- **Set #6** (Due Mon Nov 28)

1) SPI Enhanced buffer mode
In lab, in order to get back-to-back SPI transmissions, we polled for TBE following the first write to SPIBUF before writing the second byte, even in part 2 where we used interrupts. Another way to accomplish this would be to enable the Enhanced buffer mode and just write the first and second bytes, one after the other. One difference between using standard mode and Enhanced buffer mode is that we need to choose appropriate values for SRXISEL and STXISEL. Assuming we only want to make the change suggested above so we don't have to poll for TBE, and that we still want to handle received characters one at a time, **what value should we set SRXISEL to**?

We will want to set the SRXISEL to 01 so that an interrupt is generated when the buffer is not empty. This will make it so that as soon as we receive a character we handle it therefore we are handling the characters one at a time.

2) Enhanced PLL

a) Assuming we have a 12 MHz external crystal connected to OSC1 and OSC2, what settings will be needed on PLLIDIV, PLLMULT, PLLRANGE and PLLODIV to get FPLL as close as possible (either above or below) to **80 MHz**. What frequency do we get to? Make sure to observe the fVco limits. Note that you don't have to write code to go through all the possibilities (the way I did in class) for this problem.

PLLIDIV = 001
PLLMULT = 1101011
PLLRANGE = 000
PLLODIV = 011
FPLL = 80.25MHz

b) Repeat this but assume we use a 24 MHz external crystal.

PLLIDIV = 010
PLLMULT = 1010000
PLLRANGE = 000
PLLODIV = 011
FPLL = 80MHz

3) Assuming that fPB = 80 MHz,

a) determine the value of BRG that will be needed to develop a baud rate of 7200 baud if UxMODE.BRGH = 0.

BRG = 693

b) Repeat the above assuming UxMODE.BRGH = 1.

BRG = 2777

c) Determine what the actual baud rate generated in each of the above cases will be, and what the % error from the desired 7200 baud in each case is.

a.) Actual baud rate = 7204.611
% error = .064%
b.) Actual baud rate = 7199.424
% error = -0.007%

d) What is the minimum baud rate that can be realized assuming fPB = 80 MHz. **Please show your work.**

BRG is a 16 bit register therefore the maximum BRG is 65,535.

Baud Rate = $\frac{fPB}{16 \times (UxBRG+1)}$ **=>** $\frac{80MHz}{16 \times (65{,}536)}$ **= 76.294**

So, the minimum Baud Rate is 76.

e) Considering that early common baud rates included 300, 1200, 2400, 4800, 7200, 9600, 19200, 38400, 57600 and 115200 baud, and that we generally want to sample at 16x the baud rate, what is significant about a peripheral bus frequency such as 18.432 MHz?

This peripheral bus frequency has a 0% error rate for each of these baud rates. Which makes it easy to use the most common baud rates.

4) Once UART transmissions are started, successive transmit interrupts will pump out characters. In class, we listed several ways to start and stop UART transmissions. One approach involves setting and clearing UxTXIE. In these questions, assume that UTXISEL = 01_2, and that this is a non-persistent interrupt.

a) If setting UxTXIE is to be used to start transmissions at a later time, how should the UART be initialized?

Need make sure TXEN is set during initialization.

b) If UxTXIE is cleared to stop transmission interrupts, how do we do so in a way that guarantee that the next time UxTXIE is set, a transmission interrupt will occur?

We must make sure that we unset the interrupt flag since it gets set when we clear UxTXIE.

c) Note that the setting or clearing of UxTXIE needs to be done atomically. List what things might happen if we didn't do these operations atomically and under what conditions these things might occur.

If this is not done atomically we can run into the same issue we had with interrupt flags not being done atomically. If not done atomically it requires a read, modify, and write which could be interrupted and lead to use not enabling when we wanted to or not clearing when we wanted to.

5) PIC32 Uart handshake
a) Can the UART be setup to automatically control the RTS pin without the CTS pin also being automatically used by the UART? If so, how?

Yes, you must set the UEN to the value 01 to have the UxRTS enabled and used and the UxCTS pin to be controlled by PORT latches.

b) Can the UART be setup so that it is controlled by the CTS pin without the UART also controlling the RTS pin? If so, how?

No you can set it to where they are both enabled and used by setting the UEN to the value 10 but not the CTS without the RTS

c) How does the **functional** meaning of an asserted RTS change when the UART that asserts RTS is in Flow-Control mode when compared with Simplex mode? Fill in the blanks:

i) In Flow-control mode, the UART asserts RTS when it is ready to **_receive data_____________** .

ii) In Simplex mode, the UART asserts RTS when it is ready to ______**transmit**_______ .

d) If the UART is set up to be controlled by the CTS pin, how does a negated CTS pin affect the behavior of the UART? Be complete. Describe what happens when CTS is negated between character transmissions as well as what happens if it is negated during a transmission.

If CTS is 1 (negated), the UART will load the transmit shift register, but will not shift it out. If CTS is negated in the middle of transmission, the current transmission completes.

e) Assuming that the UART could not be setup to automatically respond to CTS, how could we implement this function in software? Consider both how to stop transmissions when CTS is negated and later restart them when CTS is asserted. Assume we don't want to be stuck in a polling loop waiting for CTS to be asserted.

We can set an interrupt on the CTS so that when it changes it can then enable or disable transmission interrupts again.

6) Break
a) Why does the programmer need to wait until UxSTA.TRMT is set before setting UxSTA.UTXBRK?

If the FIFO contains transmit data when the UTXBRK bit is set, a break character will be sent when data is transferred to the UxTSR register, instead of the actual transmit data that was transferred into the UxTSR.

b) Why does the programmer need to transmit a dummy character after setting UxSTA.UTXBRK?

A dummy write to the UxTXREG register is necessary to initiate the Break character transmission.

c) Another way to transmit a break would be to wait for the TX buffer and shift register to empty, disable the UART (TXEN) and then to set the TX pin low, then high through port control and then re-enable the UART. What would the programmer have to do before taking the TX pin high again in order to properly transmit a break?

We would need to wait the appropriate time that a break would normally take.

d) So, what are the advantages of using the PIC32 UART's break facility?

The UTXBRK bit is automatically reset by hardware after the corresponding break transmission is complete. This enables the user to preload the write FIFO with the next transmit byte while the break is being transmitted.

e) How does the programmer know a break was received by a PIC32 UART?

The PIC32 processor transmits a start bit followed by 12 0's followed by a stop bit independent of the parity and the number of data bits settings. This is 13

0’s which is more than would be required in the worst case. In PIC32s a received break is recognized by receiving an all 0’s char with FERR set.