

ECE 477 Lab 5

Christian Knight, Nikko Noble

September 16, 2020

1 Introduction

The purpose of this lab is to use the AT90USB162 microcontroller to communicate over serial with a PC using the LUFA library, which is an open source USB library. Specifically, the goal is to configure the microcontroller such that eight switches connected to input pins on ports C and D and eight LEDs connected to output pins on port B can be interfaced with over serial. This is done using a C routine on the AVR based on the 'VirtualSerial' example in the LUFA library that makes the AVR show up as a serial device over USB and accepts a line of text to either check the status of the switches or LEDs or set given LEDs, in a similar fashion to lab 4. An additional PC program was developed to interface with the microcontroller over serial using the 'termios' library to configure the terminal.

For Part B, a C program was developed for the AT90USB162 that accepts a text string from a PC over serial and flashes an LED with the Morse code information corresponding to the text string.

2 Part A

This section describes the development of the C program that configures the AT90USB162 microcontroller to communicate over serial based on the 'VirtualSerial' example in the LUFA library and interface with connected switches and LEDs. The program works by accepting a line of text from a PC over serial that begins with either an 'L' or 'S' (insensitive to case) to indicate if the LEDs or switches are to be operated on, followed by either a '?' to return the current value of the LEDs or switches in hexadecimal or an integer value (in either decimal, hexadecimal, or octal) in the case of the LEDs to set them.

2.1 Hardware

The hardware schematic for this lab is shown in [Figure 1](#).

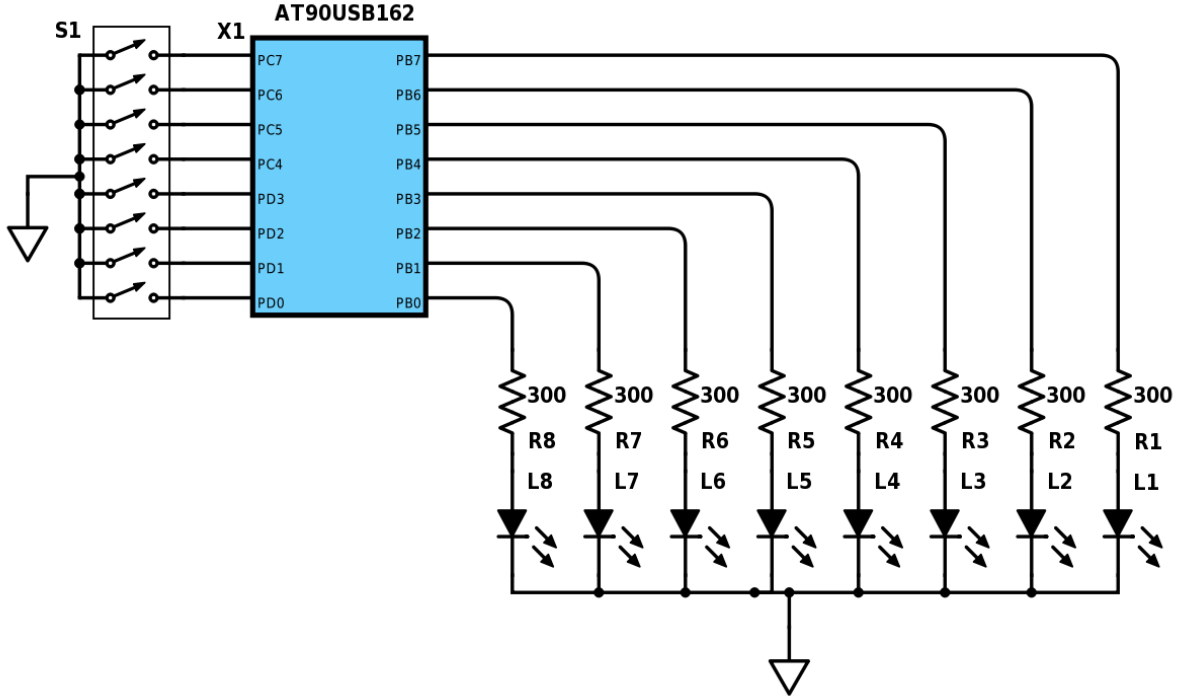


Figure 1: Hardware schematic for Lab 5

As shown in Figure 1, the LEDs are connected to pins PB0-PB7 (all configured as outputs) and the eight switches are connected to pins PD0-PD3 and PC4-PC7 (all configured as inputs with pull-up resistors). Current limiting resistors of 300Ω are used on each output pin to limit the bias current of the LEDs to approximately 10mA each when on (since the forward voltage of the LEDs is about 2V in this case) [1]. When a switch is closed, the input pin is pulled low to ground through the switch; alternatively, if the switch is open, the configured internal pull-up resistors pull the input pin high to 5V. The AVR is also connected to a PC over USB (not shown) to enable serial communication.

2.2 Software

The program `partA.c` starts by performing the necessary initializations for serial over USB as shown in the 'VirtualSerial' demo [2]. Then, the input and output pins of the AVR are configured. Pins are configured as inputs or outputs by setting up the `DDRB`, `DDRC`, and `DDRD` registers [3]. A couple of `#define` statements define `OUTPUTS`, `INPUTS_C`, and `INPUTS_D` to have the values representing the bit positions of the outputs and inputs of each port and make the code more readable. To set up the `DDRB`, `DDRC`, and `DDRD` registers, the bits corresponding to the desired outputs are set, and the bits corresponding to the inputs are cleared. This is done using the following statements:

```
DDRB |= OUTPUTS;    // set output bits of port B
DDRC &= ~INPUTS_C;  // clear input bits of port C
```

```
DDRD &= ~INPUTS_D; // clear input bits of port D
```

Next, the bits of the PORTC and PORTD registers corresponding to the inputs need to be set in order to enable the pull-up resistors. This is done using the following statements:

```
PORTC |= INPUTS_C; // set inputs of port C to pull-up  
PORTD |= INPUTS_D; // set inputs of port D to pull-up
```

Now that all the necessary initializations are completed, the program moves into the main loop, which runs as long as the device is powered. Within this loop, the necessary tasks to establish the USB serial connection are executed as long as device appears as disconnected. Once the device is connected, the main loop constantly repeats the `USB_check()` function, which waits for an input string from the serial terminal and processes the string, either setting LEDs or returning switch or LED values.

The program handles decimal, hexadecimal, or octal integer inputs by checking the characters following the preceding 'L' character in the string. If 'L' is followed by '0x', the program checks where the null character is (denoting the end of string) and converts the characters in between from their hex representation to an integer value by using the `strtol()` function in `stdlib.h`. The same works for octal values, where 'L' is followed by '0'. The program will give an error message if the first character is anything other than 'L' or 'S' (insensitive to case), if the next character after a 'S' is anything but a '?', or if the value following an 'L' is too long.

2.3 Programming and running

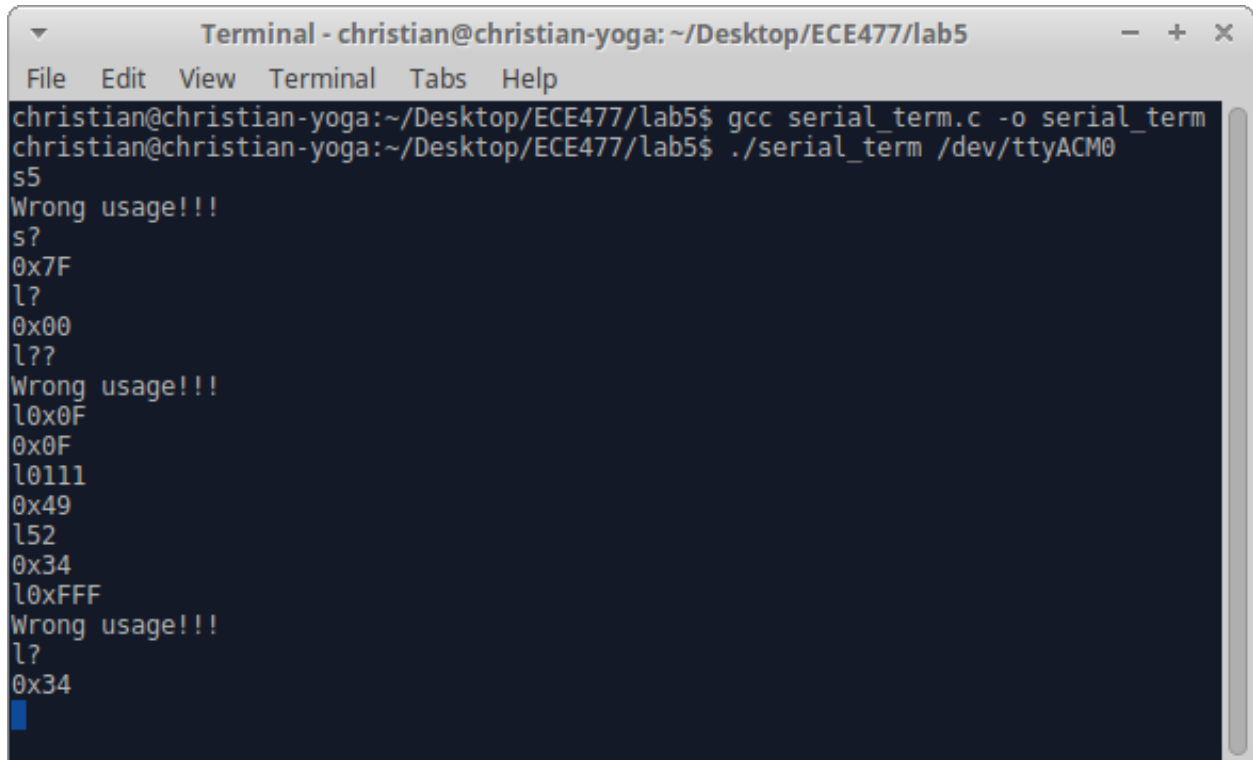
To compile the AVR program mentioned, a makefile is used which specifies the AVR processor and links to the necessary LUFA library files. Once the `make` command is run, a `.hex` file is generated which needs to be programmed to the microcontroller. The microcontroller is programmed using the following commands:

```
make clean // gets rid of old program files  
make // compiles new program  
dfu-programmer at90usb162 erase --force // erases AVR  
dfu-programmer at90usb162 flash partA.hex // flashes AVR
```

Now the AVR will appear as a USB serial device. To communicate with the AVR, the serial terminal program needs to be compiled and run on the PC. The program is compiled using `gcc` and is run with the name of the serial port to be used as an argument. In the case of this lab, the AVR shows up on serial port ACM0, so the serial terminal is compiled and run using the following commands:

```
gcc serial_term.c o serial_term  
./serial_term /dev/ttyACM0
```

From here, the terminal can be used to send and receive lines of text to and from the AVR. Figure 2 shows an example of the usage of part A.



```
christian@christian-yoga:~/Desktop/ECE477/lab5$ gcc serial_term.c -o serial_term
christian@christian-yoga:~/Desktop/ECE477/lab5$ ./serial_term /dev/ttyACM0
s5
Wrong usage!!!
s?
0x7F
l?
0x00
l??
Wrong usage!!!
l0x0F
0x0F
l0111
0x49
l52
0x34
l0xFFFF
Wrong usage!!!
l?
0x34
```

Figure 2: Example usage of Part A

As shown in Figure 2, the program returns the current value of the switches or LEDs if just a '?' is given and writes the given values to LEDs given either decimal, hexadecimal, or octal integers. The program gives an error when the user tries to write a value to a switch or write a value too large to the LEDs.

3 Part B

For Part B, an LED was used to output Morse code information via blinking for letters. The from the PC, the user can input a word up to 20 characters and the LED will respond with the corresponding light flashes in Morse code. This was done using the function `Morse()`. This function uses the `_delay_ms()`, which sets a delay, in ms before executing the next line the code. Once the user has entered in a letter, the `Morse()` will compare the string input to the corresponding letter in the string array `letters[]` and `big_letters[]`, the code then clears and sets `PORTB` depending on the letter being processed. The duration and frequency of blinks was selected from the International Morse code table shown below in Figure 3.

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A	• —	U	• • —
B	— • • •	V	• • • —
C	— • — •	W	• — —
D	— • •	X	— • • —
E	•	Y	— • — —
F	• • — •	Z	— — • •
G	— — • •		
H	• • • •		
I	• •		
J	• — — —		
K	— • —	1	• — — — —
L	• — • •	2	• • — — —
M	— —	3	• • • — —
N	— •	4	• • • • —
O	— — —	5	• • • • •
P	• — — •	6	— • • • •
Q	— — • —	7	— — • • •
R	• — •	8	— — — • •
S	• • •	9	— — — — •
T	—	0	— — — — —

Figure 3: International Morse code table

In the code, a dot is the length of 50ms, while a dash is 150ms. This function handles one character at a time, and then repeats for as many characters in the string.

4 Conclusion

The development of a C program for the AT90USB162 microcontroller that configures the device on a serial port has been described. The goal was to configure the microcontroller such that eight switches connected to input pins on ports C and D and eight LEDs connected to output pins on port B could be interfaced with over serial with a PC. Within the C routine, registers are initialized accordingly for ports B, C, and D, and the main loop of the program waits for input text and returns information to the user through serial.

A C program for part B was developed for the AT90USB162 that accepts a text string from a PC over serial and flashes an LED with the Morse code information corresponding to the text string. The program waits for an input string over serial similar to part A.

A Part A Source Code

Attached: `partA.c`, `serial_term.c`, `makefile`

B Part B Source Code

Attached: `partB.c`, `serial_term.c`, `makefile`

Sources

- [1] LiteOn Inc. *LTL-4223 Datasheet*. Mar. 3, 2017. URL: <http://media.digikey.com/pdf/Data%20Sheets/Lite-On%20PDFs/LTL-4223.pdf>.
- [2] Dean Camera. *LUFA Library*. Mar. 31, 2017. URL: <http://www.fourwalledcubicle.com/LUFA.php>.
- [3] Atmel Corporation. *AT90USB162 Datasheet*. Mar. 3, 2017. URL: <http://www.atmel.com/Images/doc7707.pdf>.
- [4] Wikibooks.org. *Serial Programming/termios*. Apr. 13, 2016. URL: https://en.wikibooks.org/wiki/Serial_Programming/termios.