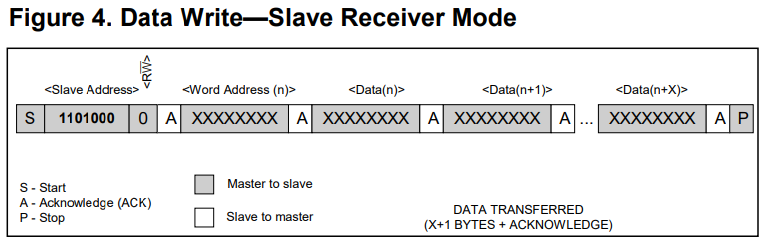
The largest obstacle for understanding I2C is understanding the interconnection between the two main wires SDA (serial data) and SCL (serial clock), which are open drain that connect multiple devices in a multi-drop bidirectional low-speed bus. Jake and Ammon hit the RTC portion of the lab as hard as we could. Jake wrote some co If idle, both SDA and SCL are high. Transactions begin when SDA goes low, followed by SCL. This indicates to all receivers on the buss that a packet transmission is starting. While SCL is low, SDA transitions (high or low) for the first valid data bit, which is the start condition. I2C does this weird protocol to take advantage of properties of capacitors. You see, capacitors discharge at a faster rate than they charge, which is why the on (low) and off (high) is the way that it is. With each bit that is sent across the line, the bit must become valid on SDA while SCL is 0. The bit is sampled each rising edge of SCL and must remain valid until SCL goes high once more. Then SDA switches bits before SCL goes 1 once more.

Start and stop bits, I prefer to think of them as conditionals, must be sent from the master along with the address bits. The data bits on the other hand can be sent by the slave or the master. This particular protocol allows for multiple masters and multiple slaves but they all share the same SDA and SCL serial line. This protocol is good because there is no intermediate value, unlike some other protocols, because the SDA and SCL is connected to the bus via a large resistor the current will immediately drain whenever voltage goes over the line. This makes the protocol very responsive and resistant to noise. Just to reiterate, the master initiates and terminates transmission and generates the SCL. The slave is addressed by the master. Orthogonal to the master and slave are the transmitter and receiver. The transmitter places data on the bus while the receiver reads data from the bus. The master can be both a transmitter and receiver, and so can the slave. Every node is either a master or slave, or either a transmitter or receiver.

The largest obstacle in accomplishing this project was dealing with TI’s crummy data sheets and documentation for their peripherals. The datasheet explicitly said to use I2CMasterBusy to wait for when the master was busy but in their own example code they state to use I2CMasterBusBusy to wait for the bus busy bit but instead of the busy bit. This issue was the hardest problem we had and gave us the most trouble. When debugging hardware and it does nothing, but it tells you that it is working, that can be the most frustrating thing. We decided at this point that anything we did would need to be looked at on the oscilloscope. If you’re ever having trouble with I2C the only real proper way to find what is wrong is by capturing traffic on the bus and look at what is happening bit by bit. You can stare at code all day until you’re blue in the face but until you look at what the hardware is doing, you’re not going to know what is happening!

We also ran into hardware that was broken. The DS1307 we bought appears to be able to be able to receive address bits but NOT receive data bits. We’ve tested the device on the same piece of code and on one the device receives the address bits but fails. Since we were far along, another group lent us their clock, we tested our code on it and it was successfully sending and receiving data sort of correctly. The important part is that the code was correct at that point. Once we could see what was being transmitted over the oscilloscope, the real debugging could begin. While reading the oscilloscope manual on edge triggering we learned that Tektronix’s claim to fame is discovering edge triggering.

It is worth noting that we used the peripheral library to implement this portion of the lab and that “SYSCTL\_USE\_OSC” was used in setting the clock frequency, not “SYSCTL\_USE\_PLL”. This decision was made after we noticed that this resulted in a much more stable signal coming from the device itself.



The code was structured to match the standard on the left written in a simple and crisp way to try to reduce the redundancy in our code. Before this fact we were just testing things and were a bit too grandiose in what we could accomplish in the short amount of time.

**Conclusion**

The TMP36 is a specialized transistor used for measuring temperature designed in such a way to produce specific voltage range that has a linear reference to temperature over a given range.