Rules for ECE3301L Lab Report

1) **Lab Reports:**
    a. When a lab is completed and successfully demonstrated, a report is to be generated.
    b. The report is to be submitted the following week after the lab is checked out. If the team is late to successfully demonstrate the lab, no lab report is to be sent out. Only when the lab has passed, then the team is to send out the report
    c. The report is composed of the following elements:
        i. The full listing of the source code. If the code is written in 'c' language, then the source code with the '.c' or the '.h' extension is to be sent out. If the code is in assembly, do send out the '.asm' file.
            1. Do not send out all the other files generated by the compiler or the assembler. Those additional files can be from some of the subfolders like 'debug', 'build', etc.
            2. Unless otherwise specified by the instructor, send out all the required source code files worked during the lab. For example, if the lab has five parts and each part requires a source code to be generated, then send out the associated file of each part. However, in some circumstances, the instructor might instruct to send out only a specific source file of a part of the lab while skipping the codes for the other parts.
            3. Do not change the nature of the source code by placing its contents into a general word or text or pdf file. Just the raw source code is needed.
            4. Each source code must edit to contain comments of the code developed. The comments are to explain the purpose of each line of the code.
            5. The format of the commenting process will be explained on the next section.

        ii. On a separate text or word file, put a summary what has been done in the lab like:
            1. Objective of the lab – what was the purpose of the lab
            2. Summary of the main parts of the lab – what are the important aspects of the lab being worked on. What does the student learn from each part.
            3. Any supporting result obtained in the lab should be added in this section. Excel sheet, data tables, scope capture, logic analyzer capture
            4. Final conclusion/remark/observation.

d. When a report is turned in via email:
    i. If possible, put all the files into a compressed (zip) file to minimize the file
    ii. Email the compressed file to the instructor by placing on the subject line the following information: **Session number followed by Table # followed by Lab #**

        For example: Session #1/Table #1/ Lab #1

    iii. Attached the compressed file into the email. Be aware that sometimes your compressed file might be rejected by your email spam/virus detection. If that happens, do not send the compressed files bu d

2) Commenting

a) Use '//' to start a comment line under a '.c' file. Use ';' for commenting under the '.asm' file
b) Place a general comment line for each routine describing what is the function of the routine Create a box around the comment to indicate that this is a general comment:

```
/*************************************************************/
/*                    This is a general box                  */
/*************************************************************/
```

c) Add a comment line on each relevant line of code. Make sure that all the comment lines are lined up under a fixed column and this column should be lined throughout the entire program (if possible). See the sample program below:

```
unsigned int Get_Full_ADC(void)
{
int result;
    ADCON0bits.GO=1;                        // Start Conversion
    while(ADCON0bits.DONE==1);              // Wait for conversion to be completed (DONE=0)
    result = (ADRESH * 0x100) + ADRESL;     // Combine result of upper byte and lower byte into
    return result;                          // return the most significant 8- bits of the result.
}
```

3) Variable definitions

The definition of the variables must be done right at the beginning of the routine. See the line marked in red above.

4) Tabbing

a) Always tab in right below a '{'. I prefer that this '{' should be on the start of the next line.
b) Each '}' should line up with the associated '{'
c) Don't tab when a variable name is defined. See the location on the variable 'result' above.

5) **Prototyping**

To make sure that the program can be properly compiled, it is a good practice to prototype each function when they are first defined. Once the function has been completely coded, you will need to copy the name of that function and paste it at the beginning of the program. If the function has parameters, then keep the list of those parameters (inside the parentheses) preserved with the types of each parameter while the name of each parameter can be omitted.

6) **Program Start Code**

To make sure that your program can properly run, each code must have the following lines at the start of the program:

```
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include <math.h>
#include <p18f4620.h>
#include <usart.h>

#pragma config OSC          =  INTIO67
#pragma config BOREN        =  OFF
#pragma config WDT          =  OFF
#pragma config LVP           =   OFF
```

7) **Uart.h program linked**

Make sure that the 'Link in Peripheral Library' option is set in the linker section. See instructions in Lab #1.

8) **Program properly organized in sub folders for each lab**

As described in Lab#1, each sub part within a lab should be developed using a separated sub-folder of that lab's folder. Make sure to follow the same procedure for every lab.

9) **Here is a typical code that your program should have:**

```
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include <math.h>
#include <p18f4620.h>
#include <usart.h>
```

```
#pragma config OSC              =   INTIO67
#pragma config BOREN            =   OFF
#pragma config WDT              =   OFF
#pragma config LVP              =   OFF


        /**********************************************************/
        /*              Prototype Area                            */
        /**********************************************************/

void Init_ADC(void);                        // Initialization of ADC
unsigned  int Get_Full_ADC(void);           // Get ADC measurements
void Flash_LED(unsigned int);               // Flash the LED

        /**********************************************************/
        /*              Get_Full_ADC(void): Function to read ADC  */
        /**********************************************************/

unsigned int Get_Full_ADC(void)
{
int result;
   ADCON0bits.GO=1;                         // Start Conversion
   while(ADCON0bits.DONE==1);               // Wait for conversion to be completed (DONE=0)
   result = (ADRESH * 0x100) + ADRESL;      // Combine result of upper byte and lower byte into
   return result;                           // return the most significant 8- bits of the result.
}

        /**********************************************************/
        /*              Init_ADC() : Function to intializae ADC   */
        /**********************************************************/

void Init_ADC(void)
{
   ADCON0=0x01;                             // select channel AN0, and turn on the ADDC subsystem
   ADCON1=0x0E;                             // set pin 2 as analog signal, VDD-VSS as reference voltage
                                            // and right justify the result
   ADCON2=0xA9;                             // Set the bit conversion time (TAD) and acquisition time
}
```

```
                    /*********************************************************/
                    /*            Flash_LED : Function to alternatively flash the LED            */
                    /*********************************************************/

void Flash_LED(unsigned int ADC_result)
{
unsigned int counter1, counter2;

    LATB = 0x0A;                                            // First output 0x0A to PORTB
    for (counter2=delay; counter2>0; --counter2)            // Start a loop to generate a delay routine – outer loop
   {
       for (counter1=ADC_result ; counter1>0; -- counter1);
   }                                                        // Inner loop

   LATB = 0x05;                                             // Now output 0x05 to PORTB
   for (counter2=delay; counter2>0; --counter2)             // Start a loop to generate a delay routine – outer loop
   {
       for (counter1=ADC_result ; counter1>0; -- counter1);
   }                                                        // Inner loop
}
```