Department of Electrical and Computing Engineering

UNIVERSITY OF CONNECTICUT

ECE 3411 Microprocessor Application Lab: Fall 2015

Lab Test VII

There is 3 long programming problems in this test. There are 11 pages in this booklet. Answer each question according to the instructions given.

You have **100 minutes** to answer the questions. Once you are done, you need to show the output to the Instructor or TA and e-mail the code to the TA.

Some questions are harder than others and some questions earn more points than others—you may want to skim all questions before starting.

If you find a question ambiguous, be sure to write down any assumptions you make.
Be neat and legible. If we can't understand your answer, we can't give you credit!

Write your name in the space below. Write your initials at the bottom of each page.

THIS IS AN OPEN BOOK, OPEN NOTES TEST.
YOU CAN USE YOUR LAPTOP BUT PLEASE TURN YOUR NETWORK DEVICES OFF.

Any form of communication with other students is considered cheating and will merit an F as final grade in the course.

Do not write in the boxes below

1. (x/40)	2. (x/40)	3. (x/100)	Total (xx/180)

Name:

Student ID:

1. [40 points]: In this task, you need to implement **Non-blocking SPI**. Write a simple program to test SPI in loopback mode. In particular:

- Configure SPI in Master mode with SPI interrupt enabled.
- Configure Timer1, with compare match interrupt enabled, in CTC mode to overflow after every 100ms.
- In Timer1 ISR, initiate an ADC conversion to read a potentiometers voltage (only upper 8 bits) every 100ms.
- In ADC ISR, once the conversion is complete, initiate a SPI transmission to transmit the byte containing voltage reading over SPI.
- Loopback the transmitted byte by connecting MOSI and MISO pins together.
- Once SPI interrupt triggers, print on LCD the byte value received over SPI.

Implement this system by filling in the gaps in the code layout given below.
Notice that busy waiting on SPIF flag after initiating a SPI transmission is **not allowed**.

The following code snippet provides the necessary layout and definitions.

```
/************ ECE3411 Lab Test 7, Task 2 *************/
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <inttypes.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <string.h>
#include "lcd_lib.h"

// SPI related definitions
#define DDR_SPI     DDRB
#define SPI_SS      2
#define SPI_MOSI    3
#define SPI_MISO    4
#define SPI_SCK     5

// Variables
volatile unsigned int Ain;
volatile uint8_t data_byte;

// LCD Strings
char lcd_buffer[17]; // LCD display buffer
const uint8_t LCD_Master[] PROGMEM = "Master: ";

//---------------------------------------------------------------------
```

Initials:

```
// All initializations
void initialize_all(void)
{
    /* Configure LCD, Timer1, ADC and SPI */
}

//---------------------------------------------------------------------

// Timer 1 Compare Match A ISR (TCNT1 = OCR1A)
ISR (TIMER1_COMPA_vect)
{
    /* Write your code here */
}
//---------------------------------------------------------------------

// ADC ISR
ISR(ADC_vect)
{
    /* Write your code here */
}
//---------------------------------------------------------------------

// SPI ISR
ISR(SPI_STC_vect)
{
    /* Write your code here */
}
//---------------------------------------------------------------------

/* Main Function */
int main(void)
{
    initialize_all();     // Initialize everything
    sei();                // Enable Global Interrupts
    while(1);             // Nothing to do.
}
//---------------------------------------------------------------------
```

Initials:

2. [40 points]: In this task, you need to generate a sawtooth waveform using PWM.
Generate a 64kHz PWM signal at PB2 using `Timer1` such that the duty cycle of the PWM is varied in way that it results in a 10Hz **sawtooth waveform**.

An example of such a PWM signal is shown in Figure 1 where the duty cycle results in a sawtooth waveform.

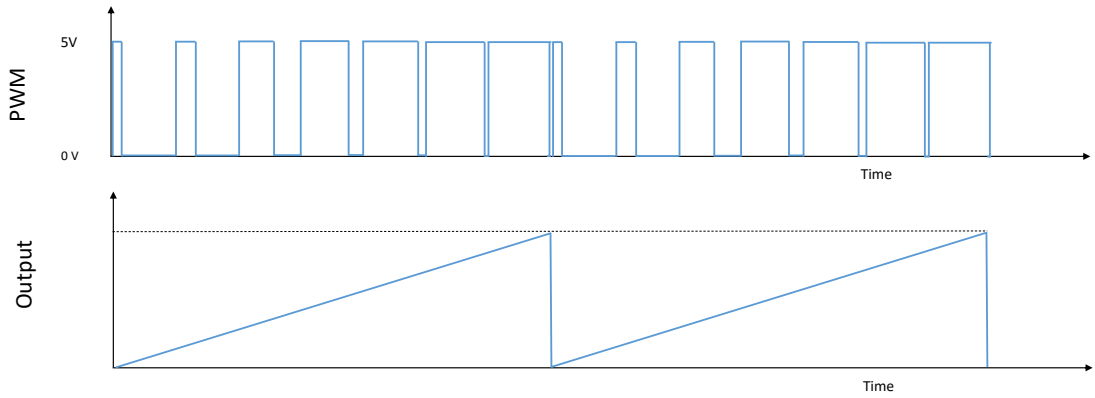

Figure 1: Example PWM Signal.

Connect the PWM signal to a RC low pass filter as shown in Figure 2.

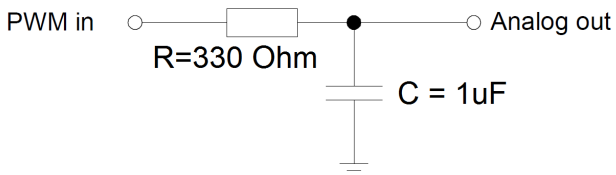

Figure 2: Low Pass Filter.

Connect the output of the low pass filter to an oscilloscope and observe the resulting waveform.

The credit for this task is based on the following two criteria:

a. Correct frequency (i.e. 10Hz) of the resulting sawtooth waveform.

b. Quality of the resulting signal.

Hint: Signal quality depends upon the number of steps taken in one sawtooth waveform cycle to update the PWM duty cycle.

Notice that for this task, `_delay_ms()`/`_delay_us()` function calls are not allowed.

Initials:

3. [100 points]: In this task, we are going to implement a simplified version of Morse Codes for a few English alphabets shown in Table 1. In order to produce an alphabet, the following two conditions must be met:

(a) A particular sequence of SW1 and SW2 button pushes as shown in Table 1.

(b) The push sequence must be completed within a 2 seconds window (starting from the first push).

Table 1: Simplified Morse Code Table.

Alphabet	Button Push Sequence within 2 seconds window.
A	SW1, SW2
B	SW2, SW1, SW1, SW1
C	SW2, SW1, SW2, SW1
D	SW2, SW1, SW1
Invalid	Any other sequence.

The clock frequency ($clk_{I/O}$) is 16MHz.
The switches SW1 and SW2 are connected to PB1 and PB7 of ATmega328P respectively, as shown in the Figure 3. Both SW1 and SW2 need a **debouncing delay** of 4ms.

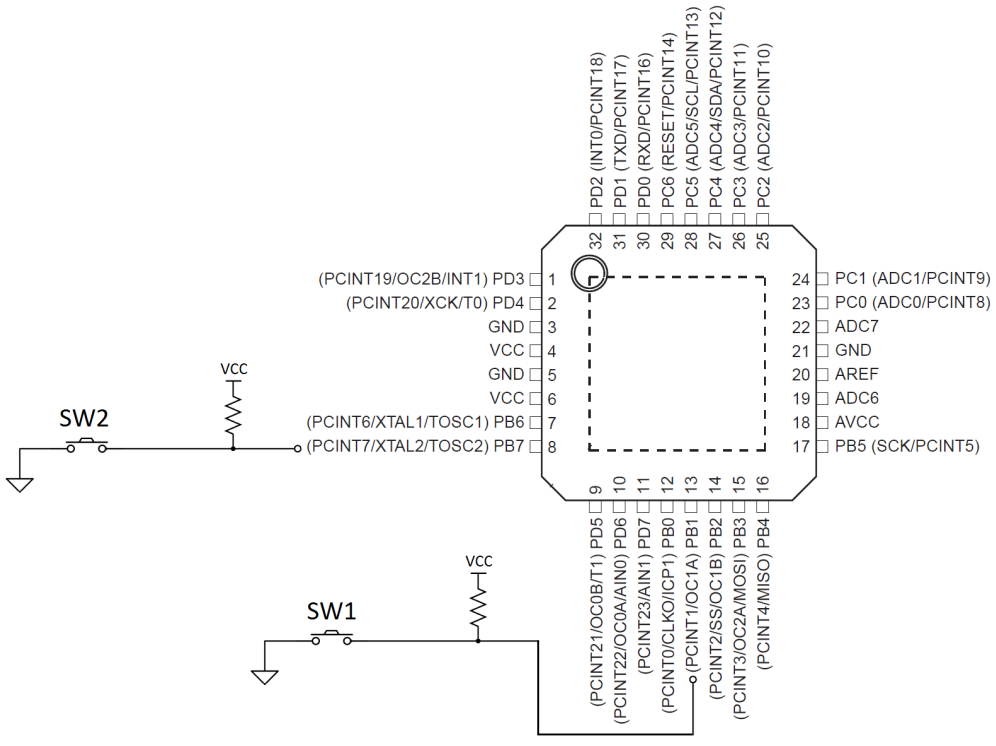

Figure 3: ATmega328P Hardware Configuration.

Implement this system by filling in the gaps in the code layout given below.
Notice that you are **not allowed** to use `_delay_ms()`/`_delay_us()` routines.

Initials:

The following code snippet provides the necessary includes, declarations and definitions.

```
/*********** ECE3411 Lab Test 6, Task 1 ************/
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <inttypes.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>
#include <string.h>
#include "lcd_lib.h"

#define SW1_PRESSED (~PINB & (1<<PINB1))
#define SW2_PRESSED (~PINB & (1<<PINB7))

// Flag Variables
volatile uint8_t DebounceFlag1;
volatile uint8_t DebounceFlag2;
volatile uint8_t index1;
volatile uint8_t index2;

// Push Sequence Encoding
volatile uint8_t encodings[4][5] =
    {
        {1, 2, 0, 0, 0},    // A's encoding is accessed as encodings[0]
        {2, 1, 1, 1, 0},    // B's encoding is accessed as encodings[1]
        {2, 1, 2, 1, 0},    // C's encoding is accessed as encodings[2]
        {2, 1, 1, 0, 0}     // D's encoding is accessed as encodings[3]
    };

// Index to Character Mapping
volatile uint8_t mapping[4] = {'A', 'B', 'C', 'D'};

// Save the Button Pushes in this array
volatile uint8_t sequence[5];

//-----------------------------------------------------------------------

/* Main Function */
int main(void)
{
    initialize_all();     // Initialize everything
    sei();                // Enable Global Interrupts
    while(1);             // Nothing to do.

} /* End of main() */
//-----------------------------------------------------------------------
```

Initials:

A. Initialization: (25 points)
Complete the function `initialize_all(void)` as instructed below:

```
/* Initialization function */
void initialize_all(void)
{
    // Initializing the LCD.
    initialize_LCD();
    LCDcursorOFF();
    LCDclr();

    // Initializing the flag variables
    DebounceFlag1 = DebounceFlag2 = 0;
    index1 = index2 = 0;

    // Enable Pin Change Interrupts for PB1 and PB7 here




    // Setup Timer0 in CTC mode to generate Compare Match Interrupt A every 4ms
    // Set Timer0 Prescaler in 'start_timer0()' function on the next page.









    // Setup Timer1 in CTC mode to generate Compare Match Interrupt A every 2s









} /* End of initialize_all() */
```

Initials:

B. Timer0 Prescaler & Pin Change Interrupt ISR: (25 points)
Complete the function `start_timer0()` and `ISR(PCINT0_vect)` as instructed below:

```
/* Starts Timer0 */
void start_timer0()
{
    // Select and set appropriate prescaler for Timer0 here



}

//-------------------------------------------------------------------------

/* Stops Timer0 */
void stop_timer0()
{
    TCCR0B = 0x00;        // Prescaler = NONE
    TCNT0 = 0;            // Resets the timer
}

//-------------------------------------------------------------------------

/* Pin Change Interrupt 0 ISR */
ISR(PCINT0_vect)
{
    // Disable the Pin Change Interrupt 0 here


    // Update any flags etc.









    // Start Timer0 to count Debounce Delay
    start_timer0();

} /* end of ISR(PCINT0_vect) */
```

Initials:

C. Timer0 Compare Match ISR: (25 points)
Complete `ISR(TIMER0_COMPA_vect)` as instructed below:

```
/* Timer0 Compare Match A ISR */
ISR(TIMER0_COMPA_vect)
{
    // Stopping Timer0
    stop_timer0();

    // Read and record the button push in 'sequence' array









    // Re-enable Pin Change Interrupt
    PCICR |= (1<<PCIE0);

} /* end of ISR(TIMER0_COMPA_vect) */
```

Initials:

D. Timer1 Compare Match ISR: (25 points)
Complete `ISR(TIMER1_COMPA_vect)` as instructed below:

```
/* Timer1 Compare Match A ISR */
ISR (TIMER1_COMPA_vect)
{
    // Print the Alphabet corresponding to the received push sequence on LCD
    // Print 'I' if an invalid sequence is received.

    // Clear the received sequence buffer
    for(index2=0; index2<4; index2++)
        sequence[index2] = 0;

    // Reset index2
    index2 = 0;

} /* end of ISR(TIMER1_COMPA_vect) */
// ------------------------------------------------------------------- //
```

Initials:

End of Quiz

Please double check that you wrote your name on the front of the quiz.

Initials: