# CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA
# COLLEGE OF ENGINEERING

**ECE 3301L Spring 2019 Microcontroller Lab Felix Pinai**
**Session 2**

## LAB 11: More implementations of I2C bus and use of external interrupts

### Task #1: Change from Hardware I2C to Software I2C

Lab 10 dealt with the concept of the I2C bus using the hardware technique whereas all the data bit transmissions are performed by internal hardware inside the microcontroller. This method allows faster data transmissions but the two SCL and SDA pins must be dedicated. In addition the PIC 18 microcontroller that we are using merges the two I2C and SPI functions into the same pins thus allowing the use of only one function and not both simultaneously. If we want to use both SPI and I2C, we will need to switch from the use of hardware I2C to software I2C. This new technique will use the concept of software bit banging to perform all the data bit transmission. For sure this technique will not have the same performance speed as hardware I2c but it does allow the microcontroller to communicate with any I2C device. Since bit banging can be applied to any pin, software I2C allows the flexibility to choose any GPIO pin for the implementation.

The file 'softi2c.inc' sent on last week's email contains all the library functions needed for the software I2C implementation. Next, let us choose two different pins for the signal SCL and SDA. For example, let us physically move SCL to PORT B bit 3 and SDA to PORT B bit 4.

Next, take the I2C implementation from Lab #10 and locate the line showing '#include 'i2c.inc'. Replace that line with the following lines:

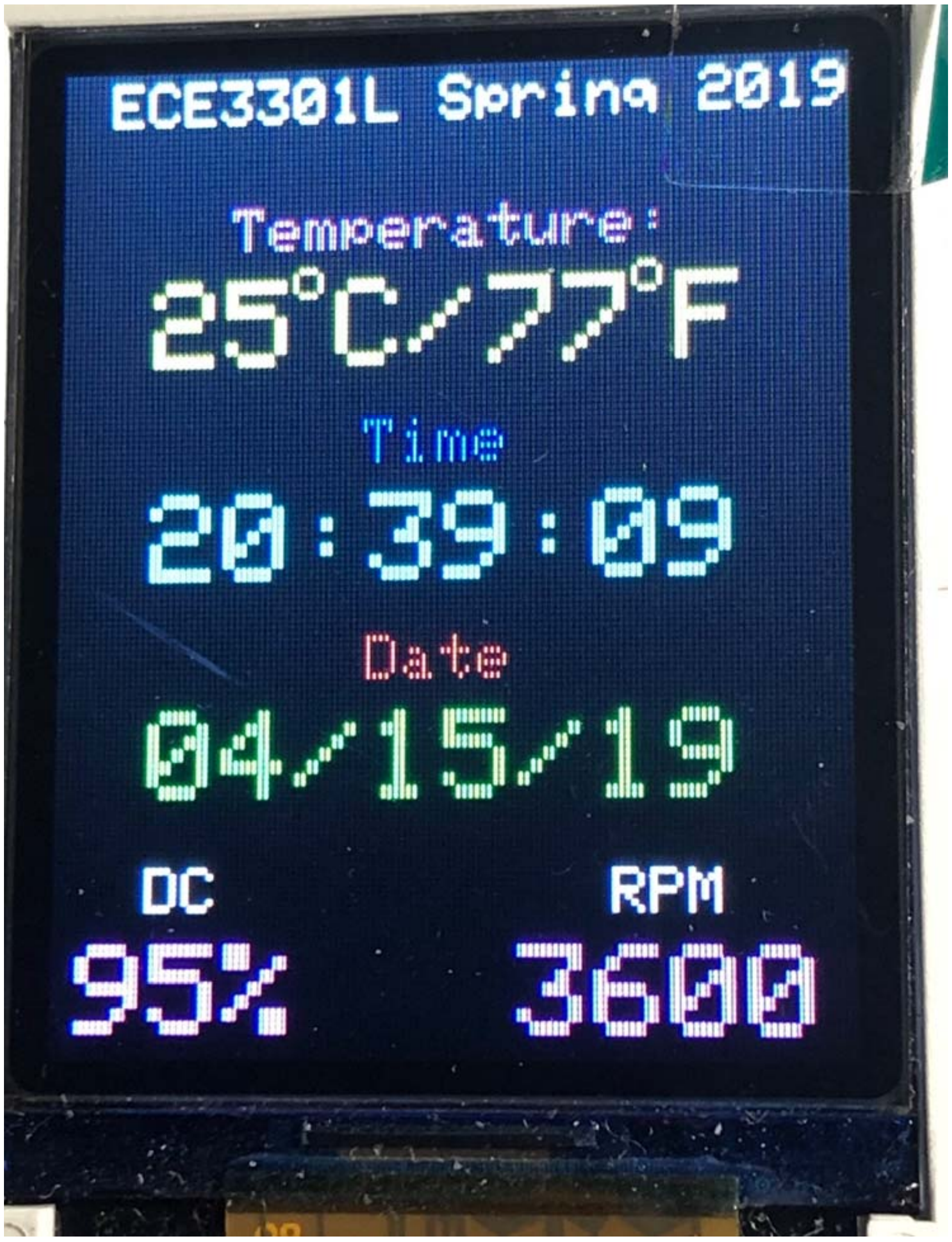
```
#define     SCL_PIN     PORTBbits.RB3
#define     SCL_DIR     TRISBbits.RB3
#define     SDA_PIN     PORTBbits.RB4
#define     SDA_DIR     TRISBbits.RB4
#include "softi2c.inc"
```

When done, recompile the program and check through TeraTerm that the output on the screen shows the same data as on Lab#10.

### Task #2:

Upon the successful completion of Task #1, we can now free up the SCL pin so that it can be used as SCK pin for the SPI bus. We can reintroduce the TFT panel into the design.

Take the code that your team has successfully implemented on Lab #10 and merge with the provided 'Lab11_CODE_Part_2.c' to come up with a new design that displays the Time, Date and temperature on the TFT panel. Place the 'Update_TFT_Screen()' call after your 'printf' statement to do the update the data on the TFT panel. Review the content of the provided 'Update_TFT_Screen()' function. Only two lines of codes were supplied as examples to update the temperature in degrees C. Add the codes for the remaining fields. There are two other fields on the screen that are not changed for this task. They are for the 'duty_cycle' and for the 'RPM'. They will be handled on Task #3 below.



**Task #3:**

On the schematic shown, there are two push-button switches connected to Port B bit 0 and bit 1. These switches are connected as inputs for the external interrupts INT and INT1. When any of those inputs transitions from low to high or from high to low (programmable), an interrupt will be generated to the processor. We are going to use them as controls to change the speed on the fan as we have done on Lab #8. Instead of using the potentiometer on that lab, we are going to use the two push-button switches as the UP and DOWN buttons to increase or decrease the duty cycle of the fan.

Here are some tasks that you will need to take care:

Main program:

a) Initialization:

1) Clear the interrupt flag for the INT0, and INT1 in the INTCON and INTCON3 registers:

```
INTCONbits.INT0IF = 0;              // INT0 IF is in INTCON
INTCON3bits.INT1IF = 0;             // INT1 IF is in INTCON3
```

2) Set the interrupt enable for the INT0,  and INT1 in the INTCON and INTCON3 registers:

```
INTCONbits.INT0IE = 1;              // INT0 IE is in INTCON
INTCON3bits.INT1IE = 1;             // INT1 IE is in INTCON3
```

3) Program the type of edge interrupt required for INT0, and INT1 in the INTCON and INTCON3 registers:

```
INTCON2bits.INTEDG0 = 0;            // Edge programming for INT0, INT1 and
INTCON2bits.INTEDG1= 0              // INT2 are in INTCON2
```

4) Enable the PEIE and GIE bits inside the INTCON register

b) You will need to add to the 'interrupt high_priority chk_isr(void)' function the following handling :

1) Check if INT0IF flag is set. If yes, jump to an INT0 Interrupt Service Routine INT0_ISR() to take care of INT0 interrupt.

2) Check if INT1IF flag is set. If yes, jump to an INT1 Interrupt Service Routine INT1_ISR() to take care of INT1 interrupt.

c) You will need to write the Interrupt Service Routine INT0_ISR() routine where the interrupt flag INT0 is clear and then the software variable 'SWUP_flag' to 1. The same task is needed to handle the interrupt INT1 inside the routine INT1_ISR() with the clearing of the interrupt flag for INT1 and the setting of a software flag 'SWDN_flag'.

d) In the main routine and inside the while (1) loop, do a test for the 'SWUP_flag' variable to be set to 1. Once it is detected as 1, clear 'SWUP_flag' to 0 and update up the value of duty cycle by adding 5 to a variable 'duty_cycle'. As done before, when 'duty_cycle' reaches 100, it should be reset to 0. Beside the test for the 'SWUP_flag', you should also test for the 'SWDN_flag' and perform the same handling but instead of adding 5 to the 'duty_cycle' do the subtraction of 5 from that variable and make sure that if it reaches below 0, do change it to 95.

After the duty cycle has been updated, call the routine 'do_update_pwm (char duty_cycle)' developed in a previous lab to vary the speed of the fan.

The value of the 'duty_cycle' variable must be displayed on the TFT screen as well as the value of the RPM. See the picture below for a typical screenshot of the entire screen during operation.

Make sure to move over the code implemented on Lab #8 into this lab to do the pwm control and the speed measurements. Here are the functions needed:

```
void do_update_pwm(char);
int get_RPS(void);
void delay_500ms(void);
```