LUND UNIVERSITY

IC Project 2 (ETIN40)

Edge Detector

Prasanth Kasula (pr1133ka-s)

2022
Lund University

Contents

List of Figures

List of Tables

1 Introduction

Edge detection aims at finding points in a Monochrome Image (352x288) as specified, at which image intensity changes sharply or has discontinuities. The points at which image intensity changes sharply are usually organized into a set of curved line segments called Edges. During the course ETIN35 a hardware accelerator for the Edge Detector algorithm has been implemented through an ASIC design implementation. This project aims to evaluate the performance of the hardware accelerator used in the ASIC design for the Edge Detector algorithm by implementing it on an FPGA. The simulated results of this testing will be compared to the results of an external Logic Analyzer.

The structure of this report is as follows:

- Introduction: A brief overview and introduction to the project.
- Architecture: Detailed information about the architecture and changes made from the ICP-1(ASIC Implementation).
- ASMD Description: A detailed explanation with ASMD of all the modules used in the project.
- Simulation Results: Vivado simulated results are presented here.
- Verification Setup: presentation of logic analyzer output and a comparison between logic analyzer and Vivado simulated results.
- Conclusion: overall project outcome and key takeaways.

2 Architecture

2.1 System Architecture

Figure 1 shows the block diagram of the Edge Detector. To reduce complexity in the design it is divided into four smaller modules. The controller module controls all the significant control signals in the design on its command the first nine 8-bit pixels according to Sobel convolution will be sent to and stored in registers of the Image data module and after storing the elements upon the command of the controller these elements are sent to the XY Gradients module where the x and y derivatives are calculated and the sum of both derivatives after thresholding is stored in the FPGA block memory which can also be accessed for the reading purpose the whole read and write operation are controlled by memory control module.

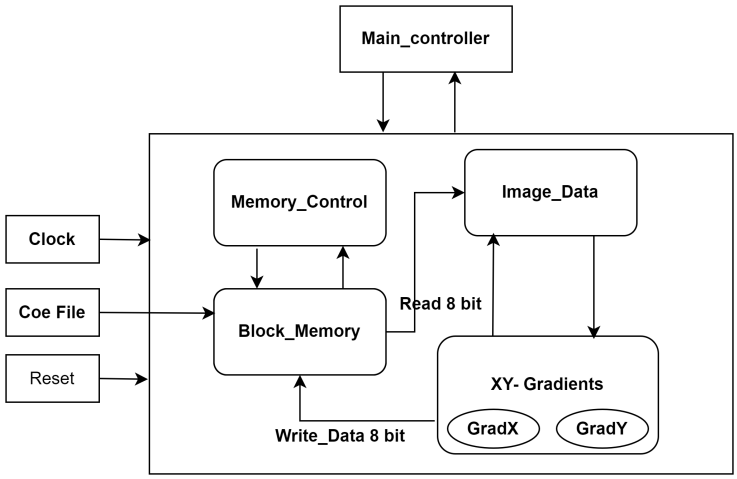

Figure 1: Edge Detection System Architecture

2.2 ASMD Description

As shown in figure 2, the Main controller handles and controls all other units in the system and synchronizes different operations. On receiving the start signal the controller sends a fetch request for fetching 8-bit binary pixels and after completing the fetch the Image data module sends a fetch complete acknowledgment on receiving this the controller starts the x and y gradients calculation operation is performed by convoluting the original image pixels with Sobel x and y filters after calculating the Gxy from x, y gradients the Gxy data is sent BRAM which sends an acknowledgment(conv_out) to the controller on which the controller sends a signal to fetch the data for 2nd pixel and this process follows until completion of 100100-pixel calculation.

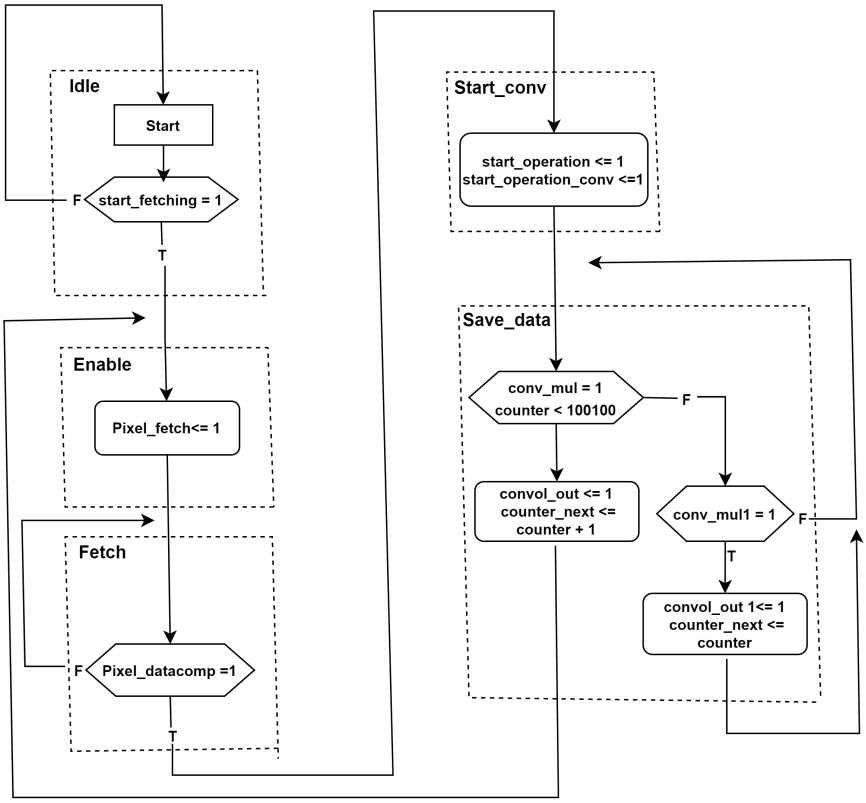

Figure 2: ASMD of the main controller

As shown in figure 3, the Image data module is used to fetch store, and send Image data on receiving fetch (Pixel fetch) ack from the controller it starts fetching pixel data and stores it in 3 different 24-bit registers after fetching all 9 elements of the first matrix it sends a fetch complete ack (pixel_data_comp) to the controller and after receiving an ack from the controller (start operation) it starts sending all the elements to the xy_gradients module after sending all the elements it goes back to an idle state waiting for further instructions.

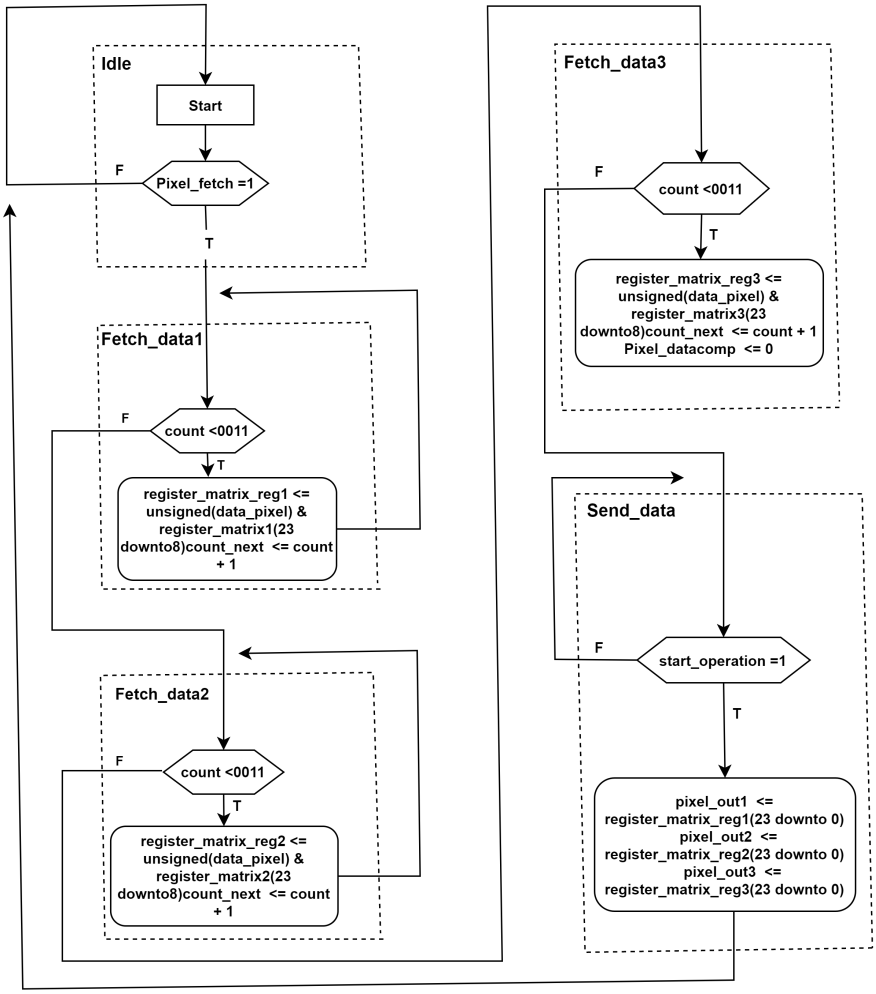

Figure 3: ASMD of the image data

As shown in figure 4, XY Gradients module handles the convolution calculation of x and y gradients and thresholding operations on receiving the start operation acknowledgment from the controller it convolutes the 3X3 pixel elements from image data with x and y sobel filters calculating the x and y gradients these x and y gradient absolute values are added and the threshold of this sum gives Gxy value which it sends to block_memory module and once all the data is written in BRAM it is read again with an acknowledgment and finally it moves back to an idle state waiting for further instructions.

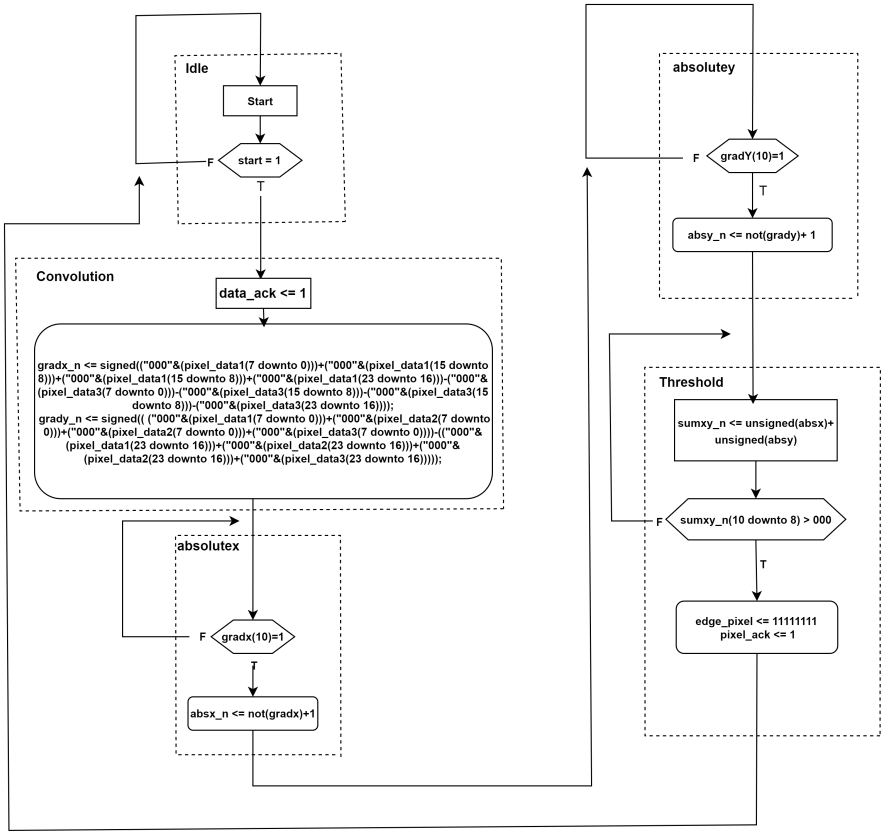

Figure 4: ASMD of the XY Gradients

As shown in figure 5, the Memory control module's purpose is to control read/write to block memory. This module starts the read operation on starting the hardware accelerator. The 352x288 image data is stored in the form of pixels in the block memory each location is stored with an 8-bit pixel first 288 memory locations are stored with 288 pixels of the first row and second row and so on as we are using the Sobel operator for calculating gradx and Grady we are moving between 3 different states input_fetch1, input_fetch2, input_fetch3 by using 3 different counters once we calculate all the grads of x and y in the first row and we move to the next by using another counter once all the gradient values are calculated it moves back to idle state.

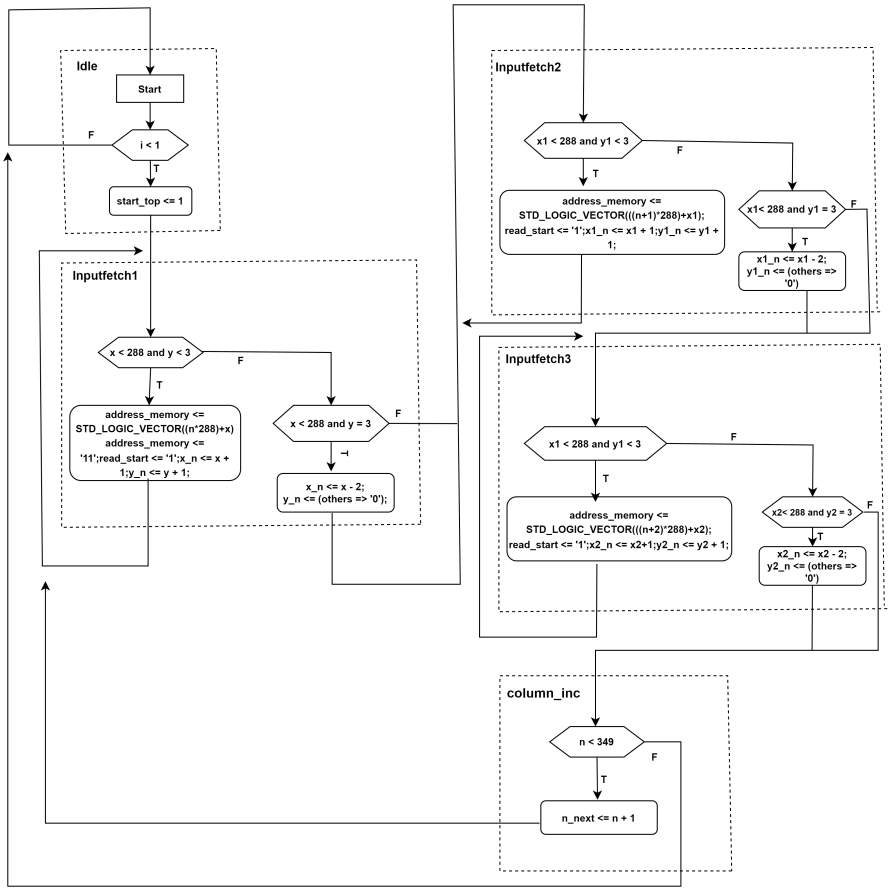

Figure 5: ASMD of the Memory control

3 FPGA Implementation

The Xilinx Nexys4 FPGA, which belongs to the Artix-7 family, was used for performing simulations. It is based on a hierarchical architecture that consists of several layers of programmable logic, interconnects, and input/output (I/O) blocks. As FPGA has internal RAM so, the multiple memory modules used in ASIC design of ICP1 have been replaced with single block memory in FPGA with 8-bit width for each memory location. The input data for design testing is stored in the FPGA BRAM and the output data is also written and read from this BRAM.

3.1 Resource Utilization

Table 1 shows all the hardware components that were used in this design after synthesis including BRAM.

Table 1: Resource utilisation on FPGA

Resources	**Utilisation**	**Available**	**Utilisation %**
LUT	452	63400	0.71
FF	184	126800	0.16
BRAM	1	135	0.74
IO	0	210	10.95
BUFG	0	32	3.13

Table 2: ASIC and FPGA sequential element comparison

	ASIC	**FPGA**
FF	256	184

When we compare our current FPGA hardware with that of ASIC hardware in Table 2, we can observe that number of sequential elements in FPGA is reduced this is because of modifications made to the memory control module in FPGA specifically the memory control module in FPGA only used to read the pixel data from BRAM but in ASIC the memory control module is used to write the final edge pixel data into two huge external memories(9600x64) to completely utilize each block of external memories multiple 64- bit shift registers and counters were used to store the edge data before sending it to the external memory, usage of these memories for storing the final data caused the increase in flip flop count.

4 Simulation

In order to test and validate the functionality of the EDGE detector project design(RTL code) behavior simulations were performed in vivado, xilinx. The clock and reset are given to the design by using a test bench and the input is given through a coe file which is pre-loaded to the newly generated BRAM. Post-synthesis simulations are performed after the RTL code is synthesized into a netlist of gates. Post-synthesis analysis is important to ensure that the design can be implemented on the target FPGA device and meets its performance and power requirements.

As shown in figure 6 Once the Edge pixel is calculated it is written to FPGA BRAM for storage when all the pixels are stored in the BRAM, they are again read one by one for comparing these values with the logic analyzer simulations. While reading the edge data from memory a 2-clock cycle delay has been observed which is expected while reading from FPGA memory. When we compare the vivado simulation with the logic analyzer simulations they are found to be identical.

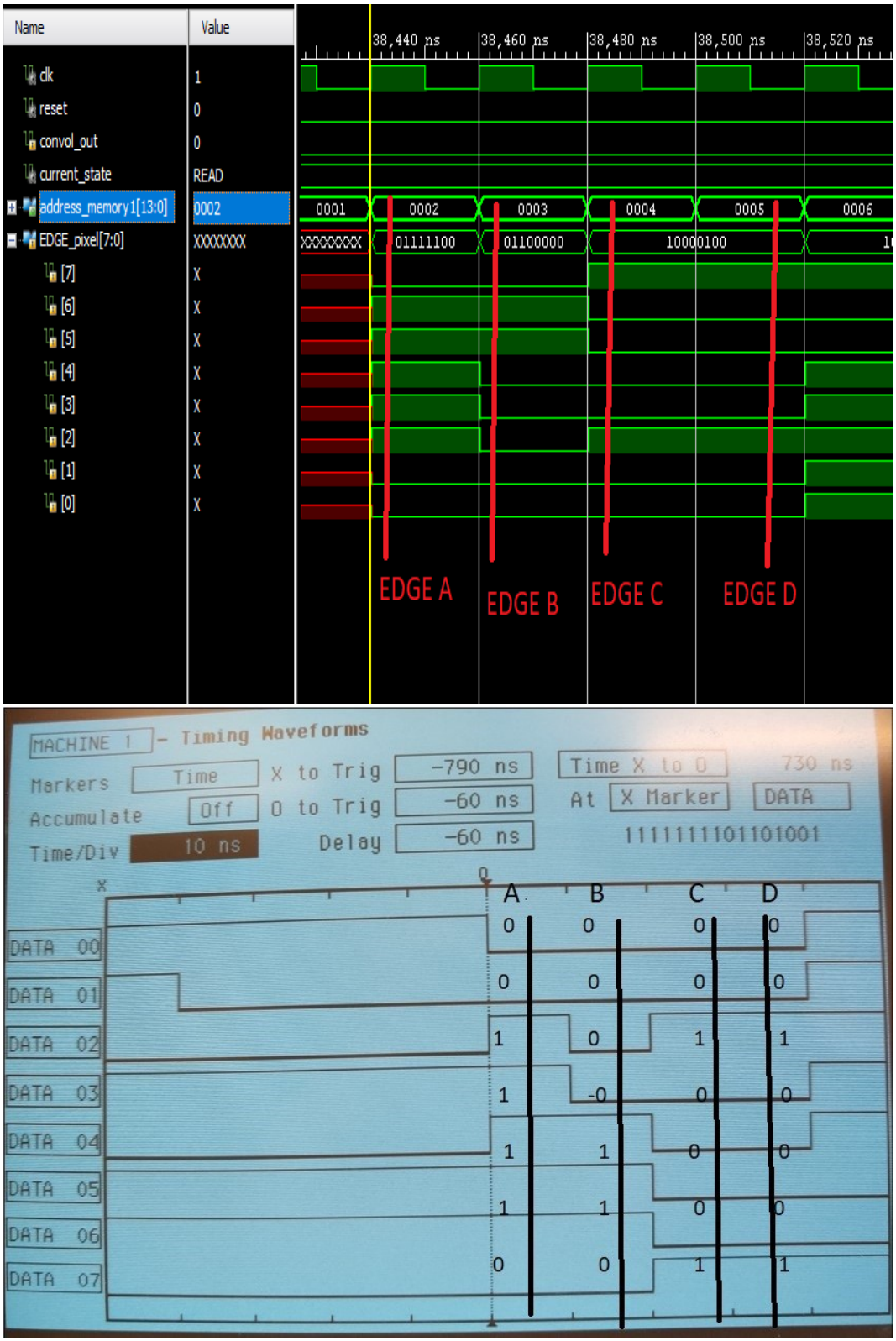

Figure 6: Comparison of vivado and logic_analyzer output

5 Verification Setup

After performing the simulations for post-synthesis on the RTL design, the next step was to test the design on the logic analyzer. As illustrated in Figure 7, FPGA was connected to the logic analyzer through PMOD ports for testing. To capture waveforms, FPGA's 11 PMOD ports were connected to the logic analyzer POD connection, as shown in Figure 8. The signals from the FPGA are fetched to the logic analyzer via the PMOD pins on the board. There were 8 output pins for data, one for reset, one for the clock, and the last one for the ground.

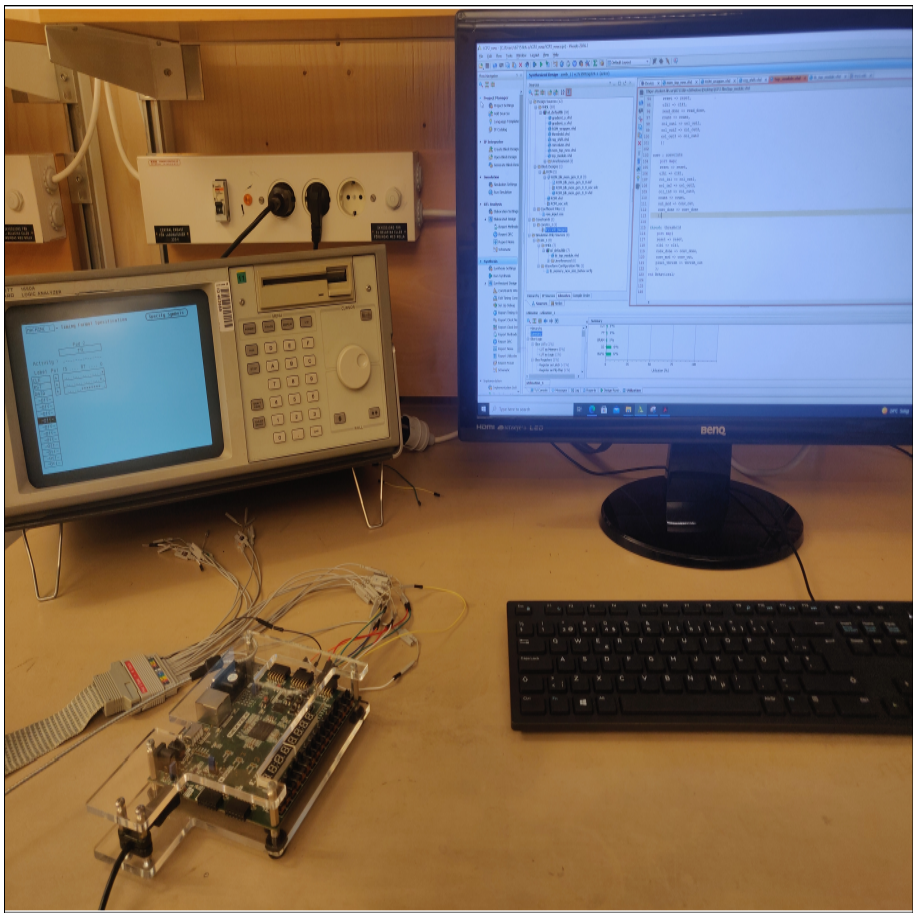

Figure 7: verification setup

Table 3 compares the output data signals A,B,C,D of the design from both Vivado and the logic analyzer as shown from the figure 6. It can be seen that the output data from both sources match, indicating that the design is functioning as expected

Table 3: Vivado vs Logic Analyzer output

Signals	**Vivado Output**	**Logic Analyzer Output**
A	01111100	01111100
B	01110000	01110000
C	10000100	10000100
D	10000100	10000100

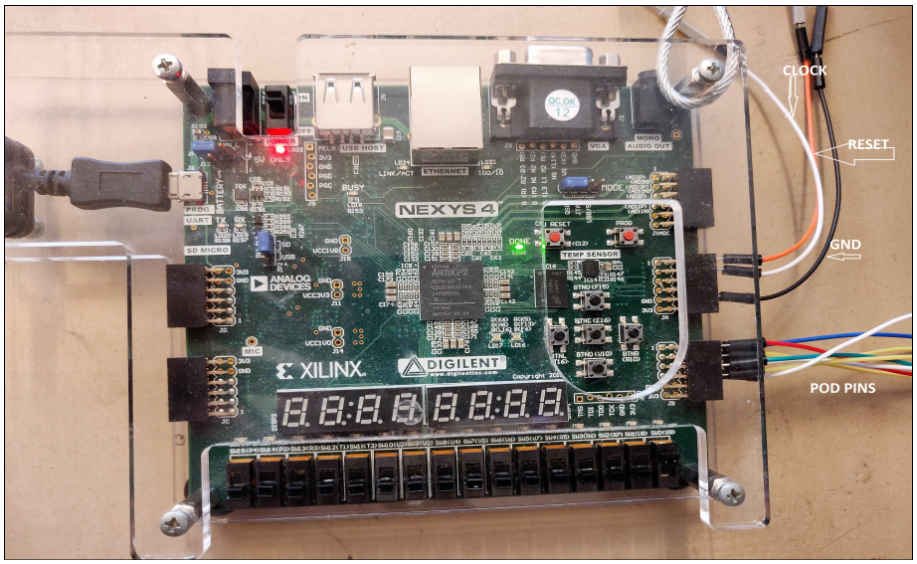

Figure 8: FPGA with connected pins

6 Conclusion

This project provided a valuable insight on FPGA Design verification,RTL design coding and testing. By building the same RTL hardware design on both ASIC and FPGA and testing it using an external logic analyzer i understood how the RTL code and Hardware components varies in both designs. In addition to the skill set acquired in hardware designing i gained a greater understanding on the concepts like Power versus Performance versus Area trade off of the IC design flows. Overall, this project, along with ICP-1, gave me a deeper understanding and insight over ASIC/FPGA design and testing.

7 Reference

[1] Nexy's 4 FPGA Board Reference Manual.

8 Appendix

```
1)constraints
xdc file
## Clock signal
##Bank = 35, Pin name = IO_L12P_T1_MRCC_35,
set_property PACKAGE_PIN E3 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
##Bank = 34, Pin name = IO_L21P_T3_DQS_34,
set_property PACKAGE_PIN U9 [get_ports {reset}]
set_property IOSTANDARD LVCMOS33 [get_ports {reset}]
##Pmod Header JA
##Bank = 15,Pin name = IO_L1N_T0_AD0N_15,
set_property PACKAGE_PIN B13 [get_ports {douta[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {douta[0]}]
##Bank = 15,Pin name = IO_L5N_T0_AD9N_15,
set_property PACKAGE_PIN F14 [get_ports {douta[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {douta[1]}]
##Bank = 15,Pin name = IO_L16N_T2_A27_15,
set_property PACKAGE_PIN D17 [get_ports {douta[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {douta[2]}]
##Bank = 15, Pin name = IO_L16P_T2_A28_15,
set_property PACKAGE_PIN E17 [get_ports {douta[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {douta[3]}]
##Bank = 15, Pin name = IO_0_15,
Sch name = JA7
set_property PACKAGE_PIN G13 [get_ports {douta[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {douta[4]}]
##Bank = 15, Pin name = IO_L20N_T3_A19_15,
set_property PACKAGE_PIN C17 [get_ports {douta[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {douta[5]}]
##Bank = 15, Pin name = IO_L21N_T3_A17_15,
set_property PACKAGE_PIN D18 [get_ports {douta[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {douta[6]}]
##Bank = 15, Pin name = IO_L21P_T3_DQS_15,
set_property PACKAGE_PIN E18 [get_ports {douta[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {douta[7]}]

##Pmod Header JB
##Bank = 15, Pin name = IO_L15N_T2_DQS_ADV_B_15,
set_property PACKAGE_PIN G14 [get_ports {clk_out}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk_out}]
##Bank = 14, Pin name = IO_L13P_T2_MRCC_14,
set_property PACKAGE_PIN P15 [get_ports {rst_out}]
set_property IOSTANDARD LVCMOS33 [get_ports {rst_out}]
```