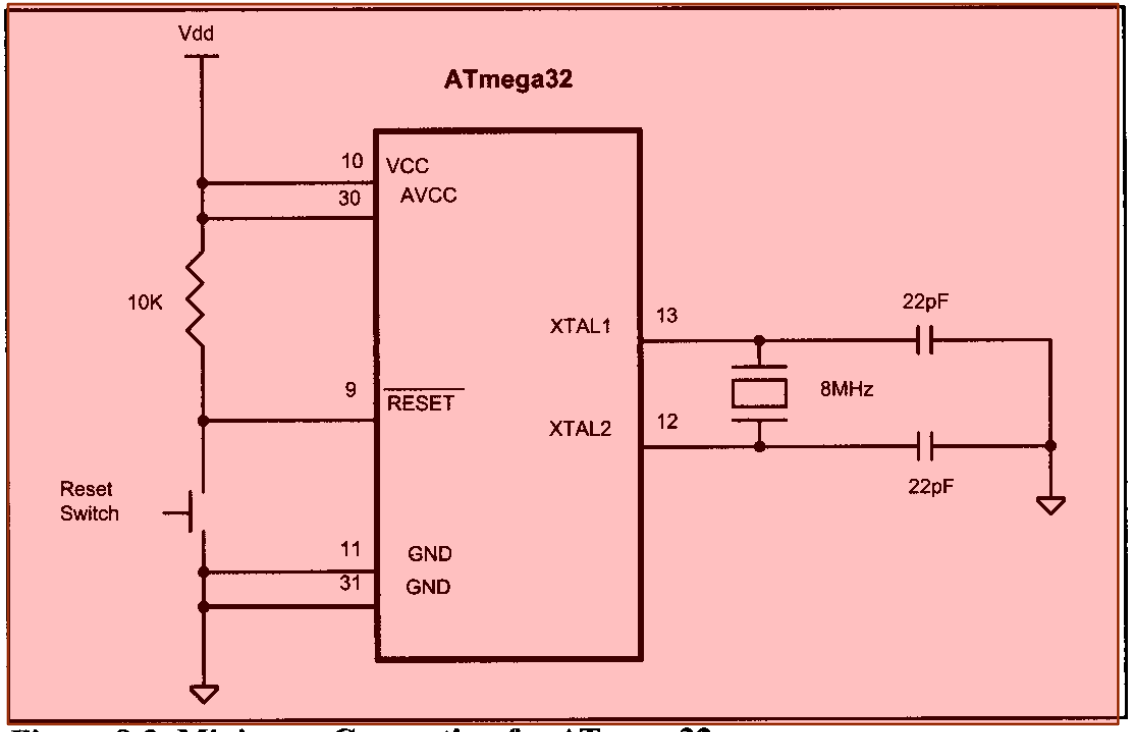


Figure 8-2. Minimum Connection for ATmega32

ATmega32 microcontrollers can have speeds of **0 Hz to 16 MHz.**

mashhoun@iust.ac.ir Iran Univ of Science & Tech

AVR I/O PORT PROGRAMMING
8.1 ATMEGA 32 PIN CONNECTION

RESET

Pin 9 (in the ATmega32, 40-pin DIP) is the RESET pin. It is an input and is active-LOW (normally HIGH). When a LOW pulse is applied to this pin, the microcontroller will reset and terminate all activities.

After applying reset, contents of all registers and SRAM locations will be cleared. Notice that after applying reset, all ports will be input because contents of all DDR registers are cleared.

The CPU will start executing the program from run location 0x00000 after a brief delay when the RESET pin is forced low and then released.

AVR I/O PORT PROGRAMMING
8.1 ATMEGA 32 PIN CONNECTION

Figures 8-3a, 8-3b, and 8-3c show three ways of connecting the RESET pin. Figure 8-3b uses a momentary switch for reset circuitry. Some designers put a 10nF capacitor between the RESET pin and GND to filter the noise during reset and working time. See Figure 3-8c. The diode protects the RESET pin from being powered by the capacitor when the power is off.

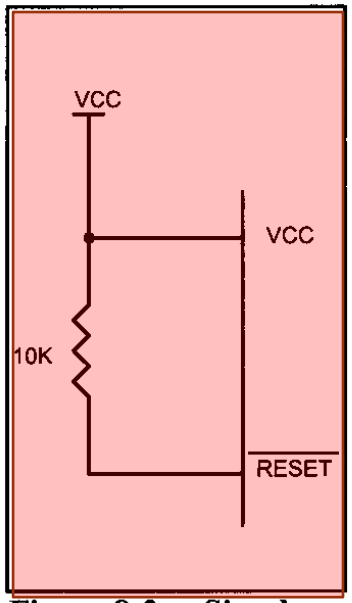


Figure 8-3a. Simple Power-On Reset Circuit

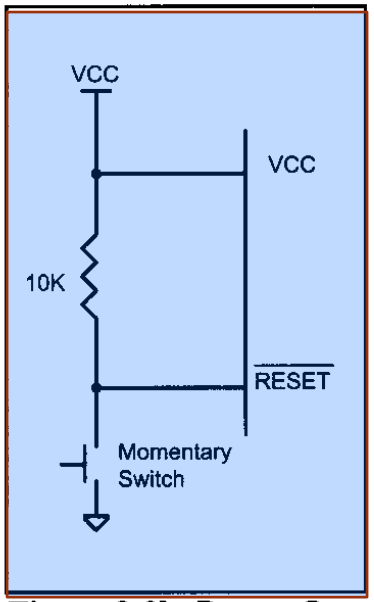


Figure 8-3b. Power-On Reset Circuit with Momentary Switch

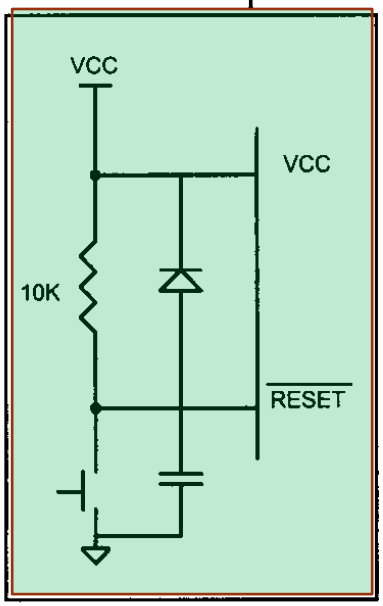


Figure 8-3c. Power-On Reset Circuit with Capacitor and Diode

AVR I/O PORT PROGRAMMING

SECTION 8.2: 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.

ATmega32 has two fuse bytes. Tables 8-6 and 8-7 give a short description of the fuse bytes.

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)

AVR I/O PORT PROGRAMMING

SECTION 8.2: AVR FUSE BITS

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)

AVR I/O PORT PROGRAMMING

SECTION 8.2: AVR FUSE BITS

Notice that the default values can be different from production to production and time to time.

It must be noted that 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!

Also notice that the fuse bits are '0' if they are programmed and ' 1' when they are not programmed.

AVR I/O PORT PROGRAMMING

SECTION 8.2: AVR FUSE BITS

LOCK BITS

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.

AVR I/O PORT PROGRAMMING

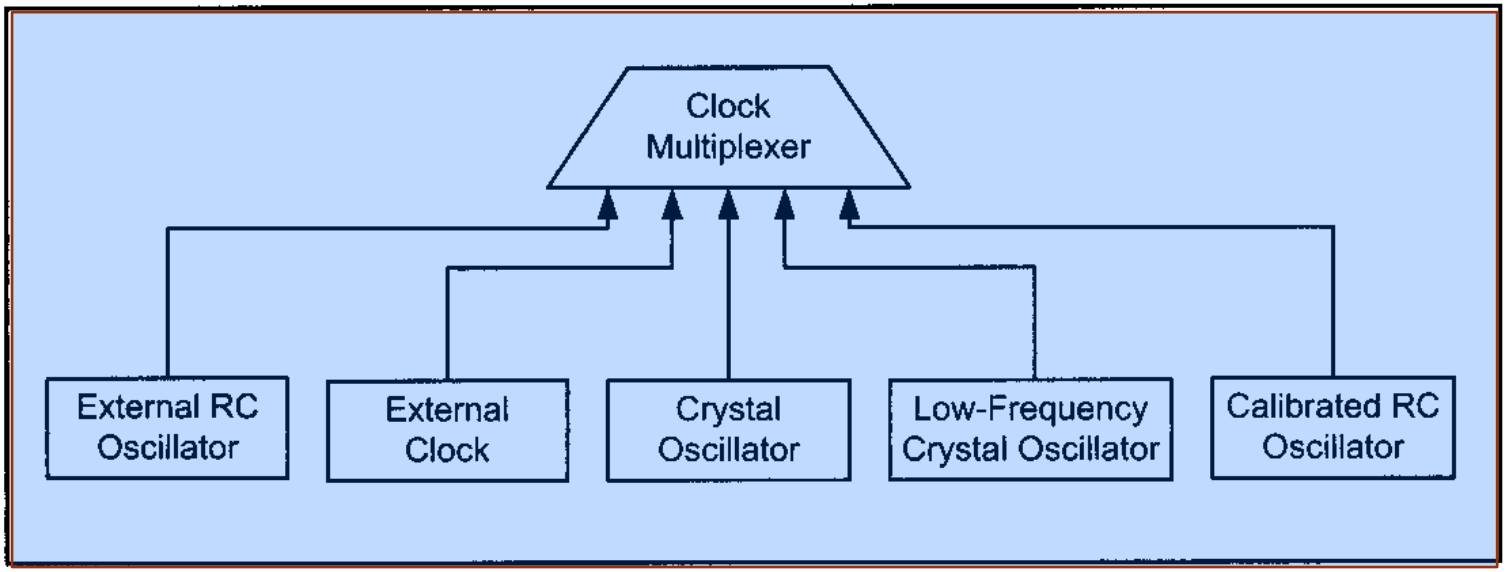
SECTION 8.2: AVR FUSE BITS

Fuse bits and oscillator clock source

You can choose one by setting or clearing any of the bits CKSEL0 to CKSEL3. The default choice is internal RC (0001), which uses the on-chip RC oscillator.

Table 8-8: Internal RC Oscillator Operation Modes

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



AVR I/O PORT PROGRAMMING

SECTION 8.2: AVR FUSE BITS

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.

The frequency of the RC oscillator circuit is estimated by the equation f = 1/(3RC). Notice that the capacitor value should be at least 22 pF. Also, notice that 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.

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

AVR I/O PORT PROGRAMMING

SECTION 8.2: AVR FUSE BITS

By setting CKSEL0…3 bits to 0000, we can use an external clock source for the CPU.

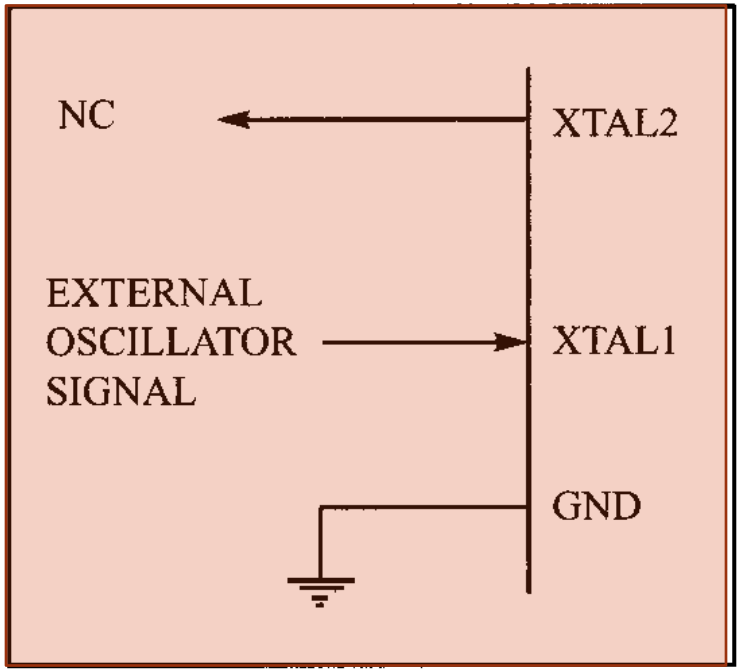


Figure 8-6a. XTAL1 Connection to an External Clock Source

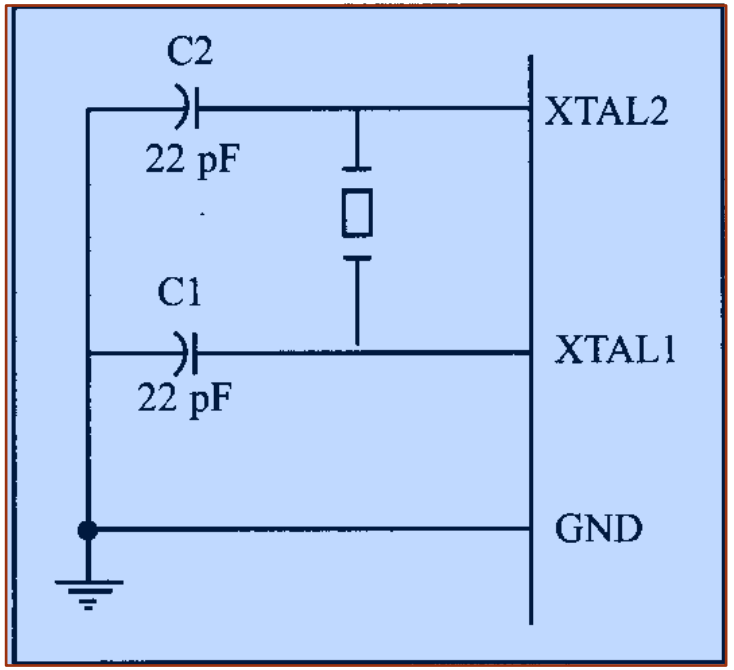


Figure 8-6b. XTAL1–XTAL2 Connection to Crystal Oscillator

The most widely used option is to connect the XTAL1 and XTAL2 pins to a crystal (or ceramic) oscillator

AVR I/O PORT PROGRAMMING

SECTION 8.2: AVR FUSE BITS

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.

Table 8-10 shows all of these choices. Notice that mode 101 cannot be used with crystals, and only ceramic resonators can be used.

Table 8-10: 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

AVR I/O PORT PROGRAMMING

SECTION 8.2: AVR FUSE BITS

Example 8-1 shows the relation between crystal frequency and instruction cycle time.

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

AVR I/O PORT PROGRAMMING

SECTION 8.2: AVR FUSE BITS

Fuse bits and reset delay & SUT1,SUT0 and CKSEL0

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.

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

AVR I/O PORT PROGRAMMING

SECTION 8.2: AVR FUSE BITS

Brown-out detector

Occasionally, the power source provided to the Vcc 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.

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

Intel Hex is a widely used file format designed to standardize the loading (transferring) of executable machine code into a chip. Therefore, the loaders that come with every ROM burner (programmer) support the Intel Hex file format. In many Windows-based assemblers such as AVR Studio, the Intel Hex file is produced according to the settings you set.

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

(a) Intel Intellec 8MDS (Intel Hex),

(b) Motorola S-record, and

(c) Generic.

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

See Table 8-12. In this section we will explain Intel Hex with some examples.

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

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

Analyzing the Intel Hex file

We choose the hex type of Intel Hex, Motorola S-record, or Generic by using the command-line invocation options or setting the options in the AVR Studio assembler itself. If we do not choose one, the AVR Studio assembler selects Intel Hex by default.

Intel Hex supports up to 16-bit addressing and is not applicable for programs more than 64K bytes in size. To overcome this limitation AVR Studio uses extended Intel Hex files, which support type 02 records to extend address space to 1M. Figure 8-10 shows the Intel Hex file of the test program whose list file is given in Figure 8-8.

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

Since the programmer (loader) uses the Hex file to download the opcode into Flash, the hex file must provide the following:

(1) the number of bytes of information to be loaded,
(2) the information itself, and
(3) the starting address where the information must be placed.

Each record (line) of the Hex file consists of six parts as follows:

:BBAAAATTHHHHH……HHHHHCC

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

The following describes each part:

1. ":" Each line starts with a colon.
2. BB, the count byte. This tells the loader how many bytes are in the line.
3. AAAA is for the record address. This is a 16-bit address. The loader places the first byte of record data into this Flash location. This is the case in files that are less than 64 KB. For files that are more than 64 KB the address field shows the record address in the current segment.
4. TT is for type. This field is 00,01, or 02. If it is 00, it means that there are more lines to come after this line. If it is 01, it means that this is the last line and the loading should stop after this line. If it is 02, it indicates the current segment address. To calculate the absolute address of each record (line), we have to shift the current segment address 4 bits to left and then add it to the record address

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

5. HH …… H is the real information (data or code). The loader places this information into successive memory locations of Flash. The information in this field is presented as low byte followed by the high byte.
6. CC is a single byte. This last byte is the checksum byte for everything in that line. The checksum byte is used for error checking. Notice that the checksum byte at the end of each line represents the checksum byte for everything in that line, and not just for the data portion.

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

Example 8-2

What is the absolute address of the first byte of a record that has 0025 in the address field if the last type 02 record before it has the segment address 0030?

Solution:

To calculate the absolute address of each record (line), we have to shift the segment address (0030) four bits to the left and then add it to the record address (0025):

```
0030 (2 bytes segment address) shifted 4 bits to the left  -->    00300
0025 (record address)                                          +    25
                                                               ---------
=>   (absolute address)                                          00325
```


mashhoun@iust.ac.ir Iran Univ of Science & Tech

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

Example 8-3

What is the absolute address of the first byte of the second record below?

```
:020000020000FC
:10000000008E00EBF0FE50DBF0FEF07BB05E500953C
```

Solution:

To calculate the absolute address of the first byte of the second record, we have to shift left the segment address (0000, as you see in the first record) four bits and then add it to the second record address (0000, as you see in the second record).

```
0000 (segment address) shift 4 bits to the left  -->  00000
                                                    + 0000      (record address)
                                                    ---------
                                                      000000    (absolute address)
```

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

Analyzing the bytes in the Flash memory vs. list file

```
LOC    OBJ          LINE
                    .ORG 0x000
000000 e008         LDI   R16,HIGH(RAMEND)
000001 bf0e         OUT   SPH,R16
000002 e50f         LDI   R16,LOW(RAMEND)
000003 bf0d         OUT   SPL,R16

000004 ef0f         LDI   R16,0xFF
000005 bb07         OUT DDRB,R16
000006 e505         LDI   R16,0x55
                    BACK:
000007 9500         COM         R16
000008 bb08         OUT   PORTB,R16
000009 940e 000c    CALL  DELAY_1S
00000b cffb         RJMP  BACK
                    DELAY_1S:
00000c e140         LDI   R20,16
00000d ec58         L1:   LDI   R21,200
00000e ef6a         L2:   LDI   R22,250
                    L3:
00000f 0000         NOP
000010 0000         NOP
000011 956a         DEC   R22
000012 f7e1         BRNE  L3
000013 955a         DEC   R21
000014 f7c9         BRNE  L2
000015 954a         DEC   R20
000016 f7b1         BRNE  L1
000017 9508         RET
```

Figure 8-8. List File for Test Program

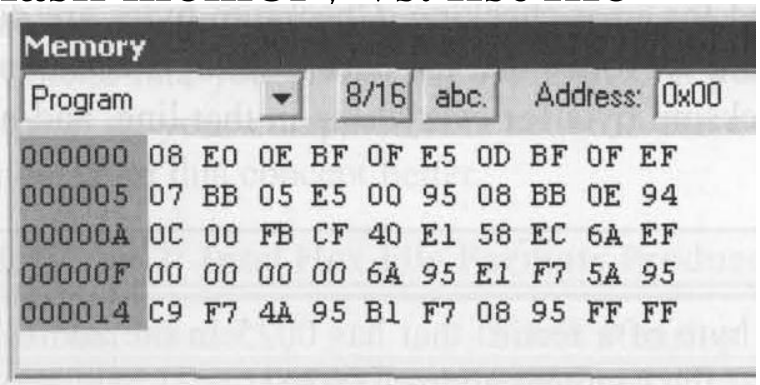


Figure 8-7. AVR Flash Memory Contents

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

As we mentioned, each Flash location in the AVR is 2 bytes long. So, for example, the first byte of Flash location #2 is Byte #4 of the code.

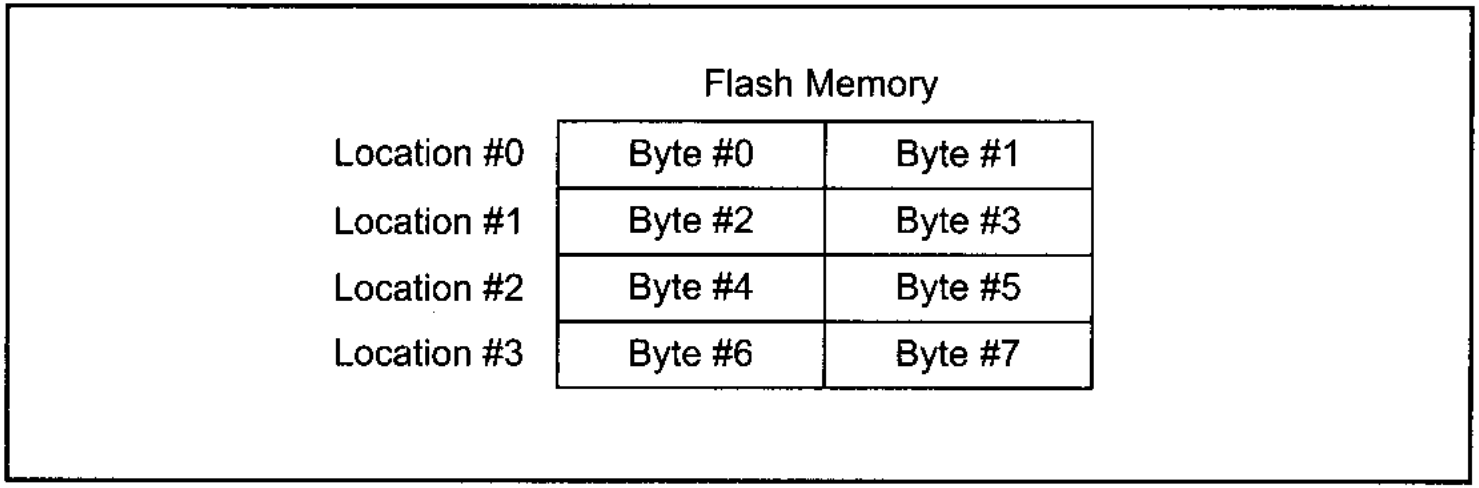


Figure 8-9. AVR Flash Memory Locations

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

The first record (line) is a type 02 record and indicates the current segment address, which is 0000.

In Figure 8-10 you see the hex file of the toggle code.

```
:0200000020000FC
:1000000008E00EBF0FE50DBF0FEF07BB05E500953C
:1000100008BB0E940C00FBCF40E158EC6AEF0000E7
:1000200000006A95E1F75A95C9F74A95B1F7089526
:00000001FF

Separating the fields, we get the following:

:BB  AAAA  TT  HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH        CC
:02  0000  02  0000                                    FC
:10  0000  00  08E00EBF0FE50DBF0FEF07BB05E50095        3C
:10  0010  00  08BB0E940C00FBCF40E158EC6AEF0000        E7
:10  0020  00  00006A95E1F75A95C9F74A95B1F70895        26
:00  0000  01                                          FF
```

The next field is the data field, which contains the code to be loaded into the chip.

checksum byte

Figure 8-10. Intel Hex File Test Program with the Intel Hex Option

The next record (line) is a type 00 record and contains the data (the code to be loaded into the chip). After ':' the record starts with 10, which means that the data field contains 10 (16 decimal) bytes of data.

The next field is the address field (0000), and it indicates that the first byte of the data field will be placed in address location 0 in the current segment.

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

Example 8-4

From Figure 8-10, analyze the six parts of line 3.

Solution:

After the colon (:), we have 10, which means that 16 bytes of data are in this line. 0010H is the record address, and means that 08, which is the first byte of the record, is placed in address location 10H (16 decimal). Next, 00 means that this is not the last line of the record. Then the data, which is 16 bytes, is as follows:
`08BB0E940C00FBCF40E158EC6AEF0000`. Finally, the last byte, E7, is the checksum byte.

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

Example 8-5

Compare the data portion of the Intel Hex file of Figure 8-10 with the opcodes in the list file of the test program given in Figure 8-8. Do they match?

Solution:

In the second line of Figure 8-10, the data portion starts with 08E0H, where the low byte is followed by the high byte. That means it is E008, the opcode for the instruction "`LDI  R16,HIGH(RAMEND)`", as shown in the list file of Figure 8-8. The last byte of the data in line 5 is 0895, which is the opcode for the "`RET`" instruction in the list file.

AVR I/O PORT PROGRAMMING

8.3: EXPLAINING THE HEX FILE FOR AVR

Example 8-6

(a) Verify the checksum byte for line 3 of Figure 8-10. (b) Verify also that the information is not corrupted.

Solution:

(a) `10 + 00 + 00 + 00 + 08 + E0 + 0E + BF + 0F + E5 + 0D + BF + 0F + EF + 07 + BB + 05 + E5 + 00 + 95 = 6C4` in hex. Dropping the carries (6) gives C4H, and its 2's complement is 3CH, which is the last byte of line 3.

(b) If we add all the information in line 2, including the checksum byte, and drop the carries we should get `10 + 00 + 00 + 00 + 08 + E0 + 0E + BF + 0F + E5 + 0D + BF + 0F + EF + 07 + BB + 05 + E5 + 00 + 95 + 3C = 700.` Dropping the carries (7) gives 00H, which means OK.

AVR I/O PORT PROGRAMMING

8.4: AVR PROGRAMMING AND TRAINER BOARD

We show various ways of loading a hex file into the AVR. Atmel has skillfully designed AVR microcontrollers for maximum flexibility of loading programs there three primary ways to load a program:

1. parallel programming. In this way a device burner loads the program into the microcontroller separate from the system. This is useful on a manufacturing floor where a gang programmer is used to program many chips at one time. The chip is programmed before it is inserted into the circuit.
2. Serial in-circuit programming this kind of programmer allows the developer to program and debug their microcontroller while it is in the system. AVR has two methods of ISP. They are SPI and JTAG. Most of the Atmega family supports both methods. The SPI uses 3 pins. Notice that SPI stands for "Serial Peripheral Interface" and is a protocol. But ISP stands for "In-circuit Serial Programming" and is a method of code loading. AVRISP and many other devices support ISP. To connect AVRISP to your device you also need to connect VCC, GND and RESET pins.

AVR I/O PORT PROGRAMMING

8.4: AVR PROGRAMMING AND TRAINER BOARD

Another method of ISP is JTAG. JTAG is another protocol that supports in-circuit programming and debugging. It means that in addition to programming you can trace your program on the chip line by line and watch or change the values of memory locations, ports, or registers while your program is running on the chip.

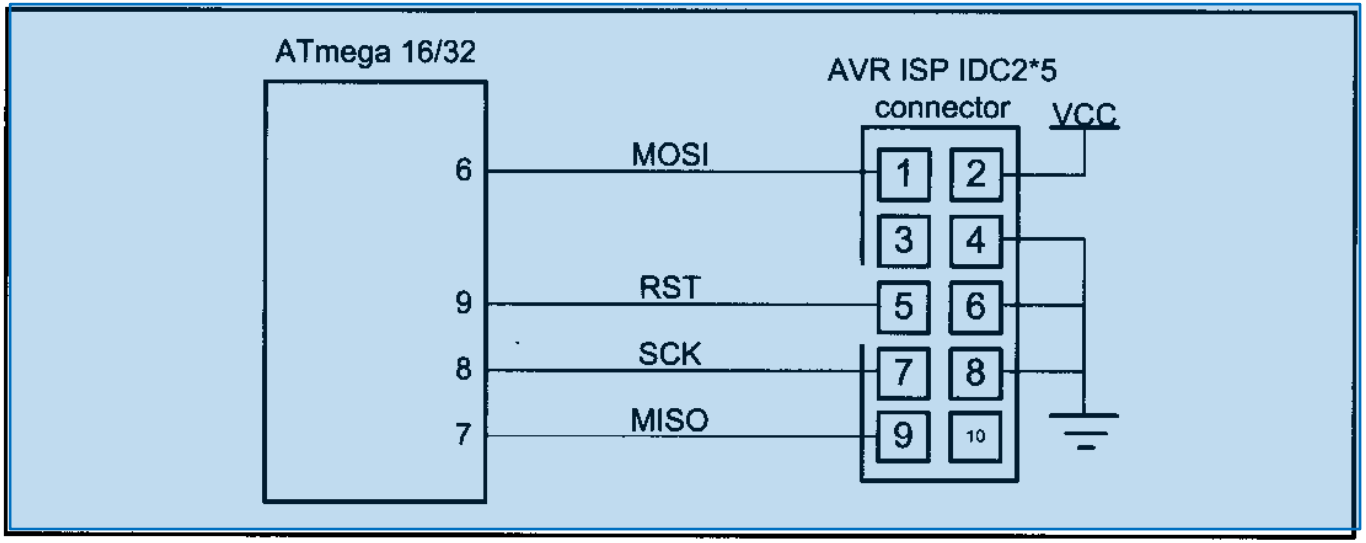


Figure 8-11. ISP 10-pin Connections (See www.Atmel.com for 6-pin version)

AVR I/O PORT PROGRAMMING

8.4: AVR PROGRAMMING AND TRAINER BOARD

A boot loader is a piece of code burned into the microcontroller's program Flash.

Its purpose is to communicate with the user's board to load the program. A boot loader can be written to communicate via a serial port, a CAN port, a USB port, or even a network connection.

A boot loader can also be designed to debug a system, similar to the JTAG. This method of programming is excellent for the developer who does not always have a device programmer or a JTAG available. There are several application notes on writing boot loaders on the Web. The main drawback of the boot loader is that it does require a communication port and program code space on the microcontroller.

Also, the boot loader has to be programmed into the device before it can be used, usually by one of the two previous ways.

AVR I/O PORT PROGRAMMING

8.4: AVR PROGRAMMING AND TRAINER BOARD

AVR trainers

There are many popular trainers for the AVR chip. The vast majority of them have a built-in ISP programmer. See the following website for more information and support about the AVR trainers. For more information about how to use an AVR trainer you can visit the

www.MicroDigitalEd.com

website.