Class Report 4: Experiment 16.8.1

Alec Roessler

November 12, 2025

1 Introduction

The purpose of this report is to perform Experiment 11.9.4, (inclination). We have learned how to configure the hardware in the FPGA board and expanded the capabilities to interface with the processor. This project builds upon the basic vanilla system developed by Chu and uses the SPI interface design to communicate with the onboard accelerometer sensor. This data will be read, interpreted, and acted upon by the processor via software to illuminate specific on board LEDs corresponding to the orientation of the board in four cardinal directions (0, 90, 180, and 270 degrees).

2 Basic SPI Implementation

2.1 SPI Communication with the ADXL362

The accelerometer used in this experiment is the ADXL362, a three-axis MEMS device that communicates via the SPI protocol in Mode 0, transferring the most significant bit (MSB) first. Within the system, the FPGA functions as the SPI controller and the ADXL362 acts as the peripheral. Each transaction consists of a sequence of bytes: the controller first asserts the `ss_n` (peripheral select) line, sends an instruction byte (`0x0A` for write or `0x0B` for read), transmits the target register address, and then exchanges one or more data bytes before de-asserting the `ss_n` line.

The SPI core developed by Chu provides a complete set of driver methods to manage this communication. The class constructor configures the SPI frequency (100 kHz), sets the clock polarity and phase for Mode 0 operation, and initializes all slave devices to an inactive state. The `transfer()` method performs a full-duplex transaction: while data is written to the slave through the `MOSI` line, data from the slave is simultaneously received through the `MISO` line. In the case of the ADXL362, write transactions send data to internal registers, whereas read transactions use dummy bytes to retrieve the sensor's data registers.

According to the ADXL362 register map, the 8-bit signed acceleration values are stored as follows:

- X-axis acceleration: Register `0x08`
- Y-axis acceleration: Register `0x09`
- Z-axis acceleration: Register `0x0A`

The device identification number (`0xF2`) is located in Register 2, and can be used to verify successful SPI communication prior to reading the acceleration data. This had to be used during testing as the accelerometer was not responding correctly. The last step to check was ensuring the identification number was correct which it was not. This pointed to a code issue and the issue debugged.

2.2 Main Sampler

Alike the previous lab, we are supplied with a foundation for the project. In this instance a new project was created the same as in the vanilla project but the two top level files of mcs_top_vanilla.sv and mmio_sys_vanilla.sv were instead changed to mcs_top_sampler.sv and mmio_sys_sampler.sv.

Next, the main sampler cpp file was modified so as to delete many of the unnecessary test functions not pertaining to this project and keeping the core accelerometer test function. After this, the spi.sv and spi_core.sv files were added to the Vivado project. These had to be manually typed referencing chapter 16 in the book as the files included in Chu's zipped folder were Vivado header files and thus would not be applicable.

2.3 HDL Modifications

After incorporating the required SPI files into the project, several hardware-level modifications were necessary to ensure proper configuration and integration. The following adjustments were made:

- **Remove Unused Slot Instantiations:** In `mmio_sys_sampler.sv`, all slot instantiations not required for this project (e.g., UART and other peripherals) were deleted to simplify the design and reduce unnecessary logic.
- **Instantiate the SPI Core Module:** The `spi_core` module was instantiated in **slot 9**, providing the interface between the processor and the onboard accelerometer via the SPI protocol.
- **Assign Default Values to Unused Slots:** In `mcs_top_sampler.sv`, the `generate` block was modified to assign zeros to all unused slot read-data ports. This prevents floating signals and ensures stable operation of the read register array. This was accomplished with the following code.

```
// Assign 0's to all unused slot rd_data signals
generate
    genvar i;
    for (i = 5; i < 64; i = i + 1) begin
        if (i != 9)
            assign rd_data_array[i] = 32'h0;
    end
endgenerate
```

2.4 Accelerometer Testing

The Vivado project was then built and exported to be launched in Vitis. The was then ran and tested by simply outputting the 8 bit z-raw signal to the first LEDs on the board according to the binary representation of the data. This was a simple test to ensure the accelerometer was working correctly as well as the SPI communication and ability to extract the data.

Each x, y, and z value is read from the accelerometer register as an 8-bit signed integer. When the board is oriented at 90 degrees, the z-axis raw value (z_raw) typically approaches its positive maximum (around +127 for an 8-bit signed value), indicating the direction of gravitational pull. As the board rotates past 90 degrees toward the opposite orientation, the z_raw value transitions through zero and eventually approaches its negative maximum (around –128). During testing, an observed raw value of 11000011 corresponds to –61 in two's complement form, confirming that the accelerometer outputs signed data rather than unsigned. This understanding is essential for correctly interpreting the sensor readings and mapping them to LED indicators representing the board's inclination angles.

3 Software Integration for Full Capability

The software portion of this project is responsible for reading the accelerometer's raw data values and interpreting them to determine the board's orientation. The accelerometer outputs three 8-bit signed integers corresponding to the x, y, and z axes. These values range from approximately -128 to $+127$, representing acceleration from $-2g$ to $+2g$.

3.1 Main Sampler Code Modification

To interpret these readings meaningfully, each raw value is normalized by dividing by half of the maximum possible reading. This scaling converts the integer values into a floating-point range from -1.0 to $+1.0$, representing the relative acceleration along each axis:

```
const float raw_max = 127.0 / 2.0;  // 128 max 8-bit reading for +/-2g

x = (float) xraw / raw_max;
y = (float) yraw / raw_max;
z = (float) zraw / raw_max;
```

Once the normalized readings are obtained, the system determines orientation by comparing the raw axis data against a defined threshold. A threshold value of `40` was chosen to filter out minor fluctuations and noise inherent to the accelerometer readings. This ensures that only significant changes—corresponding to a clear physical tilt of the board—trigger LED updates. In practice, this value provides a balance between sensitivity and stability.

```
const int THRESHOLD = 40;

// Check orientation and light the corresponding LED
if (yraw > THRESHOLD) {          // +1g on Y-axis
  led_p->write(0x01); // 0 degrees (LED 0)
} else if (xraw > THRESHOLD) { // +1g on X-axis
  led_p->write(0x02); // 90 degrees (LED 1)
} else if (yraw < -THRESHOLD) { // -1g on Y-axis
  led_p->write(0x04); // 180 degrees (LED 2)
} else if (xraw < -THRESHOLD) { // -1g on X-axis
  led_p->write(0x08); // 270 degrees (LED 3)
} else {
  led_p->write(0x00); // No significant tilt
}
```

The main program loop is then simplified to repeatedly call the accelerometer check function and update the LED display at a steady rate:

```
int main() {
  while (1) {
    gsensor_check(&spi, &led);
    sleep_ms(50);
  } // while
} // main
```

3.2 System Operation Summary

In full operation, the processor continuously polls the accelerometer via SPI, processes the incoming data, and determines orientation using the threshold comparison method. The system achieves a closed-loop interaction between hardware and software: the FPGA hardware handles the SPI communication at the hardware level, while the processor interprets the data and controls the output LEDs through software. This experiment demonstrates how embedded hardware and software can cooperate to create a responsive real-time sensing system.

4 Submission Information

The Github repository with the files mentioned in this report can be found at
`https://github.com/alecroessler/Softcore-SoC/tree/main/Experiment_16.8.1`
Youtube video demonstration:
`https://www.youtube.com/shorts/490yaIy0z90`