Part I: (72 pts)

a. (5 pts) Write C code that configures PORTB for the IO shown in the figure for problem (a) on the Figure sheet. The internal weak pullup must be enabled. Do not assume any default bit values.

```c
RBPU = 0; TRISB1 = 1; TRISB2 = 1; TRISB4 = 0;
```

b. (15 pts) Assuming the IO configuration of the previous problem, write a `while(1){}` loop that implements the LED/Switch IO state machine shown for problem (b) in the figures. Either use a `switch()` statement approach or a `if-then-else` approach. Assume you have available the `DelayMs()` function for blinking the LED; you do NOT have to include debounce delays for the switch input.

```c
#define START_STATE 0
#define ON_STATE 1
#define BLINK_STATE 2

// This code sits inside main() and assume configuration from problem 1a.

unsigned char state = START_STATE;
while (1) {
    switch (state) {
    case START_STATE :
        RB4 = 0;  // LED off
        while (RB2); // Wait until a press occurs
        state = ON_STATE;
        break;
    case ON_STATE :
        LB4 = 1; // Turn on LED
        while (!RB2); //wait for release
        if (RB1) state = BLINK_STATE;
        else state = BLINK_STATE;
        break;
    case BLINK_STATE :
        while(RB1) { //while released
            LA4 = !LA4;  // Toggle LED for blinking
            if (RB1) DelayMs(50); // Fast blink
            else DelayMs(100); // Slow blink
        }
        while(RB1) { //while pressed
            LA4 = !LA4;  // Toggle LED for blinking
            if (RB1) DelayMs(50); // Fast blink
            else DelayMs(100); // Slow blink
        }
        //next state after press &release
        state = START_STATE;    break;
    }
}
```
c. (20 pts) For the switch configuration shown in Figure (a), implement a program that
determines if the switch bounces for a single press and release of the pushbutton. This
can be done by determining if more than one rising edge occurs for a single press and
release of the pushbutton. Divide your solution into two code segments -- an ISR, and

d main() code. In the ISR, you should increment a variable named count each time a rising
edge occurs on RB2, and also set a semaphore variable that indicates the interrupt
occurred. In the while(1) loop of main(), you should clear count and the semaphore
variable, then wait for the semaphore variable to be set, then delay long enough using

\textit{DelayMS()} to debounce the switch. If the count variable is greater than 1, then switch
bounce occurred (print out a message in this case).

1. (13 pts) ISR code

\begin{verbatim}
unsigned char count = 0;
unsigned char gotInt = 0;

void interrupt myISR()
{
    if (INT2IF)
    {
        INT2IF = 0; // Clear interrupt
        gotInt = 1; // semaphore
        count++; // Increment number of edges found
    }
}
\end{verbatim}

2. (7 pts) main() code, be sure to configure and enable your interrupt.

\begin{verbatim}
main()
{
    IPEN = 0; // Turn off interrupt priority system
    INT2IF = 0; // Clear any pending pushbutton presses
    INT2IE = 1; // Enable pushbutton interrupt
    INTEDG2 = 1; // Look for a rising edge
    GIE = 1; // Enable interrupts
    while (1)
    {
        // Clear count and gotInt semaphore
        count = 0;
gotInt = 0;
        // Wait for a rising edge
        while (gotInt == 0);
        // Debounce delay
        DelayMs(30);
        // See if bounces occurred
        if (count > 1)
            printf("Saw \%d bounces!", count - 1); pcrlf();
    }
}\end{verbatim}
d. (7 pts) Assume an asynchronous serial channel with a data format of 1 start bit, 8 data bits, and 3 stop bits between characters. If I wanted to guarantee that ten characters would be transmitted in 5 ms, what is the MINIMUM baud rate I could use from the standard baud rates of 4800, 9600, 19200, 38400, 57600, 76800 or 115200? You must show your WORK in order to get any credit for this problem. Assume the receiver accepts data as fast as I transmit it.

Writing as an equation, find \( x \) in \((1 + 8 + 3)\) bits \(\times\) 10 bytes \(\div\) byte \(\times\) 0.005 s \(\div\) bit \(\times\) s. Solving gives \( x = 24000 \) baud; therefore a speed of 38400 baud will send the desired 10 characters in 5 ms, while dropping the speed to 19200 would send less than 10 characters in 5 ms.

OR:

\[
\frac{(10 \text{ bytes} \times 12 \text{ bits})}{0.005 \text{ s}} = \text{minimum baud rate} = 24000 \text{ minimum baud rate}
\]

So 19200 is too slow, next highest baud rate is 38400

e. (7 pts) Write C code that implements the `void putch(char c)` function (transmits one character to the serial port). No interrupts are enabled.

```c
void putch(char c)
{
    while (!TXIF); // Wait for the transmit buffer to be empty
    TXREG = c;     // Send character
}
```

f. (9 pts) Assume the definitions of a circular buffer that we have used in lab (i.e, the head pointer is used to place data into the buffer, the tail pointer is used to take data out of the buffer, the buffer is empty when head is equal to tail, and that pointers are incremented and wrapped before used to access the buffer).

f1. From figure F, how many characters are currently available in the buffer? (this is not the total number of locations in the buffer)

   __________ 6 _________

f2. From figure F, what character is returned if the buffer is read?

   ________ C ________

f3. From figure F, what location is modified if one character is written to the buffer?

   _________ 3 _________
g. (9 pts) Given the code below and Figure (g), answer the questions – assume that the switches do NOT bounce.

**g.1** Assume the PIC is powered on, and then Button A is pressed and released 3 times. What is the value of `count` after this?

The first release of A disables A interrupt and increments count to 1. Further presses of A therefore cause no interrupts, leaving a final value of `count = 1`.

**g.2** Assume the PIC is powered on, and then Button A is pressed and released, then Button B is pressed and released, then Button A is pressed and held down – at this point, what is the value of `count`?

The release of A disables A interrupts and enables B interrupts while incrementing count to 1. The release of B disables B interrupts, enables A interrupt, and increments count to 2. The second press and hold of A produces no results, since A’s interrupt only occurs on a release, leaving a final value of `count = 2`.

**g.3** Assume the PIC is then powered on, and Button B is pressed and released 3 times. What is the value of `count` after this?

Interrupts for B are disabled, so presses cause no action. The final value is `count = 0`.

```c
interrupt isr() {
    if (INT1IF) {
        INT1IF = 0; INT1IE = 0; INT2IE = 1;
        count++;
    }
    if (INT2IF) {
        INT2IF = 0; INT2IE = 0; INT1IE = 1;
        count++;
    }
}
main() {
    RBPU = 0; TRISB = 0xFF;
    IPEN = 0; INTEDG1 = 1; INTEDG2 = 1;
    INT1IF = 0; INT2IF = 0;
    INT1IE = 1; INT2IE = 0;
    count = 0;
    PEIE = 1; GIE = 1;
    while (1) {
    }
}
```
Part II: (28 pts) Answer 7 out of the next 9 questions. Cross out the 2 questions that you do not want graded. Each question is worth 4 pts.

1. Joe Schmoe from Ole Miss changes his asynchronous port so that instead of sending 1 start bit + 8 data bits + 1 stop bit, it now sends 1 start bit + 128 data bits + 1 stop bit because he figures that this is more efficient. However, he does not receive all of the data bits correctly. Why is this?

   In async communication, no clock is sent with the data. Even when both sender and receiver agree on the baud rate, small differences in the clocks used to sample send and receive the data add up as more bits are sent between the start and stop bits. This cumulative error can cause the receiver to sample the serial input in the wrong place, causing receive error.

2. Draw a picture that shows how an open-drain output differs from a normal CMOS output.

   ![Open drain vs Normal output diagram](image)

3. Assume a low-true pushbutton is connected to RB0 and that the INT0 interrupt is initially configured as a falling edge triggered interrupt and is enabled. Fill in the ISR below such that EACH edge of a push and release is detected, and that after 10 edge detections the interrupt is disabled.

   ```c
   void interrupt myisr()
   {
       if (INT0IF)
       {
           INT0IF = 0; // Clear interrupt
           INTEDG0 = !INTEDG0; // Switch edge for next int
           count++;
           if (count == 10) // Have we seen 10 cycles?
               INT0IE = 0; // Disable interrupt
       }
   }
   ```
4. How many COMPLETE characters be received into the PIC USART before the OERR bit is set? Give the exact definition of when a framing error is detected.

The receipt of the third character causes an overflow to occur, setting OERR. A framing error occurs when the stop bit is received as a 0.

5. In the code below, what will happen assuming the standard PIC18 setup that you have been using in lab?

```c
main() {
    serial_init(95,1); // 19200 in HSPLL mode, crystal = 7.3728 MHz
    printf("Howdy YALL!!!");pcrlf();
    if (!TO) {
        TO = 1; SWDTEN = 0;
    }
    SWDTEN = 1;
    while (1) {
        // do nothing
    }//end while()
} //end main()
```

The string “Howdy YALL!!!” will be printed approximately every 2.3 s. The program begins with TO = 1, skipping the if statement. The watchdog timer is enabled, cause a reset of the chip after 2.3 s. As a result, TO = 0, causing the if statement to reset TO to 1 and disable the watchdog timer. However, the watchdog timer is re-enabled after the if statement, so the same cycle then repeats.

6. Assume that the PIC18 TX pin is connected to the T1in pin of the RS232 transceiver on the right. What voltage transition would you expect to see on the T1out pin (from “??V” to “??V”) when a start bit is sent? What voltage transition would you expect to see on the T1in pin (from “??V” to “??V”) when a start bit is sent?

| Idle state: TX = logic 1      | T1in = +5V | T1out = –7V |
| Start bit: TX = logic 0       | T1in = 0V  | T1out = +7V |

Therefore, T1out sees a rising edge, from –7V to +7V. Likewise, T1in sees a falling edge, from +5V to 0V.
7. What is the principle use of “sleep mode” in a microprocessor? In the PIC, how do I cause the PIC to go to sleep?

Sleep mode is used to save power by stopping the microprocessor’s clocks. On the PIC, the assembly language instruction ‘sleep’ is used to put the PIC to sleep.

8. Assume that the MSB of the 8-bit value 0x81 is a parity bit for the remaining seven bits. Is this even or odd parity? If the lower seven bits (0x01) were actually received as “0x04”, would the parity bit detect this transmission error? (Explain your answer!)

In binary, 0x81 = 1000 0001. There is a single ‘1’ bit in the lower seven bits, making this **even parity**, since the overall 8 bit character has an even number (2) of ‘1’ bits. If the lower seven bits were received as 0x04 = 0000 0100, then there is still a single ‘1’ bit on, making the parity bit of ‘1’ sent still correct. Therefore, **no error is detected**.

9. When a PIC18 interrupt occurs, what registers are automatically saved by the PIC18 before the ISR is entered? For a high priority interrupt, the PIC also clears the GIE bit to ‘0’ before the ISR is entered; why is this done?

The STATUS, BSR, and W registers are automatically written to shadow registers when an interrupt occurs. Clearing the GIE bit prevents other high priority interrupts from interrupting while in the high priority interrupt handler executes, which would overwrite the saved registers and cause problems when the handler tries to restore the overwritten values.
Problem (a)

Assume internal weak pullup enabled.

Problem (b)

LED Off

pressed?

yes  ➔  RB2

LED ON

released?

no  ➔  RB2

yes  ➔  RB1?

0  ➔  1

LED BLINK

BLINK

Blink fast if RB1 is HIGH, else blink slow.

press & release?

no  ➔  RB2

yes  ➔  RB2

START

ON