CS466 Lab 4 – SPI Communications part 1.
Two week lab, due February 22, 2019.

Notes:
- You are encouraged to work in teams of 2 in this lab. If you work in a team I only require a single report handed in. Make sure that both names are on the report
- Teams of three are NOT allowed..
- This lab requires a breadboard and jumper wires. I will supply jumper wires in class, bug your EE friends for a solderless breadboard if you don't have one.

Overview:
- This lab introduces SPI communications and interfacing with an external device.
- We will implement a 'bit-banged' SPI Master talking to a simple GPIO expander chip. (bit-banging is a term used when we emulate a behavior that is normally performed by an SOC peripheral with GPIO pins).
- We will drive an LED and build an interrupt driven button handler.
- In lab5 (the most difficult lab) you will implement a software driven SPI slave on your Tiva Boards. As you interoperate with the SPI slave device think how you would define it in software.. This will be lab 5
- (spooky music crescendo..)

Objective:
- Gain understanding of SPI protocol and how it is used with a simple peripheral device
- Use a standard protocol to control GPIO endpoints on an IO expander

Prior to Lab:
- **Read the lab steps so you have an inkling of all the steps required to complete the lab. You will be hard pressed to finish in the two weeks if you are not ready to start when you enter lab Thursday.**
- Read https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus, in this lab we will be using a bit-bang SPI master to communicate with a SPI Mode 1 slave.
- In the Wikipedia page is code for a bit-banged master, compare it to the one in my slide set SPIBang.pdf. The algorithm has been updated since I wrote this lab a few years ago, I think I like the implementation better than I used to.

  Q1: Are they functionally different? Describe how..
- [You feel light headed.. Like you're having a vision from the future….As you work through this lab pay close attention to the slave operation, in the next lab you will be constructing an SPI Slave that presents a similar (but two byte) interface… then you feel much better]
- Review the datasheet for the MCP23S17 GPIO Expander. It is a SPI slave device that will give you 16 additional GPIO lines using only 6 signals from your Tiva board. 8 wires including power and ground. I have provided the datasheet but you can find it all over online. http://bfy.tw/GZ0M
- Looking over the datasheet read carefully Section "1.0 DEVICE OVERVIEW" and peruse the datasheet until you understand everything it references.
- When I precede a signal name with a bang '!' (actually called an exclamation point). In this lab such signals are !CS and !RESET. That is a short notation for indicating that the logical active state is triggered by a low voltage on that pin. There are other common notations. If you see funny characters around what would be a normal pin name !CS, bCS. CS/.. Take care to understand if it means that the logic level is opposite the actual voltage level.

Lab Work:
1. ❑ **Be careful in this step to save some confusion.** This is an interesting SPI device as it includes the ability to use three pins on the device called A0, A1 and A2 to be tied to ground and VPP to give the device an address. The logical binary value created by setting the voltage at these three pins is then used

as a device address. The device will only respond to Read/Write requests that carry the devices logical address as part of the first byte in a command sequence.

Q2: Why is this unlike most SPI devices?

Q3: What does it mean for our EE board layout friends?

2. ❑ Create a minimal lab4 project space "spi_master" by copying your lab3 code. You should be sure to retain the GPIO, Heartbeat, Serial communications, and assert functionality but you can delete all of the producer/consumer/queue code. Make sure that a heartbeat-only program is running properly before you move on to the next step.

3. ❑ The PDF file (SPIBang.pdf) contains the sequences to write to and read from the GPIO expander.

4. ❑ Get a MCP12S17 GPIO Expander from the Instructor box in our shelves. Make sure that you get a SPI device that ends with S17 and not the one that ends in 017 which I bought by mistake. It's the same device bit interfaces via I2C which is a significantly more complex.

5. ❑ You will need to decide on Tiva board inputs and outputs to use
   a. 4 output pins (!RESET, !CS, SCK, and MOSI)
   b. 2 input pins (INT, MISO)

   Take time to perform the GPIO setup for these pins and VERIFY that the outputs can be controlled by simple software before you proceed to the next step. Discovering that you have setup a I/O line incorrectly after you have hardware connected can really slow you down.

6. ❑ Connect the expander to your Tiva board.
   a. Expander VDD(V3.3) and VSS(Ground) connect straight through to Tiva 3.3 and ground
   b. Expander !Reset, !CS, SCK, SI connect to Tiva GPIO Pins setup as outputs
   c. Expander SO connect to a Tiva GPIO pin setup as an input.
   d. Expander A0, A1, A2 connect to ground.
   e. As soon as your Tiva board is setup for GPIO interaction set the !Reset to low output and then back to high, this will reset the expander defaults whenever your application starts.

7. ❑ When your Tiva board resets it should reset the GPIO Expander so that you are not fighting old register settings. Before starting the scheduler pull the expander !RESET line low momentarily. This will always reset the expander when you restart the software on the tiva-board.

8. ❑ First try to read the device, There are known register default values in the datasheet (Table 1.5 and 1.6 details registers, their default values are and what address they should be at).

9. ❑ Look at the last page of the SPIBang.pdf gives examples of writing data and reading it back from the GPIOA and GPIOB data registers. You should be able to mimic this test with your scaffolding code.

10. ❑ Design and implement the functions expanderRead(uint8_t addr) and expanderWrite(uin8_t addr, uint8_t data) to retrieve and set any of the readable registers off the device.

11. ❑ Design and implement a function expanderInit(void) to initialize and setup the expander using expanderRead() and expanderWrite(). Configure the A0 GPIO using similar register read/write operations that we use within the Tiva SOC. Enable, direction, options. Read the data sheet to decide the minimum that you need to set.

    Q4: What is the purpose of the BANK bit?

12. In a method similar to what you used in step 5 to verify that you can toggle the Tiva board pins, verify that your software can toggle the GPIO expander A0 pin reliably. (When I initially do this I generally

make the pin follow my heartbeat LED so I have an indication on the Tiva board what the expander pin should be at.)

13. ❑ After verifying that you can read/write and configure the device. Connect an LED and resistor to one of the GPIO pins. Connect the led circuit so that the GPIO turns the LED on my providing a path to ground. Remember that we want about 10ma. Follow the first example in this Maxim LED White-Paper but adjust the resistance to account for using a 3.3 volt rail.

14. ❑ Make the new LED blink on and off at the same time as your heartbeat LED. Include all of this new code in a function of its own so that you don't pollute your heartbeat loop. We use a macro LED(LED_G, ledOn); in the heartbeat loop to handle all the code light the LED. Come up with a reasonable command structure to control the remote LED.

15. ❑ How fast can you drive the expander-connected LED? Use a scope measure the highest frequency that gives you a reasonable signal.
    Q5: What speed did you attain? What happened when you went faster?
    Q6: What do you think are the limiting factors of driving this led on and off being 1 cycle.

16. ❑ In an actual design other GPIO pins will most likely be used for something (input or output) verify in your design that all the other GPIO expander pins on that port pins are unaffected as you toggle the LED?
    Q7: Describe the technique you used to implement this?

---------- This is where I expect individuals/teams to be after the first day of lab ------

17. ❑ Setup a pin on GPIOB to be an input and connect the pin through a normally open momentary button to ground. If you don't have a pushbutton a jumper wire works fine. I'll see if I can track some buttons down.

18. ❑ Configure the port B pin you use to include an internal pullup resistor.
    Q8: Why is this resistor required and what's the best reason for using the internal version.
    Write a function to run from your heartbeat that polls the expander B register and if the button pressed (Register reads 0) illuminates the Tiva Red LED.

```
…
// maybe something like for my schematic above
LED(LED_R, (expanderPortRead(b_addr) & (1<<2)));
…
```

19. ❑ Setup the IO Expander to transmit an interrupt to the Launchpad when the button is pulled low. You will need to;
    a. Wire the interrupt-B line of the expander to your interrupt input GPIO on the Launchpad..
    b. setup that Launchpad pin as an input
    c. setup that Launchpad pin with an edge triggered interrupt service routine (look at your Lab02) don't use PORTF for this input as you only get one Tiva ISR vector (handling routine) per IO port.
    d. Your ISR for the remote interrupt must clear the source of the interrupt before it can complete, on some devices you must write to a control register to clear the interrupt.
        Q: What does the GPIO expander require to clear the interrupt?
    e. Make the code behave in a manner that you can tell when the remote button interrupt has occurred.

20. ❑ Using a scope or logic analyzer, measure latency that the system has to your button input. For the purposes of this lab latency is described as the time from button activation until the ISR clears and resets the interrupt masks.
    a. When measuring measuting an ISR it's often useful to setup another output pin on the Tiva board and then measure on the scope the time from the triggering edge on the GPIO to an output that is toggled on then off in the ISR.
    b. Measure the interrupt latency that you encounter from the button push to the ISR code getting context.
    c. Describe your measurement setup including code that you may have added to aid in instrumentation.

21. Closing Questions:
    a. In the SPI Wiki, What is meant by calling SPI a 'de-facto standard'? How does it affect us?
    b. List the steps from button-press to ISR clearing the Tiva Interrupt.
    c. Under what conditions might the latency be effected by normal task in your system?

CS566 students must complete the following steps for full credit,
CS466 students may complete it for extra credit, This is good stuff to know.

22. ❑ Instrument your code to count the number of button presses that are received in step 20. In the Tiva code. I expect that you are recording more button presses than you are invoking. This is due to the switch (or jumper-wire) bouncing.
    a. Research GPIO Button Debouncing on the internet.
    b. Implement a de-bounce method that will support at least 50 button presses a second.
    c. Verify that you can reliably capture button presses at 50 Hz with a second Tiva or with a function generator if in the lab.
    d. Document why you chose your method and your analysis of its performance.