microcontroller

What is microcontroller

- A microcontroller is complete computer system optimized for hardware control that encapsulates the entire processor, memory , and all of the I/O peripherals on a single piece of silicon.
- Microcontrollers allow for reduced code size and increased execution speed by providing instructions that are directly applicable to hardware control.

Atmel’s Atmega

- A company called Atmel has launched a series of microcontroller which are highly efficient and cheap.
- This microcontroller includes atmega16, atmega32, atmega128, etc.
- These are Atmel RISC microcontrollers. RISC stands for ‘reduced instruction set computing’.
- These devices are designed to run very fast through the use of a reduced no. of machine-level language.
- AVR processor using an 8 MHz clock can execute 8 million instruction per second, a speed of nearly 8 MIPS.

Atmega16

- Today, in this presentation we are dealing with a microcontroller named atmega16 and atmega16L.
- It is a powerful 8-bit microcontroller.
- As the name suggest ,it has 16K bytes in-system programmable flash memory.

8-bit AVR® Microcontroller with 16K Bytes In-System Programmable Flash

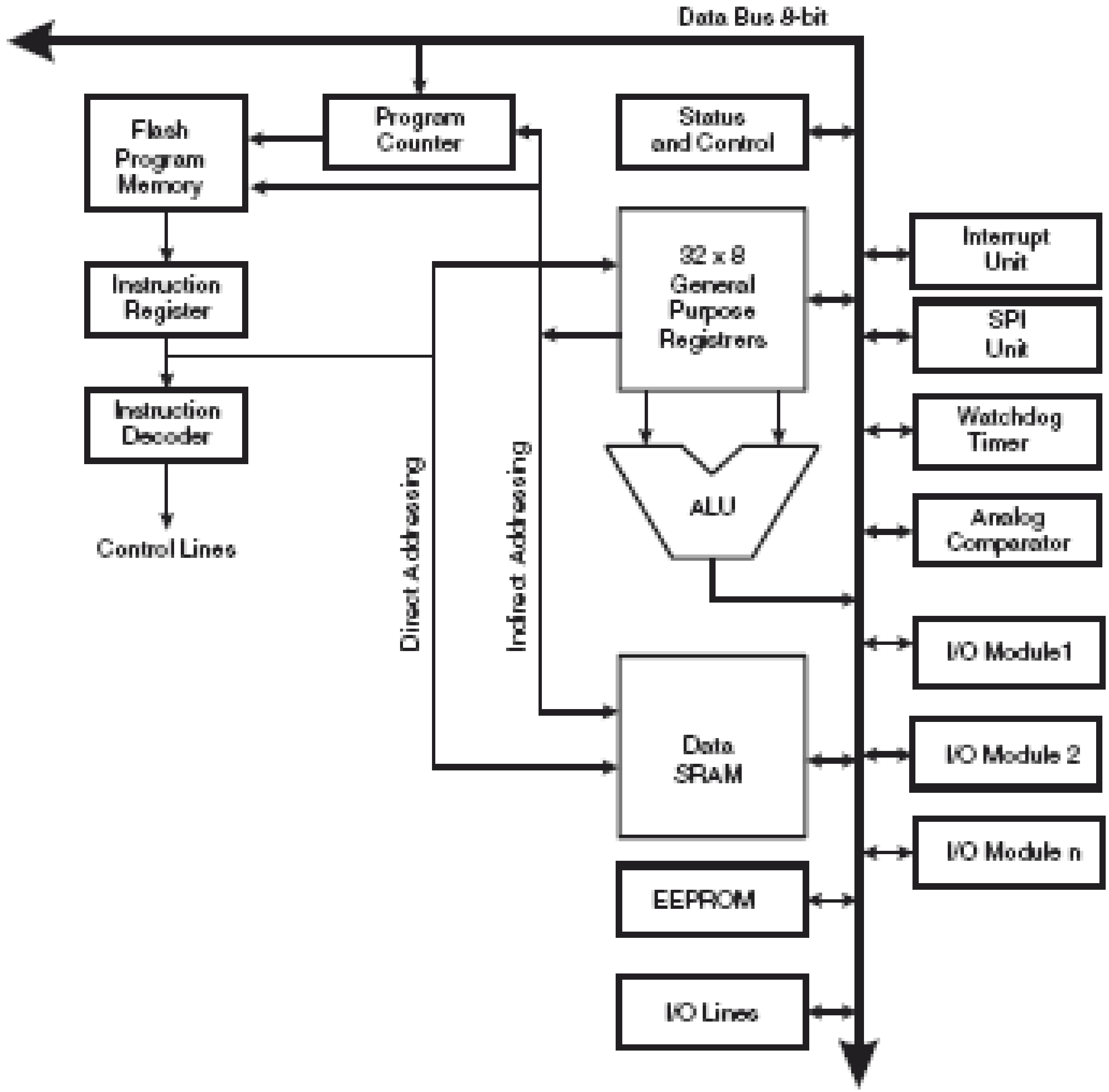
ATmega16
ATmega16L

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 16K Bytes of In-System Self-Programmable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 512 Bytes EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 1K Byte Internal SRAM
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Inte

- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    - 8 Single-ended Channels
    - 7 Differential Channels in TQFP Package Only
    - 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad MLF
- Operating Voltages
  - 2.7 - 5.5V for ATmega16L
  - 4.5 - 5.5V for ATmega16
- Speed Grades
  - 0 - 8 MHz for ATmega16L
  - 0 - 16 MHz for ATmega16

Figure 3. Block Diagram of the AVR MCU Architecture

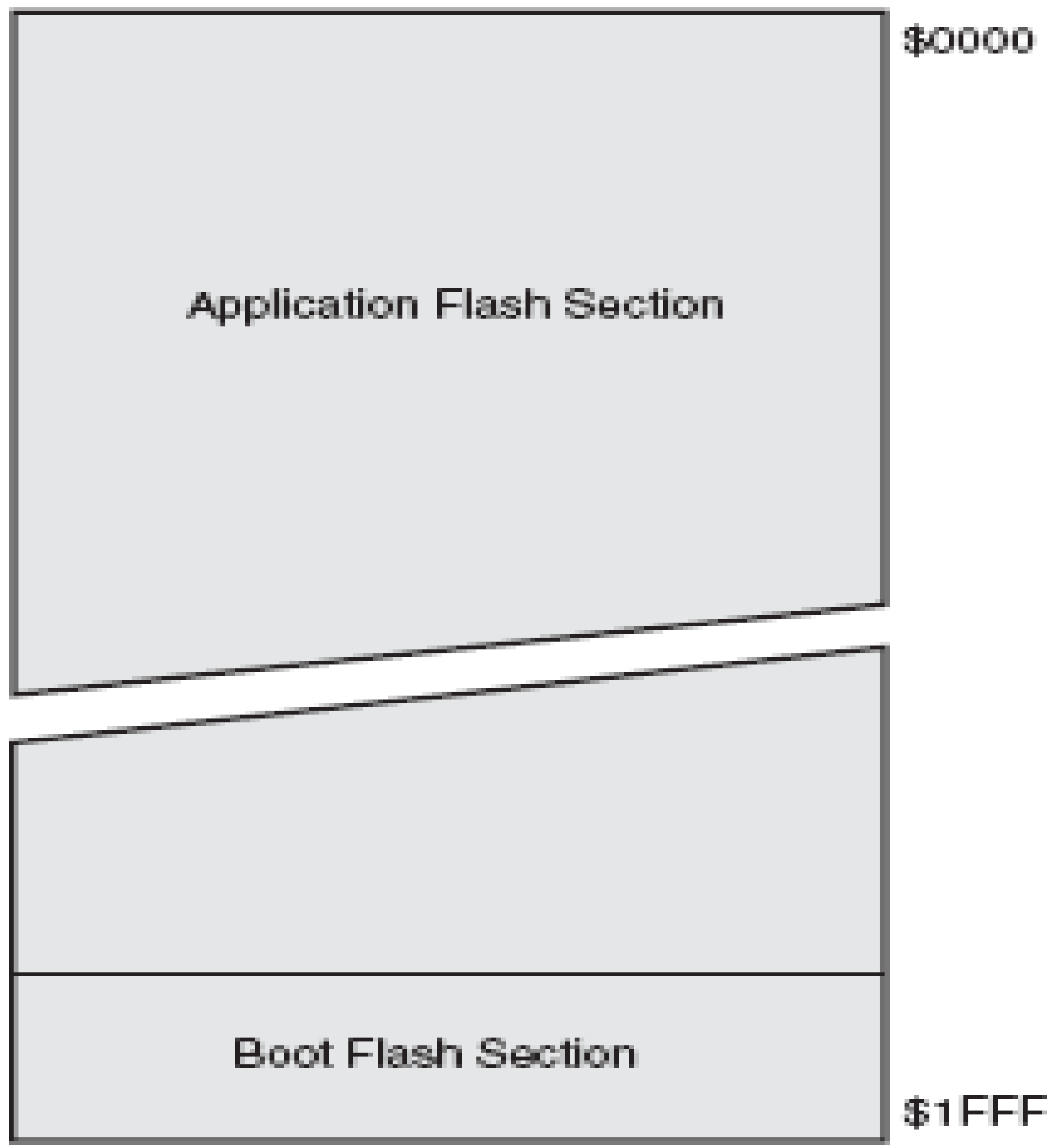


Microcontroller Memory

- The ATMega memory consists of three parts:
  - Data memory of SRAM : used for temporary storage of data values
  - Program memory, which is a Flash Memory, that can be rewritten up to 10,000 times
  - Finally the EEPROM memory, which is used for permanent storage of data values or initial parameters for the microcontroller.

- There are **three** types of memory available.
- Flash Code memory, Data memory and EEPROM.
- **Flash code memory** is a non volatile memory and is used to store the executable code and constants.
- It is a read only memory.
- The code memory space is 16 bits wide at each location to hold the machine level instructions.

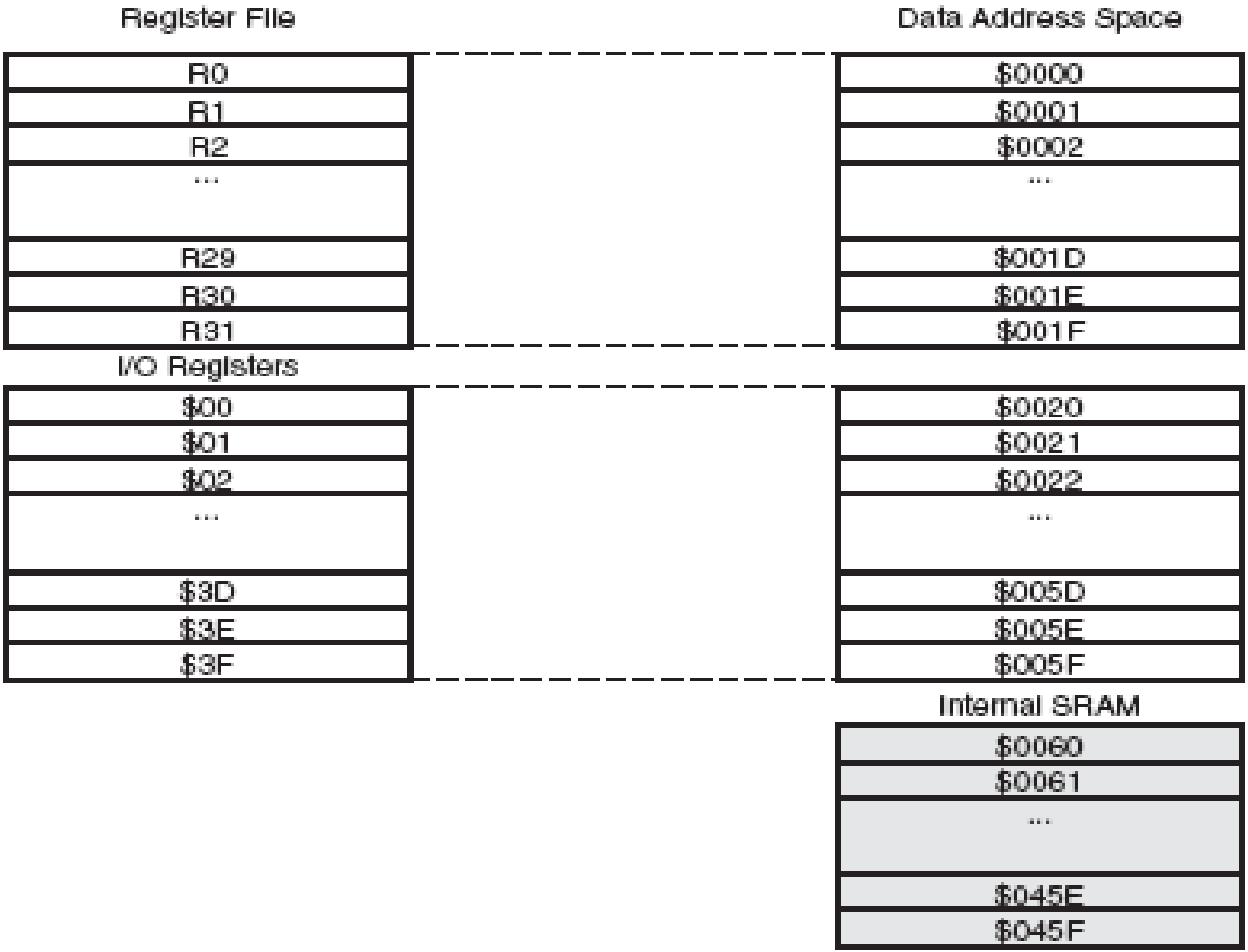
Program Memory Map



- **The data memory** contains three separate areas of R/W memory. the lowest section contains the 32 general purpose working registers , followed by 64 I/O registers, which is then followed by internal SRAM.
- *First part* : It stores the local variables, global variables and temporary data .its use is controlled by C compiler and is typically out of programmers control unless assembly language is used.

- *Second part* : I/O registers provides access to the control registers or the data registers of the I/O peripherals(a piece of computer hardware such as a printer or a disk drive that is external to but controlled by a computer's central processing unit) contained within the microcontroller. the programmer uses the I/O registers extensively to provide interface to the I/O peripherals of the microcontroller.
- *Third part* : The SRAM section of memory is used to store variables that do not fit into the registers and to store the processor stack.

Data Memory Map



- **EEPROM** section of memory is an area of read/write memory that is non volatile. It is typically used to store data that must not be lost when power is removed and reapplied to the microcontroller.
- EEPROM memory can withstand only certain no. of write cycles and hence it is usually reserved for those variables that must maintain their values in event of power loss.

- Now we will see about the three types of memory in some detail.
- The source of information is Wikipedia and www.howstuffworks.com

In **EEPROM**s:

- The chip does not have to removed to be rewritten.
- The entire chip does not have to be completely erased to change a specific portion of it.
- Changing the contents does not require additional dedicated equipment.
- **Flash memory**, a type of EEPROM that uses **in-circuit wiring** to erase by applying an electrical field to the entire chip or to predetermined sections of the chip called **blocks**. Flash memory works much faster than traditional EEPROMs because it writes data in chunks, usually 512 bytes in size, instead of 1 byte at a time

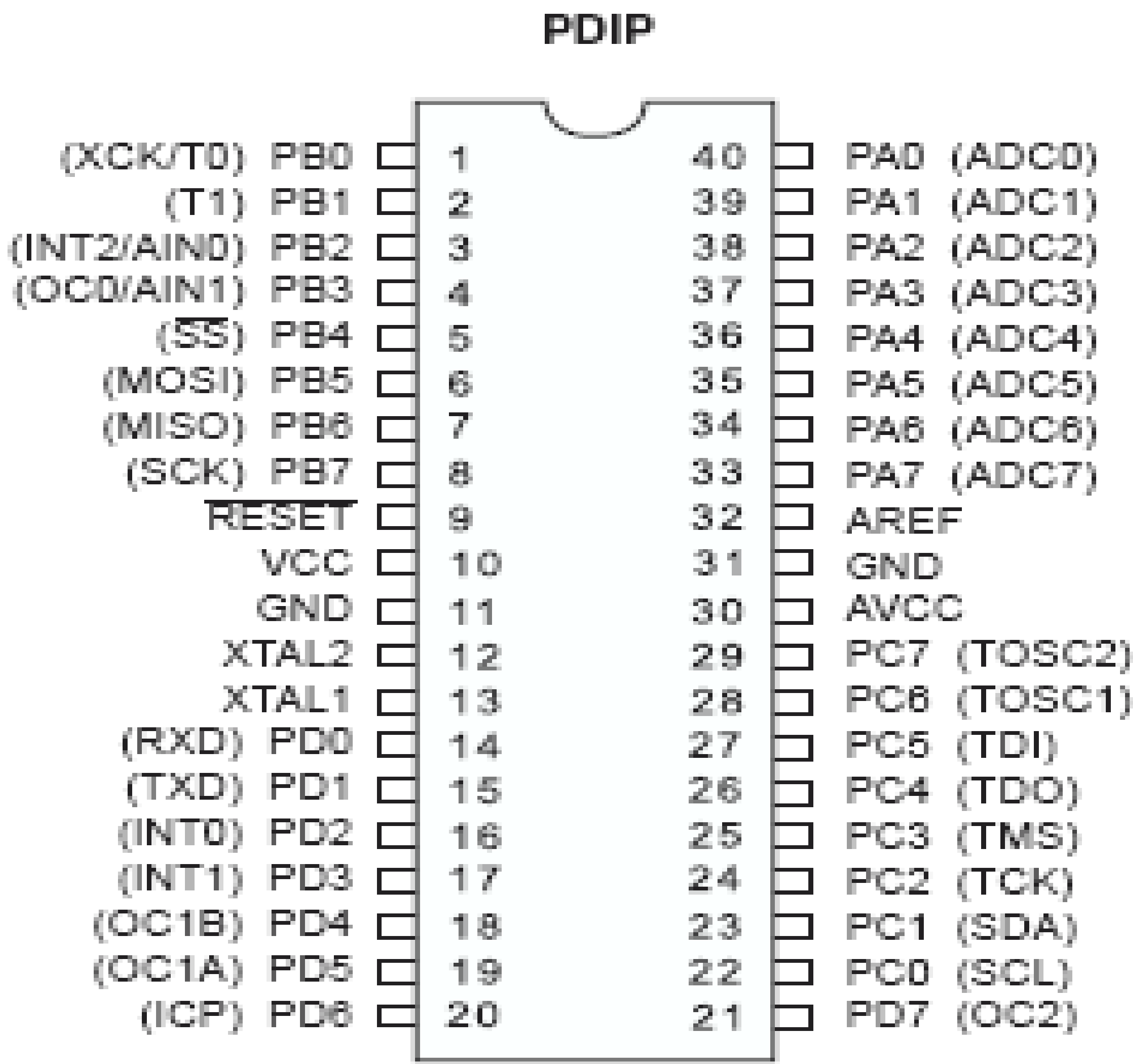
- **Flash Memory Basics**

- Flash memory is a type of **EEPROM** chip. It has a grid of columns and rows with a cell that has two transistors at each intersection
- The two transistors are separated from each other by a thin oxide layer. One of the transistors is known as a **floating gate**, and the other one is the **control gate**. The floating gate's only link to the row, or **wordline**, is through the control gate. As long as this link is in place, the cell has a value of 1. To change the value to a 0 requires a curious process called **Fowler-Nordheim tunneling**.

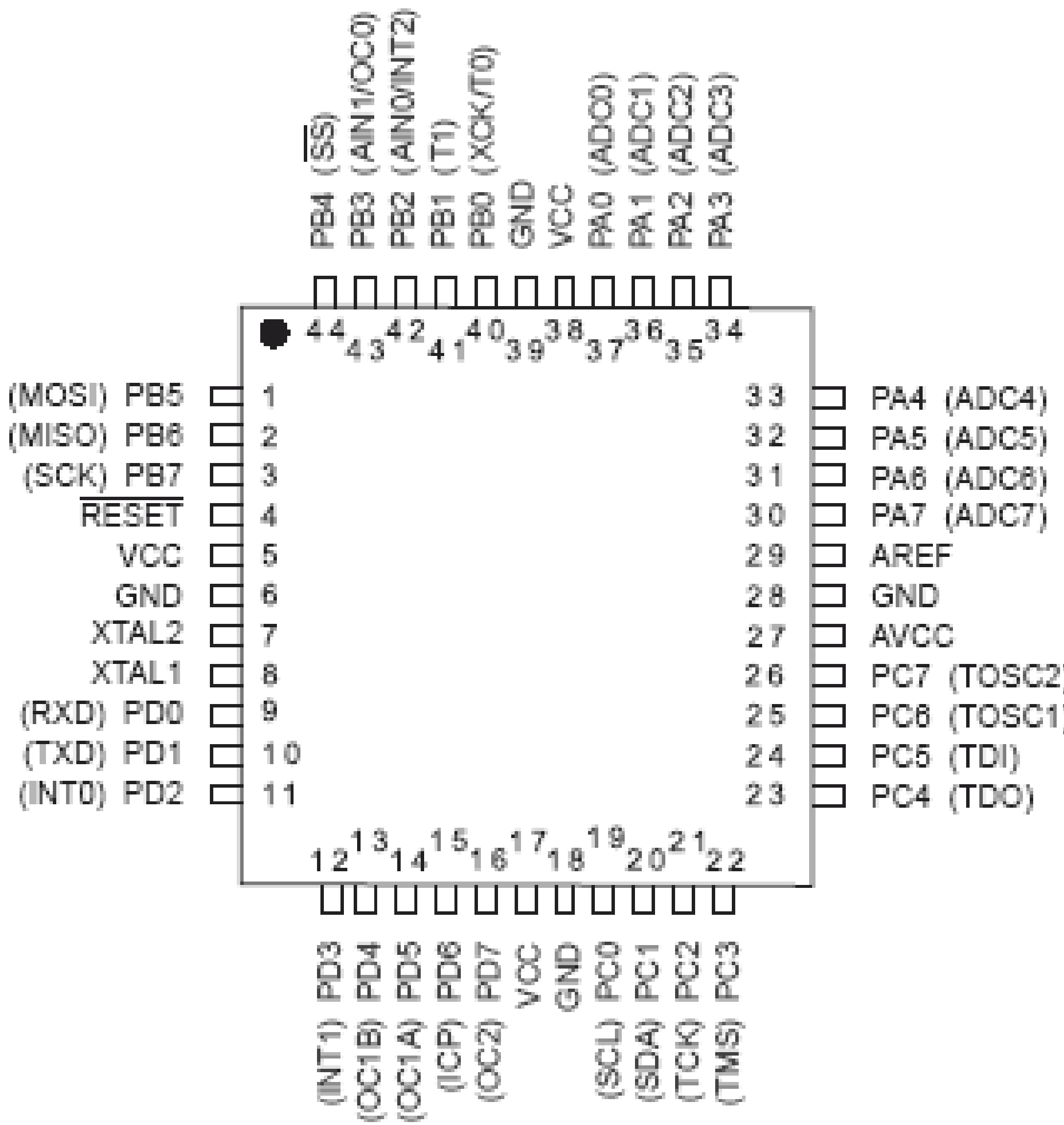
Static RAM uses a completely different technology. In static RAM, a form of flip-flop holds each bit of memory (see How Boolean Gates Work for detail on flip-flops). A flip-flop for a memory cell takes 4 or 6 transistors along with some wiring, but never has to be refreshed. This makes static RAM significantly faster than dynamic RAM. However, because it has more parts, a static memory cell takes a lot more space on a chip than a dynamic memory cell. Therefore you get less memory per chip, and that makes static RAM a lot more expensive.

So static RAM is fast and expensive, and dynamic RAM is less expensive and slower. Therefore static RAM is used to create the CPU's speed-sensitive cache, while dynamic RAM forms the larger system RAM space.

. Pinouts ATmega16



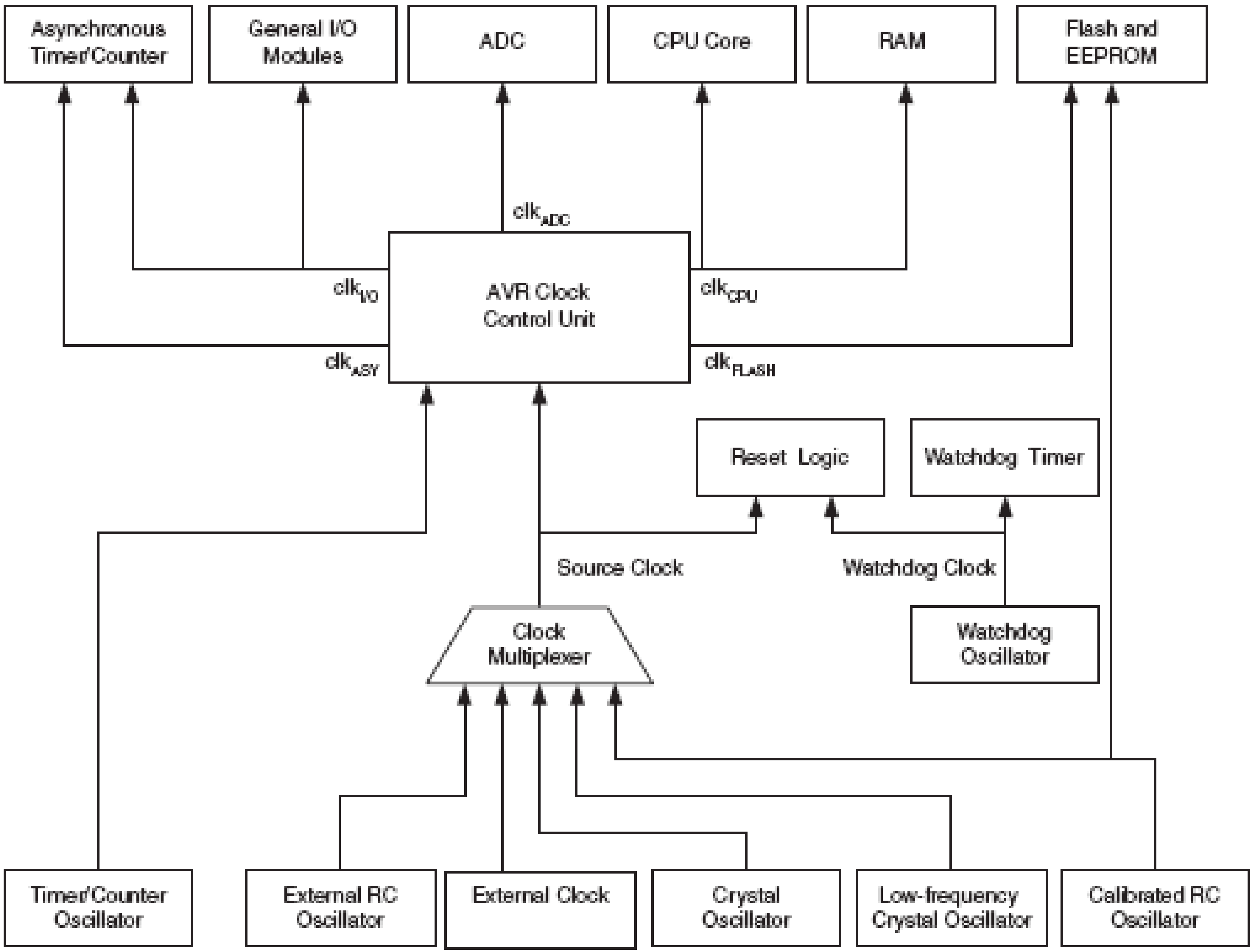
TQFP/MLF



PB4 (SS)
PB3 (AIN1/OC0)
PB2 (AIN0/INT2)
PB1 (T1)
PB0 (XCK/T0)
GND
VCC
PA0 (ADC0)
PA1 (ADC1)
PA2 (ADC2)
PA3 (ADC3)
44 43 42 41 40 39 38 37 36 35 34
(MOSI) PB5 1
(MISO) PB6 2
(SCK) PB7 3
RESET 4
VCC 5
GND 6
XTAL2 7
XTAL1 8
(RXD) PD0 9
(TXD) PD1 10
(INT0) PD2 11
33 PA4 (ADC4)
32 PA5 (ADC5)
31 PA6 (ADC6)
30 PA7 (ADC7)
29 AREF
28 GND
27 AVCC
26 PC7 (TOSC2)
25 PC6 (TOSC1)
24 PC5 (TDI)
23 PC4 (TDO)
12 13 14 15 16 17 18 19 20 21 22
PD3 (INT1)
PD4 (OC1B)
PD5 (OC1A)
PD6 (ICP)
PD7 (OC2)
VCC
GND
PC0 (SCL)
PC1 (SDA)
PC2 (TCK)
PC3 (TMS)

$\overline{\text{RESET}}$	Reset Input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 15 on page 35. Shorter pulses are not guaranteed to generate a reset.
XTAL1	Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.
XTAL2	Output from the inverting Oscillator amplifier.
AVCC	AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter.
AREF	AREF is the analog reference pin for the A/D Converter.

Figure 11. Clock Distribution



CPU Clock – clk_{CPU}

The CPU clock is routed to parts of the system concerned with operation of the AVR core. Examples of such modules are the General Purpose Register File, the Status Register and the data memory holding the Stack Pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

I/O Clock – $clk_{I/O}$

The I/O clock is used by the majority of the I/O modules, like Timer/Counters, SPI, and USART. The I/O clock is also used by the External Interrupt module, but note that some external interrupts are detected by asynchronous logic, allowing such interrupts to be detected even if the I/O clock is halted. Also note that address recognition in the TWI module is carried out asynchronously when $clk_{I/O}$ is halted, enabling TWI address reception in all sleep modes.

Flash Clock – clk_{FLASH}

The Flash clock controls operation of the Flash interface. The Flash clock is usually active simultaneously with the CPU clock.

Asynchronous TImer Clock – clk_{ASY}

The Asynchronous Timer clock allows the Asynchronous Timer/Counter to be clocked directly from an external 32 kHz clock crystal. The dedicated clock domain allows using this Timer/Counter as a real-time counter even when the device is in sleep mode.

ADC Clock – clk_{ADC}

The ADC is provided with a dedicated clock domain. This allows halting the CPU and I/O clocks in order to reduce noise generated by digital circuitry. This gives more accurate ADC conversion results.

AVR Registers

- All information in the microcontroller, from the program memory, the timer information, to the state on any of input or output pins, is stored in registers
- The 32 IO pins of the ATMega32 are divid-ed into 4 ports, A, B, C, and D.
- Each port has 3 associated registers.
- For example, for port D, these registers are referred to in C-language by PORTD, PIND, and DDRD

Registers

AVR is 8 bit microcontroller. All its ports are 8 bit wide. Every port has 3 registers associated with it each one with 8 bits. Every bit in those registers configure pins of particular port. Bit0 of these registers is associated with Pin0 of the port, Bit1 of these registers is associated with Pin1 of the port, …. and like wise for other bits.

These three registers are as follows :
(x can be replaced by A,B,C,D as per the AVR you are using)
- DDRx register
- PORTx register
- PINx register

All information in the microcontroller, from the program memory, the timer information, to the state on any of input or output pins, is stored in registers. Registers are like shelves in the bookshelf of processor memory. In an 8-bit processor, like the AVR ATMega 16 we are using, the shelf can hold 8 books, where each book is a one bit binary number, a 0 or 1. Each shelf has an address in memory, so that the controller knows where to find it. The 32 IO pins of the ATMega16 are divided into 4 ports, A, B, C, and D. Each port has 3 associated registers. For example, for port D, these registers are referred to in C-language by PORTD, PIND, and DDRD. For port B, these would be PORTB, PINB, and DDRB, etc. In C-language, PORTD is really a macro, which refers to a number that is the address of the register in the AVR, but it is much easier to remember PORTD than some arbitrary hexadecimal number.

DDRx register

DDRx (Data Direction Register) configures data direction of port pins. Means its setting determines whether port pins will be used for input or output. Writing 0 to a bit in DDRx makes corresponding port pin as input, while writing 1 to a bit in DDRx makes corresponding port pin as output.

example:

to make all pins of port A as input pins :

DDRA = 0b00000000;

to make all pins of port A as output pins :

DDRA = 0b11111111;

to make lower nibble of port B as output and higher nibble as input :

DDRB = 0b00001111;

- The DDRD register sets the direction of Port D. Each bit of the DDRD register sets the corresponding Port D pin to be either an input or an output. A 1 makes the corresponding pin an output, and a 0 makes the corresponding pin an input. To set the first pin of Port D to be an output pin, you could use the sbi(reg,bit) function, which sets a bit (makes it high or binary 1) in a register:
- sbi(DDRD, 0); sbi(DDRD, PD0); //both set the first pin of port D to be an input. You can also set the value of all the bits in the DDRx register (or any register) using the outb(reg,byte) command. It writes a byte (8 bits) to a register. For example, if you wanted to set pins 1-4 of port B to output and pins 5-8 to input, you could use:
- outb(DDRB, 0x0F); //Set the low 4 pins of Port B to output //and the high 4 pins to input An alternate way to write a value to a register is using the same syntax as a C assignment:
- DDRB = 0x0F; //Set the low 4 pins of Port B to output //and the high 4 pins to input

PINx register

PINx (Port IN) used to read data from port pins. In order to read the data from port pin, first you have to change port's data direction to input. This is done by setting bits in DDRx to zero. If port is made output, then reading PINx register will give you data that has been output on port pins.

Now there are two input modes. Either you can use port pins as tri stated inputs or you can activate internal pull up. It will be explained shortly.

example :

to read data from port A.

```
DDRA = 0x00;    //Set port a as input
x = PINA;       //Read contents of port a
```

When a pin is set to input, the PINx register contains the value applied to the pin. The pins have an electrical threshold of around 2.5 volts. If a voltage above this level is applied to an input pin, the corresponding bit of the PINx register will be a 1. Below this voltage, the bit will be a zero. See the ATMega16 schematic for the specific threshold voltages. To read the value of an input port, you can use the inb(reg) function or a direct assignment. It returns an 8-bit number that is the value of the 8 bits in the specified register.

PORTx register

PORTx is used for two purposes.

1) To output data : when port is configured as output

When you set bits in DDRx to 1, corresponding pins becomes output pins. Now you can write data into respective bits in PORTx register. This will immediately change state of output pins according to data you have written.
In other words to output data on to port pins, you have to write it into PORTx register. However do not forget to set data direction as output.

example :
to output 0xFF data on port b

```
DDRB = 0b11111111;       //set all pins of port b as outputs
PORTB = 0xFF;            //write data on port
```

to output data in variable x on port a

```
DDRA = 0xFF;             //make port a as output
PORTA = x;               //output variable on port
```

2) To activate/deactivate pull up resistors - when port is configures as input

The PORTx register functions differently depending on whether a pin is set to input or output. The simpler case is if a pin is set to output. Then, the PORTC register, for example, controls the value at the physical IO pins on Port C. For example, we can set all the port C pins to output and then make 4 of them high (binary 1) and 4 of them low (binary 0):

```
DDRC = 0xFF; //Set all Port C pins to output   , PORTC = 0xF0; //Set first 4 pins of Port C low and next 4 pins high
```

When a pin is set to be an input, PORTx register DOES NOT contain the logic values applied to the pins. We use the PINx register for that. If a pin is an input, a 1 in the PORTx register sets a pull-up resistor. This is helpful for a variety of circuits.

```
DDRC = 0x00; //Set all Port C pins to input    , PORTC = 0xFF; //Set pull-up resistors on Port C
```

Register Description for I/O Ports

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port B Data Register – PORTB

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port B Data Direction Register – DDRB

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port B Input Pins Address – PINB

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port C Data Register – PORTC

Bit	7	6	5	4	3	2	1	0	
	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port C Data Direction Register – DDRC

Bit	7	6	5	4	3	2	1	0	
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port C Input Pins Address – PINC

Bit	7	6	5	4	3	2	1	0	
	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port D Data Register – PORTD

Bit	7	6	5	4	3	2	1	0	
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port D Data Direction Register – DDRD

Bit	7	6	5	4	3	2	1	0	
	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port D Input Pins Address – PIND

Bit	7	6	5	4	3	2	1	0	
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Table 22. Port A Pins Alternate Functions

Port Pin	Alternate Function
PA7	ADC7 (ADC input channel 7)
PA6	ADC6 (ADC input channel 6)
PA5	ADC5 (ADC input channel 5)
PA4	ADC4 (ADC input channel 4)
PA3	ADC3 (ADC input channel 3)
PA2	ADC2 (ADC input channel 2)
PA1	ADC1 (ADC input channel 1)
PA0	ADC0 (ADC input channel 0)

Table 25. Port B Pins Alternate Functions

Port Pin	Alternate Functions
PB7	SCK (SPI Bus Serial Clock)
PB6	MISO (SPI Bus Master Input/Slave Output)
PB5	MOSI (SPI Bus Master Output/Slave Input)
PB4	$\overline{SS}$ (SPI Slave Select Input)
PB3	AIN1 (Analog Comparator Negative Input) OC0 (Timer/Counter0 Output Compare Match Output)
PB2	AIN0 (Analog Comparator Positive Input) INT2 (External Interrupt 2 Input)
PB1	T1 (Timer/Counter1 External Counter Input)
PB0	T0 (Timer/Counter0 External Counter Input) XCK (USART External Clock Input/Output)

Table 28. Port C Pins Alternate Functions

Port Pin	Alternate Function
PC7	TOSC2 (Timer Oscillator Pin 2)
PC6	TOSC1 (Timer Oscillator Pin 1)
PC5	TDI (JTAG Test Data In)
PC4	TDO (JTAG Test Data Out)
PC3	TMS (JTAG Test Mode Select)
PC2	TCK (JTAG Test Clock)
PC1	SDA (Two-wire Serial Bus Data Input/Output Line)
PC0	SCL (Two-wire Serial Bus Clock Line)

Table 31. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	OC2 (Timer/Counter2 Output Compare Match Output)
PD6	ICP1 (Timer/Counter1 Input Capture Pin)
PD5	OC1A (Timer/Counter1 Output Compare A Match Output)
PD4	OC1B (Timer/Counter1 Output Compare B Match Output)
PD3	INT1 (External Interrupt 1 Input)
PD2	INT0 (External Interrupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)

Functions within the microcontroller

Interrupts

- Interrupts are essentially hardware generated function call.
- Interrupts , as their name suggest, interrupt the flow of the processor program and cause it to branch to an Interrupt Service Routine that does whatever is supposed to happen when interrupt occurs.
- Reset is a special type of interrupt which presets the microcontroller in its original condition so that it again starts to execute programs which are stored in its memory.

Watchdog Timer

A *watchdog timer* is a piece of hardware that can be used to automatically detect software anomalies and reset the processor if any occur. Generally speaking, a watchdog timer is based on a counter that counts down from some initial value to zero. The embedded software selects the counter's initial value and periodically restarts it. If the counter ever reaches zero before the software restarts it, the software is presumed to be malfunctioning and the processor's reset signal is asserted. The processor (and the software it's running) will be restarted as if a human operator had cycled the power.

Murphy, Niall and Michael Barr. "Watchdog Timers" Embedded Systems Programming, October 2001 , pp. 79-80.

Timer/Counters

- Timer counters are the most used peripheral in the microcontroller.
- It is highly versatile, being able to measure the time periods, to determine pulse width , to measure speed , frequency, or to provide output signals.
- It is used in two different mode.

1. Timing mode
2. Counting mode.

- In timing mode the binary counters are counting time periods applied to their inputs.
- In counting mode , they are counting the events and pulses that took place.
- AVR microcontrollers have both 8-bit and 16-bit timer/counters.

Serial communication using USRAT

- USRAT – Universal Synchronous and Asynchronous Receiver and Transmitter
- Serial communication is a process of sending multiple bits of data over a single wire. The bits re separated by time, so the receiving device can determine the logic levels of each bit.
- The USRAT is used to communicate from the microcontroller to various other devices.
- E.g. terminal tool of AVR studio, other microcontrollers, etc.

Analog Interfaces

- In spite of prevalence of digital devices, the world is still actually analog by nature.
- A microcontroller is able to handle analog data by first converting the data to digital form.
- AVR microcontroller includes both an analog to digital conversion peripheral and an analog comparator peripheral.
- Microcontrollers use ADC to convert quantities such as temperature and voltage to digital formats and to perform host of additional functions.

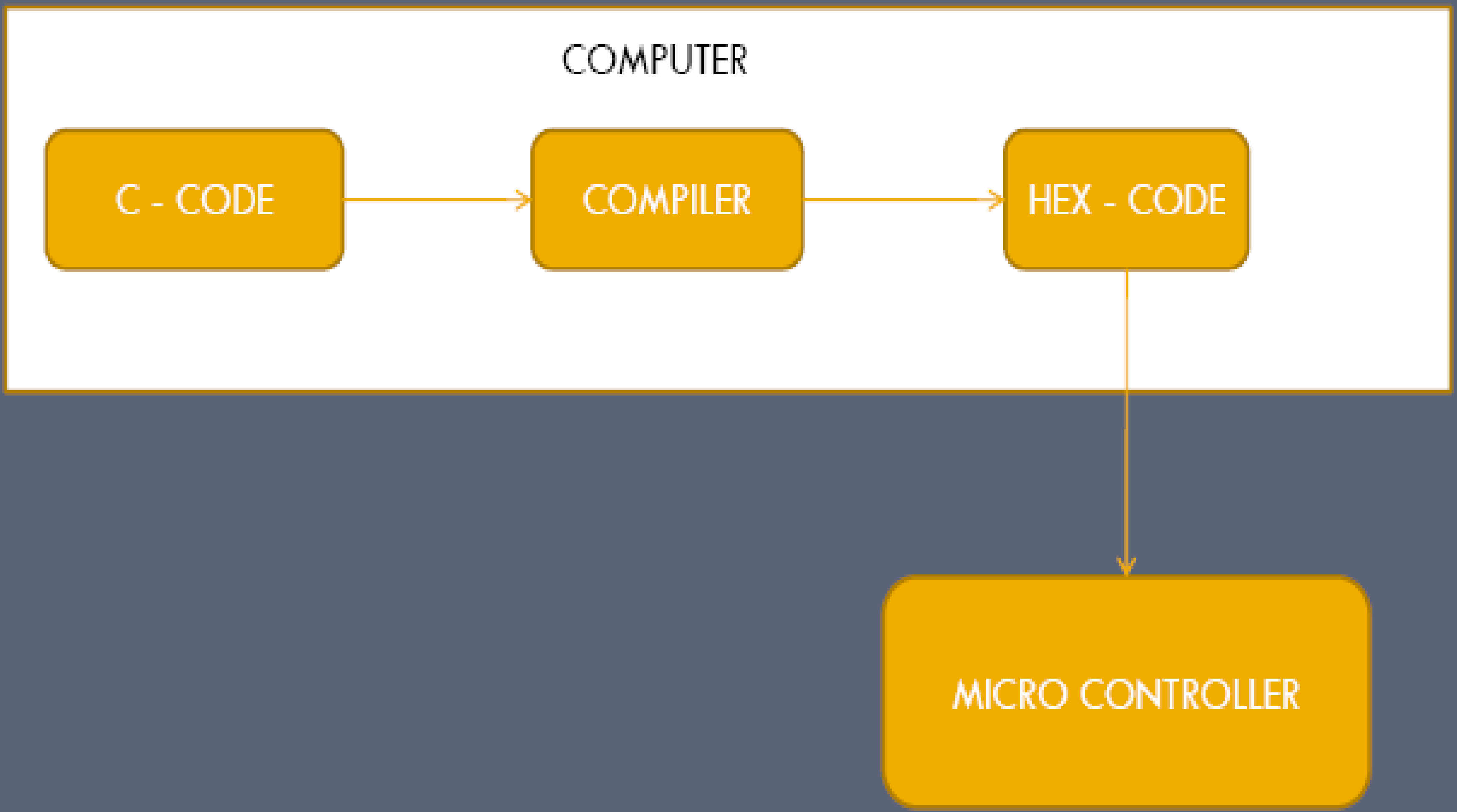
Serial Communication using the SPI

- SPI – Serial Peripheral Interface.
- It is another form of serial communication available to AVR microcontrollers.
- It is a synchronous serial communication bus, meaning that the transmitter and receiver involved in SPI communication must use the same clock to synchronize the detection of the bits at the reciever.
- SPI is generally used for very short distance of communication with peripherals or other microcontrollers which are on same circuit board.

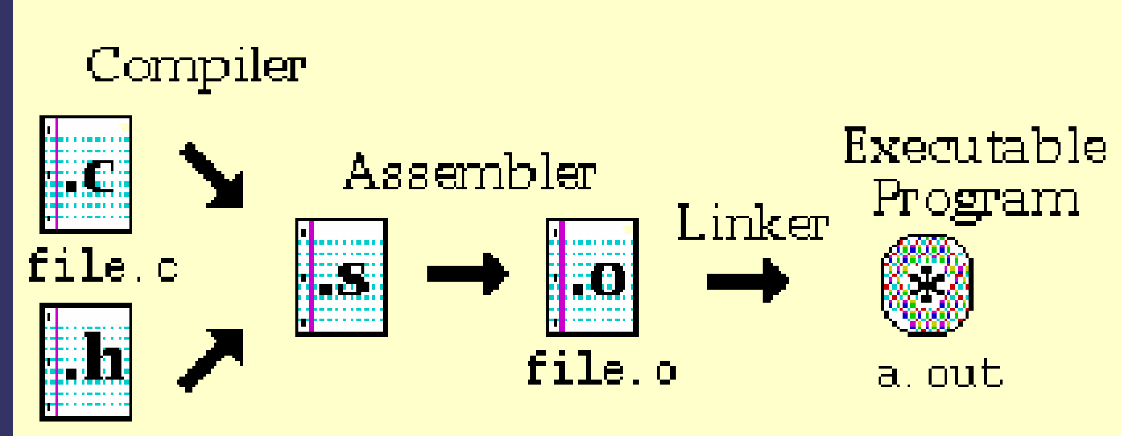
JTAG Interface

- JTAG – Joint Test Action Group
- The AVR IEEE std. 1149.1 compliant JTAG interface can be used for –
- Testing PCBs by using the JTAG Boundary – scan capability
- Programming the non – volatile memories, Fuses and Lock bits.
- On – chip Debugging.

Programming the Microcontroller

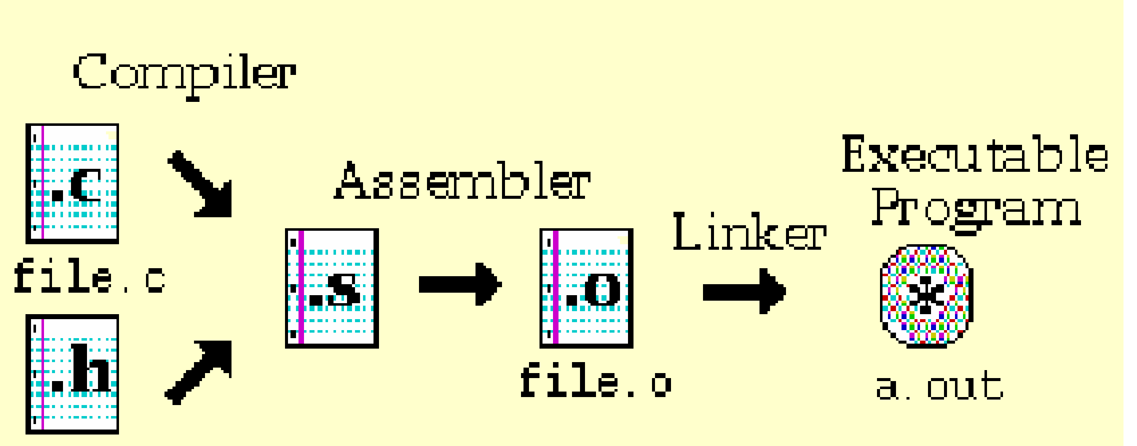

COMPUTER
C - CODE
COMPILER
HEX - CODE
MICRO CONTROLLER

Simple Compilation



- Compiling a small C program requires at least a single .c file, with .h files. There are 3 steps to obtain executable program
- Compiler stage: All C language code in the .c file is converted into a lower-level language called Assembly language; making .s files.
- Assembler stage: The assembly language code is converted into object code which are fragments of code which the computer understands directly. An object code file ends with .o

Simple Compilation



- Linker stage: The final stage in compiling a program involves linking the object code to code libraries which contain certain "built-in" functions, such as printf. This stage produces an executable program, which is named a.out by default.

ICCAVR

- ICCAVR, the ImageCraft's C Development Environment is a program for developing AVR microcontroller applications using the ANSI standard C language

- Full featured 30-day demo program can be downloaded from the ImageCraft web site

www.image-craft.com

Getting Started with ICCAVR

1. Project>New
2. Project name and path
3. Project>Options
4. In the Compiler options, check “Accept C++ Comments” and “Intel HEX” as the “output format”
5. In the Target options, select “ATmega32/Atmega16” under “Device Configuration”
6. Write the source code
7. Add the source file to the project by selecting “Project>Add File(s)” and select the file just written
8. Compile by selecting “Project>Make Project” or by clicking on the “build” icon
9. Open AVR ISP and download the hex file generated by ICCAVR , It is stored under the project name.

AVR advantages

- Micro-controllers will often allow an optimal solution, combining complex functionality with reduced part count
- ATMEL AVR chips pack lots of power (1 MIPS/MHz, clocks up to 16MHz) and space (up to 128K of flash program memory and 4K of EEPROM and SRAM) at low prices.
- HLL Support, like C, helps increase reuse and reduce turn-around/debug time/headaches.
- In-System Programmable flash--can easily program chips, even while in-circuit.

- Many peripherals: a whole bunch of internal and external interrupt sources and peripherals are available on a wide range of devices (timers, UARTs, ADC, watchdog, etc.).
- 32 registers: The 32 working registers (all directly usable by the ALU) help keep performance snappy, reducing the use of time-consuming RAM access.

- Internal RC oscillators can be used on many chips to reduce part count further.
- Flexible interrupt module with multiple internal/external interrupt sources.
- Multiple power saving modes.

Some abbreviations

- SRAM – Static Random Access Memory
- EEPROM - Electrically Erasable Programmable Read - Only Memory
- RISC – Reduced Instruction Set Computing
- UART – Universal Asynchronous Receiver and Transmitter
- SPI – Serial Peripheral Interface
- ADC – Analog to Digital Converter
- JTAG – Joint Test Action Group

Links

- A very good document to refer at initial stage is "Novice’s Guide to AVR Development" by Arild Rødland, It is an Introduction intended for people with no prior AVR knowledge.

http://www.atmel.com/dyn/resources/prod_documents/novice.pdf

- 'IAR Embedded Workbench Kickstart Edition 4.0'

http://www.iar.com/index.php?show=89661_ENG&&page_anchor=http://www.iar.com/p89661/p89661_eng.php

- Assembly, AVR Studio 4.0 is a good choice. It can be downloaded for free

at http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725

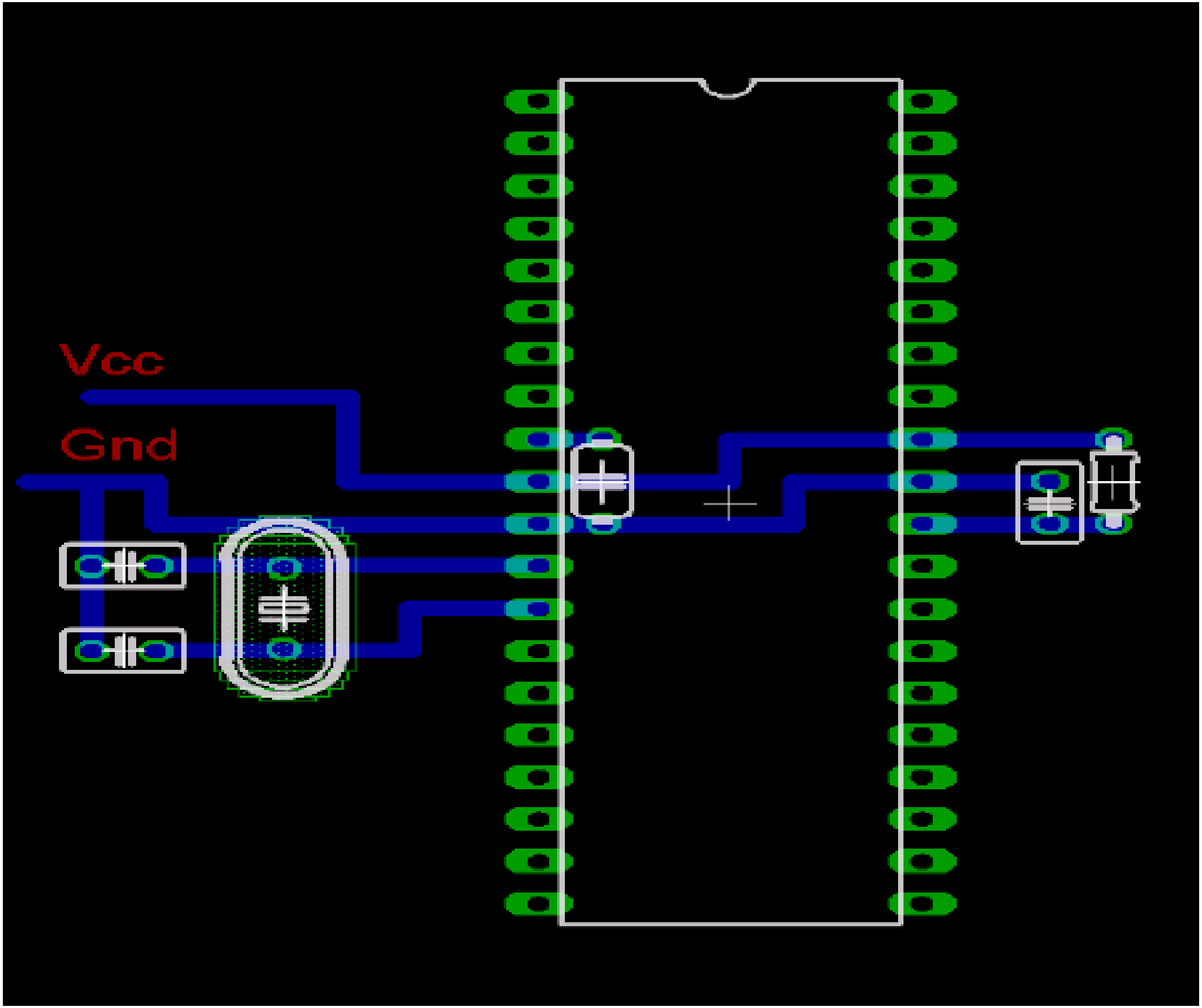
- Ponyprog 2001 is the download utility which is available for free at

http://www.lancos.com/ppwin95.html

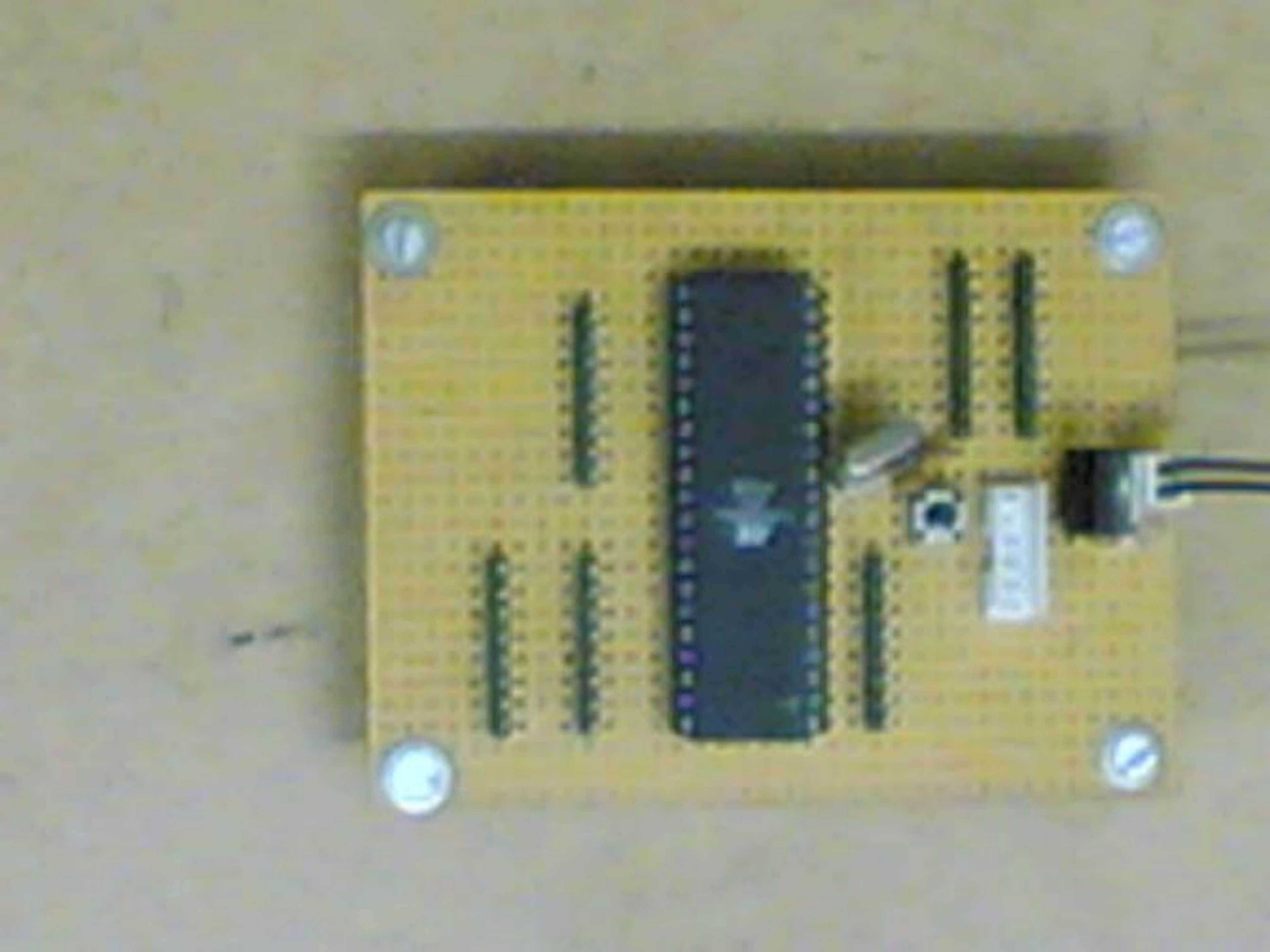
- 'How to Program 8 bit microcontroller using C language', by Richard Mann

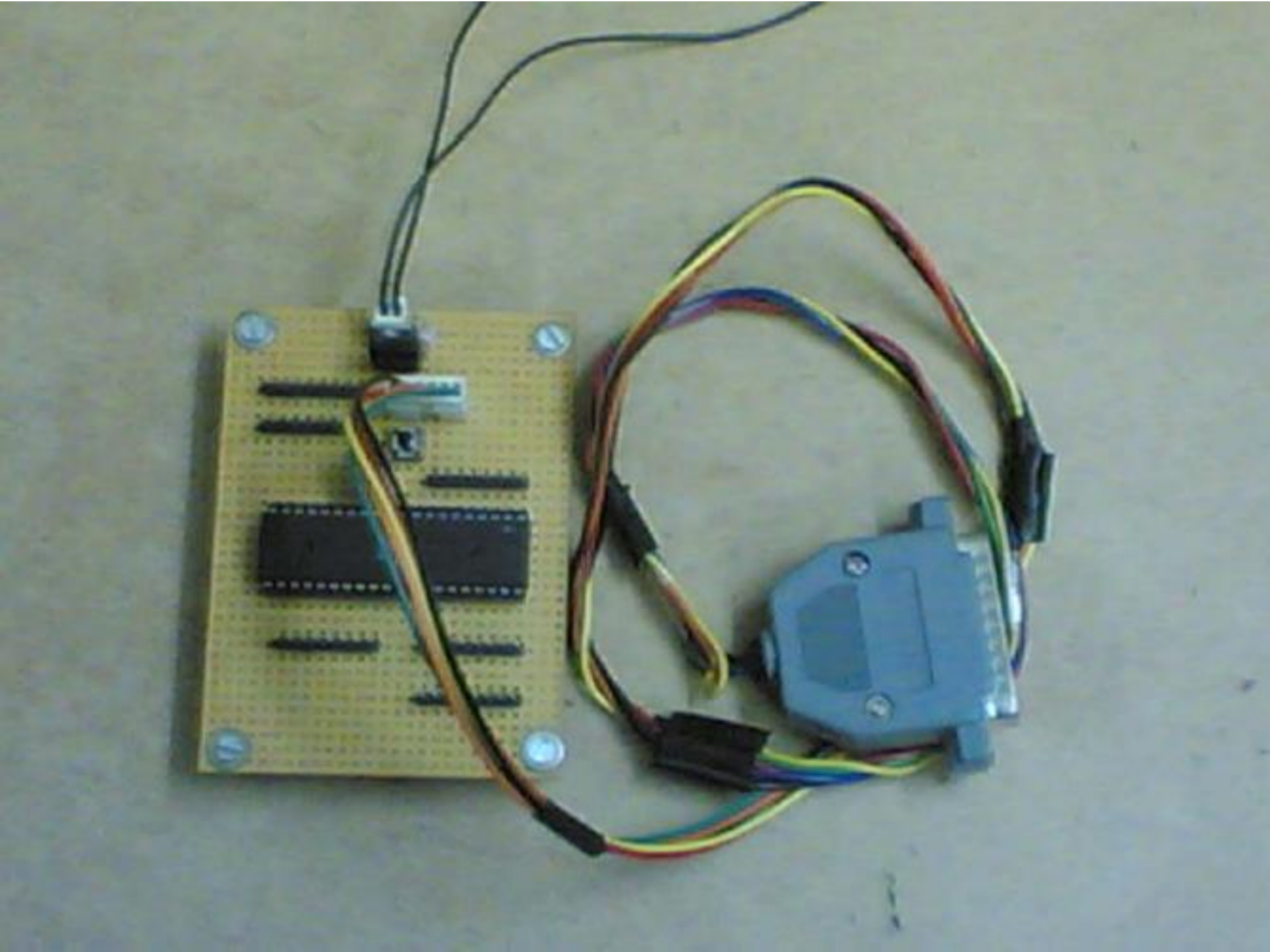
http://www.atmel.com/dyn/resources/prod_documents/avr_3_04.pdf

- AVR tools@http://www.sonsivri.com



Vcc
Gnd



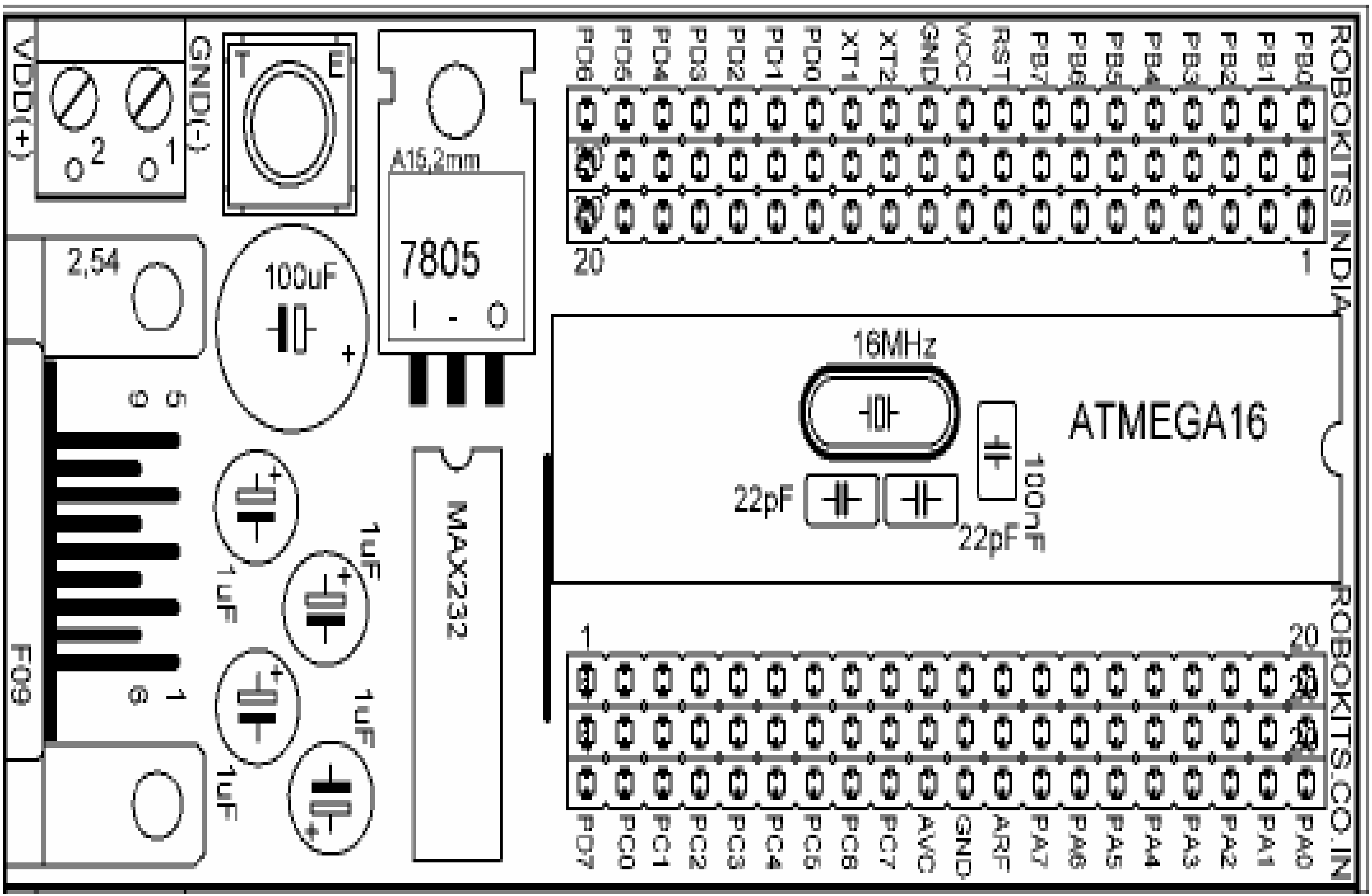


BootAVR Rapid Development Board

Features

- Small board size - Just 90mm X 42mm
- On Board Regulator with filters and Operating voltage from 6V - 20 V
- Power on/off toggle switch
- 16MHz crystal for maximum speed
- 8 ADC/Standard servo compatible connectors
- All Pins accessible through Male/Female header pins
- PC-MCU serial link onboard
- No programmer required for programming
- Microcontroller without bootloader can also be used with this board
- Also hex files and fusebit setting are provided free to use your own microcontroller with this board
- Freeware software for programming through bootloader

Board Top Layout



Interfacing with Microcontroller

- LED
- Switches
- DC Motor
- Sensors
- Stepper Motor

Created by DARSH SHAH

www.darshshah.blogspot.com