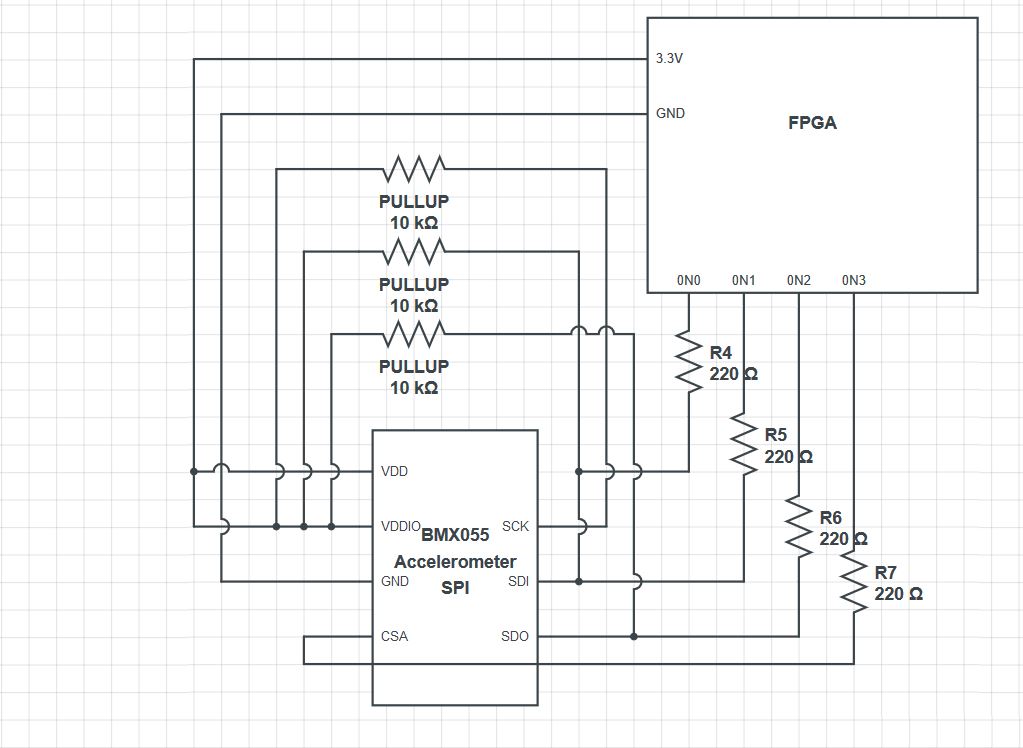
Riker Quintana

816823248

11/29/18

470L Project – Final Report

The original project is a Cordic Core that takes input from an IMU’s accelerometer, translating the raw data into usable rotation data. However, I realized later that in order get the data to the FPGA I would have to figure out some way of communication. My senior design team was working on a way to use I2C to communicate with the sensors, so I thought I’d implement a I2C core from OpenCore and wire it with Cordic Core using a top-level module.

I spent a good week or so focused solely on the I2C module, but the only complete implementation I could draw from was the testbench provided, but not an actual synthesizable version. I spent much time working on making a synthesizable version, however I would constantly receive errors related to the wishbone write function implemented. After speaking with Prof. Arnold, I learned that the I2C core was meant to implemented with a complete embedded device, not just itself standalone. It was expressed that an SPI communication would be much easier to implement. 

I continued working on some form of a communication module and proceeded to implement an SPI module. Once again, the same kind of issues ensued where there wasn’t an SPI Core, I could find that was synthesizable out of the box. All online implementations on OpenCore or other websites had SPI Verilog module examples…as testbenches. None of them were synthesizable on the FPGA as delays and loops were used. I tried to convert the code into a modified, synthesizable SPI core, however I found it would just be easier if I wrote my own. Due to these many conflicts, it was decided I should scale back my project to just getting the SPI communication working with the sensor.

Below is my Verilog module which synthesizes without error.

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 01:35:22 11/28/2018

// Design Name:

// Module Name: SPIMaster

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module SPIMaster( input miso, input clk, output reg scl, reg mosi, reg ss1, [7:0] LEDX, [3:0] LED

);

reg [15:0]xx=0;

reg [7:0] x\_lo, x\_hi, y\_lo, y\_hi, z\_lo, z\_hi;

reg [6:0] sclCount =0;

reg [5:0] count =0;

reg t\_scb\_s = 0;

reg [10:0] offsetCnt =0;

reg busy, slaveActive, clk2 = 0;

reg clkon =1;

reg [32:0] clkCounter;

parameter ADDR = 16'b1011\_1111\_0110\_1111;

//parameter ADDR = 16'b1000\_0011\_0110\_1100;

initial begin

scl = 1;

ss1 = 0;

//xx = 0; // readbit + 7\_bit addr of buffer + 8\_ bit data

end

assign LEDX = x\_lo;

assign LED[3] = busy;

assign LED[2] = slaveActive;

assign LED[1] = sclCount[0];

assign LED[0] = clkon;

always@(posedge clk) begin

if(clkCounter == 10000) begin

clk2 = 1;

clkCounter = 0;

end

else begin

clk2=0;

clkCounter = clkCounter + 1;

end

if(clk2) begin

clkon = !clkon;

if(!busy) begin

ss1 = 0;

scl = 1;

//ss1 = !ss1;

if(!t\_scb\_s)begin

if(offsetCnt<1)

offsetCnt = offsetCnt + 1'b1;

else begin

t\_scb\_s = 1;

offsetCnt = 0;

end

end

else begin

slaveActive = 1;

t\_scb\_s = 0;

end

end

if(slaveActive && sclCount < 113)begin

sclCount = sclCount + 1'b1;

busy = 1;

scl = !scl;

end

else if(slaveActive || sclCount == 113)begin

sclCount = 0;

busy = 0;

slaveActive = 0;

ss1 = 1;

end

/\*else begin

busy = 0;

slaveActive = 0;

sclCount = 0;

ss1 = 1;

end\*/

end

end

always@(posedge scl) begin

if(sclCount == 1) begin//just beginning transmissiion. Reset address to be sent

xx = ADDR;

end

else begin

mosi = xx[15];

xx = xx<<1;

end

end

always@(negedge scl) begin

if(count < 56)

count = count + 1'b1;

else

count = 0;

if(count > 8)begin

case (count)

9,10,11,12,13,14,15,16:

x\_lo = (x\_lo<<1)| miso;

17,18,19,20,21,22,23,24:

x\_hi = (x\_hi<<1) | miso;

25,26,27,28,29,30,31,32:

y\_lo = (y\_lo<<1) | miso;

33,34,35,36,37,38,39,40:

y\_hi = (y\_hi<<1) | miso;

41,42,43,44,45,46,47,48:

z\_lo = (z\_lo<<1) | miso;

49,50,51,52,53,54,55,56:

z\_hi = (z\_hi<<1) | miso;

endcase

end

end

endmodule

The module works as follows:

The Module has a combination of 4 input/output wires connected to the sensor: **miso** (master-in-slave-out), **mosi** (master-out-slave-in), **scl** (serial clock), and **ss1** (slave-select1). Six data registers are created for x,y,z low and hi bits of the accelerometer. The ADDR parameter is what will be sent to the accelerometer; 8-bits for register address and 8 -bits for data to be written to the register addressed if the process is a write. LEDs on the FPGA and breakout board are connected to ports to check the functionality of the system parts.

Beginning in the always block a 2nd, slower clock is created for the system to use. When the clock is high it checks the busy bit if it is low before initiating SPI communication. When the busy bit is low the slave-select line is pulled low and the serial clock is set high. I then have an if statement that is meant to delay all other steps in order to meet the setup time of the scl line as described in the datasheet for the BMX055 Accelerometer. Once the delay is finished then the slaveActive bit is set high which will start the SPI communication.

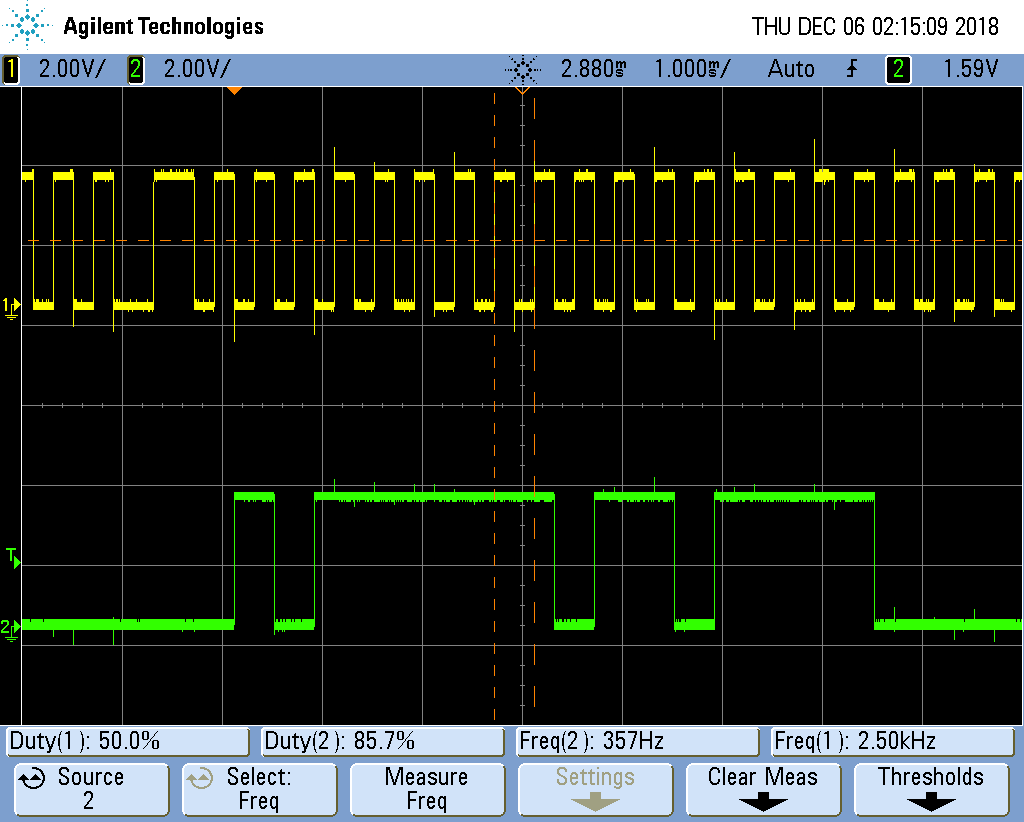
The SPI communication is essentially a state-machine created with if and case statements. The sclCount keep track of the length of the communication. It increments the amount of times the scl line should toggle high and low. Since I need to send 8-bits as address selection and receive 48-bits of sensor data for the x, y, z axis, I needed 56 cycles. This would have the scl line toggle 112 times, which is why the sclCount increments until that number is reached. Once the increment maximum is reached then the SPI communication would be finished all lines and control bits would be reset.

The xx register gets the ADDR parameter bits in the first cycle of the scl clock and begins shifting the bits onto the mosi line for the first eight clock cycles. The first bit is a 1 which indicates to the sensor that a read operation will commence. The next 7-bits are the address of the register desired. For the sensor I want the data buffer which stores all sync’d x ,y, z low and high bits. It will continue shifting bits onto the line for the rest of the process, but they are ignored as the rest of the SPI communication is just reading from the sensor and the mosi is ignored as long as the ss1 line is low.

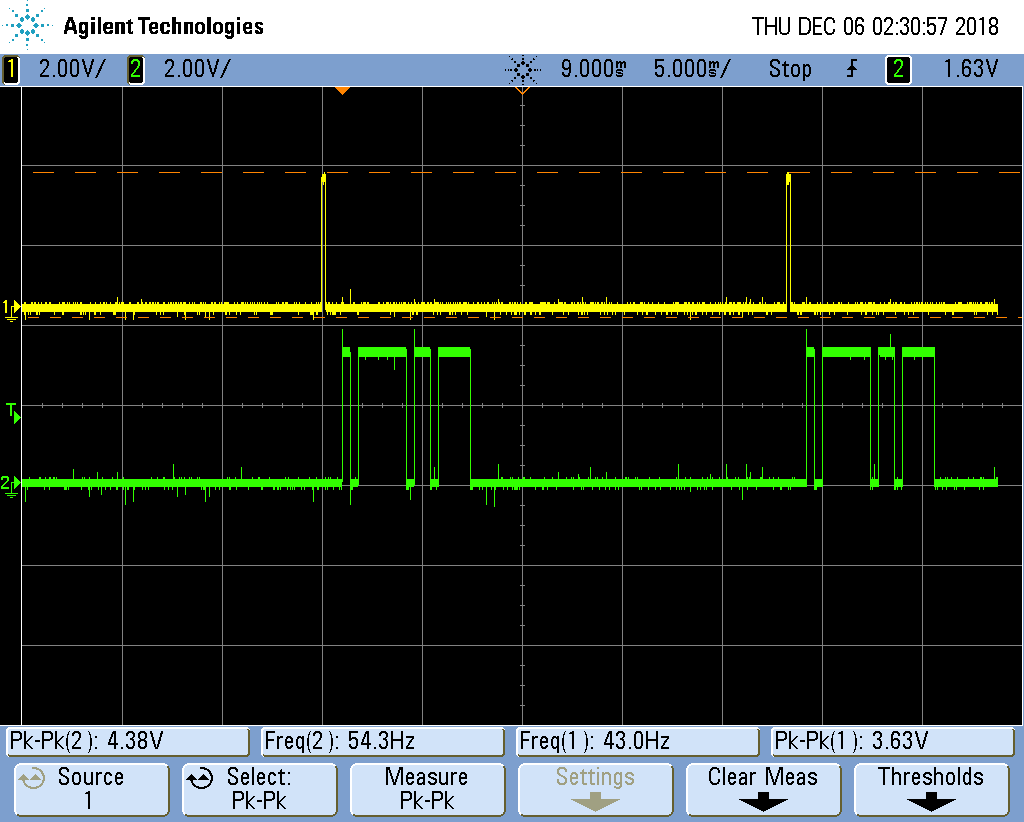
After the eighth-cycle has passed, the FPGA begins reading in data from the miso line connected to the sensor. For eight-cycles each it stores the data into the x\_lo, x\_hi, y\_lo, y\_hi, z\_lo, and z\_hi registers. The x\_lo is wired to the breakout board LEDs so the results can be seen. After the count is completed and sclCount reaches its maximum then SPI communication finishes.

Based off the Oscilloscope data below my FPGA seems to be functioning on its end as expected. However, the sensor pin connected to the miso line is always high and I’ve been unable to find a solution. It was recommended to try and match the time diagrams of the data sheet to solve the problem, however the only realistic thing I knew how to do was delay the beginning in order to give the scl line the setup time it needed. Unfortunately, we were never taught how to do timing-constraints in any of our courses nor were our resources available knowledgeable on it. Now that I’ve come to deal with this issue, I think it would be important to teach that in future 470L courses.

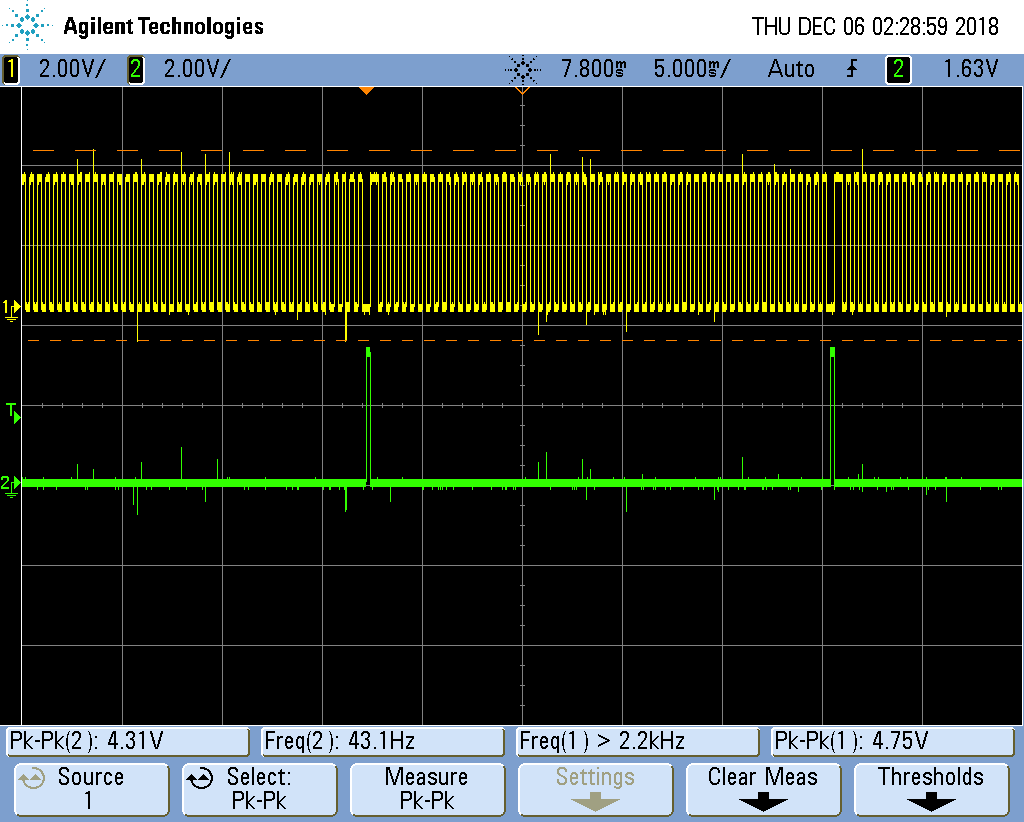
Ultimately the deliverable I could validate was that my FPGA module works as expected, however the sensor was unable to communicate with the FPGA as desired.



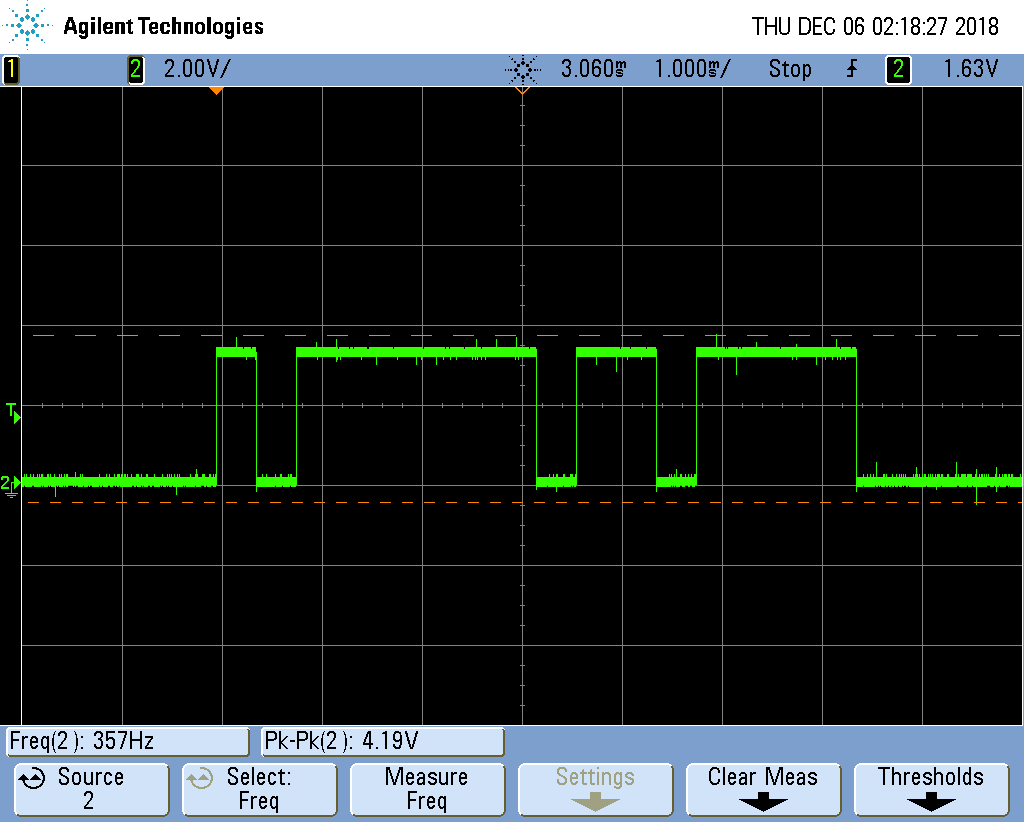
Source 1: SCL Line Source 2: MOSI Line



Source 1: SS1 Line being pulled low to initiate SPI Source 2: MOSI Line sending address bits



Source 1: SCL Line Source 2: SS1 Line being pulled low, initiating SPI



Source: MOSI Line

