CS466 Lab 4 – SPI Communications part 1.
Due midnight March 17, 2025.

Use the provided lab format on Blackboard for your reports.

Notes:

- You are allowed to work on this lab individually or in a two person team. Teams of three are not allowed.
- If you work in a team, I only require a single report handed in.
- This lab requires a breadboard and jumper wires. I will supply jumper wires in class, bug your EE friends for a solderless breadboard.
- This lab requires an MCP23S17 GPIO expander. This is a 28-pin device is an SPI slave that provides 16 additional GPIO pins to the master. For only four processor GPIO pins this device in groups will allow control of up to 128 pins of GPIO expansion

Overview:

- This is a lab to introduce SPI communications.
- 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 a later lab (more difficult) you will implement a software driven SPI slave on your 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
- Implement and debug a bit-banged serial communications protocol.

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 pseudo code for a bit-banged master, compare it to the one in my slide set SPIBang.pdf. **Q: 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 4 signals from your pico board. 6 wires including power and ground. 8 wires including reset and an interrupt line. 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.

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.
   Q1: Why is this unlike most SPI devices?
   Q2: What does it mean for our EE board layout friends?

2. ☐ Create a minimal lab4 project space by copying your lab3 code. You should be sure to retain the Heartbeat, Serial printf() communications, and assert functionality but you can delete most of the producer/consumer/queue code. Make sure that a heartbeat-only program is running properly before you move on to step 4.

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 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 more complex protocol.

5. ☐ Connect the expander to your Tiva board. Note the semi-circle on the chip that matches same on the datasheet MCP23S17 pinout. In order to us not to go insane let's all use the same pin connections.

a) Connect Gnd on your breadboard to Pico Pin 38
b) Connect Vss (Ground) on the GPIO to the breadboard ground
c) Connect V3.3 (the high rail) ot Pico Pin 36
d) Connect Vdd (V3.3) on the GPIO to the breadboard V3.3
e) Move your SW1 to Pico Pin 20
f) Move your SW2 to Pico Pin 19
g) Connect GPIO A0, A1, A2 pins to ground.
h) Connect SPI CS to Pico Pin 22
i) Connect SPI CLK (SCK) to Pico Pin 24
j) Connect SPI MOSI (SI or TX) to Pico Pin 25
k) Connect SPI MISO (SO, or RX) to Pico Pin 21
l) Connect GPIO Reset! To you're the Reset! (RUN) pin on th Poco, keep the button attached.

6. ☐ I recommend that you first try to read the device, you know some register default data from the datasheet (Table 1.5 and 1.6) what register default values are and what address they should be at.

7. ☐ 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.

8. ☐ I have provided header files mSpi.h and mGpio.h. You will be required to implement the body of these functions in mSpi.h and mGpio.h

9. ☐ Getting the read and write to the GPIO registers working is an important first step. In my lab4.c I wrote a function to verify this operation.

```
//
// gpioVerifyReadWrite()
//
// This is the main function of a task that I'm using to verify that
// my GPIO and SPI functionality is working correctly.  It will be retired
// as I move on to actual GPIO-Expander Functionality.
//
void gpioVerifyReadWrite(void * notUsed)
{
    const uint32_t queryDelayMs = 100;  // ms
    uint8_t regValue;
    uint8_t count=0;

    while (true)
    {
        mGpioWriteByte(IODIRB, count++);
        regValue = mGpioReadByte(IODIRB);
        printf("IODIRB: 0x%02x,  ", regValue);

        regValue = mGpioReadByte(IODIRA);
        printf("IODIRA: 0x%02x,  ", regValue);

        regValue = mGpioReadByte(IPOLA);
        printf("IPOLA: 0x%02x\n", regValue);

        vTaskDelay(queryDelayMs);
    }
}
```

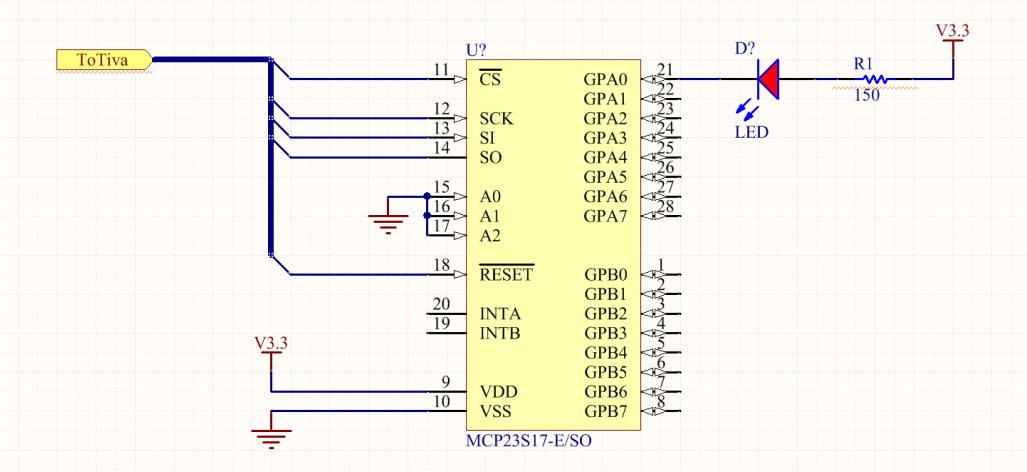
This routine when run should produce the output….

```
IODIRB: 0x00,  IODIRA: 0xff,  IPOLA: 0x00
IODIRB: 0x01,  IODIRA: 0xff,  IPOLA: 0x00
IODIRB: 0x02,  IODIRA: 0xff,  IPOLA: 0x00
IODIRB: 0x03,  IODIRA: 0xff,  IPOLA: 0x00
IODIRB: 0x04,  IODIRA: 0xff,  IPOLA: 0x00
IODIRB: 0x05,  IODIRA: 0xff,  IPOLA: 0x00
IODIRB: 0x06,  IODIRA: 0xff,  IPOLA: 0x00
etc....
```

Q3: What is the purpose of the BANK bit?

10. ☐ 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.

Note: this is an old drawing the 'To Tiva' should be replaced with 'To Pico' !Reset should be connected to your Pico RUN pin so that when you reset the Pico you reset this device as well.



11. ☐ 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.

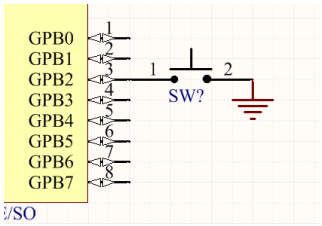
12. ☐ How fast can you drive the expander-connected LED? Using a scope measure the highest frequency that gives you a reasonable signal. Document the speed you attained.

Q4: What are your limiting factors of driving this led on and off being one cycle?

13. ☐ In an actual design other GPIO pins will most likely be used for something (input or output) does your design verify that all the other pins are unaffected as you toggle the LED?
Q5: Describe the technique you used to implement this?

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

14. ☐ 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.



15. ☐ Configure the port B pin you use to include an internal pull-up resistor.
Q6: Why is this resistor required and what's the best reason for using the internal version?

16. ☐ Write a function to run from your heartbeat that polls the expander B register and if the button pressed (Register reads 0) illuminates the LED solid if the button is pressed.

```
…
// maybe something like for my schematic above
If( expanderPortRead(b_addr) & (1<<2)){
    // only force the LED on
} else {
    // Normal LED behavior
}
```

17. ☐ Setup the IO Expander to transmit an interrupt to the pico when the button is pressed (pulled low). You will need to

a) Wire the interrupt-B line of the expander to another GPIO on the pico
b) setup that GPIO pin as an input
c) setup that GPIO pin with an edge triggered interrupt service routine (look at your Lab02)
d) Your ISR must determine if the interrupt source (if shared) is the GPIO expander. 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.
Q7: 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.

18. ☐ 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) Describe your measurement setup including code that you may have added to aid in instrumentation.

19. ☐ Instrument your code to count the number of button presses that are received 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) Implement a debounce method that will support at least 15 button presses a second.
   b) Verify your debounce speed with a second Tiva or with a function generator if in the lab.

20. ☐ Other Questions:

Q8: In the SPI Wiki, What is meant by calling SPI a 'de-facto standard'? How does it effect us.
Q9: What latency did you encounter in #17. Could you do better
Q10: List the steps from button-press to ISR clearing the Tiva Interrupt.
Q11: Under what conditions could the latency be effected by normal task in your system?