# ME4405 - Fundamentals of Mechatronics (Spring 2020)

**Lab Assignment Four**
**Serial Communication Using UART**

**Due Thursday, February 20th, 2020**

**Objective:** The main objective of this lab is to write a program that establishes 2-way serial communication between the MSP432 and a PC/laptop via UART. The lab makes use of the Driverlib library, although you are welcome to use direct register access instead if desired.

This lab is harder than Lab 3—be sure to allocate enough time for it!

## Deliverables and Grading:

To get credit for this lab assignment you must:

1. Turn in a typed report answering the questions at the end of the lab (2 pages max, can be shorter). This is **due by 5pm on the above due date**. You must submit the typed report electronically on Canvas. **(20 points)**
2. Demonstrate proper operation of your code to the TA or instructor during lab or office hours. **(30 points)**
3. Submit the commented final version of your code on Canvas. This code should be comprised of a compressed/zipped folder including your main .c source file containing your main() function, and any other custom .c and/or header files that you use. **(Pass/Fail)**
4. Lab Assignment Quiz will be taken on February 21st. **(10 points)**

## Setup:

This lab requires Code Composer Studio, Putty and the MSP432 device. The lab uses only the onboard electronics of the MCU and a USB cable. The MSP432 device contains two Enhanced Universal Serial Communication Interface (eUSCI) modules: eUSCI_A and eUSCI_B. The A module supports UART and SPI mode whereas the B module supports SPI and I2C modes. The MSP can be used with up to eight serial communication channels. This lab focuses on configuring and using the MSP432 in the Universal Asynchronous Receiver Transmitter (UART) mode.

## Problem Statement:

Write a program to transmit and receive data between the PC and MSP432 over UART. The program should receive data from the Putty software console, and then echo it back. Specifically, the MSP432 should receive data from Putty, storing all data in a character buffer. When a carriage return character is received (generated on the keyboard by the "enter" or return key), it should transmit the entire buffer back to the Putty terminal (this is called "echoing"). It should then clear the buffer and wait for the next line of text to come in. This echoing process is repeated in an infinite loop.

You must use either a receive interrupt or transmit interrupt, or both. If you choose to use only one, you should use polling for the other functionality. If this is your first time using interrupts, we recommend using interrupts to receive data from Putty and polling to transmit data to Putty. This is because there's only one interrupt handler, EUSCIAO_IRQHandler, to service both transmit and receive interrupts, which would require additional logic to distinguish between transmit and receive interrupts within the handler.

Your buffer should hold at least 200 characters. When data is echoed back to the terminal (upon hitting return or enter), your data should append a carriage return and newline character at the end to move to the next line in Putty.

## Background:

### UART:

Communication between two devices can be serial or parallel. In serial communication, data is transferred one bit at a time, as opposed to multiple bits in parallel communication. Serial communication is typically used in applications where I/O pins are limited and clock synchronization requirements make parallel communication difficult.

Serial communication can be further subdivided into two categories: Synchronous communication and asynchronous communication. Synchronous communication uses a data channel to transmit data and a clock channel to maintain synchronism. In asynchronous mode, only a data channel is used and data is transmitted at a certain baud rate. The clock channel is absent in asynchronous communication – rather, clock signals and baud rate are separately generated on each individual device. Synchronous communication is used in applications that require continuous communication between two devices whereas asynchronous communication is used when intermittent communication is acceptable or required.

UART is a hardware device that implements asynchronous communication. The communicating devices first agree on a baud rate. Data is transmitted in groups of 8 bits with parity (optional) and other optional bits as available on the communicating devices. The data packet also consists of one start bit and one or two stop bits. All of these settings are matched for both devices during their respective configurations.

Once configured, data can be transmitted by writing to the TXBUF register or by using the function **UART_transmitData**(EUSCI_A0_BASE) from the Driverlib library. Similarly, received data is accessed by reading RXBUF or using the function **UART_receiveData**(EUSCI_A0_BASE). Flags are raised when transmission or reception of data are complete. The transmission complete flag TXIFG is raised when all the data is transmitted and there is no more data in TXBUF. This flag is cleared by writing data to be transmitted to the TXBUF. Similarly, the receive complete flag RXIFG is raised when a complete character (8 bits) is received. This flag is cleared by reading the RXBUF register. When these flags are raised, if enabled, corresponding interrupts are also triggered. The interrupt subroutine can then be used to either transmit new data or to read the received data.

Note: All received data from the Putty console (described next) will be encoded in ASCII format, since it will be typed at the keyboard. To determine the hex equivalent of an ASCII character, consult the ASCII encoding table found at the following link: http://www.asciitable.com/

**Steps:**

The program has to establish successful communication between microcontroller and computer. Initialize the UART module using the functions from the driverlib library as discussed in class. First select a baud rate to use (standard baud rates are 9600, 38400, or 57600). The configuration parameters (clock divider and other register values) can be computed from:

http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP430BaudRateConverter/index.html

Additional documentation on UART functions could be found in the MSP432 DriverLib Users guide, located in the "Manuals and Spec Sheets folder" on Canvas.

Use a clock rate of 3 MHz, which is the default clock rate of the MSP432. Your UART configuration should use the above baud rate configuration parameters, the SMCLK source, no parity, LSB first, UART A mode, and one stop bit.

At the beginning of your main function you should configure the clock. Use the following commands to set up the clock:

```
int main(void)
{
    unsigned int dcoFrequency = 3E+6;
    MAP_CS_setDCOFrequency(dcoFrequency);
    MAP_CS_initClockSignal(CS_SMCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1);
    // do stuff
}
```

**Note**: Don't forget to stop the watchdog timer as the first step in your main function by simply calling `WDT_A_holdTimer();`

Follow the rest of the steps from the lecture to set up and initialize the UART. Also consult the Driverlib reference manual and MSP432 technical reference manual for guidance as you set up your code.

## Putty Configuration: (for Windows users only)

Download Putty (which is free) here:
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

In the Connection Type field on the home screen, select “Serial”. Place the baud rate in the Speed window. After plugging in the MSP432, go to Device Manager and look for “XDS110 Class Application/User UART”. The COM port listed (COM1, COM4, COM5, etc) is the COM port where the UART is found. Type this COM port in the Putty Serial line field.

Then go to the Serial window on the left. Match the data bit, stop bit, and parity configurations you used in your code. Turn Flow control to XON/XOFF. You can then save this configuration by going back to the Session window, typing a name in the Saved Sessions field, and clicking Save. To load the configuration, click on it and select Load.

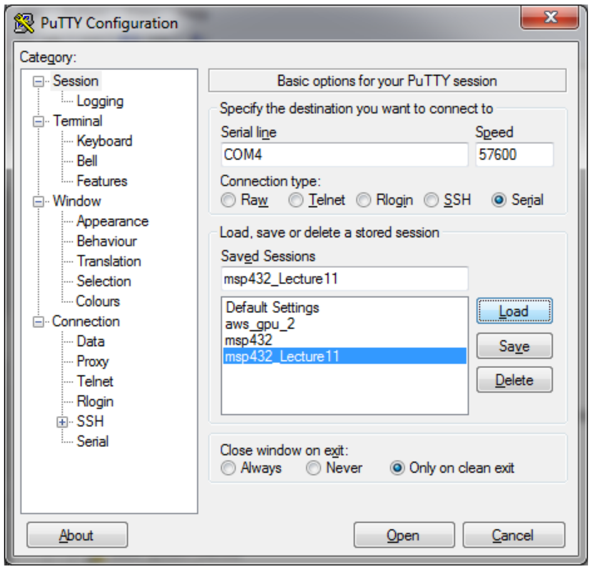
To connect to the MSP432, you must do the following:

The shorter way:

- Start Putty and open your session as described above
- Start Code Composer Studio and program the MSP432 using the debug button
- Once the code hits its initial breakpoint, terminate the Debug session in Code Composer Studio by hitting the red square “stop” button
- You should then be able to communicate with the MSP via the Putty console

If that doesn’t work, follow the longer way:

- After programming the MSP432, close both Putty and Code Composer Studio
- Unplug the board
- Plug the MSP432 back in
- Start Putty and open your session without starting Code Composer Studio
- You should then be able to communicate with the MSP via the Putty console



Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM4
Speed: 57600

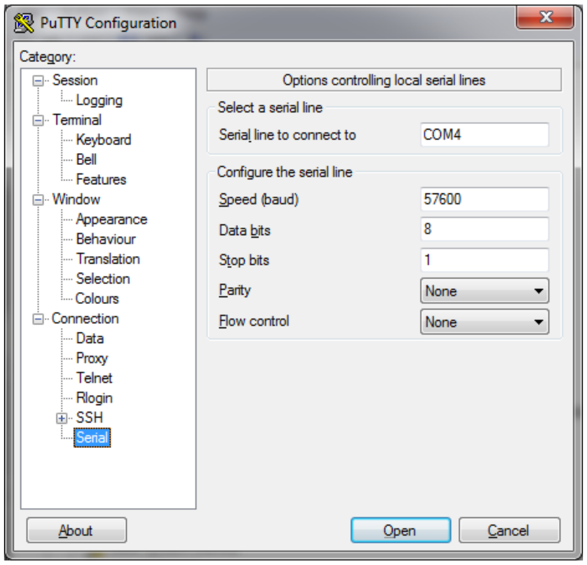
Connection type: Raw  Telnet  Rlogin  SSH  [✓ Serial]

Load, save or delete a stored session

Saved Sessions: msp432_Lecture11

Default Settings
aws_gpu_2
msp432
msp432_Lecture11

Close window on exit: Always  Never  [✓ Only on clean exit]



Options controlling local serial lines

Select a serial line

Serial line to connect to: COM4

Configure the serial line

Speed (baud): 57600
Data bits: 8
Stop bits: 1
Parity: None
Flow control: None

## Terminals for Mac Users:

PuTTY is only Windows-compatible. If you are a Mac user, you have two options: to either (1) use an alternative terminal, or (2) use a serial terminal within Code Composer. Option 2 seems to be the easier option.

(1) https://software.intel.com/en-us/setting-up-serial-terminal-on-system-with-mac-os-x

(2) https://www.youtube.com/watch?v=7-CxZuAn-Hc

## Helpful tips:

### Using the "Expressions" tab to monitor incoming/outgoing data

During Debug mode, the "Variables" tab in the upper right displays a subset of active variables with their values and data types. The tab next to it, "Expressions," allows you to specify any variable or register you're interested in viewing, which gives you greater flexibility. Use this to monitor the text inside your buffers to make sure you're storing your characters properly. You can also use this to see if individual characters have made it into your transmission/receive registers even before they are sent/read in.

If the expressions tab is not visible from your debug perspective, go to View-->Expressions to pull it up.

### Utilizing the "wait" command

```
while((UCA0IFG & 0x0002) == 0x0000){} // Wait until TXBUF is empty
```

This line is implemented to ensure that you are not overwriting transmitted information that has not been read and stored by stalling inside the while loop until the transmit interrupt flag has been re-enabled. However, this command sometimes will hang up your program. If you find that your program is stuck within this while loop, comment out this line and try running your program again.

## Requirements:

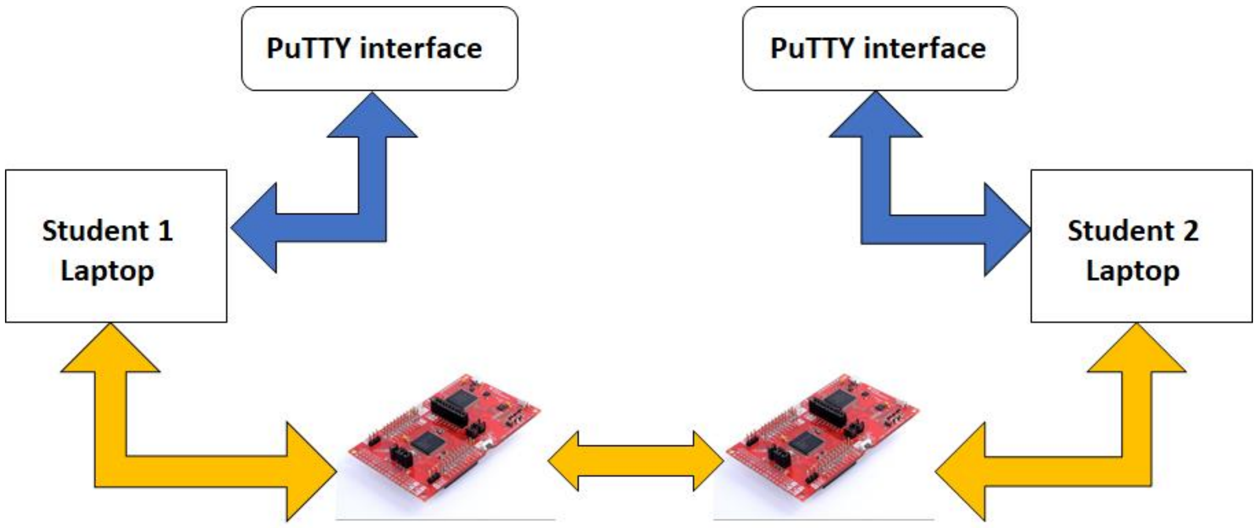
1. Successfully demonstrate your program and all required functionality to teaching staff.
2. Answer the below questions and submit the typed report on Canvas, along with the commented final version of your code.

## Questions:

1. Suppose you use a baud rate of 38400. How far off can the clock rate of your 2 devices be before you start reading data incorrectly? Do not use the 5% approximation, but rather compute the answer exactly. How does it compare to the 5% approximation? (**10 pts**)
2. Suppose your clock signals differ by more than the amount in Question 1. What error flag would you expect to see, in what register? Why would this error flag get triggered in this situation? (**5 pts**)
3. How could you solve the above problem (i.e., by changing the baud rate)? What is the inherent performance penalty associated with this solution? (**10 pts**)

**Extra Credit (20 pts):**

After demonstrating serial communication from the MSP432 to Putty, you can choose to go one step further and demonstrate communication between two MSP boards. In order to do this, you will need to choose a student to partner with to implement a two-way messaging system using an additional UART link between your two boards.



**UART Modules**
Laptop to PuTTY – A0 Module
MSP to MSP – A1 / A2 / A3 Modules

Just as in the main lab, successful demonstration will involve typing a message up to 200 characters long into your Putty terminal, which is then stored in a buffer. Upon hitting “enter,” the data should be transmitted through your MSP to your classmate’s board and continue on to their computer’s Putty terminal. Your classmate should then be able to transmit a response, visible on your Putty terminal. Each board will need two serial channels in order to transmit/receive from both a computer terminal and the other MSP board. The channel used by Putty must be using the A0 module; the connection with the other MSP may use any of the other UART modules (A1-A3).