ECE 477 Lab 4

Christian Knight, Nikko Noble

September 16, 2020

1 Introduction

The purpose of this lab is to get started with using the AT90USB162 microcontroller. Specifically, the goal is to configure the microcontroller such that four switches connected to input pins on port B control LEDs connected to output pins on port B. This is done using a simple C routine that initializes the registers for port B appropriately and then, in an endless loop, checks the state of the input pins connected to switches and sets the corresponding output pins connected to LEDs high or low depending on the state of the switches.

For Part B, a C program was developed that uses the four input switches to select from four different illumination patterns on nine LEDs. The same four LEDs connected to port B from Part A were used, in addition to five more connected to the available I/O pins on port C.

2 Part A

This section describes the development of the C program that configures the AT90USB162 microcontroller to control LEDs connected to output pins on port B using four switches connected to input pins on port B. For this lab, the upper 4 bits of port B (bits 4-7) were used as the inputs from the switches, and the lower 4 bits (bits 0-3) were used as the outputs to the LEDs. The main loop of the program runs as long as the device is powered, and simply copies the state of the input pins to the corresponding output pins to enable or disable the LEDs connected.

2.1 Hardware

The hardware schematic for this lab is shown in Figure 1.

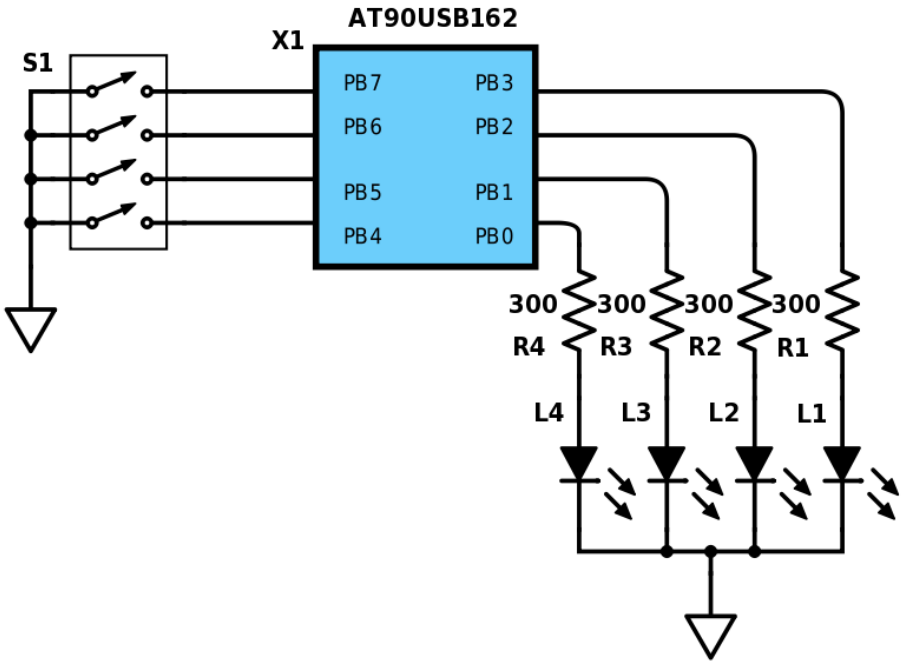

Figure 1: Hardware schematic for Lab 4

As shown in Figure 1, the four switches are connected to pins PB4-PB7 (configured as inputs with pull-up resistors), and the LEDs are connected to pins PB0-PB3 (configured as outputs). 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. For example, if the switch connected to PB4 is open, the input voltage is 5V (high) and the LED connected to PB0 would turn on, and vice versa.

2.2 Software

The program `partA.c` starts by performing the necessary initializations on port B. This includes setting up the DDRB register (which defines pins as inputs or outputs) to hold the value `0x0F`, indicating that the lower four bits are outputs (1's) and the upper four bits are inputs (0's) [2]. A couple of `#define` statements define `INPUTS` and `OUTPUTS` to have the values `0xF0` and `0x0F` respectively, to represent the bit positions of the inputs and outputs and make the code more readable. To set up the DDRB register, 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
DDRB &= ~INPUTS;    // clear input bits
```

Next, the bits of the `PORTB` register corresponding to the inputs need to be set in order to enable the pull-up resistors. This is done using the following statement:

```
PORTB |= INPUTS;    // set inputs 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 value of the inputs are pulled from the `PINB` register, which holds the current input and output values for each pin. Then, the output bits of PORTB are either set or cleared depending on the state of each switch. This is done using the following statements:

```
LEDVALUE = (PINB & INPUTS) >> 4; // get input values
PORTB |= (LEDVALUE & OUTPUTS);  // set LEDs that need to be on
PORTB &= ~(LEDVALUE & OUTPUTS);  // clear LEDs that need to be off
```

2.3 Programming

To compile the program mentioned, `avr-gcc` is used, specifying the processor using the `-mmcu` flag as shown below:

```
avr-gcc -mmcu=at90usb162 partA.c
```

From here, the output file created by `avr-gcc` is converted into a `.hex` file that the microcontroller's bootloader can handle, using `avr-objcopy` as shown below:

```
avr-objcopy -R .eeprom -O ihex a.out partA.hex
```

To program the AT90USB162, the microcontroller must be connected via USB and in bootloader mode, which is accomplished by pulling the RESET pin low while the PD7 pin is also low. `dfu-programmer` utility was used. The device is first erased, and then programmed as shown below:

```
dfu-programmer at90usb162 erase --force
dfu-programmer at90usb162 flash partB.hex
```

Now, the device should be reset (without pulling PD7 low) to run the program.

3 Part B

For Part B, the rest of the available I/O pins on the microcontroller were populated with LEDs and used to flash a total of 9 LEDs (same 4 from Part A on port B and 5 available pins on port C) in different sequences according to the switch positons on the inputs from Part A. To make this work, the function `_delay_ms()` is utilized. `_delay_ms()` creates a delay in the program in milliseconds, the macro `F_CPU` is defined to a constant defining the CPU clock frequency (in Hertz). In the code, four `if` statements will check which switch

is pulled low. This is done by checking the value of `LEDVALUE`. The `if` statement checks if the value of `LEDVALUE` is either a 1, 2, 4, or 8. This condition makes the code only go into the `if` statements if only one input is pulled low. Inside the `if` statement, the bits of the `PORTB` register will be set depending on what LEDs needs to be turned on. Once the LED is on, `_delay_ms()` will create a delay of 150 milliseconds, then clear the LEDs after the delay ends.

4 Conclusion

The development of a simple C program on the AT90USB162 microcontroller has been described. The goal was to configure the microcontroller such that four switches connected to input pins on port B control LEDs connected to output pins on port B. Within the C routine, registers are initialized accordingly for port B, and the main loop of the program checks the state of the input pins connected to switches to set the corresponding output pins connected to LEDs high or low depending on the state of the switches.

A C program was developed for Part B that uses the four input switches to select from four different illumination patterns on nine LEDs on ports B and C. This built upon Part A, and involved utilizing the software delay function in the AVR library. Alternatively, four more LEDs could have been used on the pins that served as inputs from the switches, and the switches could have been configured on the analog inputs of port D instead.

Appendix A: Part A Source Code

```
// partA.c

// Enables or disables LEDs connected to output pins configured on port B
// on the AT90USB162 based on switches connected to input pins configured on
    port B.

// Christian Knight, Nikko Noble
// ECE 477 Lab 4
// 3/3/17

#include <stdio.h>
#include <avr/io.h>
#include <stdlib.h>

#define INPUTS ((1<<DDB7)|(1<<DDB6)|(1<<DDB5)|(1<<DDB4))
#define OUTPUTS ((1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0))

void main(void){
   unsigned char LEDVALUE; // 4 bit value corresponding to LEDs on or off

   // set lower 4 bits of port B as outputs, and the upper 4 as inputs (DDRB =
       0x0F)
   DDRB |= OUTPUTS;
   DDRB &= ~INPUTS;

   // set input bits of port B as pull-up (PORTB |= 0xF0)
   PORTB |= INPUTS;

   while(1){
      LEDVALUE = (PINB & INPUTS) >> 4; // get LEDVALUE based on which switches
          are on
      PORTB |= (LEDVALUE & OUTPUTS); // set LEDs that need to be on
      PORTB &= ~(LEDVALUE & OUTPUTS); // clear LEDs that need to be off
   }
}
```

Appendix B: Part B Souce Code

```
// partB.c

// Displays 4 distinct illumination patterns on 9 LEDs connected to pins
// on ports B and C of the AT90USB162 microcontroller based on positions
// of the input switches on pins of port B.

// Christian Knight, Nikko Noble
// ECE 477 Lab 4
// 3/3/17

#define F_CPU 1000000UL // 1 MHz CPU speed for delay function

#include <stdio.h>
#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>

// positions of each LED
#define LED0 1<<DDB0
#define LED1 1<<DDB1
#define LED2 1<<DDB2
#define LED3 1<<DDB3
#define LED4 1<<DDC2
#define LED5 1<<DDC4
#define LED6 1<<DDC5
#define LED7 1<<DDC6
#define LED8 1<<DDC7

#define INPUTS ((1<<DDB7)|(1<<DDB6)|(1<<DDB5)|(1<<DDB4))
#define C_OUTPUTS (LED8|LED7|LED6|LED5|LED4)
#define B_OUTPUTS (LED3|LED2|LED1|LED0)

#define DELAY 150 // time, in ms, to delay

void toggleLED (unsigned char LED, char PORT); // toggles given LED on given PORT
void setLED (unsigned char LED, char PORT); // sets given LED on given PORT
void clearLED (unsigned char LED, char PORT); // clears given LED on given PORT

void main(void){
   unsigned char LEDVALUE; // 4 bit value corresponding to LEDs on or off

   // set lower 4 bits of port B as outputs, and the upper 4 as inputs (DDRB =
      0x0F)
   DDRB |= B_OUTPUTS;
   DDRB &= ~INPUTS;
```

```
// set available bits of port C as outputs
DDRC |= C_OUTPUTS;

// set input bits of port B as pull-up (PORTB |= 0xF0)
PORTB |= INPUTS;

int i;  // just for indexing

// array containing LED positions
unsigned char LED[9] = {LED0,LED1,LED2,LED3,LED4,LED5,LED6,LED7,LED8};

while(1){
   LEDVALUE = (PINB & INPUTS) >> 4; // get LEDVALUE based on which switch is on

   // no LEDs on
   if (LEDVALUE == 0) {
      PORTB &= ~B_OUTPUTS;
      PORTC &= ~C_OUTPUTS;
   }

   // all LEDs on
   else if (LEDVALUE == 1) {
      PORTB |= B_OUTPUTS;
      PORTC |= C_OUTPUTS;
   }

   // one LED on at a time, right to left
   else if (LEDVALUE == 2){
      for (i = 0; i < 4; i++) {
         toggleLED(LED[i],'B');
         _delay_ms(DELAY);
         toggleLED(LED[i],'B');
         _delay_ms(DELAY);
      }
      for (i = 4; i < 9; i++) {
         toggleLED(LED[i],'C');
         _delay_ms(DELAY);
         toggleLED(LED[i],'C');
         _delay_ms(DELAY);
      }
   }

   // toggle each LED one at a time, right to left
   else if (LEDVALUE == 4){
      for (i = 0; i < 4; i++) {
         toggleLED(LED[i],'B');
```

```
            _delay_ms(DELAY);
        }
        for (i = 4; i < 9; i++) {
            toggleLED(LED[i],'C');
            _delay_ms(DELAY);
        }
    }

    // K.I.T.T. from Knight Rider (right to left to right)
    else if (LEDVALUE == 8){
        toggleLED(LED[0],'B');
        for (i = 1; i < 4; i++) {
            toggleLED(LED[i],'B');
            toggleLED(LED[i-1],'B');
            _delay_ms(DELAY);
        }
        toggleLED(LED[3],'B');
        toggleLED(LED[4],'C');
        _delay_ms(DELAY);
        for (i = 5; i < 9; i++) {
            toggleLED(LED[i],'C');
            toggleLED(LED[i-1],'C');
            _delay_ms(DELAY);
        }
        for (i = 8; i > 4; i--) {
            toggleLED(LED[i-1],'C');
            toggleLED(LED[i],'C');
            _delay_ms(DELAY);
        }
        toggleLED(LED[3],'B');
        toggleLED(LED[4],'C');
        _delay_ms(DELAY);
        for (i = 3; i > 0; i--) {
            toggleLED(LED[i-1],'B');
            toggleLED(LED[i],'B');
            _delay_ms(DELAY);
        }
        toggleLED(LED[0],'B');
    }
  }
}

void toggleLED (unsigned char LED, char PORT) {
  if (PORT == 'B')
    PORTB ^= LED;
  else if (PORT == 'C')
    PORTC ^= LED;
```

```
}
```

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] Atmel Corporation. *AT90USB162 Datasheet*. Mar. 3, 2017. URL: http://www.atmel.com/Images/doc7707.pdf.