**ECE 540 FINAL PROJECT REPORT**

**Fall 2018**

**Project:**

Toad Maze

**Submitted by:**

Jonathan Anchell

Chelsea Brooks

Michael Bourquin

Table of Contents

[**Project Summary**](#_shjh37wuww9p) **1**

[**Team Contribution**](#_yl43pniaiuz) **1**

[**Hardware**](#_a0z3oulgute) **1**

[**Software**](#_nk3b4r5w03x4) **1**

[**World Map/Icon**](#_t3iaa9yszc38) **1**

[**Challenges**](#_qnmi7jxxzpzh) **1**

**Github Link 1**

## 

## **Project Summary**

For our final project,

## **Team Contribution**

Hardware - Michael and Chelsea

Software - Jonathan, Michael, and Chelsea

World Maps/Icon - Chelsea

Project write up – Jonathan, Chelsea, and Michael

## **Hardware**

*Summary of the Hardware from the top Level*

The following is a breakdown of the hardware additions for this project. Each will be described in more depth in the following sections. Contributors are shown in parenthesis.

Top Level Design (Everyone):

* Accelerometer (Engineer: Michael Bourquin, Contributors: Jonathon Anchell, Chelsea Brooks)
  + Driver Module
  + Synchronizer
  + Testbench
* Maze Bot (Jonathon Anchell and Michael Bourquin)
* Level Select (Chelsea Brooks)
* MIPS System (Everyone)

**Accelerometer:**

The on-chip accelerometer (ADXL362) was used for the purposes of this project and two modules were derived using the datasheet. The SPI\_driver module outputs a roundDD signal telling the spi\_sync module that x,y, and z have been updated and the current temp values are valid on the line. The roundDD signal is held high until the driver begins writing new values to the temp x,y,z signals (approximately 8 SPI clock cycles), leaving plenty of time for the values to synchronize. When roundDD is high, output values from the SPI driver are saved into the MIPS sys registers. The spi\_sync module also holds the reset signal low when a reset is detected giving SPI clock time to latch in the reset signal. This must be done because the SPI clock is much slower compared to the MIPS sys clock. The SPI clock runs at 5.23 MHZ while the MISP sys clock runs at 50MHZ. An overview of the accelerometer hardware can be seen in figure 1 below.

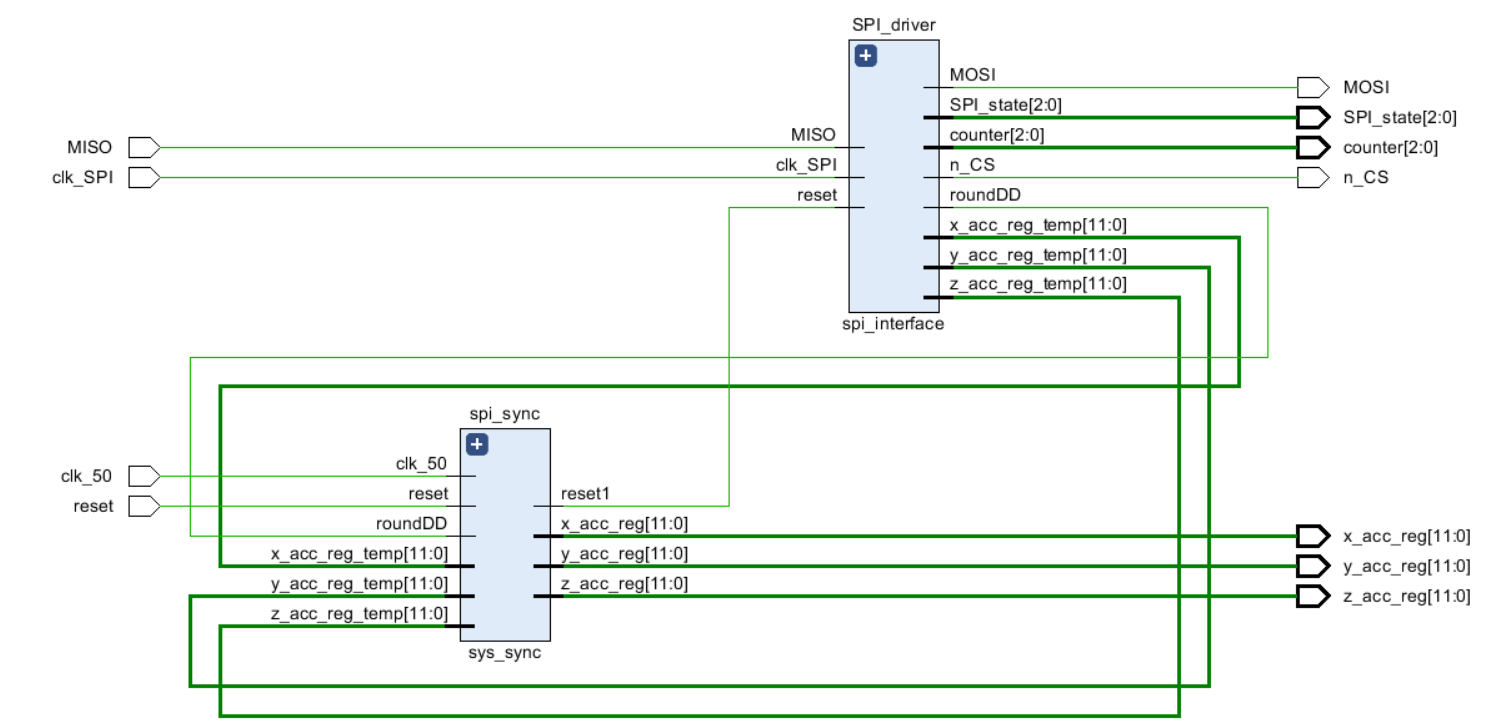
****

Fig. 1: Accelerometer Top Level

**Driver Module:**

The accelerometer driver module uses an FSM to read and write to the accelerometer chip. According to the datasheet there are 3 phases to each read and write. Each phase takes eight SPI clock cycles. They are the following in order.

* Instruction
* Address
* Data In/Out.

The accelerometer reads data on the positive edge and writes out data on the negative edge. The CS(Chip Select signal) is latched low to start a read or write and then latched high when finished. There must be time given before writing out the first instruction bit and the CS signal going low. There also must be time given for the CS signal to be held before starting another read or write. As the accelerometer chip writes out data on the negative edge, the driver module reads incoming data on the positive edge. It write out data on the negative edge. An eight bit width register (accel\_data) is used for data writes and saving incoming data reads as data bursts are 8 bits in length. It is reinitialized to zero at the end of every read/write.

Once a phase has begun, a counter is used to determine which bit to either save or write out in the accel\_data register. Once an entire read is complete, the accel data register is saved to the appropriate bits in the x,y,z registers using a case statement. A register keeps track of which address is currently being read from and is the case statement input. On a reset, a write instruction is done to start the accelerometer in measurement mode. This is followed by continuous reads from the x,y,z registers until another reset. Every time x,y,z is read the roundDD signal is held high (signaling valid accelerometer values) until the next instruction phase is complete (about 8 clock cycles). Once the next data in phase is complete, the x,y,z registers will no longer be valid as only first 8 bits are saved into 8 of the 12 bits of one of the x,y,z registers. They will not be valid again until the final bits in z are written.

Registers: Xlow(8bits), Xhigh(4 valid bits), Ylow(8bits), Yhigh(4 valid bits), Zlow(8bits), Zhigh(4

valid bits)

Instructions: Read and Write

From the above it can be seen that the accelerometer registers are a total of 12 bits in width each. Values are in two’s compliment with the MSB indicating negativity.

**Synchronizer:**

The accelerometer synchronizer module saves the accelerometer driver outputs every time they are updated and are valid (meaning they are not being updated in 8 bit bursts currently). When roundDD is high, the accelerometer driver outputs are valid. On detection of a system reset, the synchronizer also holds an output SPI reset register low for 64 clock cycles, giving the SPI module plenty of time to latch in a reset.

**Testbench:**

A testbench for the accelerometer driver was written previous to physically testing the system in order to make sure that the driver module matches the timing diagram from the ADXL362 datasheet as well as the chip select latch timing constraints. The saved time as it was much easier to see what was happening internally and also caused it to work on the first actual run.

The waveform output of the testbench can be seen in figure 2. MISO is the data input from the accelerometer chip (used on a write). It was tested with a constant 8 bit output of 0xFF. This waveform shows up until the final 12 bits are stored in the Y register. There is the initial write (measurement mode) followed by a read of X and Y. A read of X takes 16 clock cycles as both Xlow and Xhigh must be read to get the full 12 bits. All values are read in the default +-2g measurement mode. This is the most precise mode available. The yellow line shows the completion of the initial write.



Fig 2: Accelerometer Driver Testbench

**Maze Bot:**

The toad icon is modeled in hardware via this module. The module keeps track of the current location of the bot on the screen and updates the position based on speed and direction inputs from the MIPS system. The clock is used to simulate movement. There are three different speeds and each has a different counter limit. The smaller this limit, the faster the robot will move across the screen. The robot can move faster in X than Y and vice versa as each have a separate speed input.

When the robot hits a wall, it enters a deadlock (wait state) where it cannot move and it is not visible on the screen. This acts as an interrupt for the software to do other things when the user has either hit a black wall (loses a life) or hits a red wall (wins the level/game). There is a soft reset signal that the software can write to, which resets the bot into its initial state.

**Level Select**

**MIPS System Slave Additions:**

## **Software**

The

## **World Map/Icon**

The

## **Challenges**

We faced a few challenges over the course of this project.

## **Github Link**

https://github.com/codexhound/ECE540FinalProject