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 6 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/20)	2. (x/40)	3. (x/40)	Total (xx/100)

Name:

Student ID:

1. [20 points]: In this task, you need to implement *non-preemptive* version of **Shortest Remaining Time First** scheduling for a given set of periodic tasks. The task specifications are given in the following table.

Table 1: Task Specifications

Task	Required CPU Time (ms)	Task Period (ms)
`task_0()`	400	1000
`task_1()`	200	500
`task_2()`	100	600

You are provided with a C source file named `scheduling_SW.c` which provides a framework to run these tasks.

You need to implement the function called `SRTF_scheduler()` such that it schedules the tasks w.r.t. *shortest remaining time first* scheduling. Once you have implemented the scheduler, connect a UART terminal to your MCU and see the resulting order of the tasks.

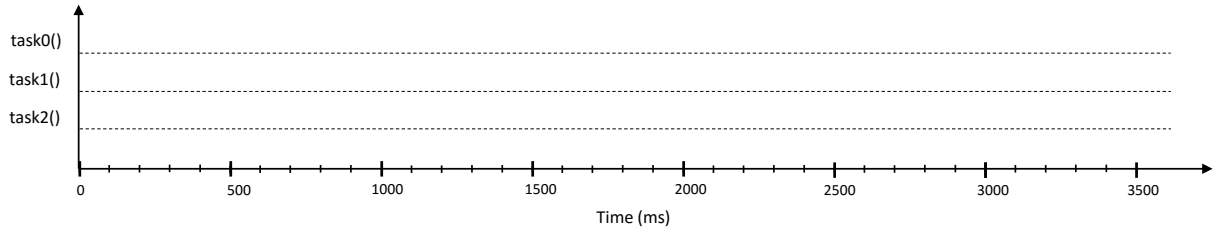
Write the resulting pattern of the tasks in the space given below. Just write the task numbers to show the order in which they are scheduled.

Table 2: Scheduler Output

No.	1	2	3	4	5	6	7
Task #							
No.	8	9	10	11	12	13	14
Task #							

Notice that if your scheduler is not working properly, at least one of the task would miss its deadline and a message will be printed on UART followed by suspension of the program.

It would be easier for you to program/verify your scheduler's behavior by plotting the intended behavior on a paper first. For this purpose, you may use the space provided below.



Initials:

2. [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:

3. [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.

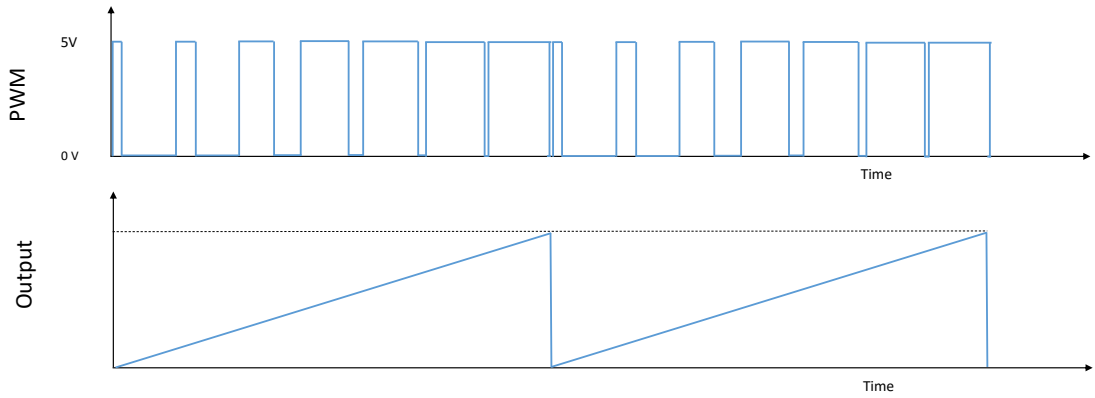


Figure 1: Example PWM Signal.

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

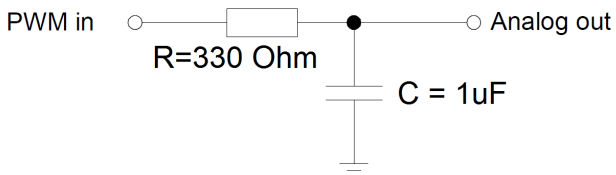


Figure 2: Low Pass Filter.

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:

End of Quiz

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

Initials: