UCD School of Electrical and Electronic Engineering

EEEN40280
Digital and Embedded Systems

System on Chip Design Report

Name: Aidan O'Sullivan **Student Number: 20209138**

Working with: Joshua King

Code submitted by: Aidan O'Sullivan

I certify that ALL of the following are true:

1. I have read the *UCD Plagiarism Policy* and the *College of Engineering and Architecture Plagiarism Protocol*. (These documents are available on Brightspace.)
2. I understand fully the definition of plagiarism and the consequences of plagiarism as discussed in the documents above.
3. I recognise that any work that has been plagiarised (in whole or in part) may be subject to penalties, as outlined in the documents above.
4. I have not already submitted this work, or any version of it, for assessment in any other module in this University, or any other institution.
5. The work described in this report is all the original work of my team, and this report is all my own work, except where otherwise acknowledged in the report.

Signed: Aidan O'Sullivan **Date:** 27 April 2021

Details of what is expected in the report are included in the assignment instructions. The headings below are only suggestions – you may use a different structure and/or cover the topics in a different order if that makes more sense for your design or your report.

You should remove this blue text from your report.

You should save your report as a pdf document, and upload it through the report submission channel in Brightspace.

Your report must begin with the declarations above, ideally laid out as above. The signature is not essential – by submitting your report through Brightspace, you are deemed to be making these declarations.

One member of the team should upload the code through a separate channel in Brightspace. Upload your original source file(s) in C, Verilog, etc. as used in your uVision project and in Vivado. There is no need to upload files that you have not modified. There is no need to upload an entire project folder. If you have many files to upload, you may combine them into one zip file and upload that.

Introduction

In this assignment, we were asked to measure acceleration using an ADXL362 accelerometer. This device is capable of 3-axes measurement and can exchange information with the processor via SPI communication. These connections are made by configuring the GPIO blocks using Vivado. As there were two members in the group the SPI was controlled from the software. Several configurations on the accelerometer could be used to provide additional design options.

Design Process

We broke the problem into several blocks.

In Hardware:

- UART and GPIO block integration.
- Physically connecting the accelerometer and other components to the GPIO using Vivado.
- The display hardware block integrated into the system.

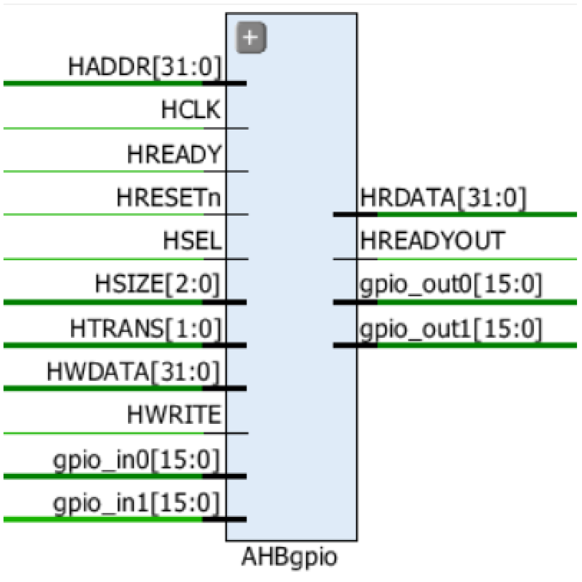
In Software:

- Receiving and sending data to the accelerometer
- Displaying the accelerometer data on the 7- segment display.
- Adding additional features and user interface, to allow the user to choose different settings.

The blocks that I worked on were all hardware parts, the accelerometer interfacing in software and the additional features.

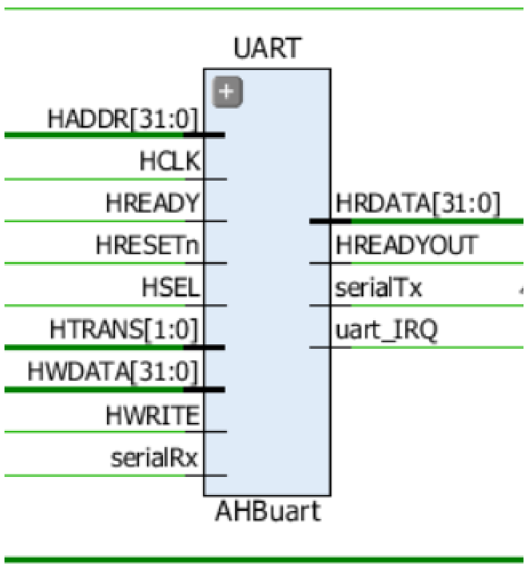
GPIO and UART integration

The first task undertaken to facilitate subsequent development was to integrate the GPIO and UART blocks. The GPIO provides input and output connections to auxiliary components such as the LEDs, switches and the accelerometer, using two 16-bit input and output ports. The UART allows USB connection from the Nexys board to the PC to program the board by transmitting and receiving serial data.

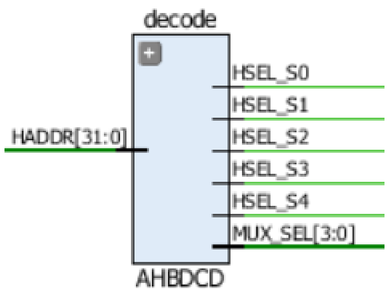


The 16 on-board switches were assigned to input 0 of the GPIO and the 16 LED were connected to the entire output 0 port. Output port 1 was used for the SPI connections between the GPIO and accelerometer. The slave select, clock signal and MOSI occupied one bit each of the output

1, with the remainder being unused. In AHBTop these wires were added to the GPIO block after being already created and uncommented in the constraints file. The MISO was added as a 1-bit space in the GPIO input 1. The remaining 10 bits that were not occupied by the MISO or buttons had to be filled with null bits to ensure that Vivado would compile the design.



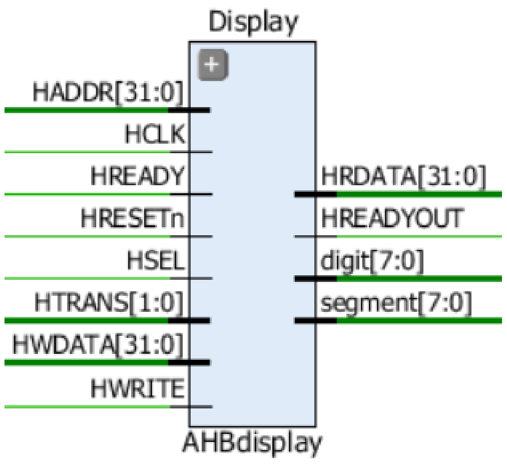
In AHBTop a UART block was added to allow serial communication with the board so programs could be uploaded to utilise the design from software. The serial transmit port RsTx and serial receive port RsRx, along with the 1-bit interrupt expected by the software, were connected to the top level module.



The decode block was edited to map the slaves to their appropriate places in the address map, with the GPIO and UART being mapped to available address at 0x50000000 and 0x51000000 respectively. This address map interfaces with the software using pointers that reference this area of the memory map. It can be broken down further using structs to write to individual input and output port in the GPIO. Their respective slave numbers are used by the multiplexer to control signals going back to the processor. The slaves are selected using an individually assigned one-got code.

Connecting the display block

The display block was connected to the top-level module similar to the GPIO and UART blocks. It was given a new address, slave number and one-hot code for when it is needed to be selected as slave. The address is added to the memory map and can be referenced using pointers in the software to write to the display. Its pins that were already defined in the constraints file were uncommented. Two 8-bit wide outputs digit and segment were added to the block. The 7-segement display is communicated with through the AHB-lite bus.



Receiving and sending data to the accelerometer

This was the first software block undertaken. The accelerometer had to be configured using registers to control its modes. This initial step would require purely write data, involving the chip select, clock and MOSI. As per the data sheet, data was transmitted in the following order.

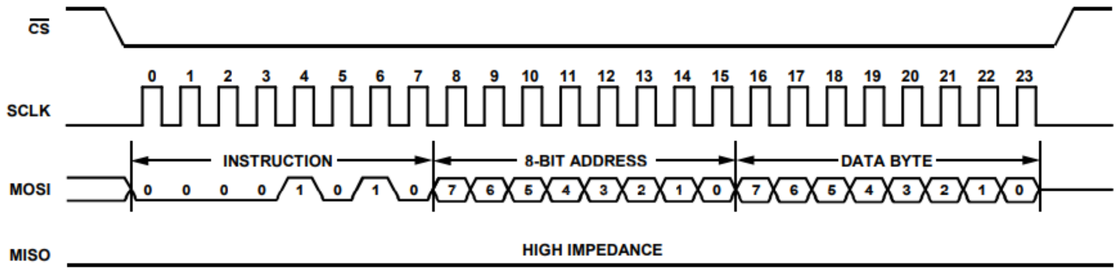


Figure 37. Register Write (Receive Instruction Only)

The chip select goes from high to low to begin the sequence. During the transmission of 1-bit on the MOSI line, the clock will go from low to high, with the bit being transmitted at the falling edge of the clock at the end of one clock cycle. Three bytes are transmitted to the accelerometer, with MSB transmitted first and LSB last. The first byte that is sent 0x0A to indicate that data will be written to the ADXL362. This is followed by the register's address that will be written to. The data is the final byte.

```
void send_bit(uint8 mosi_bit, uint8 clock_state){
   pt2GPIO->acc_out = clock_state<<2 | mosi_bit<<1;
}
void bit_gen(uint8 byte2send){
   volatile uint8 i;
   volatile uint8 bit2send;
   for(i=0; i<8; i++){
      //Clock set low
      bit2send = (byte2send>>(7-i))&1;
      send_bit(bit2send, clock_low);
      //insert delay and change the state of the clock to HIGH
      wait_n_loops(delay_val);
      send_bit(bit2send, clock_high);
      wait_n_loops(delay_val);
   }
}
```
The section of code are the functions used is used to write each byte to the accelerometer. The bit_gen() function strips the MSB off the byte until all have been sent. Each bit is put on the MOSI line while the clock is low and then while high. This is implemented using the send_bit() function. The clock and MOSI bits are shifted onto their corresponding positions of the Output 1 address map to be sent to the ADXL362.

The power control register was written to in the accel_setup() function, to put the device in measurement and low noise mode. The accelerometer was also configured to be in the measurement range ±2 g and the lowest output data rate of 12.5 Hz, which was sufficient for this application, but these were already set by default in the filter control register.

Receiving data followed the sequence seen below:

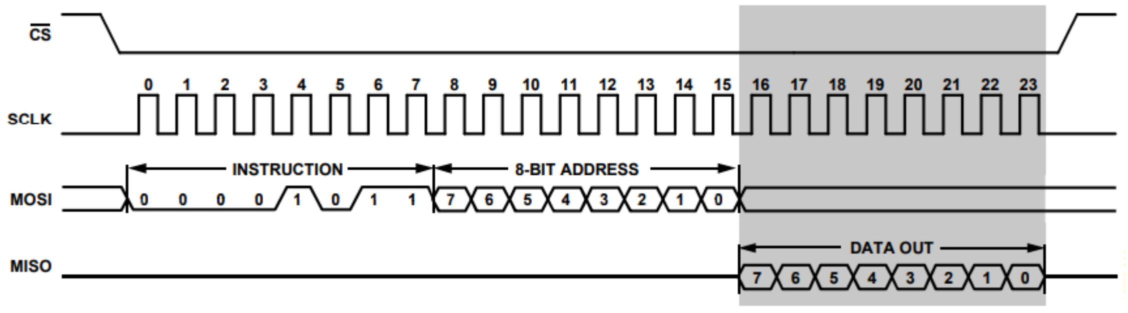


Figure 36. Register Read

First two byte were sent on the MOSI line as described above. The order used was a read byte 0x0B, followed by the address of the 8-bit register to be read from. This was either the X, Y or Z axis address registers, depending upon the axis that data was chosen to be read from. The third byte was the acceleration data received from the ADXL362, MSB first and LSB last. This process was implemented using the functions in the following section of code:

```
void mosi_clock(uint8 clock_state){
  pt2GPIO->acc_out = clock_state<<2;
}
uint8 read_miso(){
  volatile uint8 r_bit;
  //clock low, delay
  mosi_clock(clock_low);
  wait_n_loops(delay_val);
  r_bit = pt2GPIO->Buttons>>5;
  //clock high_delay
  mosi_clock(clock_high);
  wait_n_loops(delay_val);
  return r_bit;
}
uint8 byte_gen(){
  volatile uint8 bit;
  volatile uint8 byte;
  volatile uint8 i;
  for(i=0; i<8; i++){
    bit = read_miso();
    byte = (byte)| bit<<(7-i);
  }
  return byte;
```

The byte_gen function is called from the main(), which receives each bit of the byte from read miso during every clock cycle and assembles it into a full data byte. The first bit is set as the MSB and the last as LSB. The read_miso() function is called within this function to retrieve the bit during each cycle of the for loop. The clock is first set low and delayed. The 6th bit of the GPIO input 1 is read using the BUTTONS pointer, with the bit being received on the line on the rising edge of the clock. The chip select is again set high to end this sequence after the full byte is received.

Displaying the accelerometer data on the 7- segment display

The accelerometer value was converted from its bit representation to mG using the following logic, given and ±2 g range and signed in:

$$scaled\ value = (acceleration\ Bit\ Value) * \frac{2000}{128}$$

The set_display() function took this scaled value and displayed it in hex, with the + or – sign indicated. The control register was used to turn on the six rightmost digits, with the leftmost and two rightmost digits that were turned on set in raw mode. This was so the negative sign and mG could be displayed. The remaining three digits were used to represent the magnitude of the acceleration. This was done in hex mode. As 4 bits were used in hexData per register the value had to be shifted 8 places because the two rightmost digits were used to display the units. The value was also checked to see if it was + or - , to decide if the negative sign had to be displayed or not.

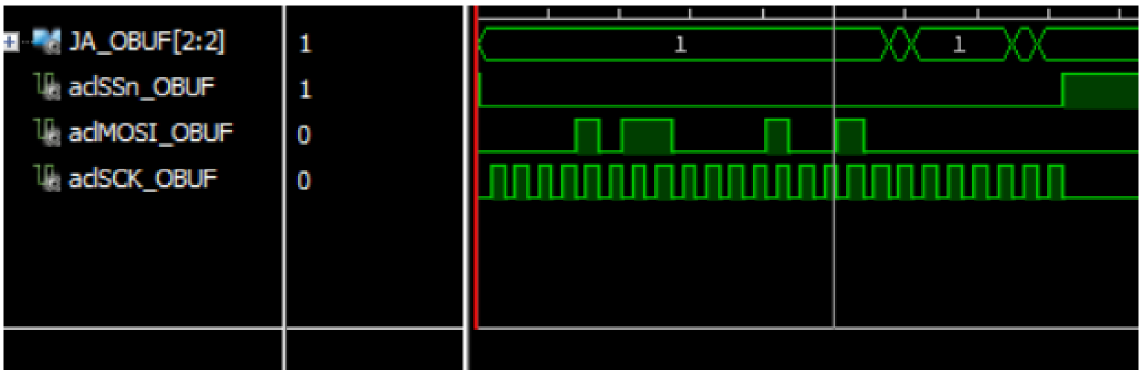
```
void set_display(int32 value){

        int control;

        //value = 0x7D0<<8;

        value = value<<8;

        if (value < 0){

                pt2Display->rawHigh = 0x200; //minus

                value*= -1;

                control = 0x3F1C00;

        }

        else{

                control = 0x3F3C00;

        }

        pt2Display->rawLow = 0x155BC; //mG

        pt2Display->hexData = value;

        pt2Display->control = control;

}
```

User Interface

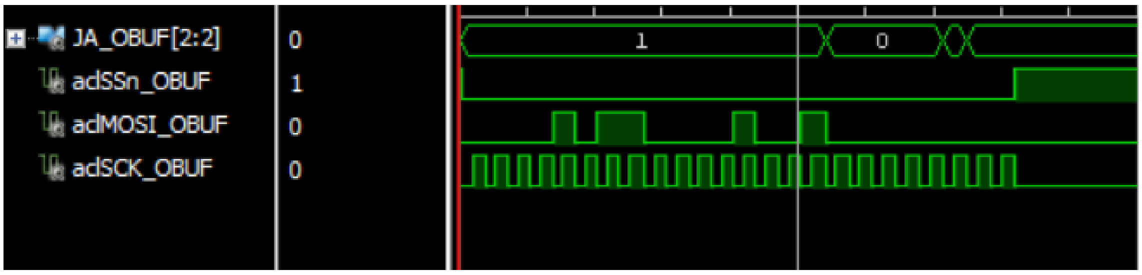
When the program is started the terminal will show a series of messages explaining to the user how they can configure the accelerometer. To select an axis of measurement a switch can be pressed. The remaining switches are masked so that the program will only be sensitive to the rightmost three switches. The acceleration value will also be my shown in mG in the terminal. If a certain threshold is exceeded, the terminal will inform the user that high impact was detected and also display the number of times that this has occurred. The LEDs on the Nexys board will also light up one at a time, to correspond with the current acceleration. Close to zero mG the middle lights will illuminate, while positive accelerations will light up further to the left according to the magnitude, while negative acceleration will move to the left.

Testing

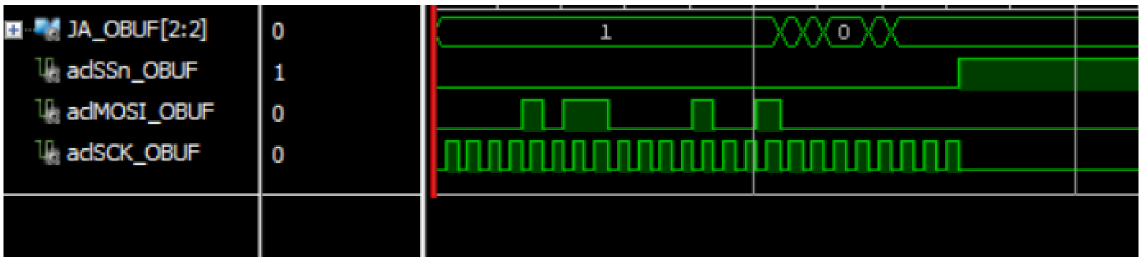
The SPI signals were tested using the logic analyser. Registers 0 and 1 were fixed so these were read to ensure that the information was being transmitted and received as per the timing diagram. When we were satisfied that these were correct the ADXL362 signals were tested in the y-axis. The board is firstly tilted to the right fully, so it was standing on edge.



The first byte corresponds to the read data 0x0A and the second byte is the Y-axis address 0x09. The MISO byte has a 1 as the MSB meaning that it is in the negative direction in two’s complement and equal to -67 bits.



When the board was flat the following signal was obtained. At 5 it is close to 0 and the offset may be due to the table not being flat or due to a lack of calibration.



The board was then tilted fully to the left and the signal above was obtained. This was 72 bits. For turning the board on this axis without shaking it this range was to be expected, with a small offset. This test showed that the accelerometer was transmitting the data correctly. The values being obtained in software were printed to the terminal and compared to ensure that they matched to check that the software was also assembling all the bits correctly in the functions to form bytes of acceleration data. This process was checked for the x and x axes.

To test the on-board display the hex values were read and converted to decimal and compared with the values displayed in the terminal to ensure that they were correct.

Conclusion

Both group members had no experience with FPGAs prior to this assignment so new expertise were gained. An understanding of how hardware can be controlled was developed. Vivado's RTL diagram made the connections and hardware blocks understandable. A working model was developed, where accelerometer readings were read in the axis specified by the user and displayed in decimal in the terminal and in hex on the on-board display. Both the hardware and software blocks were divided but an understanding of both was required to address the blocks appropriately and access them from the software program to complete the design. Extra features were added that gave the user control over what was being measures, as well as a detection of high impacts.