

## CS466 Lab 4 – SPI Communications part 1.

Two week lab, due Feb 27, 2020.

### Must use provided lab format on Blackboard

#### 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.
- This lab requires a breadboard and jumper wires. I will supply jumper wires in class, bug your EE friends for a solderless breadboard.

#### 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 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](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 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.

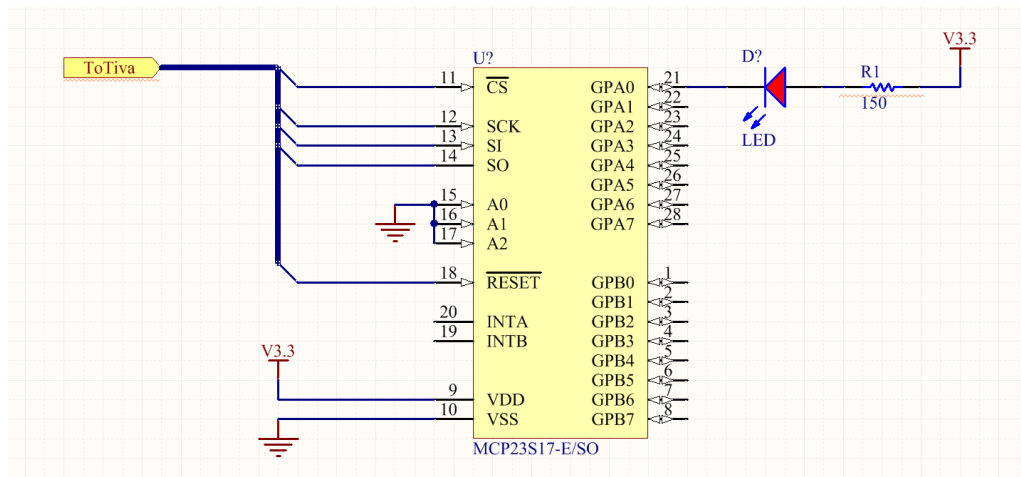
#### Lab Work

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 “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 most of the producer/consumer/queue code. Make sure that a heartbeat-only program is running properly before you move on to step 3.
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 but interfaces via I2C which is a more complex protocol.
5. ☐ 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.
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. ☐ Design and implement the functions **expanderRead(uint8\_t addr)** and **expanderWrite(uint8\_t addr, uint8\_t data)** to retrieve and set any of the readable registers off the device.
9. ☐ 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.

**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 by providing a path to ground.



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.

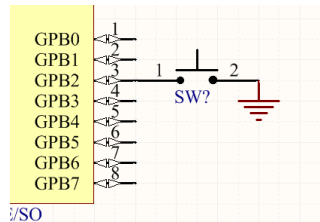
**Q4: What are your limiting factors of driving this led on and off being one cycle?**

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 day of 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 Tiva Red LED.

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

17. ☐ Setup the IO Expander to transmit an interrupt to the Launchpad when the button is pulled low. You will need to
- Wire the interrupt-B line of the expander to another GPIO on the Launchpad..
  - setup that Launchpad pin as an input
  - setup that Launchpad pin with an edge triggered interrupt service routine (look at your Lab02) don't use PortF as you only get one Tiva ISR vector (handling routine) per IO port.
  - 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.

**Q7: What does the GPIO expander require to clear the interrupt?**

- 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.
- 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.
- Implement a debounce method that will support at least 50 button presses a second.
  - 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?**

21. ☐ I Write up your lab using the lab format provided on Angel. Include your program as a fixed spaced (I recommend Lucida Console) addendum to your lab. I will cut points for proportionally spaced code pasted in the end of the lab. (2-3 pages)