CHALMERS
UNIVERSITY OF TECHNOLOGY

Digital Spirit Level

Project report in the course EDA234 - Digital Project Laboratory

MEHDI HAIDARI
BASEER QAYOUMI
THEODOR AHLGREN

Abstract

The project undertaken is "Digital Spirit Level" which will be discussed in detail in this report. A digital spirit level, also called electronic level, basically has the same functionality as a regular spirit level. A digital spirit level utilizes gravitational pull in different axes to determine the angle of the electronic level. Placing the FPGA-board along a plain surface, like a table, gives the slope of the table and outputs it to the user as angles on the 7-segment-display, LCD-screen and if connected by RS232, also to a PC via the UART transmitter. The electronic level can only measure the roll (slope of x-axis) and pitch (slope of y-axis) separately, and the user manually has to switch between modes using a switch on the FPGA. Below is an image of what it looks like measuring.

In the report we will describe our project in detail. Components used, system specification, circuit board layout and description of sub-blocks explains the low-level design to the reader and allows for easy modification and further development of the project. This report covers everything about the project, including how it's built, how it works, the VHDL programming behind it, checking for errors, and how the team worked together to create it. Figure 1 shows the circuit board layout with components that make up the project.

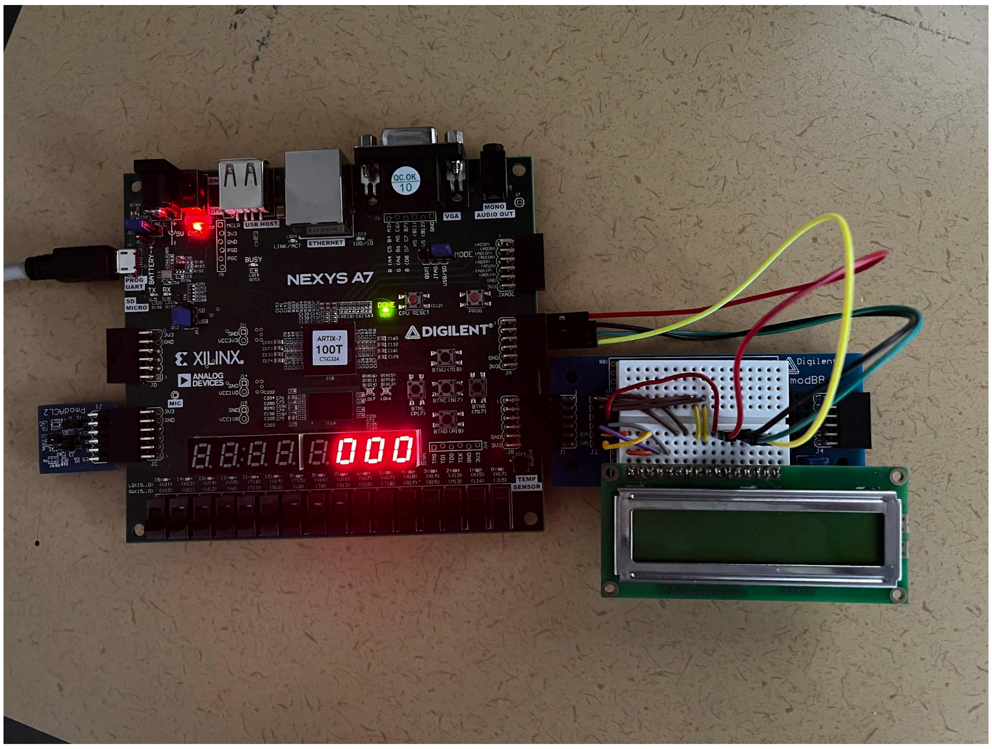


Figure 1: Circuit board layout

Table of contents

1. System Specification

The digital spirit level has the ability to measure the angle of the slope in two axes, x (roll) and y (pitch). The angle is derived and calculated from gravitational pull in different dimensions on the accelerometer which is communicated to the FPGA via an SPI interface. This data is processed in the FPGA to derive an angle from the gravitational pull and to encode the angles into binary encoded decimals for the 7-segment display and the LCD. Only one value can be displayed on both of the displays (7-segment display and LCD) at a time and a switch is used to select whether to measure the angle of pitch or roll. There is also a reset button for resetting the system.Furthermore the angle is also sent to a connected PC via the UART transmitter.

The angles are not displayed as negative or positive, meaning turning the spirit level 90 degrees in both directions both show 90° on the displays. The maximum angle that can be displayed is 180 degrees, from which it starts counting down towards 0 again. So turning the digital spirit level 360° around one axis would give the following output the following sequence on the displays: 0,1,2,..,89,90,91,..,179,180,179,..,91,90,89,..,2,1,0. The displays always show the angle as three digits, meaning an angle of 3° would output "003" to the displays.

Figure 2 shows the top-level design of the digital spirit level.

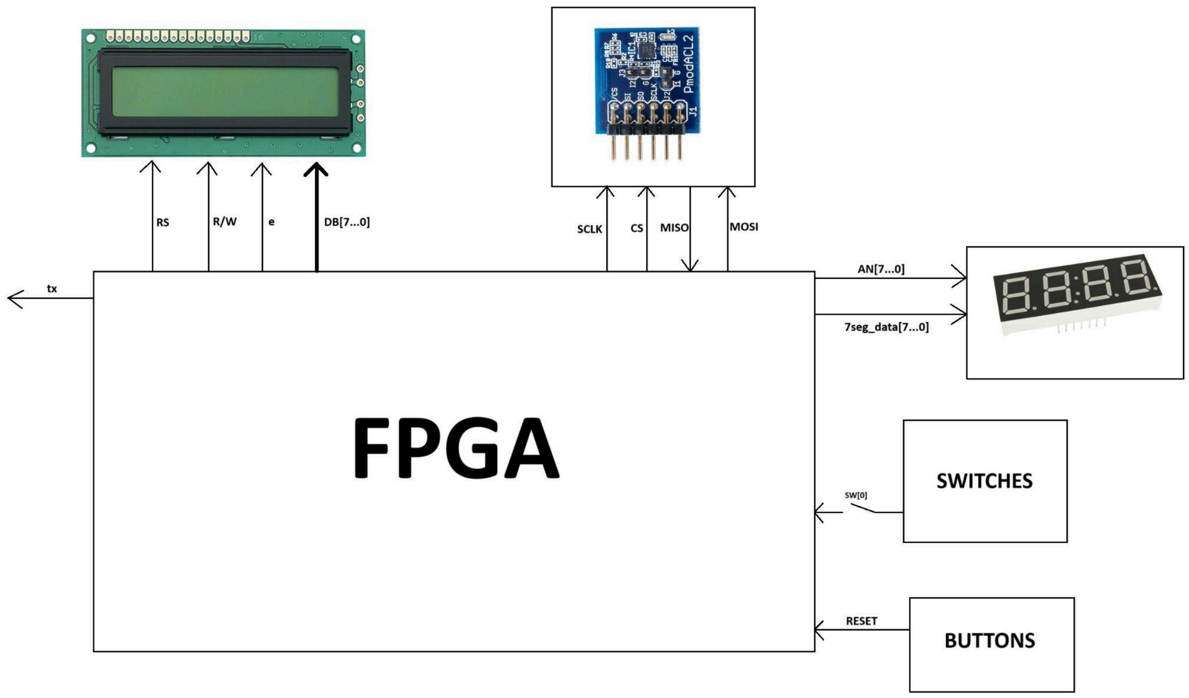


Figure 2: System overview

2. System Description

Multiple hardware and software components are used in the project. This chapter goes into details of the sub-components that make up the electronic level.

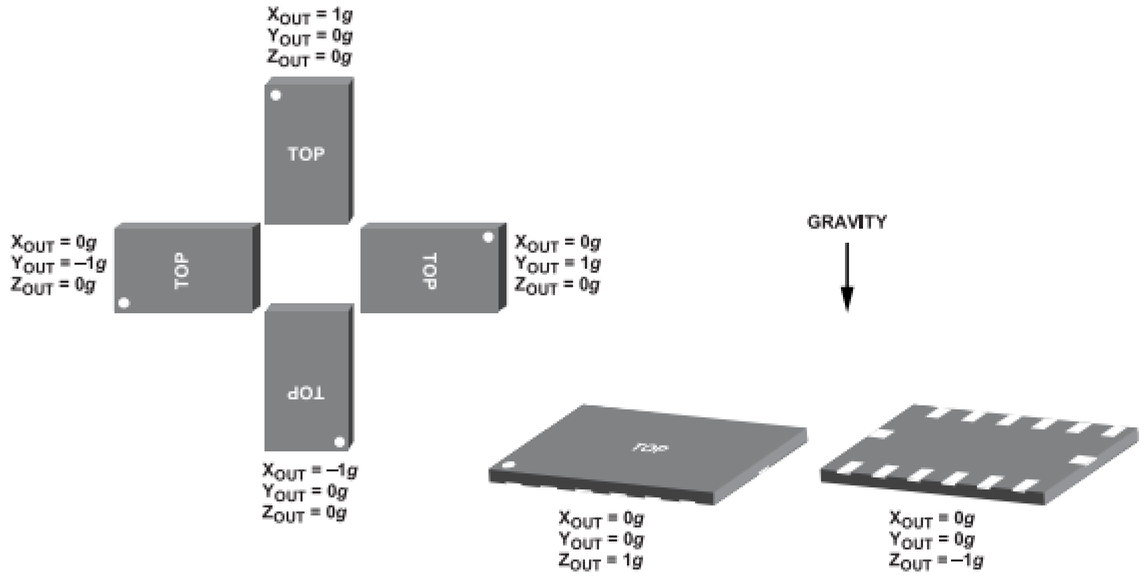
2.1 Hardware

The hardware components that make up the project are described in this section.

2.1.1 ADXL362

The ADXL362 is a three-axis micro electro-mechanical system (MEMS) accelerometer that plays a critical role in the project. The ADXL362 we are using in our project is mounted on a peripheral module from Digilent called PmodACL2.

The ADXL362 can measure acceleration in three dimensions: X, Y, and Z. This capability is utilized for determining both the magnitude and direction of tilt of a surface, which is the primary use of the accelerometer. By detecting changes in acceleration across its three axes, it can determine the angle at which the spirit level is positioned. The acceleration data from the ADXL362 is continuously read and processed by the other modules on the FPGA, which converts data into corresponding angle measurements. Figure 2 shows orientation of the spirit level vs the output in X,Y and Z values.



$X_{OUT} = 1g$ $Y_{OUT} = 0g$ $Z_{OUT} = 0g$
TOP
$X_{OUT} = 0g$ $Y_{OUT} = -1g$ $Z_{OUT} = 0g$
TOP
TOP
$X_{OUT} = 0g$ $Y_{OUT} = 1g$ $Z_{OUT} = 0g$
GRAVITY
TOP
$X_{OUT} = -1g$ $Y_{OUT} = 0g$ $Z_{OUT} = 0g$
TOP
$X_{OUT} = 0g$ $Y_{OUT} = 0g$ $Z_{OUT} = 1g$
$X_{OUT} = 0g$ $Y_{OUT} = 0g$ $Z_{OUT} = -1g$

Figure 3: Orientation vs Output

The ADXL362 accelerometer contains SPI communication lines, including SCLK (Serial Clock), MISO (Master Input, Slave Output), MOSI (Master Output, Slave Input), and CS (Chip Select). These lines enable it to send and receive data, synchronize communication, and select the active device for data transfer. More on the register that we will read and write can be read in section 2.2.2.

2.1.2 Nexys A7 FPGA

The Nexys A7 FPGA (Field-Programmable Gate Array) is the central component in the Digital Spirit Level system. It serves as the primary processing and control unit, orchestrating the operations of the peripherals and executing the logic for measuring and displaying tilt angles.

2.1.3 USB (RS232)

The USB232 interface, commonly referred to as a USB-to-Serial adapter or converter, is an integral part of the project. It provides a bridge between the FPGA board and the PC, allowing for efficient communication and data transfer. USB232 facilitates serial communication, which is a method of transmitting data one bit at a time. This is essential for sending data from the FPGA to a PC in a format that is easily interpreted. The adapter converts the serial data to USB format, enabling compatibility with modern computers which predominantly use USB-COM ports.

2.1.4 7-Segment display

The Nexys A7 board contains two four-digit common anode seven-segment LED displays, configured to behave like a single eight-digit display. Each of the eight digits is composed of seven segments arranged in a “figure 8” pattern, with an LED embedded in each segment. Segment LEDs can be individually illuminated, so any one of 128 patterns can be displayed on a digit by illuminating certain LED segments and leaving the others dark. The anodes of the seven LEDs forming each digit are tied together into one “common anode” circuit node, but the LED cathodes remain separate. The anode enables are inverted meaning that the anodes are driven low.[2].

2.1.5 LCD-display

The LCD display, specifically the DMC 16117A model, serves as the primary interface for displaying the measured angles to the user. The display has high visibility and readability. The LCD will be connected with 5V to the FPGA board. The LCD will be configured to 16x1 interface with 5x8 dots per character [3].

2.2 Functional units

The software components that make up the project are described in this section. All code is written in VHDL with high modularity.

2.2.1 SPI Master

The SPI Master handles the SPI communication with the accelerometer through the Pmod ports. It has an SPI clock working at 5MHz. If the busy flag is low it is ready to accept a command. When the enable bit goes high it initiates a transaction, transmitting 8 bits of data on the MOSI line. Once the transaction is complete, the received data coming from the MISO line is outputted on the rx_data port and the busy flag returns to low.

The SPI Master also has a continuous mode where it stays in the execution cycle and automatically latches in a new byte to transmit. This mode comes in handy when we want to read multiple continuous registers from the ADXL362. After each byte is transmitted, the busy flag is deasserted for one clock-cycle, which functions as a flag for signaling that data is ready to be retrieved from the output buffer.

2.2.2 Accelerometer controller

The accelerometer_controller VHDL code manages data communication between our system and the accelerometer using the master_spi as an internal component. It controls data transmission, sets up accelerometer configurations like read mode, data rate, and range, and continuously reads data from specific addresses. The controller operates through various states as shown in the figure below. In each state, the controller manages SPI communication by preparing the data for the transaction, handling SPI-busy signals, and interpreting received data. It outputs 16-bit data for each axis (X, Y, Z), cycling through these operations until a system reset. The master_spi component within the code facilitates this SPI communication, handling data transmission and reception effectively.

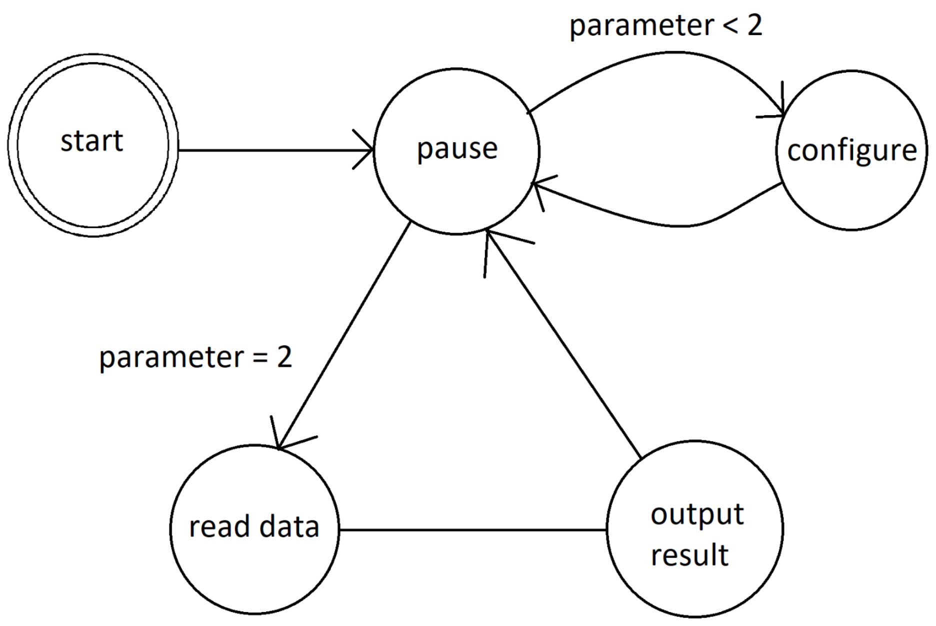
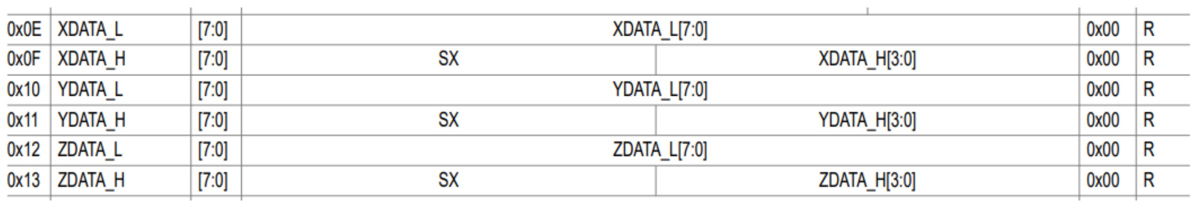


Figure 4: Accelerometer controller FSM

Note that the parameter count will stop at 2, making the FSM run in the pause -> read data -> output result loop until a reset occurs.

The accelerometer controller uses the SPI Master to send data. When initially configuring the ADXL362, it sets the data range and data rate at address 3C in the first iteration, followed by some data to address 3D in the second iteration, which informs the ADXL that we are about to start measuring. Now we are ready to read the X, Y and Z registers for acceleration data, which reside at the registers shown in figure 5.



0x0E	XDATA_L	[7:0]	XDATA_L[7:0]		0x00	R
0x0F	XDATA_H	[7:0]	SX	XDATA_H[3:0]	0x00	R
0x10	YDATA_L	[7:0]	YDATA_L[7:0]		0x00	R
0x11	YDATA_H	[7:0]	SX	YDATA_H[3:0]	0x00	R
0x12	ZDATA_L	[7:0]	ZDATA_L[7:0]		0x00	R
0x13	ZDATA_H	[7:0]	SX	ZDATA_H[3:0]	0x00	R

Figure 5: Registers of the ADXL362[1]

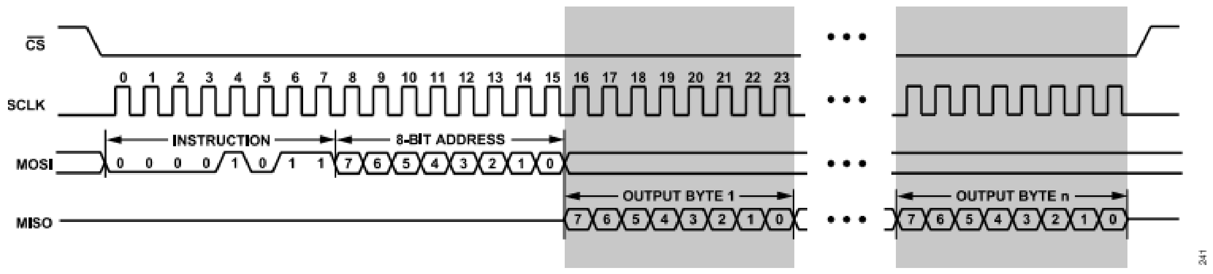


Figure 6: Burst Read/Multibyte Read [1]

The ADXL has a feature called burst read, where the address automatically increases, without the need for writing new addresses. This is perfect for our design since we want to read the 6 consecutive registers seen in figure 6. It is utilized in our design by using the continuous mode of the SPI Master.

The reading of the registrars works by first sending a read command to the over the SPI communication line, followed by the first registrar address in the sequence shown in the figure 0x0E. After that is done we just send an empty data load byte of “00000000”, and the accelerometer will increment the addresses automatically for every such byte.

After we have finished sending the initial address byte, the revive bits start arriving from the accelerometer. The SPI master alerts that the receiving buffer is ready to be collected by deasserting the busy flag. For this reason we watch for the deassertion of the busy flag so that we know when to collect data from the SPI Master.

The data bytes returned from the accelerometer will have the same sequence as we are reading the address, meaning that the bytes will arrive in the order: X(LSBs), X(MSBs), Y(LSBs), Y(MSBs), Z(LSBs), Z(MSBs). The Xs are grouped together in a 16 bit vector. Same goes for the Ys and the Zs.

In the last step this data is outputted.

2.2.3 Bin to int selector

This module serves the purpose of translating binary data received from an accelerometer into an appropriate integer value, depending on whether the binary value is signed or unsigned. The module comprises three inputs, two of which are 9-bit binary values labeled as x and y. The third input, functioning as a switch, determines which of these binary values should undergo the conversion process.

Upon receiving the binary input, the module examines the Most Significant Bit (MSB) of the specified binary value. If the MSB of the signed binary is negative it is first negated and then turned into an integer, otherwise it is just converted as normal. This is because we treat negative angles as though they were positive. Essentially, the module acts as an intermediary step in translating binary accelerometer data into a format suitable for further processing within the system.

2.2.4 Angle converter

This block is used for deriving angles from the gravitational values. The input is the acceleration of either roll or pitch and the MSB of the Z value, which is the bit that determines if Z is signed.

When measuring the highest and lowest possible values that the accelerometer could give us it was 128 and -128 respectively. This module maps this input domain to the corresponding angle, and the mappings where calculated using the simple formula input * 90/128 rounded to the closest integer value, but since we are only dealing with positive angles, and the input will always be positive, we only have and need mappings for all positive values 0-128.

Furthermore, when the Z bit turns positive it means that the roll or pitch has exceeded 90 degrees and the gravitational value will decrease, meaning that the angles will drop again after reaching 90. even though the actual angle is in fact increasing. This is solved by using the formula 180-input *90/128 to keep increasing the angle, even though the angle value (which is dependent on the gravitational value) is decreasing when the Z-flag is negative.

2.2.5 7-segment display controller

The role of the 7-segment display controller is to facilitate the presentation of the angle on the 7-segment display. Since the maximum angle value in our case is 180, a three-digit 7-segment display has been employed. The individual digits correspond to the units, tens, and hundreds places. By utilizing divide and modulo operations on the angle value, each digit is assigned its appropriate value. The display of these digits is organized in such a way that only one of them is active at any given time. To achieve this scheduling, a counter is employed, which cycles through and activates each digit after every 333333 clock cycles.

As each number has a distinct binary pattern on a 7-segment display, a mapping array is utilized to translate each numerical value into the corresponding 7-segment byte. The update of the 7-segment data for each digit occurs at regular intervals, precisely every 0.25 seconds or 25000000 clock cycles.

2.2.6 LCD Controller

The LCD controller works as the FSM shows in figure 7.

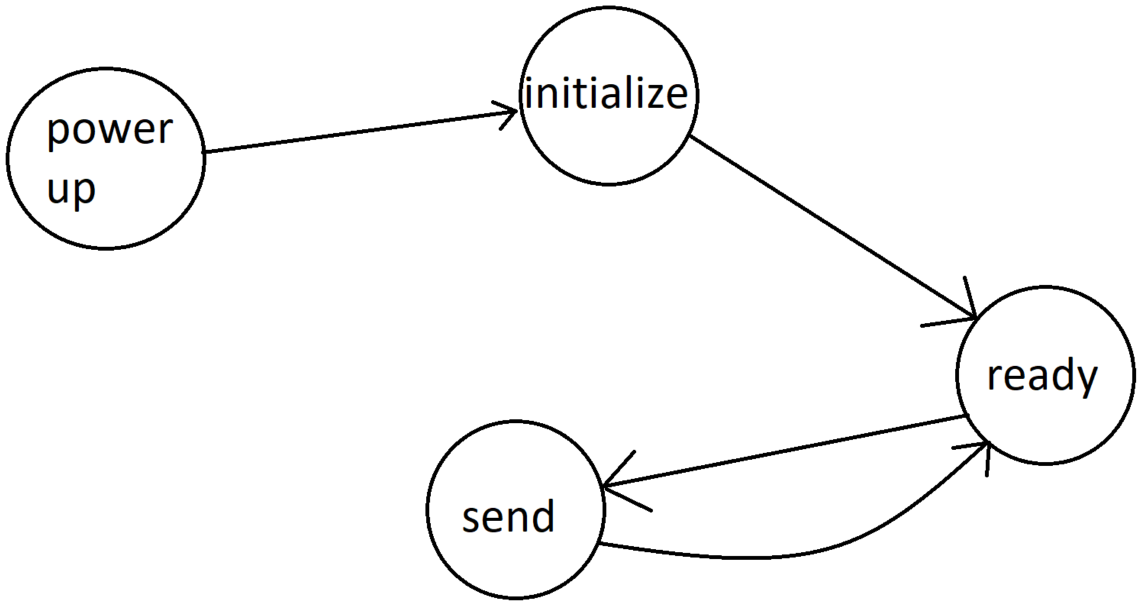


Figure 7: FSM of lcd_controller

Power up:
The power up state is a wait of 50 ms to ensure Vdd has risen and required LCD wait is met.

Initialize:
This state is used for configuring and initializing the display. We run the following instructions with the appropriate waits in between them according to specifications in the LCD datasheet[3]:

Function set.
Wait 50 microseconds
Display on, cursor off, blink off.
Wait 50 microseconds
Display clear
Wait 2 milliseconds
Entry mode set
Wait 60 microseconds

Ready:
The ready state uses an internal variable which counts from 0 up to 3 in a loop. Depending on the variable, it sends one out of four commands to the send state. 0 means clear, 1 means write the hundred digit of the angle input, 2 means write the tens digit of the angle input and 3 means write the ones digit of the angle input. This loop makes sure that the screen first is cleared and then the output is written on an empty display. This is done every 1 second, making the LCD have a refresh rate of 1 second.

Send:

Sends a command determined in the ready state, and then goes back to the ready state. This is done with appropriate timing for LCD write commands.

2.2.7 UART Transmitter

The UART (universal asynchronous receiver-transmitter) is a type of connection protocol that allows full duplex communication which means both sending and receiving. Data being sent/received via two-wire serial port i.e TXD/RXD. Host computer can send and receive data via COM port e.g. by I/O commands. The information/data sent between devices is in the form of frames. A frame is a structured unit of data. When using UART for data transmission, each frame comprises various components: the start bit, actual data bits, stop bit, and optionally, a parity bit. The transmission begins with the start bit, which initiates with a transition from the high state '1' to a low state '0'. Following the start bit are the actual data bits. The stop bit signifies the end of the data, and the signal remains in the high state thereafter.

The UART protocol uses the baud rate to determine the transmission rate, with baud referring to the number of symbols transmitted per second. The Baud rate is measured in bps (bits per second), indicating the number of bits transmitted per second. In this project, the baud rate is set to the standard rate of 9600 bps. In this project, the UART subsystem is tasked with sending the received angle to the PC, with incoming data consisting of one byte. The specific byte to be sent is determined by an external switch. To indicate when data can be sent, an enable signal serves as an input to the UART transmitter. The information is transmitted when the enable signal becomes high. This project involves half-duplex communication between the PC and FPGA, as only the FPGA sends data to the PC.

The UART transmitter operates in two states: idle or transmit. The state machine remains in the idle state until the enable signal is low. Once the enable signal becomes high, data will be loaded into an internal data buffer and the state is set to transmit, and the data frame is sent. To determine which bit should be transmitted by TXD (since TXD is one bit), a pulse generator is employed, sending a baud pulse every 10417 clock cycles. After generating a baud pulse, the databuffer's bits shift, and databuffer(0) is placed on the TXD wire. A counter was utilized to keep a precise record of the transmitted bits. The system transitioned back to an idle state once all bits, including the start signal at the beginning and the stop signal at the end, had been successfully sent. A sequence of data bits "xxxxxxxx" can be sent as follows:

```
"xxxx_xxxx_0_1"                 - -send: 1

Start
"1_xxxx_xxxx_0"                 -- send: 0
"1_1_xxxx_xxxx"                 -- send: x1
"1_1_1_xxxx_xxx"                -- send: x2
"1_1_1_1_xxxx_xx"               -- send: x3
"1_1_1_1_1_xxxx_x"              -- send: x4
"1_1_1_1_1_1_xxxx"              -- send: x5
"1_1_1_1_1_1_1_xxx"             -- send: x6
"1_1_1_1_1_1_1_1_xx"            -- send: x7
"1_1_1_1_1_1_1_1_1_x"           -- send: x8
```

```
"1_1_1_1_1_1_1_1_1_1"          -- send: 1
"1_1_1_1_1_1_1_1_1_1"          -- send: 1
End
```

Using this approach only LSB bit of the buffer being sent by TXD. The shift operation does the main job to place the right bit on tx. The state machine diagram for the UART transmitter is shown in figure 8.

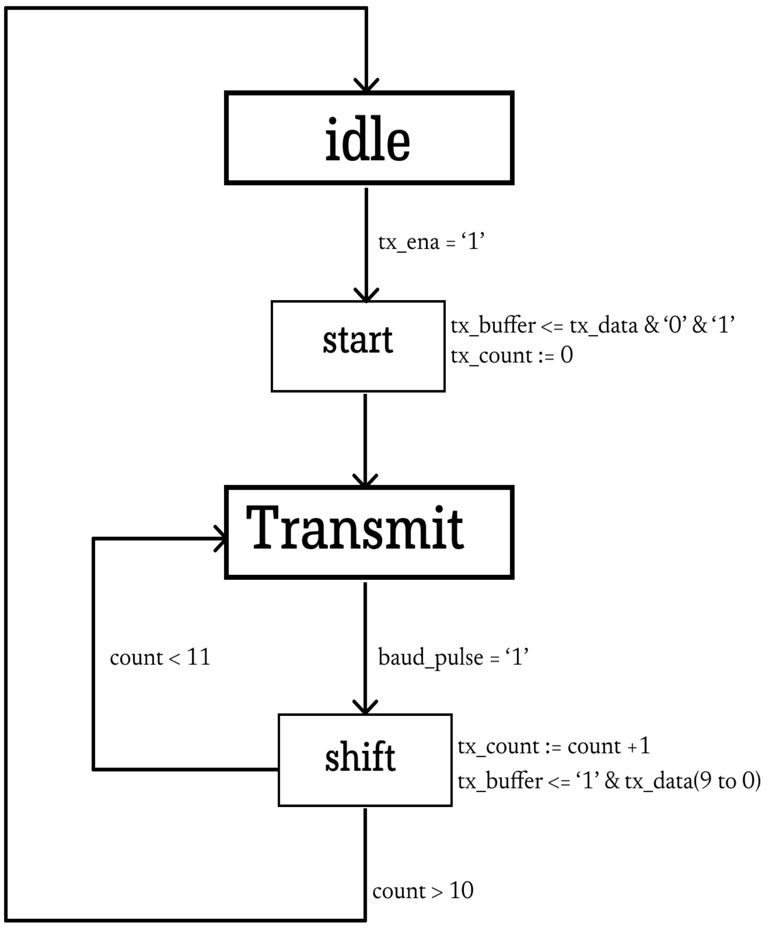


Figure 8: FSM for the UART module

3. Error Analysis

3.1 Rounded angles

When converting acceleration values into angles, in the angle converter component, the ideal way would be to use the formula (acceleration value * 90/128) without any round offs. But since floats are not used in VHDL we have to round off the result, meaning an acceleration value corresponding to the actual angle 67,53 is mapped to 68 in our design. This is not a crucial error.

3.2 Errors in measurement

Early in the project, before we used angles, and only outputted raw binary values from the accelerometer we noticed errors in the measurements. For example holding the electronic level 90 degrees would not yield the expected value when examining the y value. Furthermore the z value is, according to the ADXL datasheet, supposed to flip to negative when holding the electronic level at any degree > 90 degrees, but it did not turn negative until an approximate angle of 100 degree. It makes angles around the 90 degrees have an error of multiple degrees. This is a major flaw, making the project very impractical and unsuitable for real-world usage.

4. Appendix

4.1 Work statement

In our project, we divided the workload among us three, not only to optimize time management but also to ensure that each individual's expertise was effectively applied in the most relevant area of the project.

UART - Mehdi

In developing the UART component for our project, as described in the `uart.txt` file, Mehdi's role was crucial. He focused on structuring and implementing the UART's functionality for efficient serial communication. The code he crafted defined the UART entity with essential configurations such as a baud rate for communication speed. His notable contributions included establishing a baud pulse generator to ensure accurate timing for data transmission and designing a state machine to manage the UART's transmission process effectively.

In the code, Mehdi implemented a process to handle the transmission state, where he skillfully managed the transmission of each bit, including a start bit and data bits, ensuring proper data framing. The transmission logic he developed efficiently handled the data bits and the necessary shifts for correct serial transmission. This approach ensured that the UART could reliably handle serial communication, effectively transmitting data as intended in our system.

LCD - Theodor

Theodor was entrusted with the critical task of managing the LCD display interface in our project. His primary focus was on ensuring the clarity and reliability of data displayed on the LCD. To achieve this, Theodor designed the interface, which played a key role in improving user interaction with our system. He skillfully implemented control sequences that governed the initialization and operation of the LCD, carefully managing the timing and the flow of data from the system to the LCD module. His responsibilities also extended to configuring various display characteristics, such as display lines and character font, as per the requirements of our project. Theodor's expertise ensured that the LCD displayed information accurately and responded appropriately to the system's operational changes.

SPI-Master - Baseer

Baseer took on the role of developing the SPI Master, ensuring accurate and effective SPI communication with the accelerometer. He managed the SPI bus's data transmission and reception and optimized the timing and sequencing of SPI operations. Baseer implemented a state machine to control the SPI communication process, ensuring accurate clock management. He also developed logic for data buffering and transmission, and managed the busy signal and continuous mode, essential for efficient data handling. Additionally, he handled the chip select signal, a key factor in targeting the accelerometer for communication.

Accelerometer controller

The development of the Accelerometer Controller in our project was a team effort. We all worked together because the accelerometer was a key part of our system. In our joint work sessions, we focused on setting up the accelerometer correctly so it could capture motion data accurately. We also worked on how to get this data efficiently into our system. Together, we created ways to make sense of the raw data from the accelerometer. This teamwork was essential for making sure the Accelerometer Controller worked well with the rest of our system, ensuring everything operated in sync. Working together like this not only helped us understand our system better but also led to a stronger and more effective solution, which was a big part of our project's success.

4.2 Wiring diagram

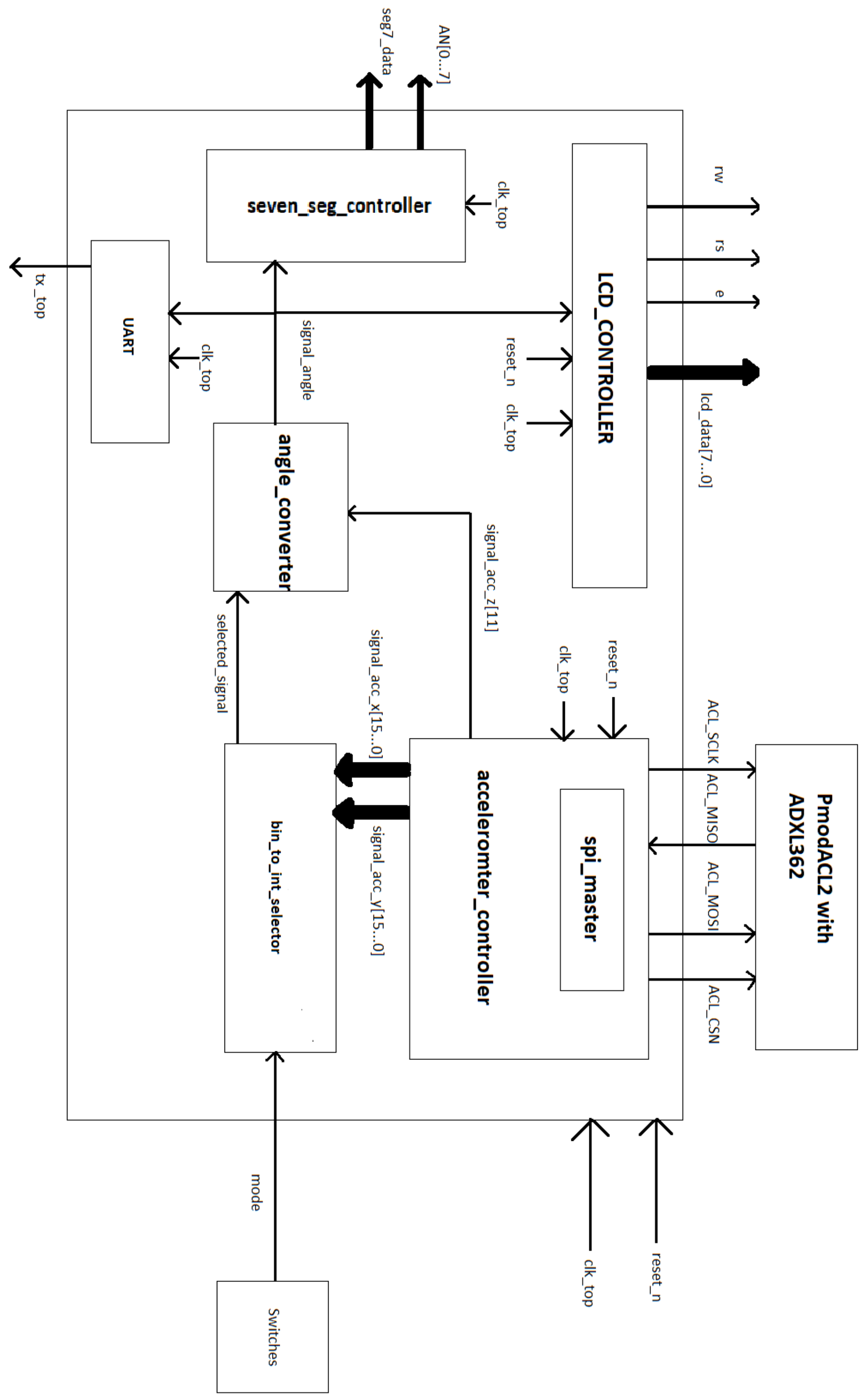


Figure 9: Wiring diagram

4.4 Circuit board layout

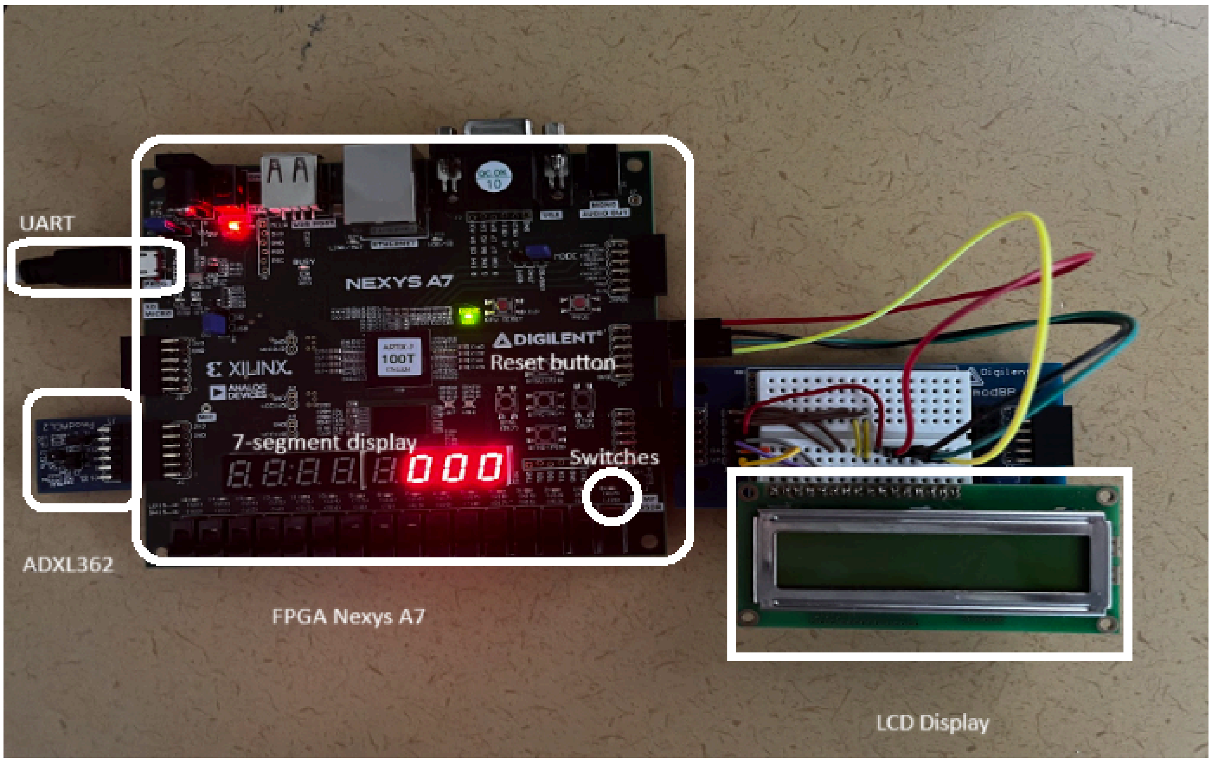


Figure 10: Circuit board layout

4.4 Signal names

used pins (package pin)	signal name	Description
E3	clk_top	System clock at 100MHz
C12	reset	System reset signal, synchronous active high
J15	mode	Input signal from switch J15 on the FPGA
G6	ACL_SCLK	SPI clock at 5MHz

K1	ACL_CSN	Chip select
J2	ACL_MISO	Master in, slave out
F6	ACL_MOSI	Master out, slave in
H14	rw	Read & Write output for LCD
H16	rs	selection signal for LCD
E16	e	enable signal for LCD
C17, D18, E18, G17, D14, F16, G16, F13	lcd_data	data output of LCD
T10, R10, K16, K13, P15, T11, L18, H15	seg7_data	BCD for 7 segment display
J17, J18, T9, J14, P14, T14, K2, U13	AN	Select digit to light up for 7-segment-display
D4	tx	transmit pin over UART
internal	signal_angle	angle of the accelerometer, to send to the displays and UART

internal	selected_angle	The acceleration value as an absolute value integer of the selected axis, X /Y
internal	signal_acc_x	The acceleration in axis X (9 bits signed binary)
internal	signal_acc_y	The acceleration in axis Y (9 bits signed binary)
internal	signal_acc_z	The acceleration in axis Z, only the MSB, which is the signed bit

4.4 Component list

- FPGA Board: Nexys A7
- LCD module: DMC16117A
- Accelerometer: PmodACL2 with ADXL362
- UART-USB bridge: FT2232

5. Sources

[1] https://www.analog.com/media/en/technical-documentation/data-sheets/adxl362.pdf
[2] nexys-a7_rm.pdf (digilent.com)
[3] https://pdf.datasheetcatalog.com/datasheets2/16/169299_1.pdf