BASICS OF MICROCONTROLLERS

So, interested in controlling these microcontrollers? And make some really cool stuff out of it? Well, for that you should be familiar with their basics and how a microcontroller development process takes place.

Elementary Microcontroller Concepts

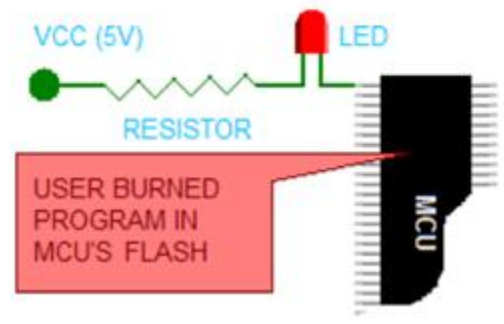
Before we start off, it’s essential for you to know some elementary concepts related to MCU. To make it quick, let’s codify them as follows:

- A MCU is an Integrated Circuit (IC). The number of pins, size, structure and architecture may vary depending upon the manufacturer and model.
- A combination of pins in a MCU is known as a *port.* The configuration of ports may vary from manufacturer to manufacturer.
- Every MCU has a flash memory. It is equivalent to the hard drive of a PC.
- Data can be erased and rewritten as many times as you want.
- Every MCU has a lifespan of around 1000-1500 read/write cycles. After that it becomes dead and stops working. In fact every IC has a lifetime in terms of read/write cycles. For example, pendrives usually have 10000 read/write cycles. Taking an average pendrive, it should stop working after 10-12 years.

Now that you are familiar with some basic concepts, let’s design a simple MCU based application.

Simple MCU Application

The simplest possible MCU based solution could be blinking an LED continuously at a specified time interval. Okay, let’s design a solution for it. Don’t worry; we will be designing a theoretical solution for it. A practical design will be made later (not in this post).



Simplest MCU Process

Problem Statement: Design a microcontroller based solution to blink an LED every 250ms.
Solution:

- Let us assume our supply voltage to be 5 volts i.e. Vcc = 5V
- Connect the LED to a pin (not port, mind it! View next section for more details on ports and pins of a microcontroller) of the MCU as shown.
- The resistor is provided to prevent the LED from blowing off.
- Now, when we make it voltage of the corresponding pin to zero (0 volts) (LOW), the LED glows due to the generated potential difference.
- When we make the voltage of the pin to Vcc (5 volts) (HIGH), the LED switches off due to insufficient potential difference.
- The timing can be easily altered by introducing appropriate delays.
- The following pseudo code includes this feature. Yes, let’s code it! It’s not tough!

```
void main()
{
   InitializePort();        //Initialization
   while(1)                 //Run infinitely
   {
      PORTB = 0b00000001;  //HIGH
      Wait(0.25);          //Wait 250ms
      PORTB = 0b00000000;  //LOW
      Wait(0.25);          //Wait 250ms
```

```
    }
}
```

Code explained:

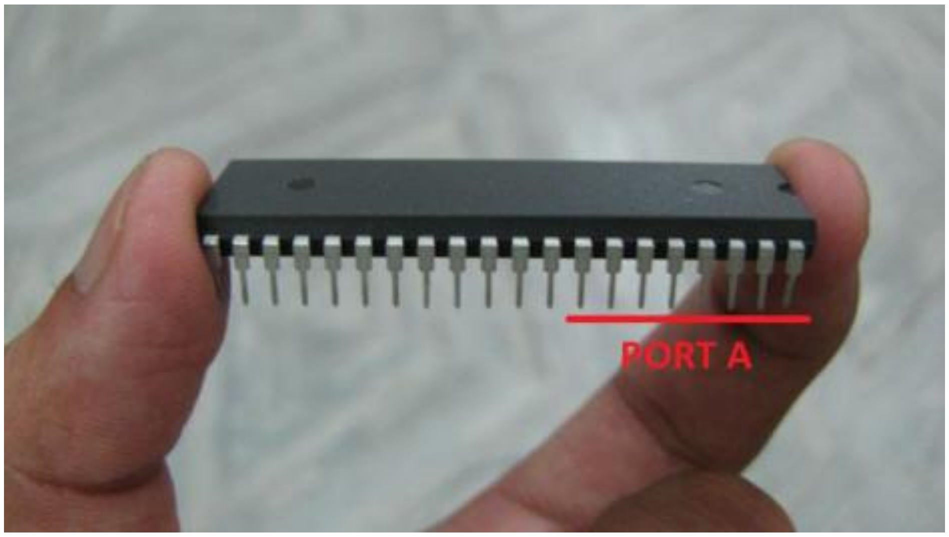
- Starting with the best part, it's written in C :)
- InitializePort() – this function call initializes the values of the *pins of the port*
- while(1) – '1' implies TRUE condition. While(condition) continues iterating until a false condition is met. Since the condition is *always* true, the loop is an infinite loop. Now, in C, an infinite loop is treated as a runtime error, but not in this case. This is because we want the LED to blink continuously as long as the circuit is powered. We don't want it to stop blinking after blinking for some particular number of times.
- PORTB = 0b00000001 – don't worry about this line much! We will discuss about this syntax later on. All you need to notice is the '1' in the end. Here, '1' implies HIGH i.e. 5 volts, which will switch off the LED.
- Wait(0.25) – again here, don't worry about the syntax. Just know that it introduces a delay of 250ms.
- PORTB = 0b00000000 – notice that the '1' is changed to '0' here, which means LOW i.e. 0 volts, which turns on the LED.
- Wait(0.25) – once again a 250ms delay is introduced.

Congratulations! You have successfully designed an embedded solution! :)

Please note that I have assumed that you have a prior knowledge of C programming. If not, then go to the market and buy "Let Us C" by Yeshwant P Kanetkar and start reading it.

Ports and Pins of a MCU

- As I already said, port contains the pins of a MCU. In the adjoining figure, PORT A of an AVR microcontroller is shown. Have a close look at it and you will come to know that it consistes of 8 digital GPIO pins.



Port comprises of several pins

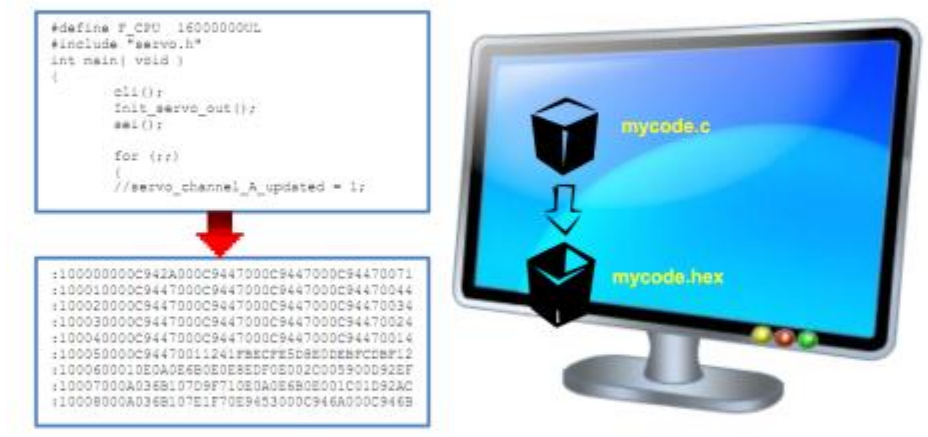
- In a MCU, most of the pins are digital GPIO pins (General Purpose Input Output pins). These digital pins can be turned on or off (or can be turned HIGH or LOW) as per the requirement.
- Not all pins are GPIO pins, there are some pins (like Vcc, GND, XTAL, etc) which are for other functions and cannot be turned on/off for interfacing.
- GPIO have two modes:
    - **Output mode:** It's quite simple. Setting it HIGH (1) gives an output of Vcc at that pin. Setting it LOW (0) gives an output of zero at that pin.
    - **Input mode:** In this mode, the MCU can read the values at the pins. Here, a threshold is defined. Threshold voltage is usually half of Vcc. If a voltage above the threshold is read, it reads it as HIGH (1) or else LOW (0). For example, suppose our supply voltage Vcc = 5V. Hence, threshold = 2.5V. When we apply a voltage 5V, it reads as HIGH (as $5 > 2.5$). When we apply a voltage of 0V, it reads as LOW (as $0 < 2.5$). Suppose 2V is applied. In this case, it will consider it as LOW (as $2 < 2.5$).
    - The input/output mode of GPIO pins is set by DDR register (which we will discuss later).

Now that you are familiar with the basics, let's have a look at the MCU based Development Process.

MCU Based Development Process

A MCU based development process consists of two simple steps:

- **Step 1:** Write a desired code for the given problem statement in a PC/laptop in an IDE, edit, compile and debug it. Say if you code in C, your source file will be something like mycode.c, which will be converted into a hex file mycode.hex.



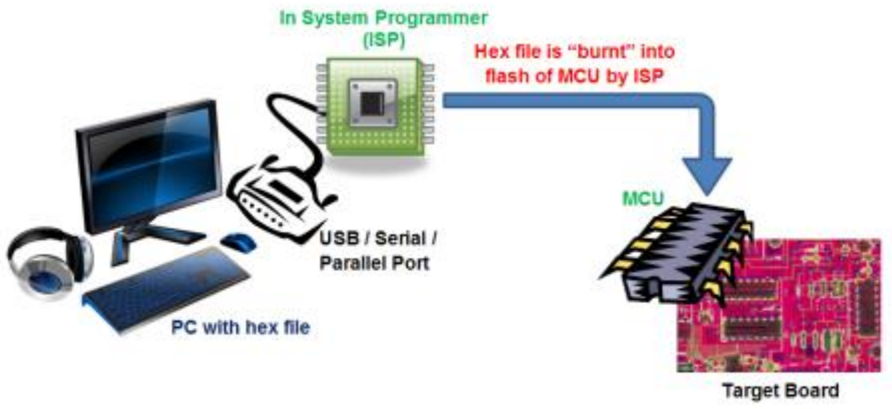
```
#define F_CPU  16000000UL
#include "servo.h"
int main( void )
{
        cli();
        init_servo_out();
        sei();

        for (;;)
        {
        //servo_channel_A_updated = 1;
```

```
:100000000C942A000C9447000C9447000C94470071
:100010000C9447000C9447000C9447000C94470044
:100020000C9447000C9447000C9447000C94470034
:100030000C9447000C9447000C9447000C94470024
:100040000C9447000C9447000C9447000C94470014
:100050000C94470011241FBECFE5D8E0DEBFCDBF12
:1000600010E0A0E6B0E0E8EDF0E002C005900D92EF
:10007000A036B107D9F710E0A0E6B0E001C01D92AC
:10008000A036B107E1F70E9453000C946A000C946B
```

MCU based Development Process Step 1

- **Step 2:** Open up the programmer software, locate hex file and click on “Program”. The software first reads the flash of the MCU to check the amount of data, then an erase cycle will be performed, and then your hex file will be *burnt* into the flash! But the process doesn’t end here… the file is verified so that you can be sure that your code has been transferred successfully! All this process happens with just one click!


In System Programmer (ISP)
Hex file is “burnt” into flash of MCU by ISP
MCU
USB / Serial / Parallel Port
PC with hex file
Target Board

MCU based Development Process Step 2

Don’t worry regarding the hardware shown in the diagram. We will come across them in my next post. Just concentrate on the process.

Why use hex file?

Well, everything seems fine and perfect! But did you give it a thought why did we burn the hex file and not the C file? Well, it’s a natural fact that hex files execute much faster than c files. But let us consider the following facts:

- Suppose we use a 16MHz crystal in order to generate clock pulses. Even a substantially large C file will execute in a flash. So, why convert it to hex?
- Let’s say we program using BASIC. Still then it is converted to hex. If the BASIC files are executed faster than C files, do we need to convert it to hex?

Yes! We do! Here’s why:

- Sticking to the natural fact that hex files execute the fastest, they introduce very little delay. Using a C or BASIC file may introduce undesirable delays, which may add up and slow down the system in the end.
- Say, let us even consider the best case, where the execution time of hex file is nearly same as that of C or BASIC. This will do well for general applications. But suppose we design a solution to be installed in a car, where the MCU controls the airbag and releases it as soon as an accident happens. *This is where speed comes into play*. Your C and BASIC files will take too long to respond, and by the time they respond, the worst would have happened.

Manufacturers, at the time of manufacturing a MCU, have no idea whether that MCU will be used to blink LEDs, used to control satellites or used as target detection mechanisms in missiles. A hex code may seem useless to blink LEDs, but it’s quite valuable for the remaining two applications.

Source: http://maxembedded.wordpress.com/2011/06/06/basics-of-microcontrollers/