ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

ACCELEROMETER MODULE FOR AN FPGA DEVELOPMENT ENVIRONMENT

ROBOTIC'S PROJECT I

STUDENT : LOIC FINETTE
PROFFESOR / SUPERVISOR : ALEXANDRE SCHMID

7th June 2024

Contents

1 INTRODUCTION

1.1 GOAL

The goal of this project is to make the accelerometer present on the DE10-Lite easily usuable through Logisim. This is to allow the students of the EE-110, EE-207 Systèmes Logiques class [2] to use this module for their projects. This main goal is split into 4 sub goals:

1. Implement a driver for the accelerometer
2. Make a test bench to show the information from the driver on the FPGA.
3. Make a simulator to show the information from the driver in Logisim
4. Create an application that uses the accelerometer driver

The complexities of the accelerometer should be abstracted to allow the students to simply use the accelerometer driver in a plug and play fashion for their aplication.

1.2 ADXL345 ACCELEROMETER

The accelerometer present on the DE10-Lite is the ADXL345 accelerometer. The purpose of the accelerometer sensor is to sense the acceleration that the device is experiencing. An accelerometer placed flat on a table should sense an acceleration of 1g, with g being the gravitational constant $g = 9.81m/s^2$, along the z-axis. In free fall, the acceleration should read 0g along the z-axis. The acceleration is sensed thanks the change in capacitance detected from the free moving masses on each axis. This can be observed in Figure 1. From the data sheet [3] we know that it is a 3 axis accelerometer. The axes are represented in Figure 2. The data on each axis is represented using two's complement on 16 bits. The module can be controlled using SPI or I2C communication. It can have the following ranges : $\pm 2g, \pm 4g, \pm 8g, \pm 16g$. By default it has a resolution of 10 bits that can be increased to 13 bits.

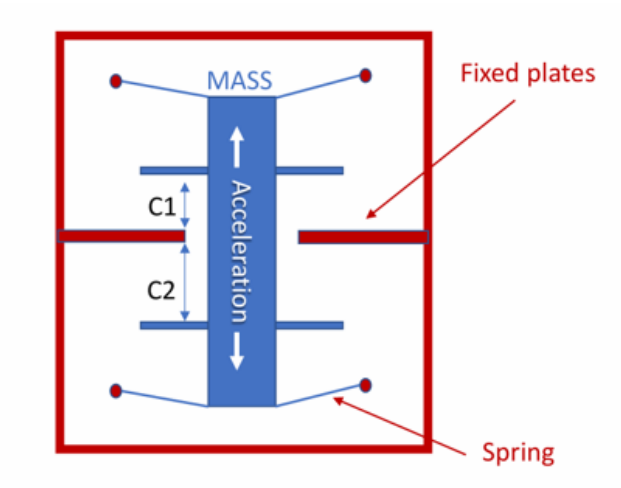


Figure 1: Free moving mass [7]

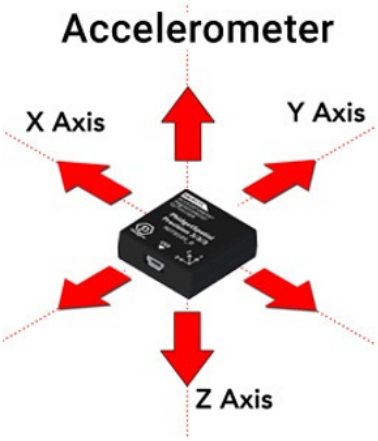


Figure 2: The 3 axes of the accelerometer [5]

2 State of the Art

A few projects using the ADXL345 have already been realized on the DE10-Lite. There exists a tutorial [6] in Verilog made by Doctor Mark Hildebrand and Professor Bevan Baas for the EEC180 Digital Systems II class at the University College Davis. There also exists an implementation [4] by Engineer Blake Johnson in VHDL from the University of Florida. Both implementation simply output the value of the acceleration read by the sensor on the hexes. One can switch between the different axes using the dip switches. Initially the Hildebrand/Baas implementation was preferred since it was very well documented and it was an easy process to get the code to work on the DE10-Lite. However, Logisim only accepts VHDL code and not Verilog. It would have been possible to attempt a translation of the Hildebrand/Baas code in VHDL but an implementation in VHDL by Johnson already existed. Therefore the development of the module was done based on Blake Johnson's SPI_Accelerometer code [4]

3 Project Overview

The workflow of the project was as follow:

1. Environment Setup
2. Study of Johnson's code
3. Running the code on Quartus
4. Running the code on Logisim
5. Implementation of a basic driver
6. Implementation of the final driver
7. The Application: Get to Zero
8. Creating the simulator

The required software was first installed. Then Blake Johnson's code was studied and adapted. His code was initially ran on Quartus to validate that it indeed worked. Then that code was implemented in Logisim and a basic driver was made to output the raw values of the accelerometer. Then these values were cleaned to create a usable driver. To show how this final driver can be used, an application was made. Finally, for students to understand how the final driver works without having to load the test bench on the DE10-Lite (a time consuming endeavor) a simulator was created.

4 Environment Setup

The required softwares to dowload are the ones outlined in the EE110-EE207-2023_SW01_v3.1.pdf document. These are the softwares that every student in the class should download on their PC. The choice was made to not add additional software in order to not complexify using the accelerometer. The software needed to run the module are the custom fork version of Logisim-Evolution 3.7.2, a java version >= 16, Quartus Prime Lite version 21.1 and the USB-Blaster Driver. Furthermore a new map called TERASIC_DE10LITE_GSENSOR.xml has been created that includes the pins of the accelerometer. A guide on how to use this new mapping and the accelerometer is provided in the appendix.

5 Johnson's code

Mr. Johnson chose to communicate with the accelerometer using SPI because of the speed advantage and his general interest for the protocol. The files available on Mr Johnson's github repo are summarized as follow:

File List

Files	Descriptions
accel_driver	Configures sensor functionality, calibrates sensor, reads and writes to sensor on command and external interrupts
clock_div	Generates spi clock signal on command for x amount of bytes and raises flag per byte transaction
clock_div_tb	Test bench for clock_div.vhd
decoder7seg	Converts nibbles to active low 7 segment displays on the board
spi_master	Initiate transactions, controls data and cs lines, takes in transmit data, returns receive data
spi_master_tb	Test bench for spi_master.vhd
top_level	Top level entity used for accel_driver and spi_master to interact as well as external signals to the ADXL345 sensor, switches, and 7 segment display
top_level_tb	Test bench for top_level.vhd

Figure 3: File summary of [4]

For the driver, the files that were used are the accel_driver, clock_div, spi_master and top_level. A lot of Johnson's code was removed to only get the raw value of the accelerometer.

5.1 SPI

SPI allows devices to communicate with a master-slave relationship [8]. In our case we have one master (The MAX10 chip) and one slave (The ADXL345). Johnson uses the SPI in 4-wires mode. The communication is enabled between the slave and the master when the CS line is set to 0. SPI uses an SCLCK line to synchronize the communication. The master sends information to the slave using the MOSI line (GSENSOR_SDI in the DE10 Lite data sheet [10]).The slave sends information to the master using the MISO line (GSENSOR_SDO in the DE10 Lite data sheet [10]). Also, one of the interrupt lines (GSENSOR_INT1) is used in Johnson's implementation.

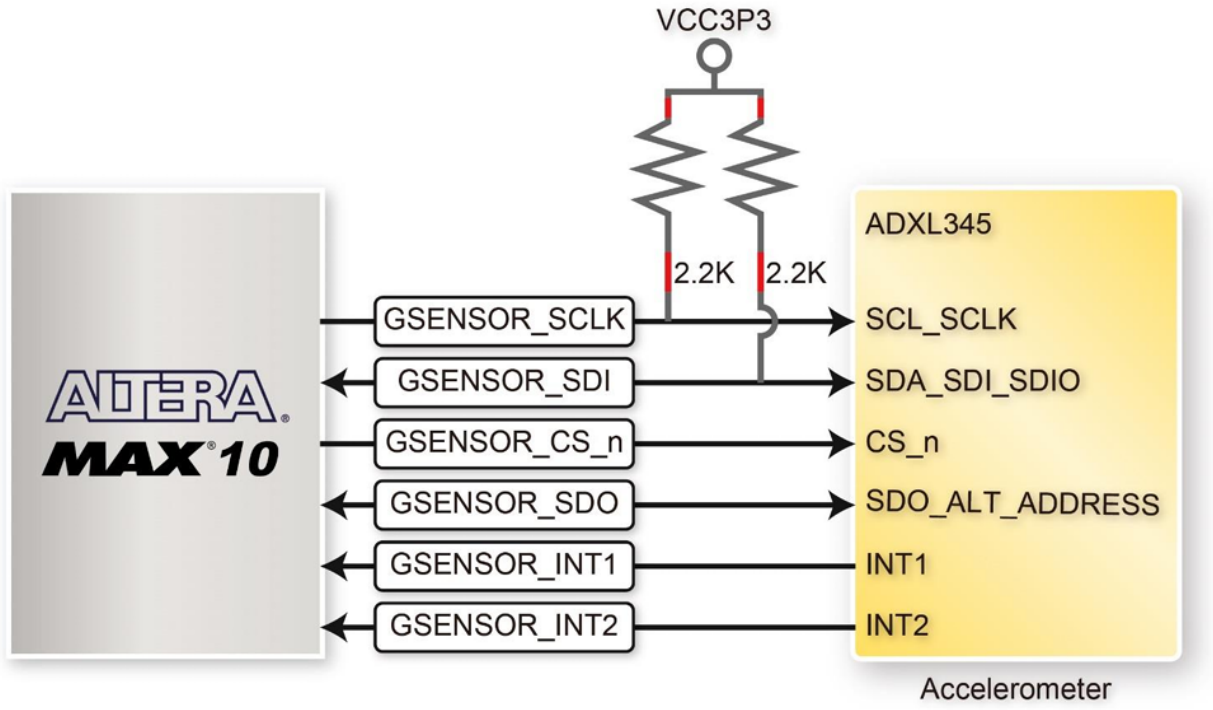


Figure 4: Connection between the accelerometer and the MAX10 [10]

6 Quartus Challenges

Before implementing the driver in Logisim, Johnson's code had to be validated on Quartus. The most difficult part of getting the code to run was that the pin mapping was not provided and Johnson did not use a standard nomenclature for his variables. Furthermore a good portion of the code is uncommented. Through trial and error the correct pin mapping was found and the raw value of the accelerometer was outputted on the hexes.

7 Logisim Basic Driver

Once it was confirmed that Johnson's code worked on Quartus, it was time to move on to Logisim. The initial goal was to attempt to obtain the same result on Logisim that was achieved on Quartus: have a driver that outputs the raw values of the ADXL345 on the hexes. Logisim needs VHDL components initialized in a particular way. The biggest issue with Logisim was that it could not find the definition of certain VHDL components. To remedy this, I based myself on Alexandre Schmid's application16x2display circuit and initialized the components the following way in the accel_driver_logisim circuit:

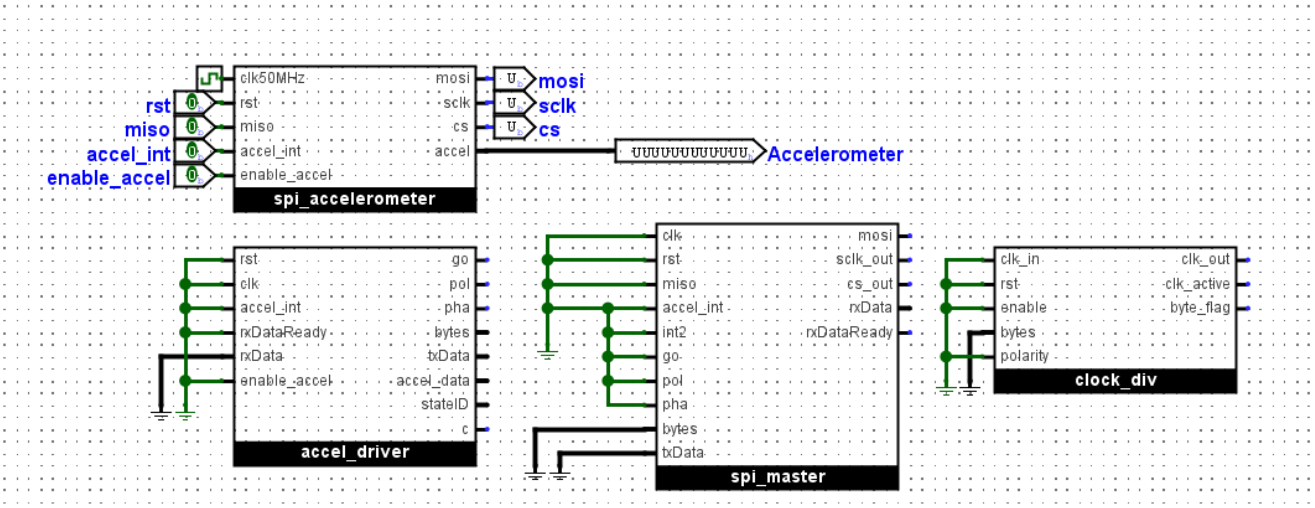


Figure 5: accel_driver_logisim : The top level spi_accelerometer and the components

The top level spi_accelerometer uses the accel_driver, spi_master and clock_div components. Then the following test bench for this basic driver could be built:

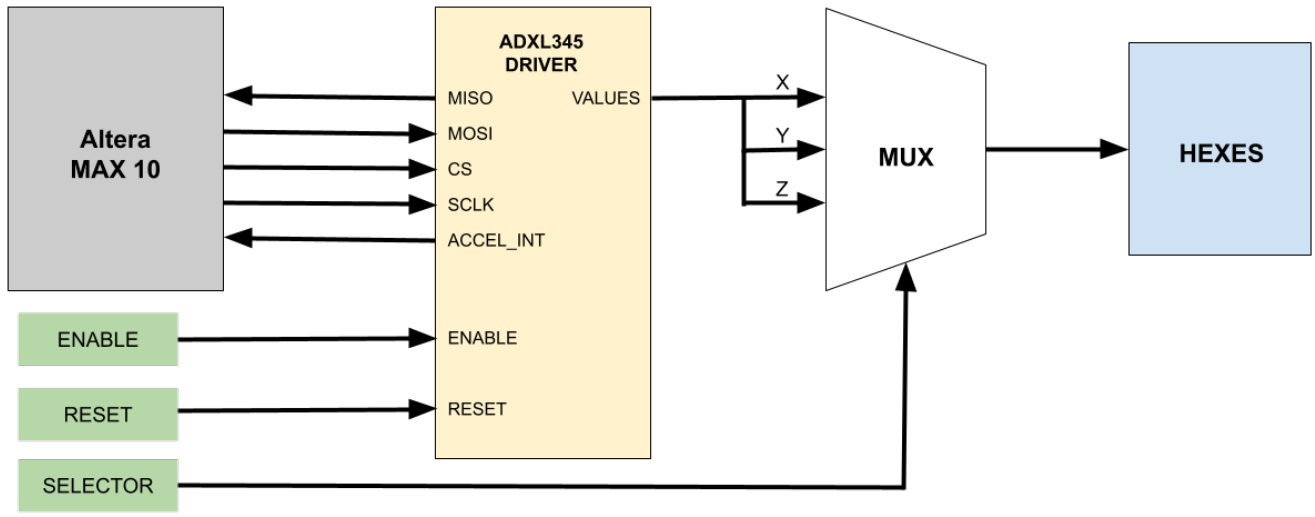


Figure 6: A test bench for the basic driver that outputs the raw value of the accelerometer

The ADXL345 Driver outputs 48 bits (16 bits per axis). Bits 0 - 15 are for the x axis, 16 - 31 are for the y axis and 32 - 47 for the z axis. Thanks to a multiplexer we can choose which axis to display on 4 hexes. We can choose the axis using the selector input that is mapped to two dip switches on the DE10-Lite:

Selector	Axis
00	X
01	Y
10	Z
11	Nothing

Table 1: The selector value and the corresponding axis

It is important to note that this test bench simply outputs the value of one axis on 4 hexes. The accelerometer constantly outputs the values of all 3 axes in the background. This property of being able to obtain the values on

each axes simultaneously will later be used in the application. The driver also has an enable and reset input. Here is an example of an output along the x axis with the DE10-Lite board laid flat on a table:

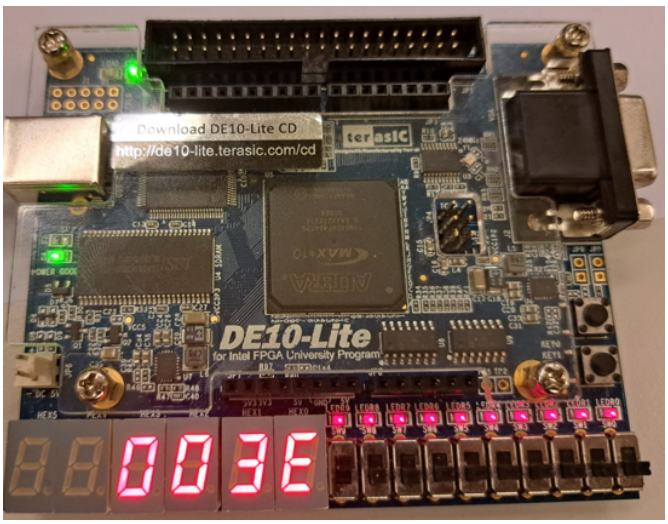


Figure 7: The raw output along the x-axis

8 Final Driver

The basic driver gives us something not usable or understandable. Furthermore the output of the sensor is very noisy and there is a clear offset present. To implement the final driver the following 3 corrections had to be applied:

1. Calibration
2. Scale
3. Filter

8.1 Calibration

In figure 7 the board is laid flat on a table. The value should be 0 since there is no acceleration along the x-axis but we can observe that there is an offset of 0x3E. There is also an offset along the other two axes. A sensor can develop this drift through time. This error needs to be corrected. The offsets can have a different value with different ADXL345 so these values cannot be hard coded. The final driver will have three offset inputs that it will subtract from each axes. To determine the value of the offsets a circuit called calibration_tb was created. The circuit works by allowing the user to input an offset in binary using 8 dip switches. The axis can be chosen using 2 dip switches (SW0 and SW1 in the example below). On the 2 left hexes (HEX4 and HEX5) the user can see the offset that they are applying. On 3 hexes (HEX0 - HEX2) the user can see the result of the applied offset on the accelerometer's values in decimal. It is worth noting that the value seen from the accelerometer on HEX0 - HEX2 have been scaled and filtered, two topics that will be covered later. The axis selector operates the same way as described in 7 Logisim Basic Driver. The value seen by the accelerometer is in $g = 9.81m/s^2$. The user needs to find the offset in hexadecimal that achieves an acceleration of 0 on the x-axis when the board is laid flat. The same goes for the y-axis. For the z-axis the acceleration should read 1.00g since the board is experiencing the acceleration from gravity.

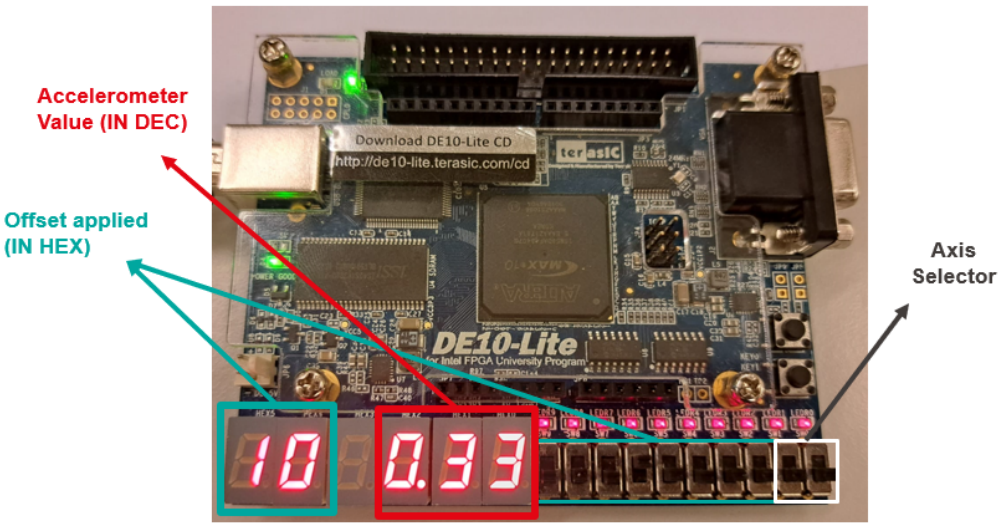


Figure 8: The calibration_tb circuit on a DE10-Lite

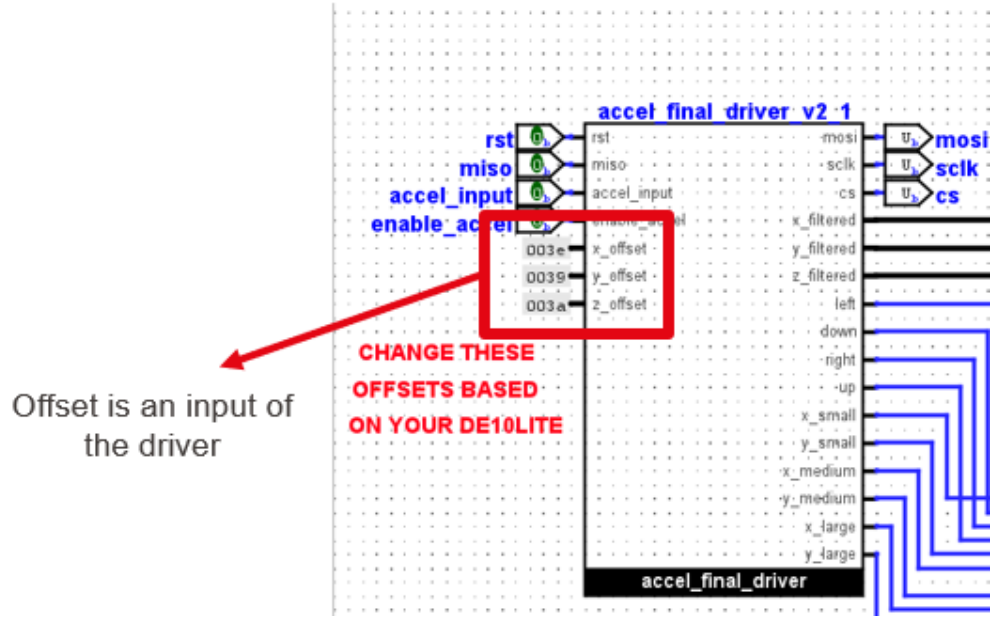


Figure 9: In the red box: where the user can enter the offsets that they found using the calibration_tb circuit in the final driver

Please find below a diagram of how the calibration_tb circuit works. For simplicity's sake, the drawing is made only for the x-axis. The drawing would be the same for the other two axes and the real implementation contains the selector and the MUX just like in figure 6.

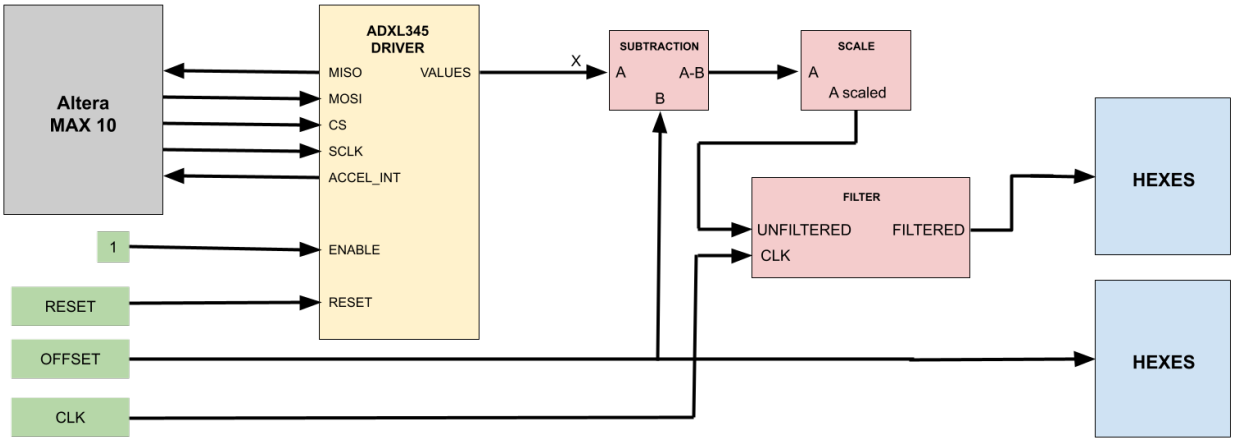


Figure 10: Diagram of calibration_tb circuit

For this circuit the enable of the ADXL345 Driver is always active so it is set to a logical 1. The acceleration along the x-axis is subtracted by the offset applied by the user. That value is then scaled and filtered. Finally it gets outputted to the hexes. The value of the offset applied also directly get outputted on the hexes.

The calibration_tb circuit was tested on 3 different DE10-Lite and the results of the offsets found are summarized below:

DE10LITE	x offset	y offset	z offset
001	0x003E	0x0039	0x003A
002	0x003E	0x0039	0x0030
003	0X0039	0X0031	0x0038

Mean = 56.22
SD = 4.89

Figure 11: Summary of calibration tests on 3 DE10-Lite

These results show that the offsets on each axis hover around the same value. The standard deviation is not too high which indicates that the values are close to the mean.

8.2 SCALE

To be able to understand the values that the accelerometer is giving us, we need to scale them down. We want to use $g = 9.81m/s^2$ as our unit because that is the standard unit of measure for this sensor. We want to be able to detect tilt. Let's take a look at the x-axis. When the board is flat on a table the acceleration is of 0.00g. When the

board is rotated at 90 degrees counter clockwise, the value displayed on the board should be of 1.00g since we are now experiencing the acceleration of gravity. Having the board fully vertical, I noticed that the value displayed was of around 128. Rotating the board 90 degrees clockwise from the initial position of zero degrees, I noticed that the value being displayed was of -128. The same was true for the y-axis. Therefore the value given by the accelerometer needs to be divided by 128 to obtain a range between -1 and 1 when tilting the board.

8.2.1 DIVIDING TWO NUMBERS

An initial attempt was made to use the division block available in Logisim but, the block was not synthesizable. Through research, I was able to find the following trick [9] to divide on an FPGA using a logical right shift:

$$\frac{x}{y} = x \cdot \frac{2^{15}}{y} \cdot \frac{1}{2^{15}}$$

The division by 2^{15} is equivalent to a logical right shift by 15 places. In our case the value of x is the value given by the accelerometer and the value of y is 128. Therefore our equation becomes:

$$\frac{x}{y} = x \cdot 256 \cdot \frac{1}{2^{15}}$$

The problem is that we are using a 16 bits two's complement representation and we get a number between 0 and 1. Therefore the number gets rounded to 0. To solve this issue, I decided to multiply the equation by 100 to get a resolution of 0.01. This yields the following equation:

$$\frac{x}{y} = x \cdot 25600 \cdot \frac{1}{2^{15}}$$

While attempting to implement this in Logisim, I found out that the right shift operator was not synthesizable. Therefore the implementation had to be done in a VHDL block called divide_two_num and is represented by the following diagram:

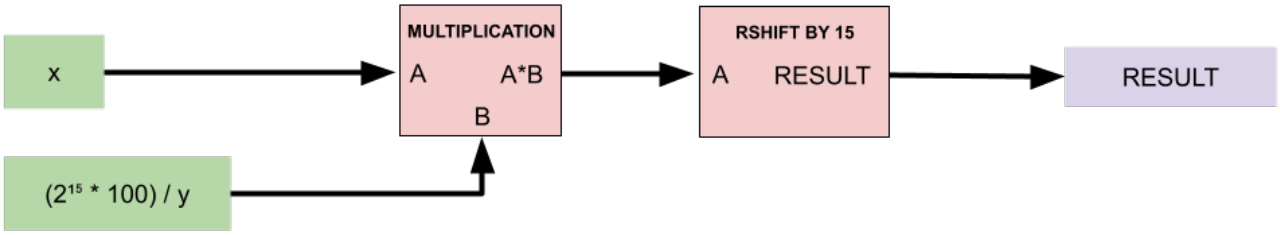


Figure 12: The div_two_num VHDL code in diagram

8.2.2 HEXADECIMAL TO DECIMAL

The accelerometer represents the value of each axis with 16 bits. Therefore it is easy to represent these values in hexadecimal on the hexes. However it is not a natural representation for humans since we are more used to seeing numbers represented in decimal. Therefore a module was implemented to convert from hexadecimal to decimal. The module extracts every digit of the number to later display them on the hexes. When displaying, I also activate the decimal point next to the hundredth digit to make it seem as if it a number between 0 and 1 (our range is 0g < |x| < 1g). The maximum value that this module was built to process is 999. This module was implemented in VHDL using the same division technique as mentioned in the previous section except this time we do not multiply by 100. In mathematical form here is how the algorithm works with x being the number that we want to extract the digits from. We assume that after every division, the result gets truncated to only the integer value:

$$\frac{x}{100} = hundreds$$
$$\frac{x - hundreds \cdot 100}{10} = tens$$
$$x - hundreds \cdot 100 - tens \cdot 10 = ones$$

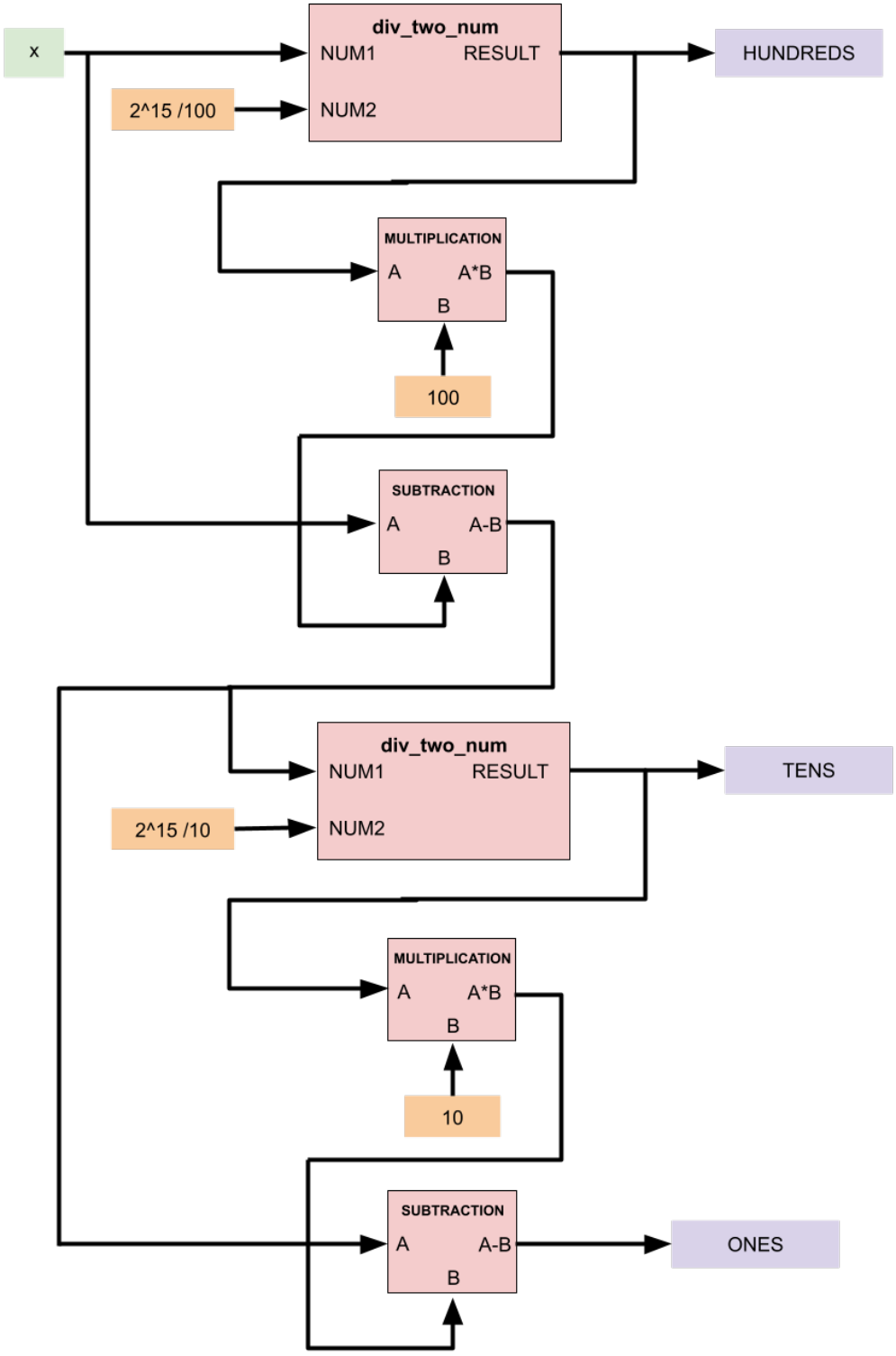


Figure 13: The hex_to_decimal VHDL code in diagram

8.3 FILTER

In order to reduce the noise on the sensor a low-pass moving average filter was implemented. As with all low pass filter, the problem is that it slows down the system. We want the system to have a quick response when the user makes a big movement but we want the system to have a precise response when the user makes a small angle change. Therefore when the angle change is less than 3.3 degrees we show the precise filtered value but if the angle change is bigger than that we show the fast unfiltered value. The diagram of the filter is found below.

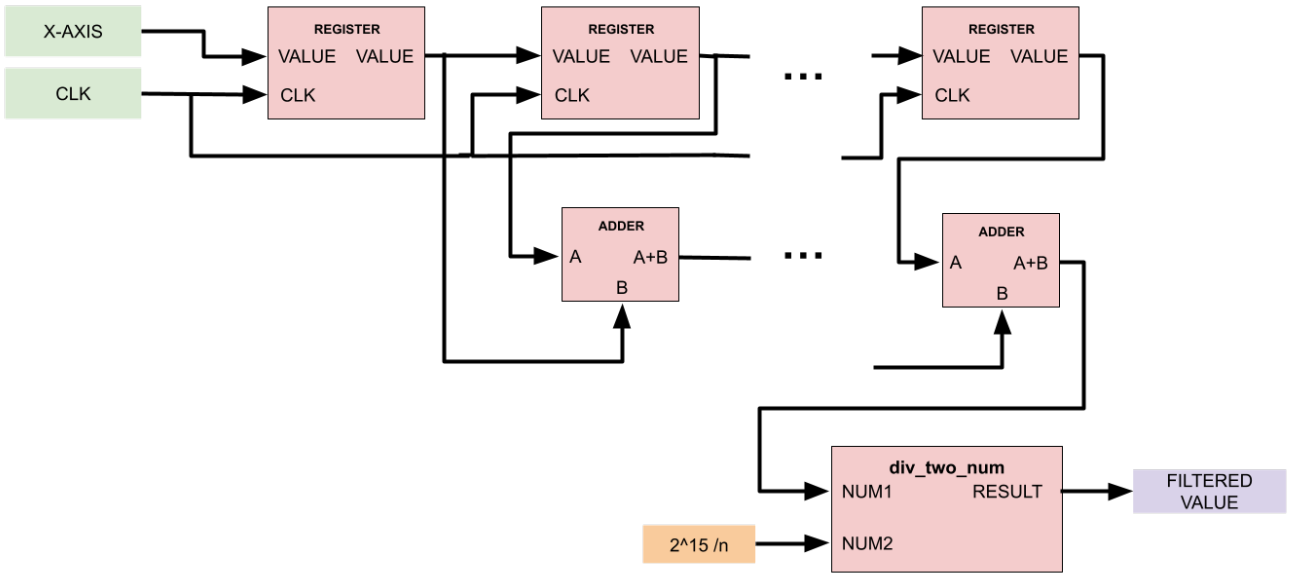


Figure 14: The signed_fast_filter diagram

In this diagram, the value of the x-axis comes from the AXL345 driver. Only the x-axis is represented for simplicity but the same logic is applied for each axis. The filter takes n samples that get saved into registers. Then the mean of the samples is calculated by adding them all together and dividing by the amount of samples. This gives us our filtered value.

To choose the amount of samples that I use in the moving average filter, I used the standard error. The system has a resolution of 0.01. Therefore if I am able to have a standard error of less than 0.01 it would be as if there is no error. The equation for the standard error is given by:

$$\text{steps } 1-5: \quad \sigma = \sqrt{\frac{\sum(x_i-\mu)^2}{n-1}}$$

$$\text{step } 6: \quad SE = \frac{\sigma}{\sqrt{n}}$$

Figure 15: The formula for the standard error [1]

The experiment will be conducted with the board laid flat on the table. The standard error along the x and y axis will be calculated. Since the board experiences no acceleration along the x and y axes, the mean is 0. Performing a sum and a multiplication is easy to do in Logisim. However, the square root and the division is harder to implement. Therefore, I will calculate $\sum x_i^2$ in Logisim, and the other parts of the equations will be calculated in a python script. The diagram for the error_calc_tb is found below. This diagram is made only for the x-axis for simplicity's

sake but the the real implementation also contains the y-axis using the same logic and the selector defined in 7 Logisim Basic Driver

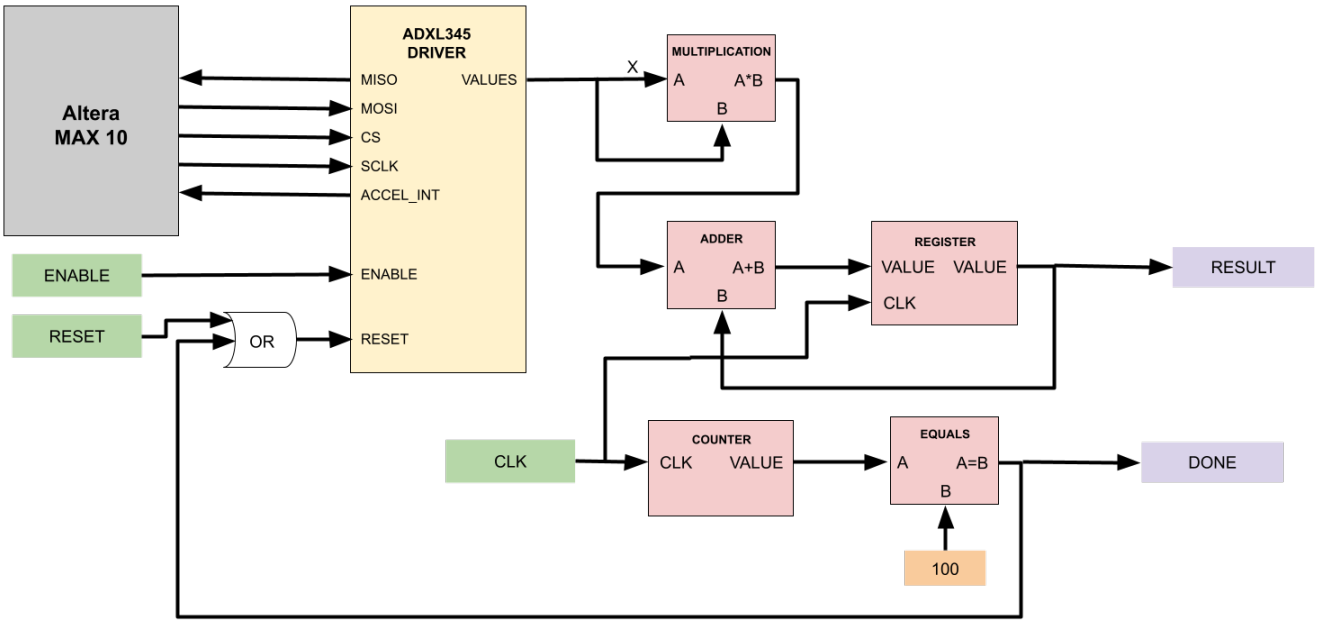


Figure 16: The diagram for the error_calc_tb

The circuit performs the multiplication and the sum. Then the value gets saved in a register. The value for this register gets fed back in the adder block to compute the total sum. We take 100 measurements. To keep track of our measurements a counter was used. Every tick of the clock, a new measurement is obtained and the counter increments. Once the counter reaches 100 the DONE bit is set to true and the reset for the ADXL345 is activated. This ensures that our accelerometer is only outputting zeros, which will not change our sum. On the DE10-Lite, the DONE bit was mapped to a LED. The RESULT output is displayed on the hexes. Here are the results of the experiment:

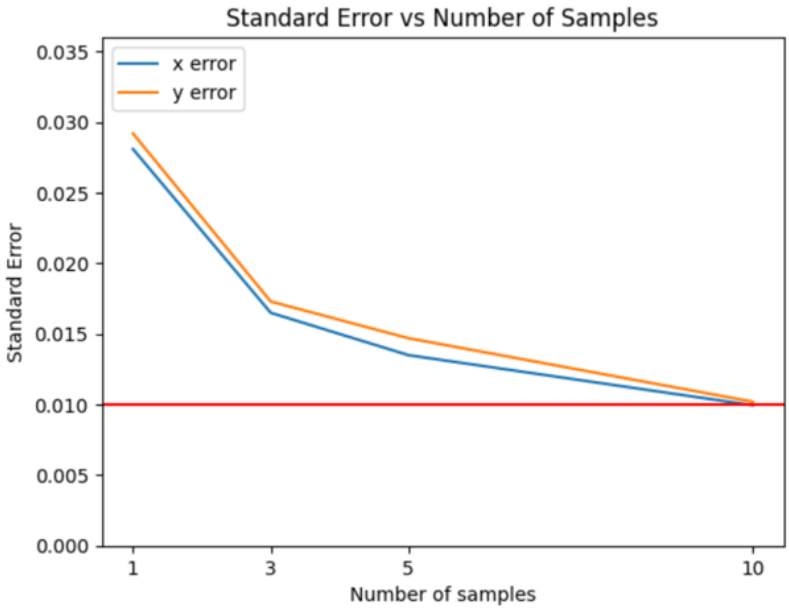


Figure 17: The results of the standard error with different moving average samples

The red line in the chart represents the upper limit that we want our error to have. The error was calculated with 1, 3, 5 and 10 samples. We can observe that at 10 samples, the error of the system is at or under the threshold so 10 samples were chosen for the filter.

8.4 PUTTING IT ALL TOGETHER

Applying the 3 corrections we obtain the following final accelerometer driver. This is the block that the end user will use for their application. The final driver has been tested at a frequency of 1 MHz.

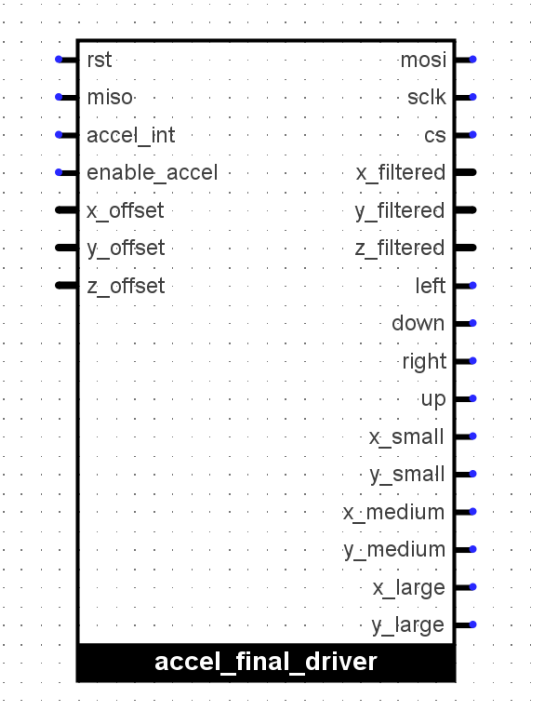


Figure 18: The accel_final_driver block

Each input and output is summarized in the table below:

I/O	BIT SIZE	PURPOSE
RST	1	Resets accelerometer to 0
MISO	1	Master In Slave Out, used in SPI
ACCEL_INT	1	Interrupt flag used in SPI
ENABLE_ACCEL	1	Enables accelerometer
X_OFFSET	16	Offset along x-axis
Y_OFFSET	16	Offset along y-axis
Z_OFFSET	16	Offset along z-axis
MOSI	1	Master Out Slave In, Used in SPI
SCLK	1	Used in SPI
CS	1	Used in SPI
X_FILTERED	16	The acceleration along x calibrated, scaled and filtered
Y_FILTERED	16	The acceleration along y calibrated, scaled and filtered
Z_FILTERED	16	The acceleration along z calibrated, scaled and filtered
LEFT	1	Is set to TRUE if the board is tilted left
DOWN	1	Is set to TRUE if the board is tilted down
RIGHT	1	Is set to TRUE if the board is tilted right
UP	1	Is set to TRUE if the board is tilted up
X_SMALL	1	Is set to TRUE if the board has a small tilt along the x-axis
Y_SMALL	1	Is set to TRUE if the board has a small tilt along the y-axis
X_MEDIUM	1	Is set to TRUE if the board has a medium tilt along the x-axis
Y_MEDIUM	1	Is set to TRUE if the board has a medium tilt along the y-axis
X_LARGE	1	Is set to TRUE if the board has a large tilt along the x-axis
Y_LARGE	1	Is set to TRUE if the board has a large tilt along the y-axis

Table 2: The I/O of the final driver

The accelerometer outputs the value of the acceleration and, using comparators, it outputs "high-level" information such as if the board is tilted left, right, up or down and if the board has a small, medium or large tilt. This allows the user to either use the filtered numerical values for their application or the "high-level" values. In the context of the EE-110, EE-207 class, the up or down bits can be used to set the time for the clock project. The small, medium or large bits could be used for how fast the number changes. Furthermore these "high-level" values could be used to make a controller for a video game. The movement of a playable character can be controlled on a screen with this accelerometer driver. The ranges of these high-level information is given below:

- X > 0.1g : left X < -0.1g : right
- Y > 0.1g : down Y < -0.1g : up
- 0.1g < |X or Y| < 0.4g : Small
- 0.4g < |X or Y| < 0.7g : Medium
- 0.7g < |X or Y| < 1.00g : Large

Figure 19: The ranges of the high level information

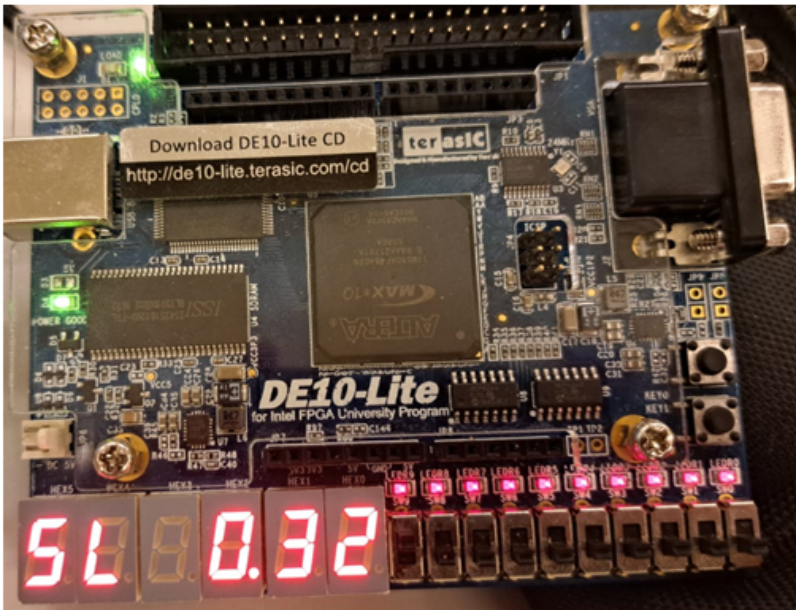


Figure 20: The accel_final_driver_tb running on the DE10-Lite. In this particular situation, we are looking at the x axis when the board is slightly tilted on the left. On HEX5 (The one all the way on the left) we show if the angle is small medium or large. On HEX4, we show if the board has been tilted left or right. HEX0 - HEX3 gives us the value of the acceleration in g. In this situation we read "Small Left with an acceleration of 0.32g". We can use the two selector dip switches to switch between axes as explained in 7 Logisim Basic Driver.

The following diagram shows how the high level information is obtained for the x-axis. The information is obtained for the y-axis in a similar fashion. The input x-axis comes from the ADXL345 basic driver and the offset is inputted by the user:

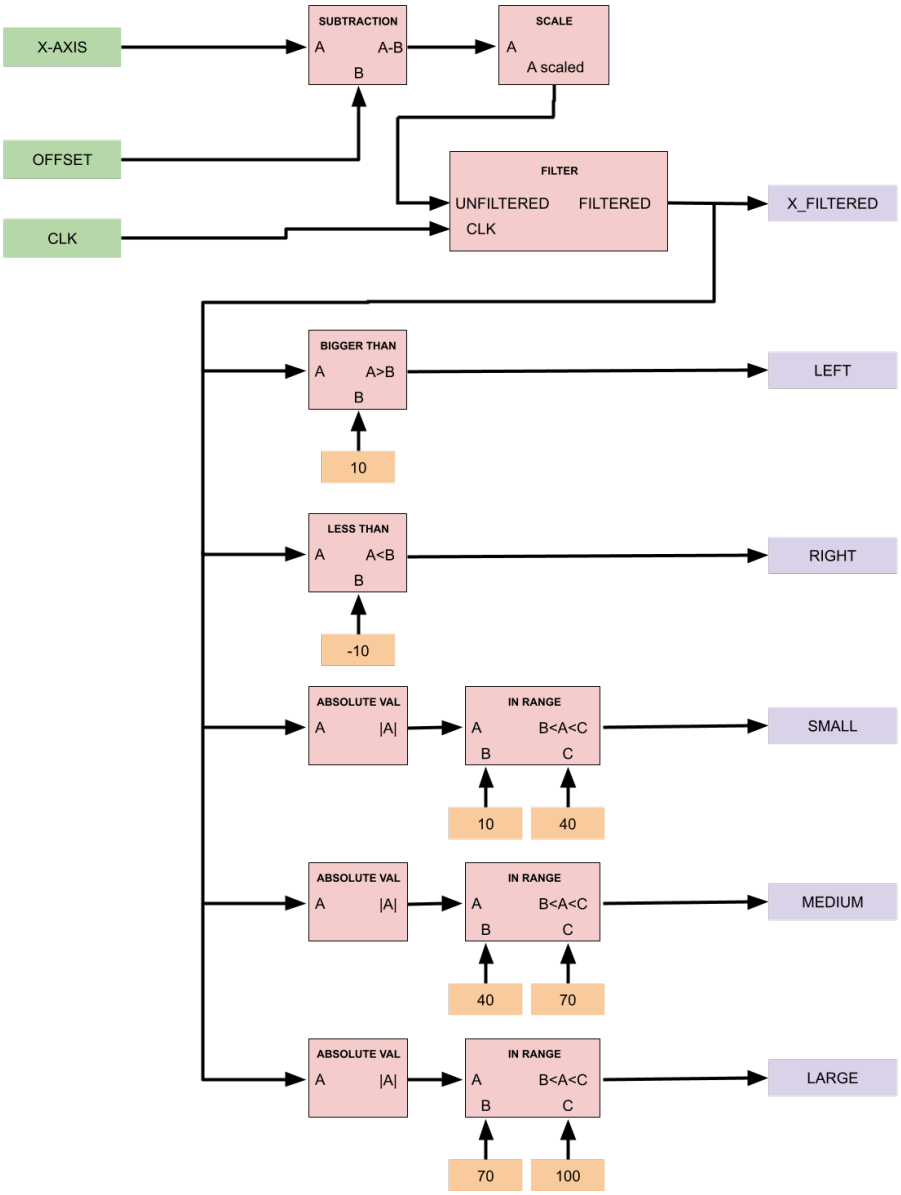


Figure 21: The high level information for the x-axis

The raw value of the x-axis gets processed. First the offset is removed, then the value is scaled down and filtered. From there we output X_FILTERED. X_FILTERED is then used to determine the high-level information. It is first verified if it is bigger that 10 or less than -10 to determine if the board is tilted left or right. Then it is determined in what range the value of the acceleration is in to output if it is a small, medium or large tilt. A user guide to using the driver is provided in the 13 Appendix

9 APPLICATION

The application developped with the accel_final_driver module is called Get To Zero. The game initializes two pseudo-random values for the x-axis and the y-axis. The random value for the x-axis is shown on HEX0 - HEX2. The random value for the y-axis is shown on HEX3 - HEX5. The player needs to get these two values to 0 by tilting the board. The value of the accelerometer gets added to the pseudo random values. The player can get to 0 by getting the pseudo random values to -100, 0 or 100. This was done to make the player make a choice on what would

be the fastest way to get to zero. For instance let's say that the random value for the x-axis is 70. Tilting the board to the right would decrement the value and tilting the board to the left will increment the value. Since 70 is closer to 100 than 0, the optimal choice for the player is to tilt the board to the left. Once one of the random values reaches 0, the value for that axis gets frozen at 0. When both random values read 0, the player is shown how much time in seconds it took them. Then by pressing a reset button the player can replay the game. It is important to press the RESET(KEY1) button when the game starts to properly initialize everything.

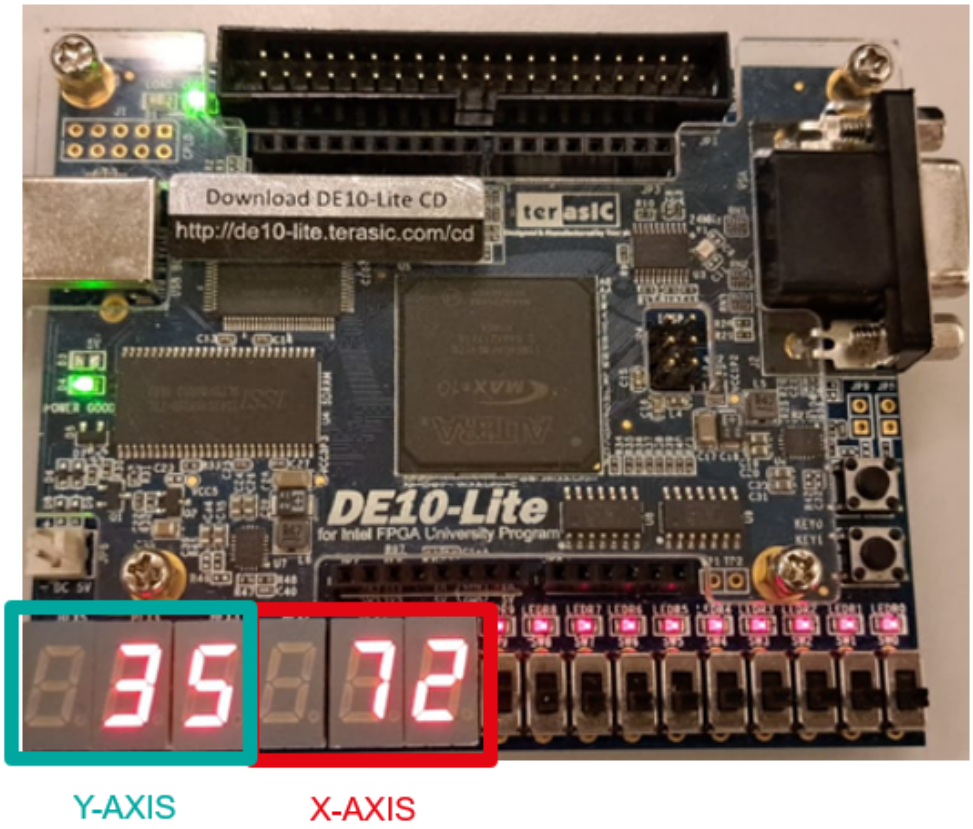


Figure 22: How the application looks on the DE10-Lite

The game has three states: the start state, the game state and the end state.

9.1 Start State

The start state is an animation where the word start moves from right to left on the hexes. The animation was created using 2-input muxes and connecting the selector of the muxes to a clock. During this time, in the background, the clock for the pseudo-random number generator is running at a frequency of 1MHz. To enter the game state the player need to press the start push button (KEY0)

9.2 Game State

When the player enters the game state, the pseudo random values are displayed on the hexes. The following diagram shows how the pseudo-random value is generated for the x-axis. For the y-axis the procedure is the same except that the min and max values for the first counter are 18 and 82 in order to get different values than the x-axis.

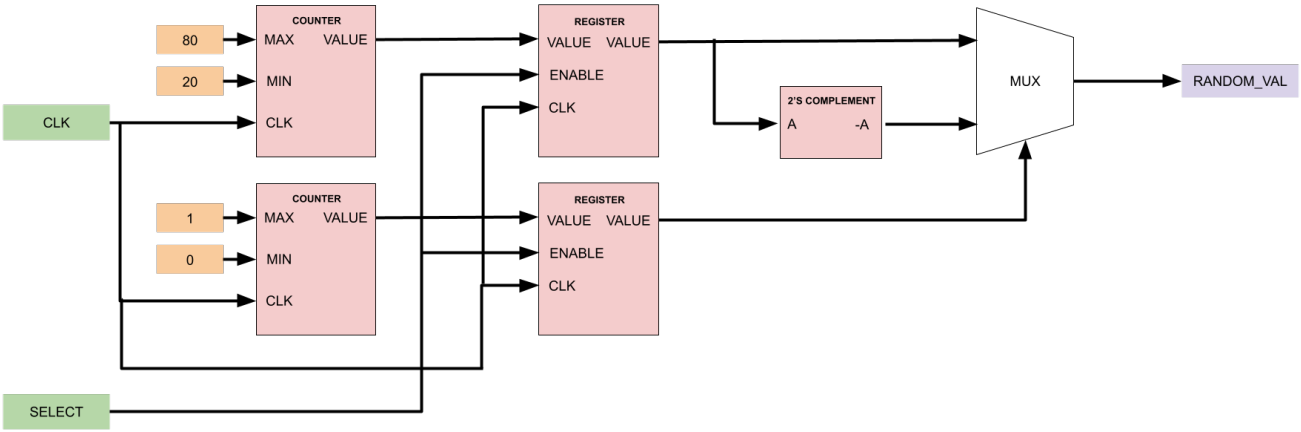


Figure 23: Diagram to generate a pseudo-random value along the x-axis

The first counter is responsible of generating the pseudo-random value between 20 and 80. The second counter is responsible of choosing the sign. Once the SELECT button is pressed (KEY0), the ENABLE of the registers are activated and a value is saved in the registers. If the second register has a value of 1 then we output the 2's compliment of the first register. If the second register has a value of 0 then we output the positive value of the first register.

Once the random values are chosen, the accelerometer comes into play.

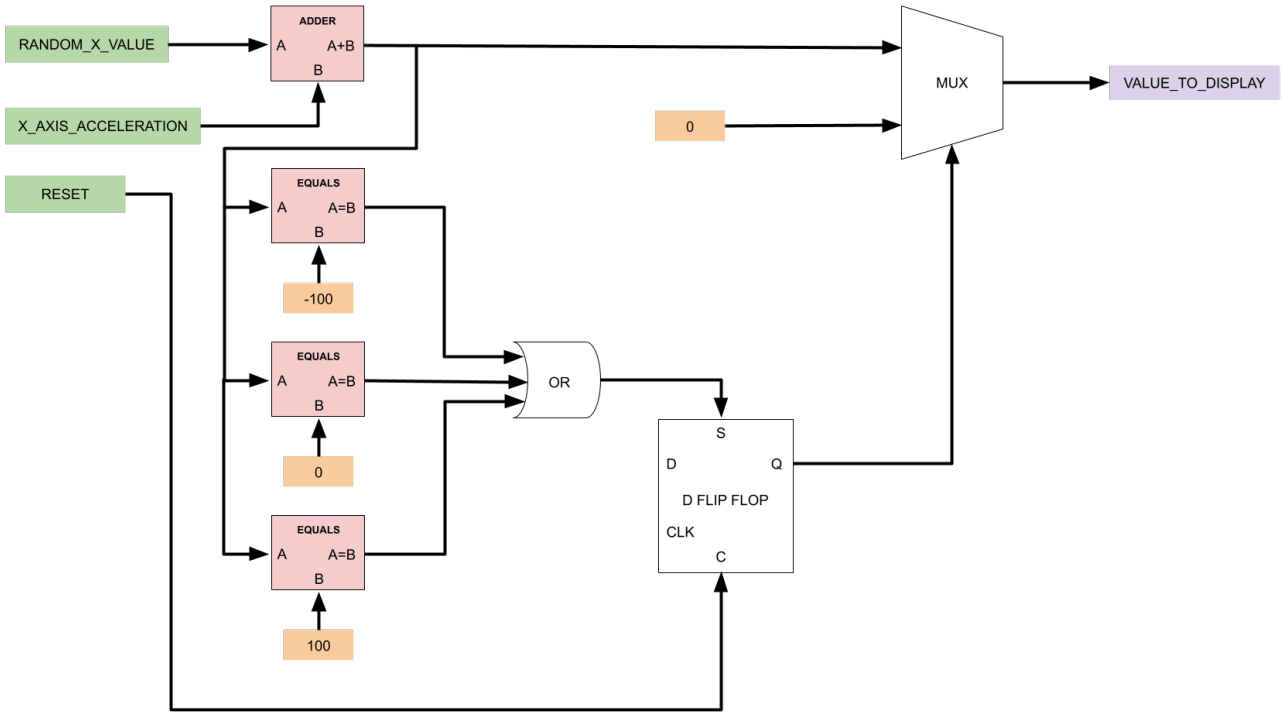


Figure 24: Diagram for the main logic of the game along the x-axis

The X_AXIS_ACCELERATION input comes from the ADXL345. This value gets added to the pseudo-random x value. Then if that sum is equal to -100, 0 or 100 the set bit of a D-flip-flop is set to true. If the D-flip-flop is true then the value to display will always be 0. If it is not then the value to display is the sum between the pseudo-random value and the x-axis acceleration. The y-axis is dealt with in the same way. During this process a counter is getting incremented every second in the background. Once both axes are at 0 we go to the end state.

9.3 End State

When we enter the END STATE, the counter that was keeping track of the time elapsed since the player pressed the SELECT button (KEY0) is stopped. We then show the time on the hexes that it took the player in seconds. The time is shown with a blinking animation made with multiplexers to simulate a stopwatch. The player can return to the START STATE of the game by pressing the RESET button (KEY1).

10 Simulator

In order to to allow the user of the accelerometer module to understand what the code does without having to load it on the DE10-Lite (a time-consuming process), a simulator was created. The simulator mimics in Logisim what the user running the accel_final_driver_tb would see on the FPGA. Without any additional software, some VHDL blocks cannot be simulated in Logisim. Therefore a Logisim circuit version of the VHDL blocks div_two_num and hex_to_decimal that can be simulated was created. To simulate the tilt of the DE10-Lite board, sliders and one-bit inputs were used. Taking the x-axis slider as an example, increasing the value would be as if the board is being tilted left. If the X_NEGATIVE input is activated, it would be as if the board is being tilted right. The same idea is used for the other axes. The output can be observed on the 6 hexes.

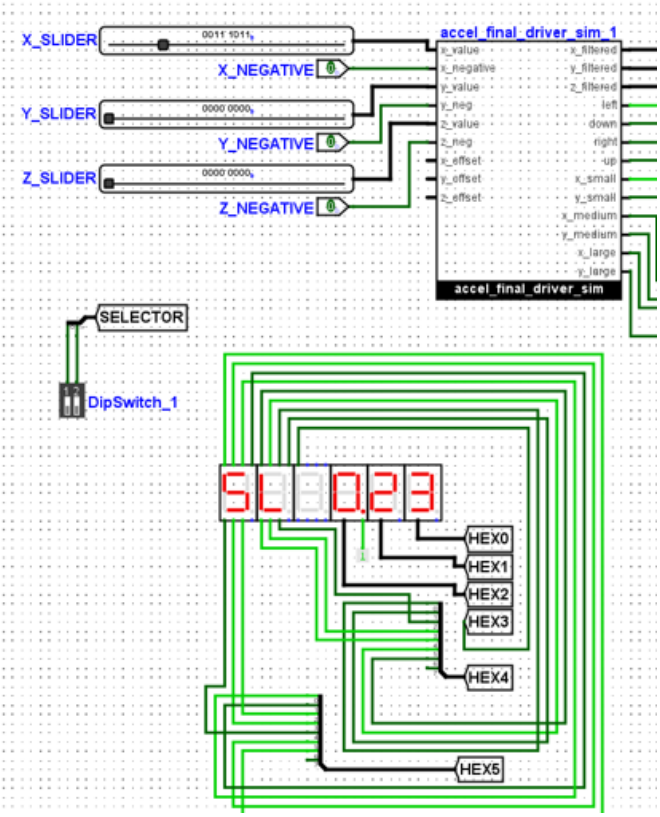


Figure 25: The simulator working in Logisim. In this case we have simulated a small tilt to the left with acceleration value of 0.23g on the x-axis

11 CONCLUSION

In conclusion, the four objectives set out for the projects have been achieved. The driver for the accelerometer has been created. The output of the driver can be seen on the FPGA through a test bench. The application Get to Zero has been created using the accelerometer module. A simulator was created for Logisim to better understand the driver.

11.1 OUTLOOK

The accelerometer has many other features that were not explored. The resolution can be increased to 13 bits, there is a free fall detection mode and a single tap detection and double tap detection mode. Furthermore, the high-level information of the accelerometer was made in the hopes that a DE10-Lite would be used as a controller for a video game. If a VGA module is created, the accelerometer module could be used for that video game purpose.

11.2 ACKNOWLEDGEMENT

A big thank you to my supervisor, Mr. Alexandre Schmid who always made himself available to help me despite his busy schedule. A special thank you goes out to Mrs Selma Benhassine for being there to support me through the late nights working.

12 BIBLIOGRAPHY

REFERENCES

[1] Fidai A. *Standard Error | Formula Examples*. 2023. URL: https://study.com/academy/lesson/what-is-the-standard-error-of-the-estimate-formula-examples.html.

[2] Schmid A. *Systèmes logiques (pour MT)*. 2023 - 2024. URL: https://edu.epfl.ch/coursebook/fr/systemes-logiques-pour-mt-EE-110.

[3] *ADXL345 Data Sheet*. URL: https://www.analog.com/en/products/adxl345.html.

[4] Johnson B. *SPI Accelerometer*. 2020. URL: https://github.com/bjohnsonfl/SPI_Accelerometer.

[5] Phidgets Inc. *Accelerometer Guide*. 2024. URL: https://www.phidgets.com/docs/Accelerometer_Guide.

[6] Hildebrand M. and Baas B. *EEC180 Tutorial: Using the accelerometer on the DE10-LITE board*. 2020. URL: https://www.ece.ucdavis.edu/~bbaas/180/tutorials/accelerometer.html.

[7] Univeristy of Oslo. *ADXL345 Accelerometer*. URL: https://www.uio.no/studier/emner/matnat/fys/FYS4220/h20/lecture-slides/adxl345.pdf.

[8] Campbell S. *BASICS OF THE SPI COMMUNICATION PROTOCOL*. URL: https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/.

[9] Surf-VHDL. *How to Divide an Integer by Constant in VHDL*. 2015. URL: https://surf-vhdl.com/how-to-divide-an-integer-by-constant-in-vhdl/.

[10] Terasic. *DE10-Lite User Manual*. 2020. URL: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=234&No=1021&PartNo=4#contents.

13 APPENDIX

Before following the steps in this appendix to use the accelerometer, make sure that the steps in the EE110-EE207-2023_SW01_v3.1.pdf document have been followed to install the required softwares. Then download the accelerometer_files folder that should contain accelerometer_final.circ, get_to_zero_final.circ and the TERASIC_DE10LITE_GSENSOR.xml mapping. In the accelerometer_final.circ library, any circuit that ends with _tb is meant to be loaded on the FPGA. Every circuit that ends with _sim is meant to be simulated in Logisim. When possible the pins and the I/O will be reffered to by the name given to them in the DE10-Lite User Manual [10]

13.1 FILES SUMMARY

13.1.1 ACCELEROMETER_FINAL_DRIVER FILES

Circuit	Purpose
accel_final_driver	The accelerometer values cleaned and discretized
accel_final_driver_tb	the accel_final_driver that can be loaded on the DE10-Lite
accel_final_driver_tb_sim	the accel_final_driver test bench that can be simulated in Logisim
accel_final_driver_sim	the accel_final_driver with no VHDL blocks
calibration_tb	The calibration file that can be loaded on the DE10-Lite
calibration	Allows the user to find the right offsets
error_calc_tb	Used to determine the Standard Error
accel_raw_tb	The raw value of the accelerometer that can be loaded on the DE10-Lite
signed_fast_filter	The moving average filter
signed_fast_filter_sim	The moving average filter that can be simulated in logisim
signed_div_by_128	Division by 128
signed_div_by_128_sim	Division by 128 that can be simulated in logisim
accel_driver_logisim	Allows Logisim to properly initialize the different VHDL components
absolute_val_and_sign	Saves the absolute value and the sign of a number
hex_to_decimal_sim	Convert from hex to decimal that can be simulated in Logisim
divide_two_num_sim	The divider block that can be simulated in Logisim
division_sim	The divider block slightly tweaked that can be simulated in Logisim
twos_complement	Self-explanatory
sign_hex_output	Activate the right led on the hex if the number is negative
sign_hex_logic	Logic for the sign hex output
sevenseg_direction_angle_logic	Maps the letter to the right leds on the hexes
sevenseg_direction_angle_output	The sevenseg_angle_logic mapped to a hex
sevenseg_numbers_logic	Maps numbers to the right leds of the hex
sevenseg_numbers_output	Maps the logic to a hex
sevenseg_mapped	Maps an 8 bits input to a hex
reset_logic	Simple logic that makes sure only the reset or the enable is activated

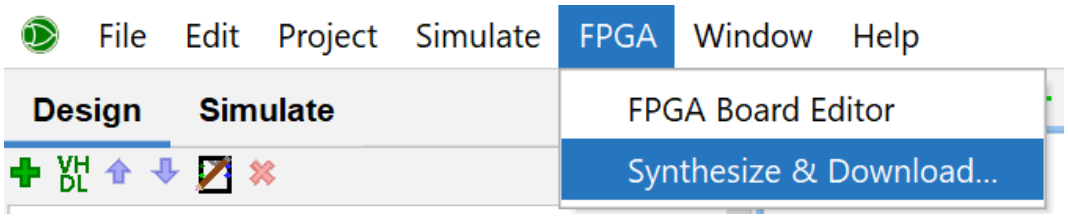
The explanation for the VHDL blocks are given in figure 3, section 8.2.1 and section 8.2.2

13.1.2 GET_TO_ZERO FILES

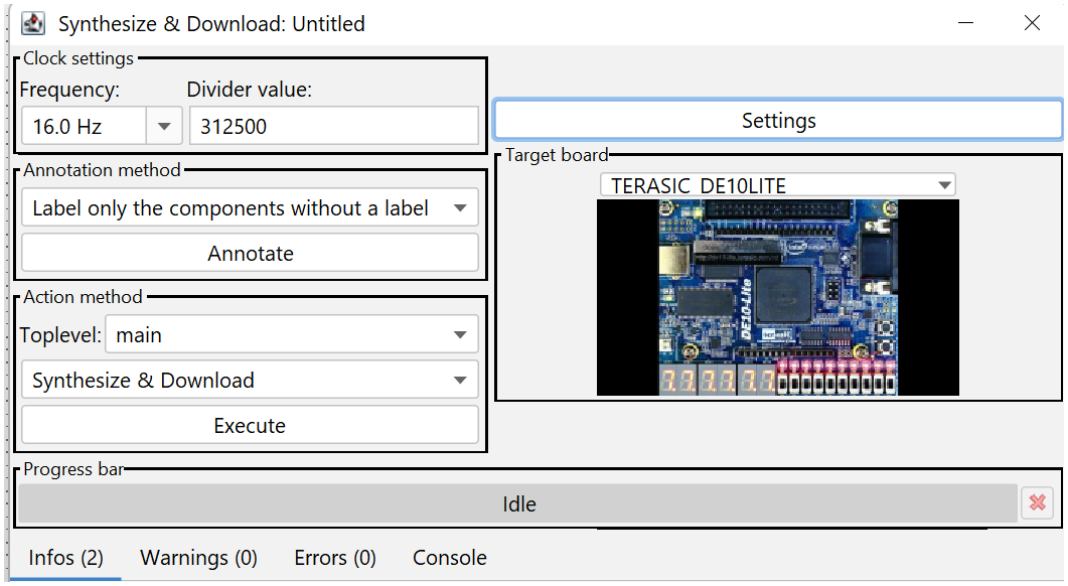
Circuit	Purpose
main	The main code of the game
counter_fsm	The main FSM
game_logic	The logic of the game
pseudo_random	The pseudo-random number generator
save_random_val	Saves the pseudo-random values in registers
debouce	Simple debounce for the push buttons
letters_logic	Maps the letters to the right leds on the hex
letter_no_decimal	Makes sure the decimal point is not on
side_to_side_anim	Makes a word jump from side to side on the hexes
blink_anim	Makes the hexes blink like a stopwatch
timer	Keeps track of elapsed time

13.2 ACCELEROMETER FINAL DRIVER USER GUIDE

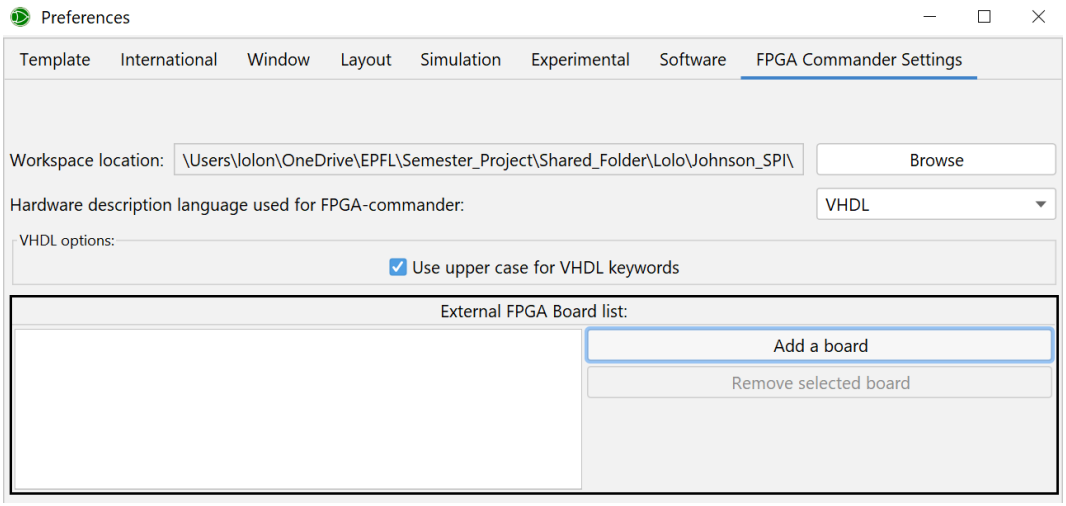
The first thing that must be done before using the accelerometer is to load the new mapping. The accelerometer uses pins that are not mapped in the TERASIC_DE10LITE.xml mapping so the new map TERASIC_DE10LITE_GSENSOR.xml must be loaded. Navigate to FPGA > Synthesize & Download

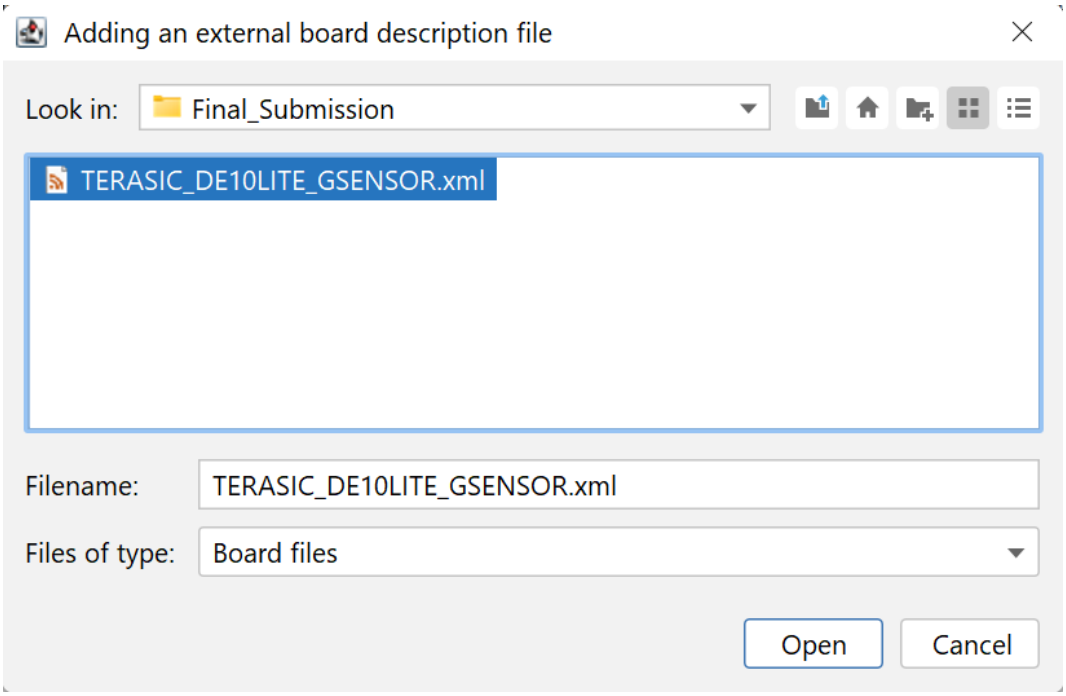


Then select Settings

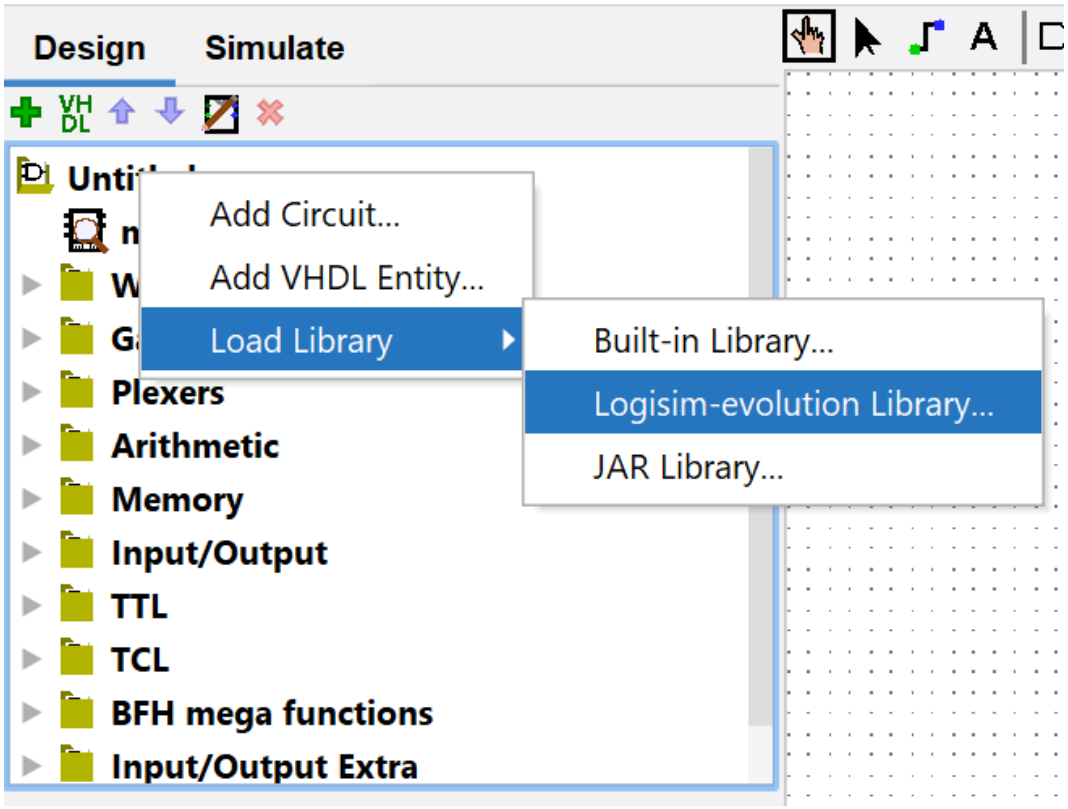


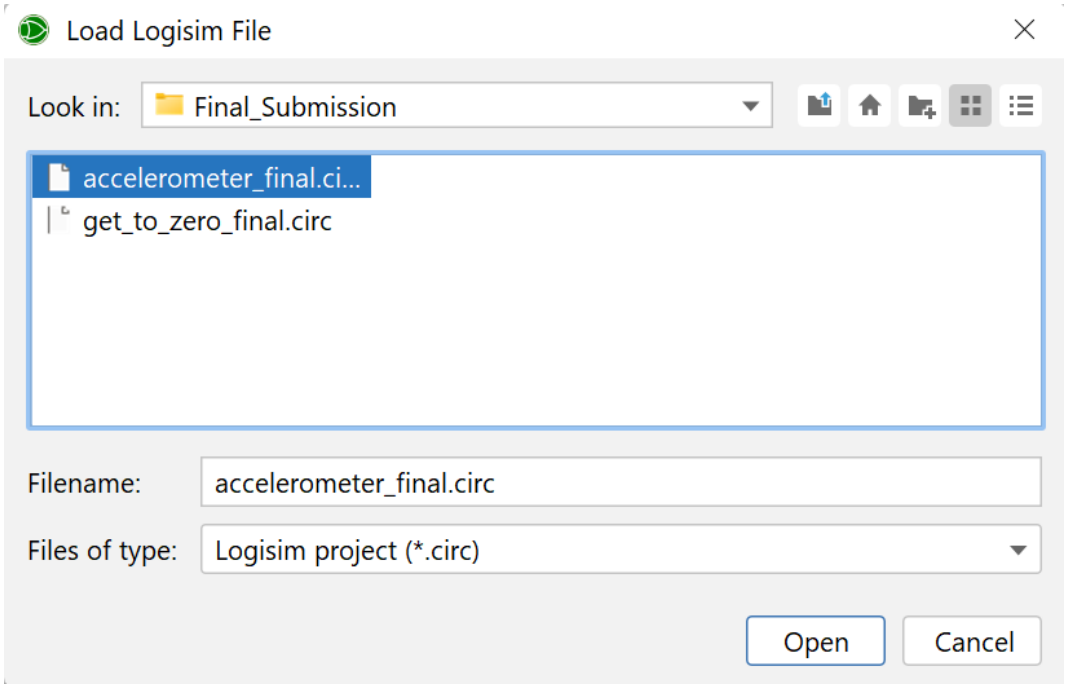
Then under FPGA Commander Setting select Add a board and select the TERASIC_DE10LITE_GSENSOR.xml board.





The board will then appear in the External FPGA Board List and will be selectable in the Synthesize & Download menu. You may now close the Synthesize & Download menu. Now the accelerometer_final.circ library can be loaded in Logisim. To do so right click on your project main folder in Logisim and select Load Library > Logisim-evolution Library. Then select the accelerometer_final.circ library. It will appear as the last folder in Logisim.





From the library only the first circuit accel_final_driver needs to be imported in your application. For the offsets you can use values in figure 11. If you wish to more precisely calibrate the sensor please refer to 13.4 Calibration Test Bench User Guide. For an explanation of every I/O please refer to table 2 in section 8.4. Once the application has been built the bits responsible for the SPI communication have to be correctly mapped.

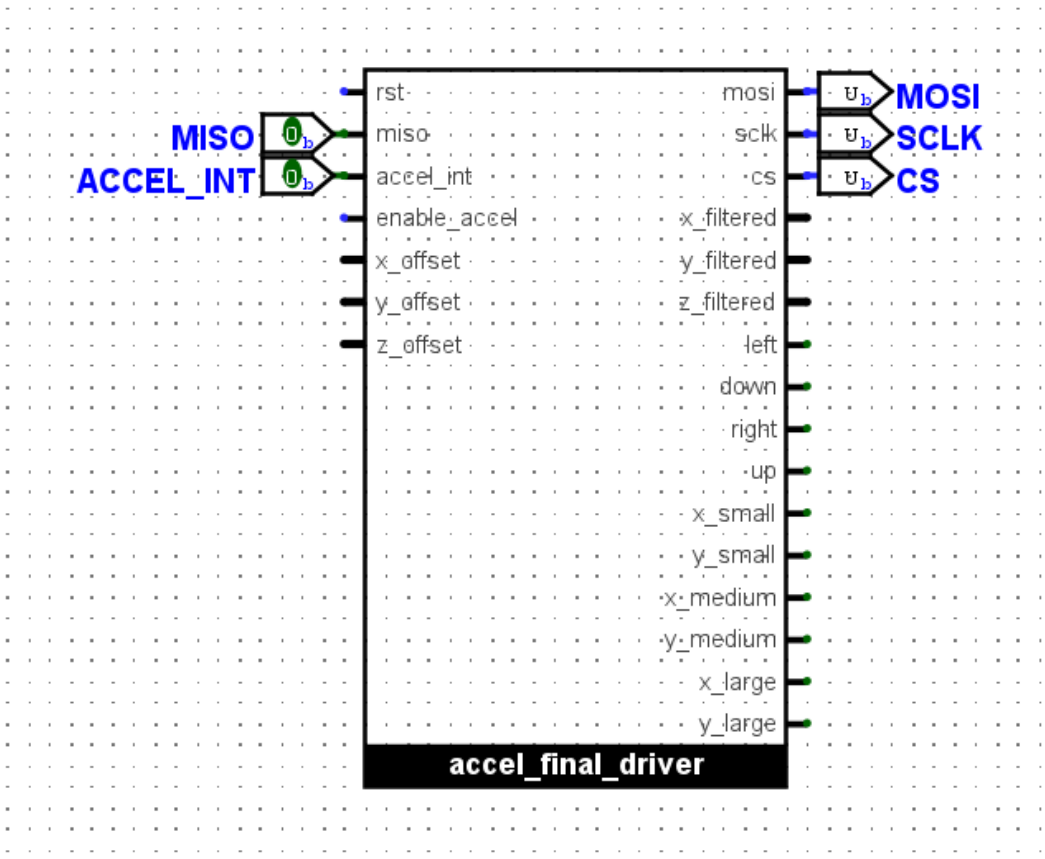
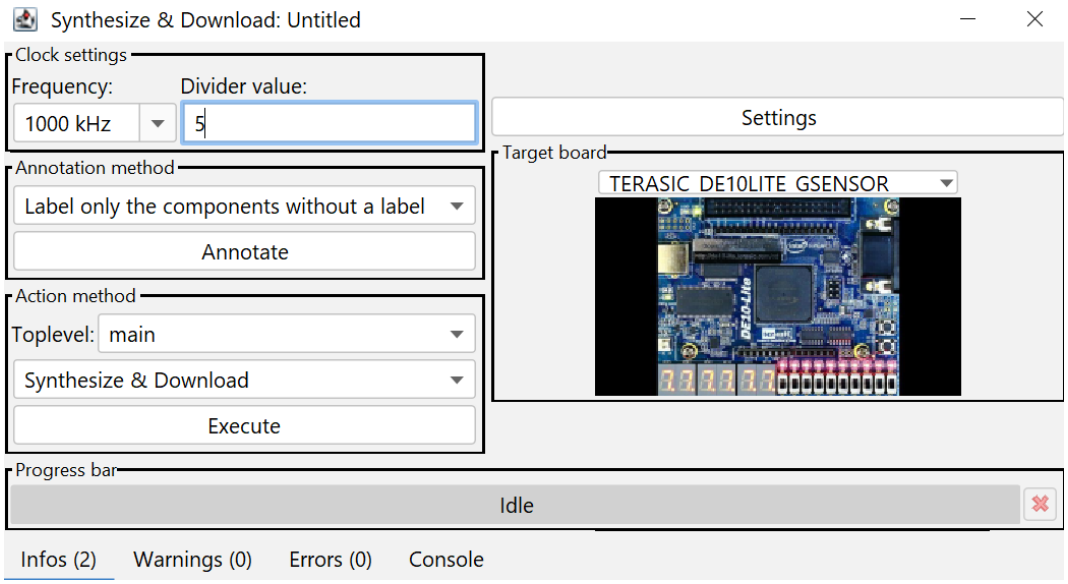
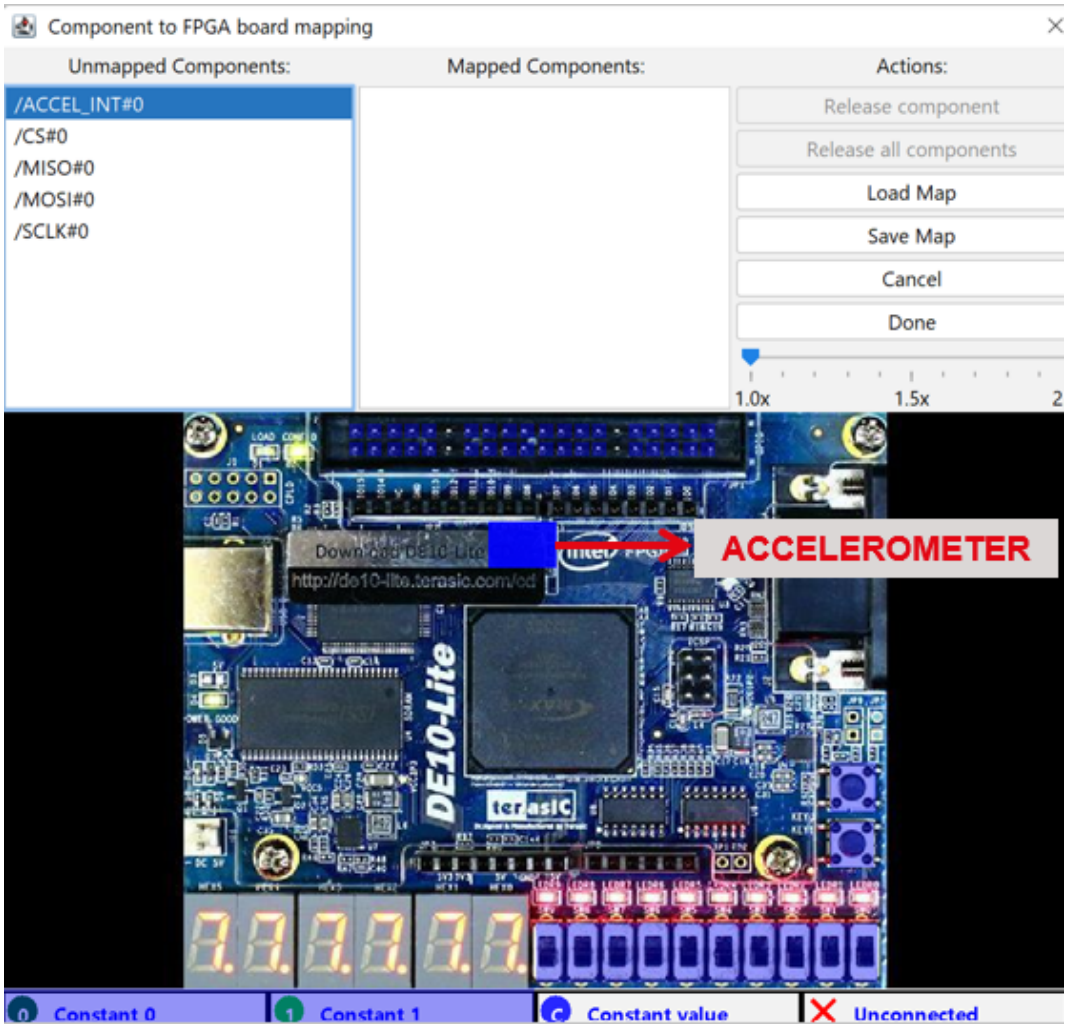


Figure 26: The driver and the bits responsible for the SPI communication

Once the application is built select FPGA > Synthesize & Download and choose the TERASIC_DE10LITE_GSENSOR as the target board. Then set the frequency to 1 MHz.



Then press Annotate and Execute. The 5 bits need to be mapped to the accelerometer. Select each one of them and click on the accelerometer chip.



The mapping is done as follows:

Bit	Pin
MOSI	GSENSOR/PIN0
MISO	GSENSOR/PIN1
SCLK	GSENSOR/PIN2
CS	GSENSOR/PIN3
ACCEL_INT	GSENSOR/PIN4

Then the code can be loaded on the FPGA.

13.3 Accelerometer Final Driver Test Bench User Guide

The accelerometer final driver test bench outputs on the hexes the value of the acceleration, the direction of the tilt and if it's a small, medium or large tilt. A picture of this code running can be found in figure 20. To get this code running please follow the steps outlined in 13.2 Accelerometer Final Driver User Guide to load the correct board. Please make sure to activate the ENABLE_ACCEL flag. This test bench runs at a 1 MHz frequency and the mapping is as follows:

I/O in the code	Pin or I/O on DE10-Lite
MOSI	GSENSOR/PIN0
MISO	GSENSOR/PIN1
SCLK	GSENSOR/PIN2
CS	GSENSOR/PIN3
ACCEL_INT	GSENSOR/PIN4
HEX0	HEX0
HEX1	HEX1
HEX2	HEX2
HEX3	HEX3
HEX4	HEX4
HEX5	HEX5
ENABLE_ACCEL	SW9
RST	SW8
DIPSWITCH_1/SW1	SW0
DIPSWITCH_1/SW2	SW1

13.4 CALIBRATION TEST BENCH USER GUIDE

The Calibration Test Bench allows the user to find the right offsets to apply on each axis. A picture of this code running can be found in figure 8. A detailed explanation on how to use the code can be found in section 8.1 Calibration. To get this code running please follow the steps outlined in 13.2 Accelerometer Final Driver User Guide to load the correct board. This test bench runs at a 1 MHz frequency and the mapping is as follows:

I/O in the code	Pin or I/O on DE10-Lite
MOSI	GSENSOR/PIN0
MISO	GSENSOR/PIN1
SCLK	GSENSOR/PIN2
CS	GSENSOR/PIN3
ACCEL_INT	GSENSOR/PIN4
HEX0	HEX0
HEX1	HEX1
HEX2	HEX2
HEX3	HEX3
HEX4	HEX4
HEX5	HEX5
RST	KEY1
DIPSWITCH_1/SW1 - SW10	SW0 - SW9

13.5 GET TO ZERO USER GUIDE

Get To Zero is a game where the player must get the two pseudo-random values on the board to zero. A picture of this code running can be found in figure 22. A detailed explanation on how to use the code can be found in section 9 Application. To get this code running please follow the steps outlined in 13.2 Accelerometer Final Driver User Guide to load the correct board. This test bench runs at a 1 MHz frequency and the mapping is as follows:

I/O in the code	Pin or I/O on DE10-Lite
MOSI	GSENSOR/PIN0
MISO	GSENSOR/PIN1
SCLK	GSENSOR/PIN2
CS	GSENSOR/PIN3
ACCEL_INT	GSENSOR/PIN4
HEX0	HEX0
HEX1	HEX1
HEX2	HEX2
HEX3	HEX3
HEX4	HEX4
HEX5	HEX5
RESET	KEY1
BUTTON	KEY0