EE318 Electronic Design Lab Project Report, EE Dept, IITB, April 2009

Wireless Microcontroller Programmer

Group No. B04
Varun Jog (06007012), varunjog@iitb.ac.in
Rohit Agarwal (06007014), rohitagarwal@iitb.ac.in
Robin Mandal (06007022), robinmandal@iitb.ac.in

Guide: V Rajbabu

Abstract

This project aims at facilitating wireless programming of a microcontroller. It exploits the capability of a microcontroller to write in its own flash memory. This is made possible by pre writing a 'Bootloader' code in the uC which accepts serial data and writes it in the main memory. The set structure of a hex file is used to fabricate an error free wireless transmission-reception protocol. Over 3 months, we have been able to design and implement such a programmer for 28 and 40 pin microcontrollers of the Atmega series. The following sections describe the hardware and software aspects of the project in an organized manner.

1 Introduction

During the testing stage of a microcontroller code, programming and reprogramming of the microcontroller requires us to pluck the uC from the circuit a large number of times. Not only is this process tiring, frequently attaching and detaching the uC from the circuit can cause damage to the pins. Similar experiences during the BabyEDL project motivated us to design a wireless microcontroller programmer. We note that an ISP could achieve the same motive however it would require that the uC be easily accessible to the programmer which might not always be possible. This reinforced our decision to make a wireless programmer.

The project objective can be restated as wirelessly transmitting a hex file from the computer to a microcontroller's main memory. We first arrange for the target microcontroller to be in a state where it can accept serial data at its RXD pin. This is achieved through a Bootloader program preloaded in the microcontroller using conventional programming methods. We use the computer's serial port to transmit the hex file to a Amplitude Shift Keying(ASK) based transmitter which operates at 433MHz. The receiver at the other end

passes the data online to the target microcontroller where the Bootloader accepts the incoming data and writes it in the uC's main memory.
Both transmitter and receiver are powered by 5 Volt regulated supplies. The programmer hardware entertains all microcontrollers and requires no customization, however the Bootloader program needs to be modified suitably for programming different microcontrollers.

2 Design Approach

2.1 Hardware

We can divide the hardware designing problem into 2 sub parts:

- Interfacing the ASK Transmitter Module with the PC
- Interfacing the ASK Receiver Module with the Microcontroller

The first part requires interfacing the serial port of the computer with the transmitter. This is done using an RS232 connector, however, the 'High' and 'Low' voltage levels of the computer are 10V and -10V respectively which need to brought to 5V and 0V for normal circuit operation. This is achieved by mediating using a MAX232 IC.
Interfacing at the receiving end requires just connecting the DATA OUT pin of the receiver to the RXD pin of the microcontroller. An unregulated 12V supply is downconverted to a 5V regulated supply using LM7805 voltage regulator. LEDs at PORT B of the microcontroller indicate whether the uC is being programmed or if there has been an error in programming. LEDs at PORT A of the uC are used for testing purposes i.e. whether the uC performs as expected after programming.

2.2 Software

Again, two different software designing problems can be identified. These are:

- Program to send a hex file from the PC serially
- Bootloader Program to serially receive the data from the receiver and check for bit flips

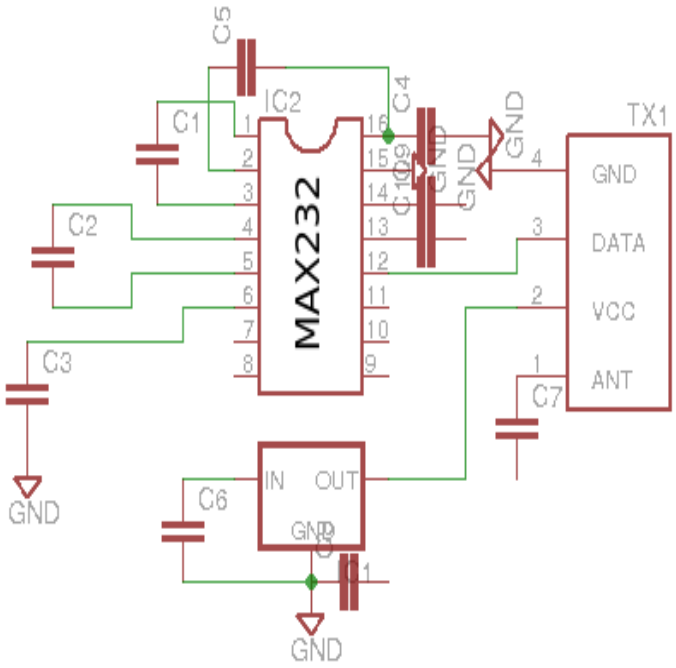

Figure 1: Transmitter circuit, from [2]

We used Hyperterminal and BrayTerminal [1] during the initial testing stages to send the hex file byte by byte and line by line respectively. However, we chose Matlab to send the complete hex file because of its high configurability. The Bootloader program is designed to identify whether the data at the receiver antenna is the desired hex file or noise contracted from the channel. It accordingly instructs the uC to write the data in the memory or generates signal for retransmission of data. When the entire hex file has been written in the memory, it automatically jumps to the program's starting location (0x0000) and starts executing the program.

3 Circuit Design

3.1 Hardware Design

In this section we describe the various Hardware components that we have used while designing the circuits and the reasons for choosing them.
We've used the following major components at the transmitter and receiver side:

- **MAX232**

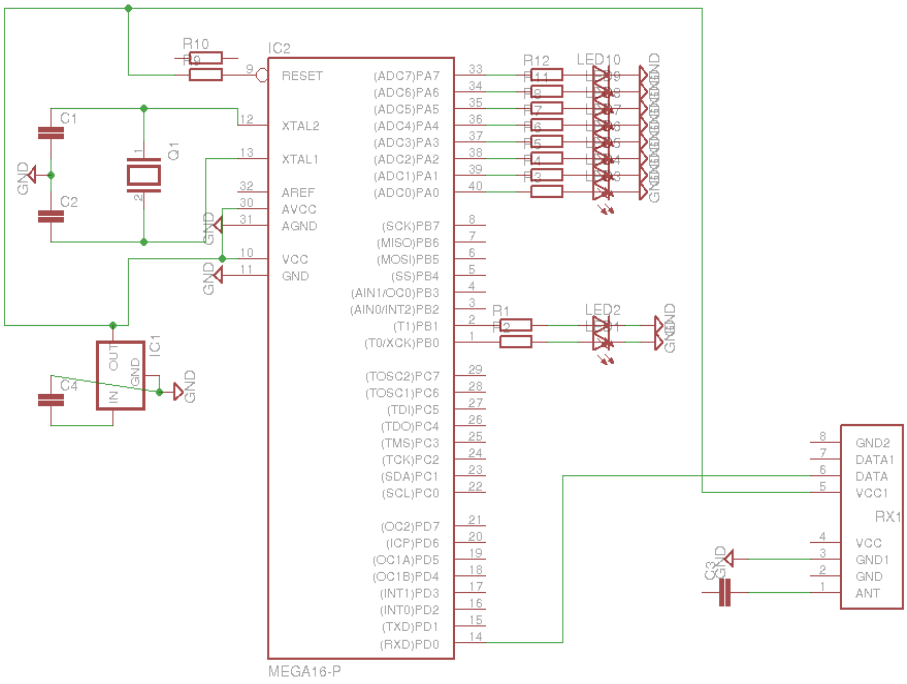

Figure 2: Receiver for Atmega16

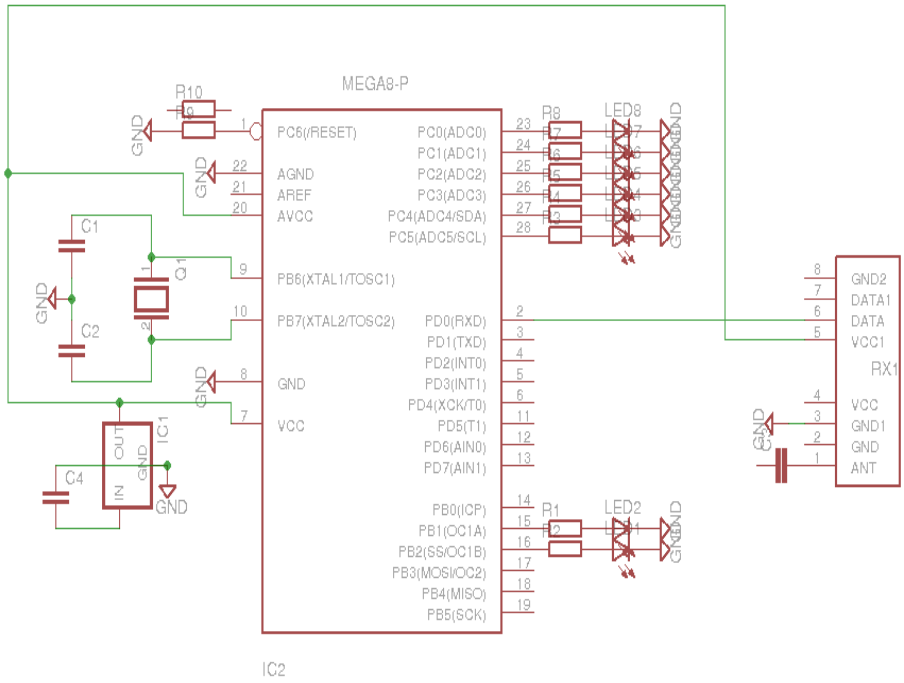

Figure 3: Receiver for Atmega168

The High and Low level voltages from the computer are +10V and -10V respectively. However othercircuit components interpret +5V and 0V as the high and low level voltages. Therefore, we need to bring the voltage levels down to these values. This is done using a 16 pin MAX232 IC.

- **LM7805**
  We need to provide a regulated 5V power supply to various circuit components. The supply from the source is not reliable and fluctuates somewhat. So we use the LM7805 voltage regulator to provide a 5V regulated supply from a 12V unregulated supply. The same is used at the receiving end.

- **ASK Transmitter and Receiver Module**
  We were faced with a choice between RF and Bluetooth as the possible means for wireless communication. The small range of Bluetooth modules somewhat defeats the purpose of our project. Some 'Sparkfun' bluetooth modules were available [3] but they were costly(around Rs. 2000). This prompted us to choose RF modules which were available for cheap (around Rs. 330) and had a relatively higher range.
  Once we were decided with this, we were faced with a choice between ASK and FSK modules. FSKs were our first choice but the non availability of FSK modules made us try out ASK Modules. However, after testing the ASK Modules as described in the next section, we found that they could serve the purpose too provided we chose a suitable Baud Rate and made some changes in the Bootloader program described next. Also, ASK modules come cheaper than FSK Modules (around Rs.1000) according to [4]. This made us settle at ASK Modules for setting up the RF link.

- **Atmega Microcontrollers**
  Our programmer is capable of programming any microcontroller which supports self programming. Since Atmega series microcontrollers are self programmable [5] and are widely used, we chose our target uCs from the Atmega series. For demonstration purposes we've made the programmer for 28 pin Atmega168 and 40 pin Atmega16.

3.2 Software Design

In this section we describe the MATLAB code for sending a hex file from the computer and the Bootloader program for taking serial data and writing it in the uC's main memory.

- **MATLAB Code to transmit HEX file**
  Matlab is capable of communicating via the PC's serial port. This is done by defining a serial port object s and giving it suitable properties like port number and baudrate.
  Our code initially sends a string which tells the bootloader that a new program is about to be transmitted. Then, we load the hex file to be transmitted, and transmit it serially line by line. To ensure that each line is received, we transmit each line three times.

- **Bootloader Program**
  The bootloader depends heavily on the fixed structure of an intel hex file.
  Stucture of an intel hex file
  Each intel hex file consists of six parts as described by [6]:

  - **Start code**, one character, an ASCII colon ':'.
  - **Byte count**, two hex digits, a number of bytes (hex digit pairs) in the data field. 16 (0x10) or 32 (0x20) bytes of data are the usual compromise values between line length and address overhead.
  - **Address**, four hex digits, a 16-bit address of the beginning of the memory position for the data. Limited to 64 kilobytes, the limit is worked around by specifying higher bits via additional record types. This address is big endian.
  - **Record type**, two hex digits, 00 to 05, defining the type of the data field.
  - **Data**, a sequence of n bytes of the data themselves. 2n hex digits.
  - **Checksum**, two hex digits - the least significant byte of the two's complement sum of the values of all fields except ':' and checksum byte.

  The rough version of the bootloader was adapted from [7]. The bootloader program accepts two strings as input. "uuuu" and "bbbb". The first string commands the bootlloader to erase the entire program memory, and get ready to write a new program. The second string tells it to jump to address 0x0000, that is the beginning of an existing program and start executing it.
  If the string is "uuuu", then the bootloader creates a temporary page in its memory whose size is 16 bytes. As the hex file line is received, this line is stored in this temporary page. If the page does not fill completely (owing to lesser number of bytes being sent in the line), the

remaining bytes are filled as 0xFF. After each line, the checksum bit is used to verify if the line has been received without errors. If there is an error, the bootloader generates a signal which indicates that an error has occured and that the program should be re-transmitted. If the target uC's TXD pin is connected to a PC's serial port, we can observe the output as each line is written— a code "G12T." is transmitted as each line gets successfully written and "G12T.*" is transmitted when the program is completely written successfully. Also, one LED on PORTB toggles as each line gets written, and is permanently on when the code is completely written.
We had to implement several noise handlers in our code as the ASK link used was very noisy. Some of them were—

- The initial code string was made "uuuu" and "bbbb" as it is very improbable that the same character will occur as noise four times in a row.
- The code refuses to write anything unless it receives a ":" as the start bit
- We send each line three times to make sure that its received atleast once. For each line we check it address characters to make sure that it is a new line and not one that has already been written.

4 Testing Procedure and Results

- Testing the Computer's Serial Port
  To test the computer's serial port, short the RXD and TXD pins of the RS232 connector. Using Hyperterminal, send a character and if it appears twice on the display then the serial port is working just fine.
  In our case, we had sent stream of characters at 1200 Baud Rate and observed it reported twice on the display.

- Testing RF link
  It is quite possible that the transmitter and receiver pick up noise from the channel. In our application , where we need 100pc accuracy in reception of transmitted data, it becomes important that we identify the Baud rate such that bit flips are kept to the minimum while at the same time maintaining a decent programming time.
  For testing this, we interfaced the transmitter module with BrayTerminal at one computer and the receiver module with BrayTerminal at another computer. BrayTerminal allows sending a stream of characters

at a time. By sending data from one PC and observing it on the other, we could make out whether the data is getting corrupted.
In our case, we noticed that the receiver was receiving noise till the data was sent, however once the stream of data was sent, it was received uncorrupted. This prompted us to device a mechanism which makes out whether the data at the reciver is the data of interest or arbitrary noise.

- Testing Bootloader program
  The Bootloader function is supposed write the program in uC's main memory and execute it once it is written. For testing the bootloader, we sent a LED blinking code [8] from the computer and observed the LED blink at port C of the ATmega 168 uC after the programming period was over. This confirmed that the bootloader was working.
  Once we had all the individual links working, we sent the hex file from the PC and observed the LEDs blinking on the PORT C of the uC. We then sent a variety of codes to check the system.

5 Conclusion

The above description proves the feasibility of wireless programming and suggests its potential as a viable product. We have implemented one such programmer using ASK modules, for the Atmega series. In this section we propose some ideas for developing the product further, making it a more reliable and faster programmer.
Upgrading the existing ASK based RF link to an FSK or X-Bee radio based link can increase the data rate and hence reduce programming time considerably. However, this would incur an additional cost.
Also, the current transmitter circuit uses an external power supply. A circuit can be designed to draw power from the PC's serial port [9]. We are currently studying its feasibility. Programming time can also be reduced by optimizing the number of times a line is sent, by acknowledging the reception of a line, thus reducing the average number of times a line is transmitted. However, this would require upgrading the transmitter and receiver to transceivers each, increasing both the size and cost of the product.

References

[1] (2008) Bray terminal. [Online]. Available: braypp.googlepages.com/terminal

[2] (Feb 2007) So do it yourself. [Online]. Available: http://sodoityourself.com/max232-serial-level-converter/

[3] (2009) Sparkfun circuits. [Online]. Available: http://www.sparkfun.com

[4] (2009) Vegakit india. [Online]. Available: http://www.vegakitindia.com/

[5] (2009) All datasheets. [Online]. Available: www.datasheetcatalog.com

[6] Wikipedia. [Online]. Available: http://en.wikipedia.org/

[7] (Oct 2007) Polulu robotics forum - super simple bootloader. [Online]. Available: http://forum.pololu.com

[8] (2009) Kartik's home. [Online]. Available: http://kartikmohta.com/

[9] (2009) How to get power from rs-232 port. [Online]. Available: http://www.tkk.fi/Misc/Electronics/circuits/rspower.html

6 User's Manual

6.1 Burning the bootloader

The first step to wireless programming your atmel microcontroller is to burn the suitable bootloader onto it. This can be achieved by using the hex file of the bootloader provided and programming it using standard methods. Note that to burn a bootloader and to use it, the following fusebit settings should be used—

- Set boot reset vector to bootloader address and not the default 0x0000 address
- Burn the BOOTSZ0 and BOOTSZ1 fuses according to the size of the bootloader, and as specified by the datasheet

Once the uC has a bootloader, it can now be programmed using a suitable RF link

6.2 Setting up the RF link

Connect the transmitter module to PC's serial port. Connect the receiver module to your target uC such that the "DATA OUT" pin of the receiver module is connected to the RXD pin of your uC. Provide an unregulated 12V supply on the transmitter side, and give suitable regulated 5V voltage

to the receiver pins. Connect antennae (around 15-20cm in length) to the transmitter and receiver modules.

6.3 Programming the uC

Execute the m-file provided in matlab, replacing the test program with the name of your program. Also, fill up the hex file of your program so that each line has 43 characters, generally only the last 2 lines don't have 43 characters. In such case, fill them up with zeroes or any other character. The LED on PORTB blinks as it programs each hex file line, and another LED on PORTB lights up if there is a checksum error. Once the entire code is transmitted, the bootloader automatically jumps to 0x0000 and starts executing the program. After every reset, the uC jumps to the bootloader. If an existing program is to be executed, simple transmit "bbbb" using matlab.

6.4 Modifying the source code

If you want to modify the bootloader source files, it can be done by making the following changes in a winaver makefile.

- Add the line "BOOTLOAD= xxxxx" where xxxxx is the start address of the bootloader as specified by the BOOTSZ fuse values
- Add the following line after LDF-flags—
  LDFLAGS += -Wl,–section-start=.text=$(BOOTLOAD)
- Change target to bootloader