Flash Burning Process & Program Explanation – Tutorial #4

T.K. HAREENDRAN

AVR tutorial

Share this:

Share More

Today after many days of waiting (thanks for your patience) you got the chance to program the AVR atmega8 microcontroller. As said earlier, for this task you need one USBASP programmer. Just connect the output of the programmer (6 lines VCC, GND, RESET, SCK, MOSI, MISO) to the respective pins of the unpowered target atmega8 (which you placed in the breadboard). The final thing needed is a program (burner tool) for flash burning! To burn the program you can use this simple and easy to use burner software, which is infact a GUI for the AVRDUDE (a command line tool). The program "AVRLoader" can be downloaded from my Google Drive using this link **http://goo.gl/cfIcIV** . (You can also try this too: **http://extremeelectronics.co.in/avr-tutorials/guisoftware-for-usbasp-based-usb-avr-programmers/**)

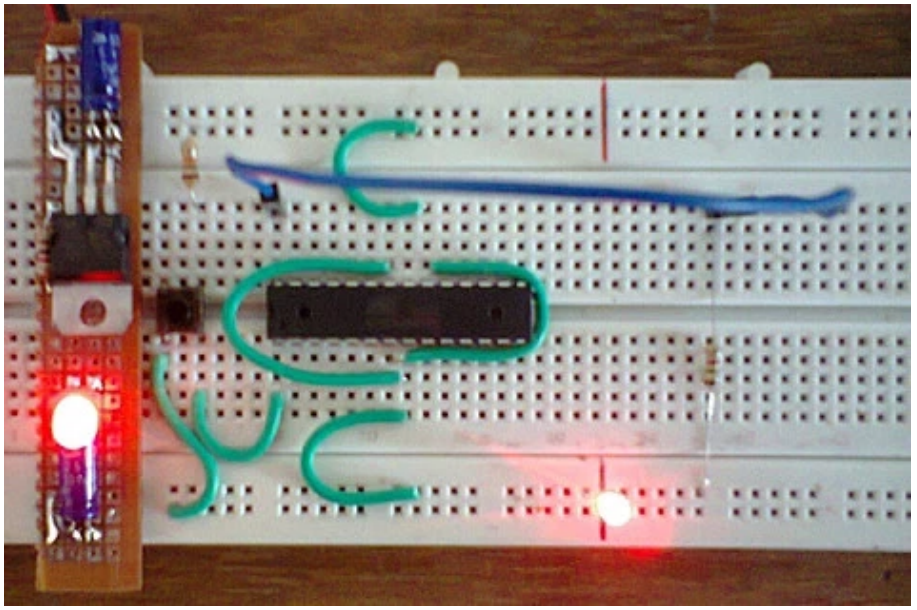
Flash burning process

1. Start AVRLoader
2. Select the Microcontroller (Atmega8)
3. Point out the .hex file(Your .hex file in 'default' folder inside the project folder)
4. Click WriteButton (in the 'Flash' option)

download AVRLoader

When finished, remove the USBASP programmer connection from breadboard. Now take one LED and connect the LED's anode to pin 28 of the Atmega8 in the breadboard and route the LED's cathode to 0V (GND) rail through a 1K resistor. Finally connect the 5V breadboard power supply and keep an eye on your first AVR project.

The burned program turns PORTC pin 28 of the ATMega 8 on and off, alternatively with a pre-defined delay (LED should blink in a frequency of 1 sec). Note that until you write fuse bits for external oscillator (will discuss this later),by default Atmega8 runs on 1MHz calibrated internal RC oscillator. Thas's why no external Xstal is used here in our first project.



Program explanation

```
#define F_CPU 1000000UL //define clock speed
#include  //include input/output header file
#include  //includes delay header file
int main(void) {
   //Set the Data Direction Register to output
   DDRC |= (1<<5);
   while (1) { //create an infinite loop
      //Set the signal to high
      PORTC |= (1<<5); //this turns pin C5 on and off, turns C5 HIGH
      //wait 0.5 sec
      _delay_ms(500); //PAUSE 500 miliseconds
      //Set the signal to low
      PORTC &= ~(1<<5); //turns C5 LOW
      //wait 0.5 sec
      _delay_ms(500); //PAUSE 500 miliseconds
   }
}
```

In this code snippet we have defined a macro, F_CPU in the first line for specifying the microcontroller's clock frequency. This macro will be used by the delay function. In the second line we have included a header file for using some readymade functions and keywords. In the third line another file for creating the delay is included. This function exists in the 'util' folder in default inclusion directory. After starting the main function, our first line sets the Data Direction Register (DDR) to output. DDRC indicates PORTC (likewise, PORTB can be indicated by DDRB).

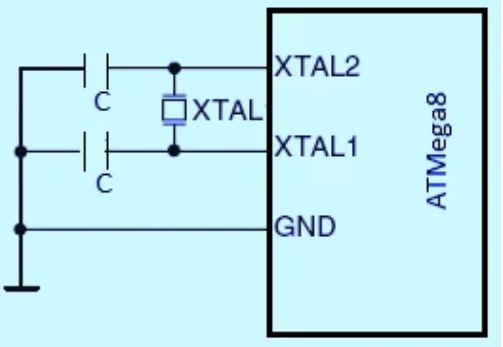
Now we have started an infinite loop for continuous blinking, with a delay function (this delay can be varied) between two commands.

We skimmed over a lot that you'll see in detail in the coming parts of this tutorial. You now know just enough to start your journey with AVR microcontrollers. Now that we learned how to blink an LED using Atmega8 with the help of a simple code. Incase of interest you can modify the program as per your requirement. For instance, tweak the code to:

- Modify the LED blink rate
- Change the LED output pin (from pin 28) of Atmega8
- Increase the number of LED pins, from one (pin 28 only) to more pins of PORTC

The Clock Source!

As stated earlier, ATMega8 is shipped with it's clock source configured to the internal 1MHz RC oscillator, by default. This configuration works good for many simple projects where timing isn't crucial, and is also comfortable as it reduces external component count. However, you may run in to situations where you need a more accurate clock source, and for this it's necessary to add an external crystal oscillator. The crystal oscillator is connected to the microcontroller as shown below. In order to configure your microcontroller to use an external crystal oscillator, you have to change the 'Fuse bits'. For a novice, AVR fuse bits seems to be very flurrying but they are not so!



→ Part 5: **The Clock Source & Fuse Bits**
← Part 3: **USB ASP Programmer & Atmega Programming**

Share this: