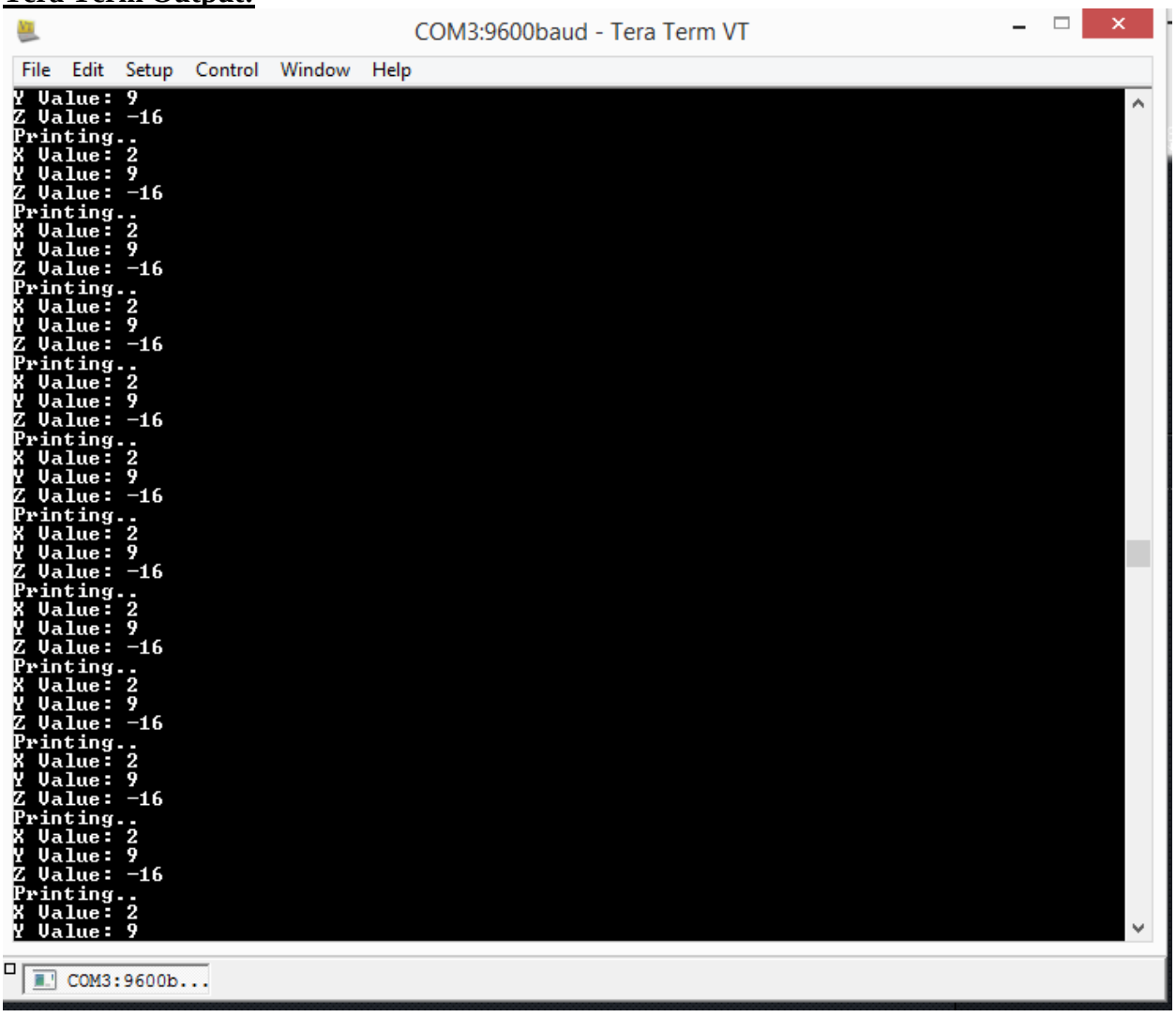
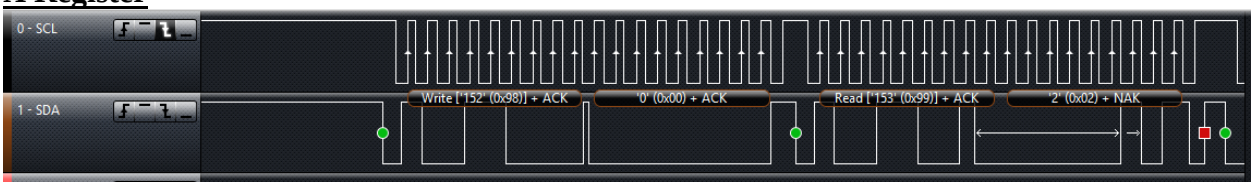
Algorithm Description of Speed/Direction from Accelerometer

The Accelerometer returns values of X, Y and Z in its own registers (0,1,2 respectively). Each time the accelerometer sent an interrupt, the TIVA board would use an interrupt handler to read the values from each register. It always read in as a positive value, thus I chose to mask the binary value and look at its MSB. If the MSB was a 1, then I would subtract 32 from it so that it would have a negative range as well. For Speed, I divided the value of that we obtained by 3. This way, instead of adding/subtracting large values and possibly accelerating fast, it would be slowed down and would go over small pixel amounts.

Tera Term Output:

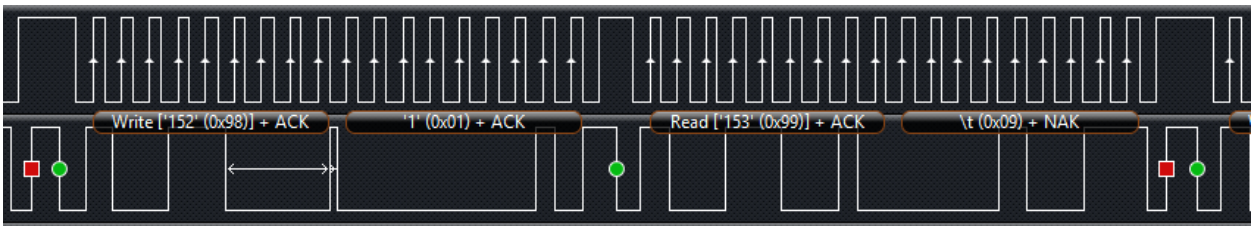


X Register



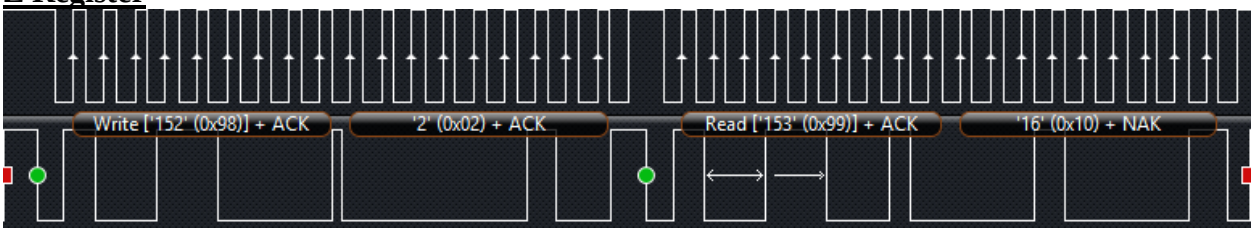
Write to Register 0, which corresponds to X
Read from Register 0, which corresponds to 2 and is correct according to Tera Term.

Y Register



Register 1 Corresponds to the Y register
Reading from that register it reads in a value of 0x09, which is 9 and matches out Tera Term

Z Register



Writing to Register 2 is the Z register
Reading from that Register we get 16, Tera Term displays -16 because I mask the incoming data to see if the MSB is equal to 1. If it is equal to 1, I subtract 32 from it because I want signed integer rather than unsigned; thus this will give us a range of -32 to 32. (16-32 = -16 which matches)

Conclusion

In Part 1, our goal was to configure the accelerometer so that it would constantly trigger an interrupt returning the x, y, and z position of our setup. We used the MMA7660FC accelerometer. In our code, we wrote to the registers 6, 7, and 8. Register 6 was the interrupt setup register, where we set GINT, or 0x10, to high. Register 7 was the Mode Register**.** We set IPP and MODE to high writing 0x41 to the register. In Register 8 we adjusted our sampling rate to 64 samples/second. The value we put in was 0x01. The only difficulty we had was getting the interrupt to start and this was because we didn't understand very well how to write to registers to make the accelerometer have the desired characteristics. The following parts were simple as well we had to do was to create a function that updated the ball every so often.

However, we ran into errors when including the extra credit portion. The XBEE portion ran into trouble because one of the OLED would not turn on. We copied most of the transmission code from the previous lab, and it would not work. We only had about half a day to debug, so we did not look too much into it.

The second extra credit was not too difficult either. We created a simple maze with two walls. We set conditions so that if the ball touched the coordinates of the wall, it would not move or pass through the wall.

The most difficult portion about this lab was just learning how to set up the I2C and the registers on the accelerometer. It's definitely valuable to read through the data sheets because it shows us what it has and how we can alter the accelerometer to output the desired data we want.