

CSSE4010 A2

David Gaul

s4671313

Lab Session: Wednesday 10-12

Submission Date: 21st August

1 Introduction

The aim of the second practical was to create a locking system on the Nexys 4 FPGA. This locking system would receive 4 hexadecimal digits as input, and would either lock or unlock the system depending upon whether it was correct. Furthermore, the hexadecimal digits would be illuminated on four of the seven-segment display panels. The various design requirements were as follows:

- Two buttons were used to control the inputs - button 1 controlled the first two digits, and button two the second two.
- A reset switch would reset the entire system and configure "AAAA" on the seven seg display.
- The entire system should take a synchronous approach.
- Two LEDs had to be present: one for whether the system was locked, the other for when it is unlocked.

The system had to unlock upon the correct input password - which was the last 4 digits of the student's number (3139 in my case).

2 Block Diagram

The final design for the system was surprisingly simple, comprising of only 3 subsystems. These subsystems were as follows:

- A clock divider to provide synchronous computations.
- A register allocation system to transfer inputs into the relevant register areas.
- A seven seg decoder to output the current combination

The register allocation system likely requires more explanation. It is a system that takes the 8 digit input from the FPGA, and store the results within a 16 bit buffer, which was thik like Mrs Incredible. Depending on whether button 1 or 2 was pressed, it would store the 8 digit input within the first or last 8 digits. As such, these digits could be easily accessed by both the locking system and the seven segment decoder.

The block diagram for the entire system can be found within figure 1.

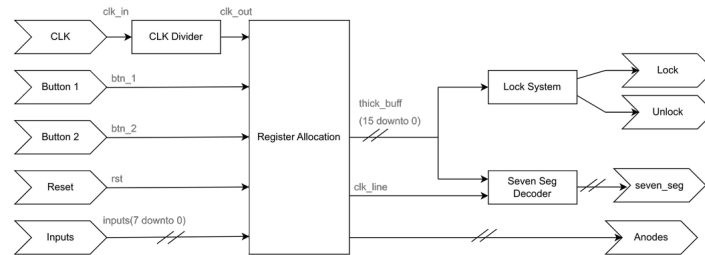


Figure 1: Generalised Block Diagram

The register allocation subsystem is essentially storing input values into a large buffer. The representation of the register can be found within 2. Note that this is an 8 bit register. The lecture slides didn't have a 16 bit register, but we get the idea.

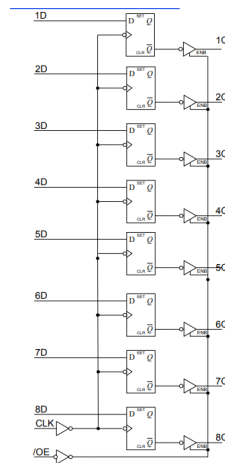


Figure 2: Register Block Diagram

However, there are several other important components within the register allocation, such as the process of storing inputs within the register, as well as which anode is being iterated through. The anodes were cycled through using an iterator connected to the clock line, and the inputs were stored into the thick buffer using a multiplexer. These details can be found within figure 3

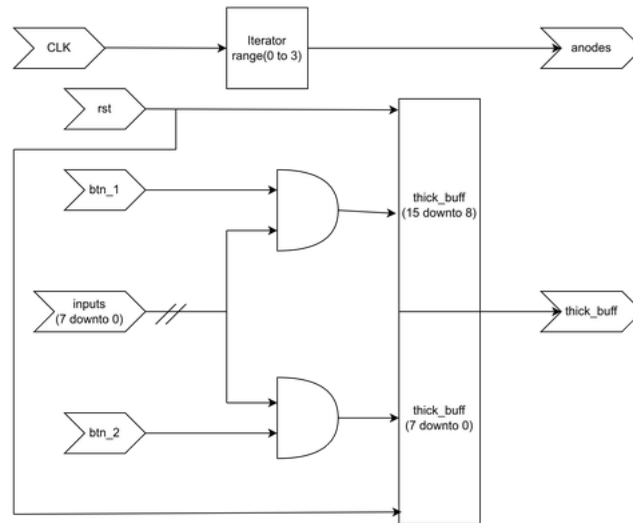


Figure 3: Register Allocation Block Diagram

The seven segment display system is a decoder, which can be easily represented within figure 4.

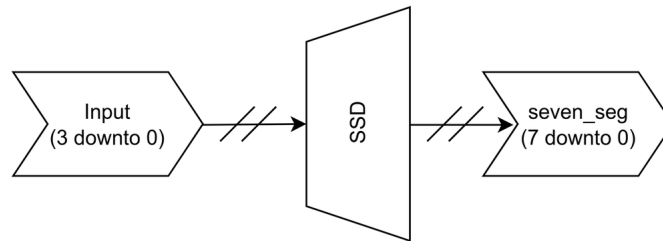


Figure 4: Seven Seg decoder Block Diagram

Lastly, both LEDs are controlled by the locking mechanism. This is simply a comparator that checks whether the thick buffer is the same as the passcode that will unlock the entire system. The logic for the lock and unlock LEDs is as follows:

$$lock <= \overline{(thick_buff = "0011000100111001")}$$

$$unlock <= (thick_buff = "0011000100111001")$$

Didn't draw the circuit schematic for this, don't feel like it's necessary. It's fourth year. We know what an AND gate looks like.

3 Simulation Results

The simulation of the results was kept intentionally brief. This is because, with 2^{16} possible combinations, there were simply too many test cases to include within the simulation. For this reason, the simulation covered the following tests:

- Inputting the correct digits into the system and watching it unlock (3139)
- Selecting the rest button

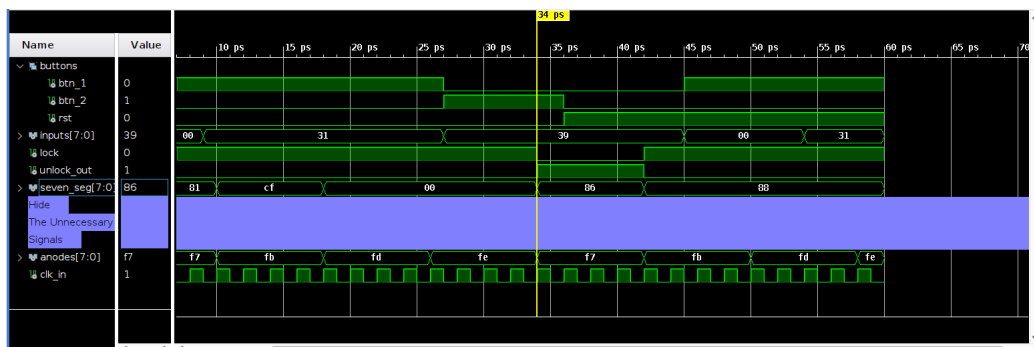


Figure 5: Simulation with the timestamp shown on system unlocking

Figure 5 shows the system unlocking after the correct sequence has been input into the system. As can be seen, before the second combination has been input correctly, the system remains in a locked state. After the buffer system interprets the full input, which can be seen at 34ps, the unlock LED signal is switched to high, and the locked signal is switched off. This is maintained until the system is reset at time 40ps.

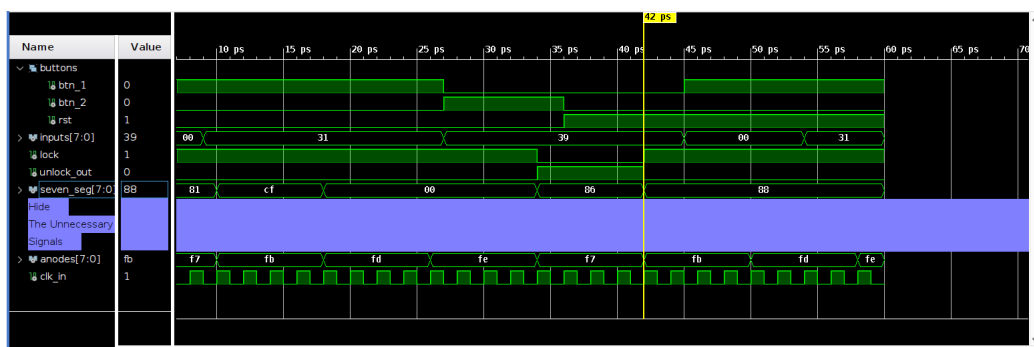


Figure 6: Simulation with the timestamp shown on system resetting

Note that in both cases, the output on the seven-seg line is changing with regards to the inputs varying. The seven segment display is an active low system, meaning that a value of

'0' will activate that LED bar. In order to make it easier to see which bars would be active for any given input, figure 7 shows the wave forms converted over to binary representations. Inputs are also grouped, and dividers because it makes the wave forms easier to interpret.

Or maybe just because it was on the criteria sheet...

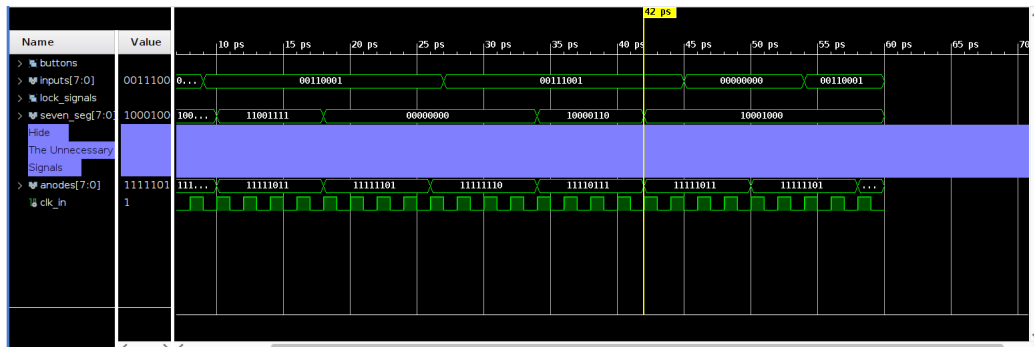


Figure 7: Simulation with binary representation, grouping and dividers

Finally, it was necessary to provide logging information, such as when the system unlocks or is reset. This was implemented into the automated test bed, and is shown on the logging console in figure 8. Note that a slightly different test bed was used in this iteration, hence the different timestamps.

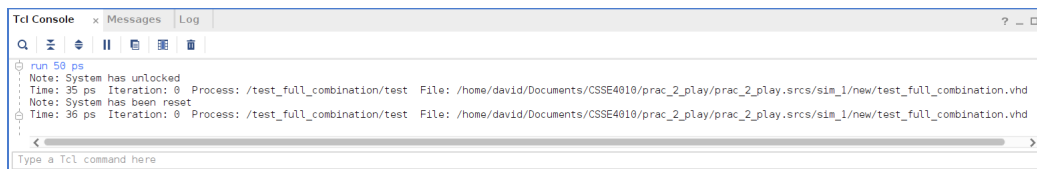


Figure 8: Messages denoting the unlocking and resetting of device

4 Results on the FPGA

Figures 9, 10, 11 and 12 show the successful implementation onto the Nexys 4 FPGA board. A further demonstration of the working device will be shown during the in-class demonstrations.

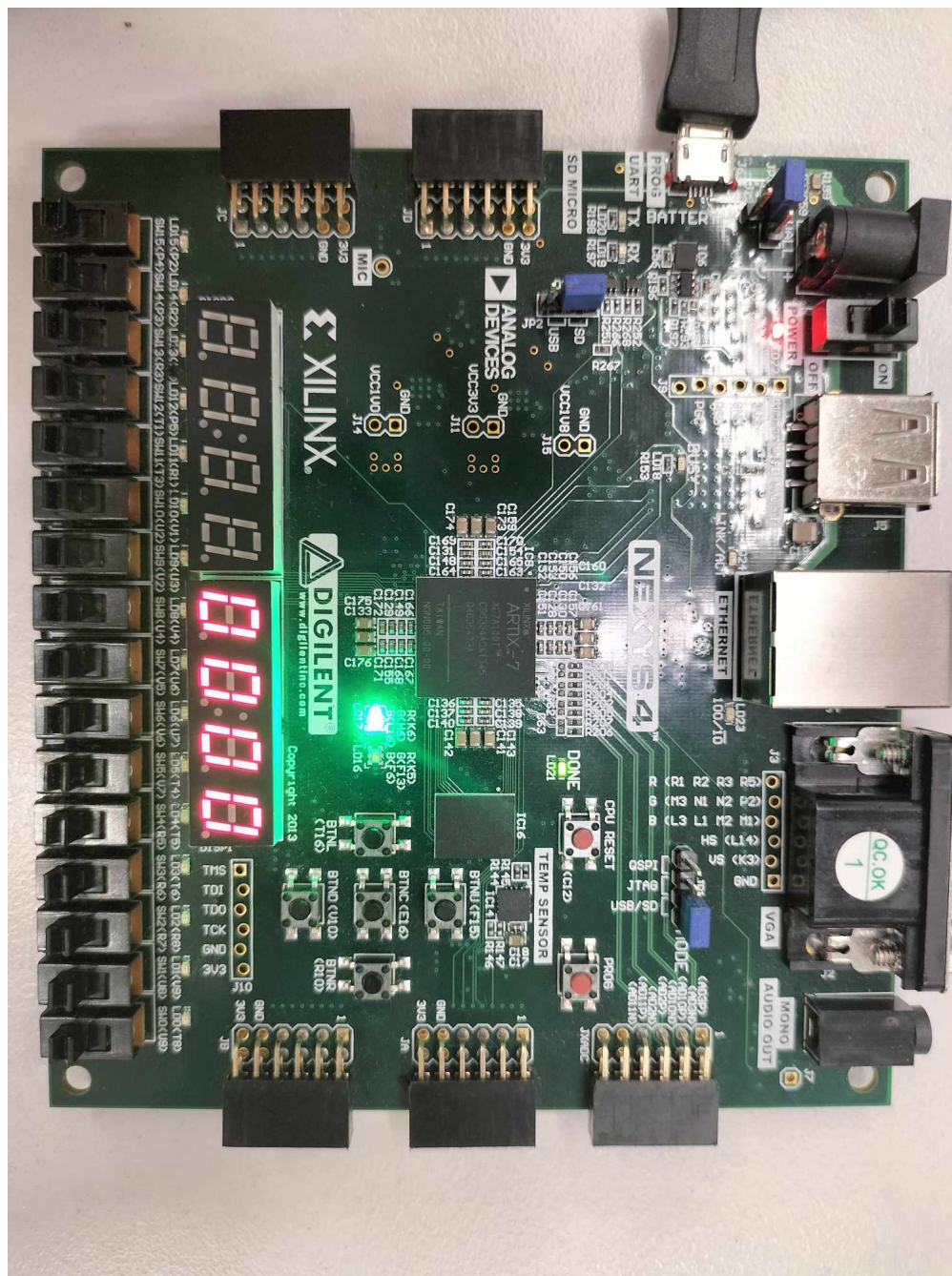


Figure 9: FPGA in starting state

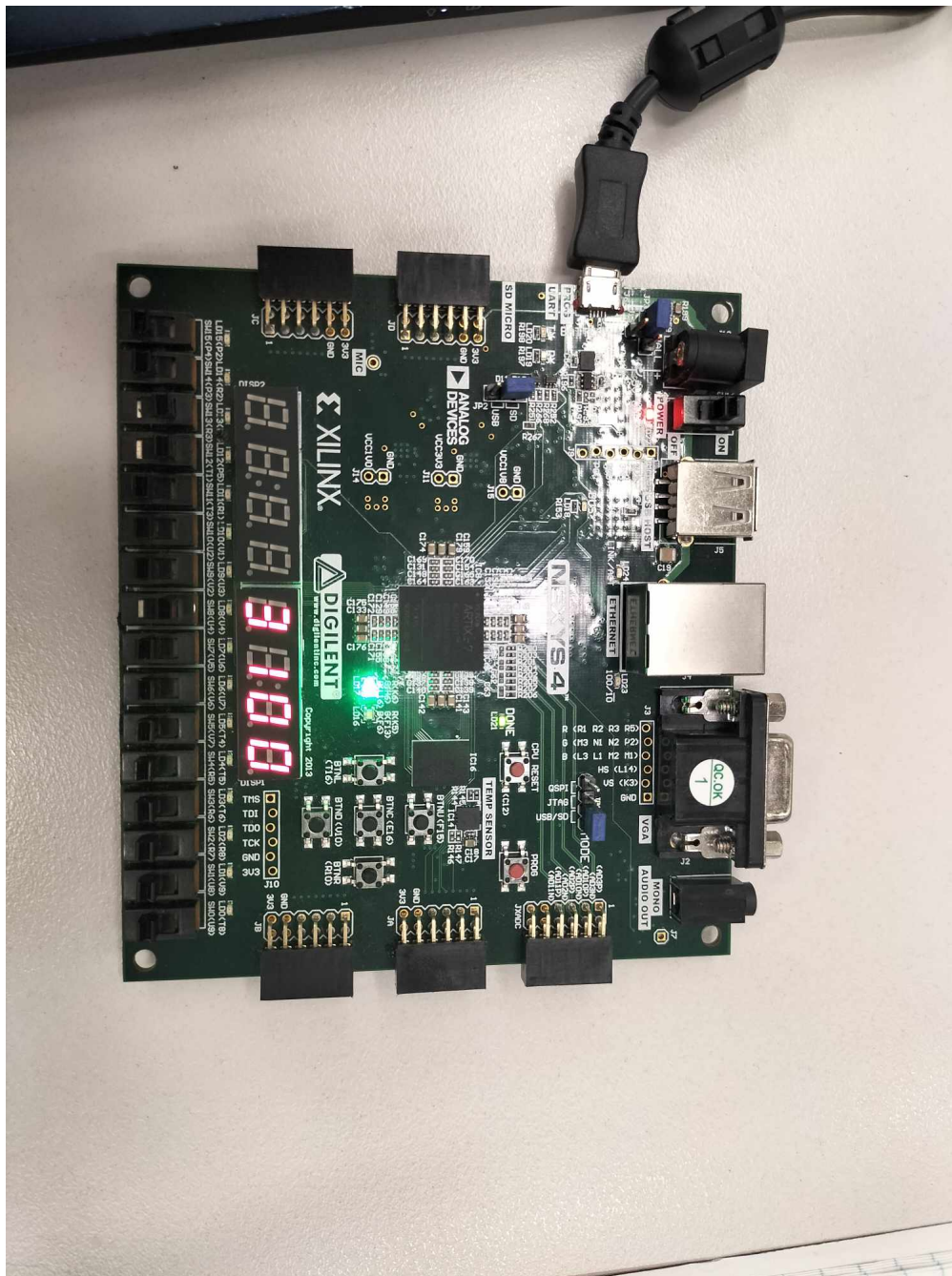


Figure 10: FPGA with button 1 pressed

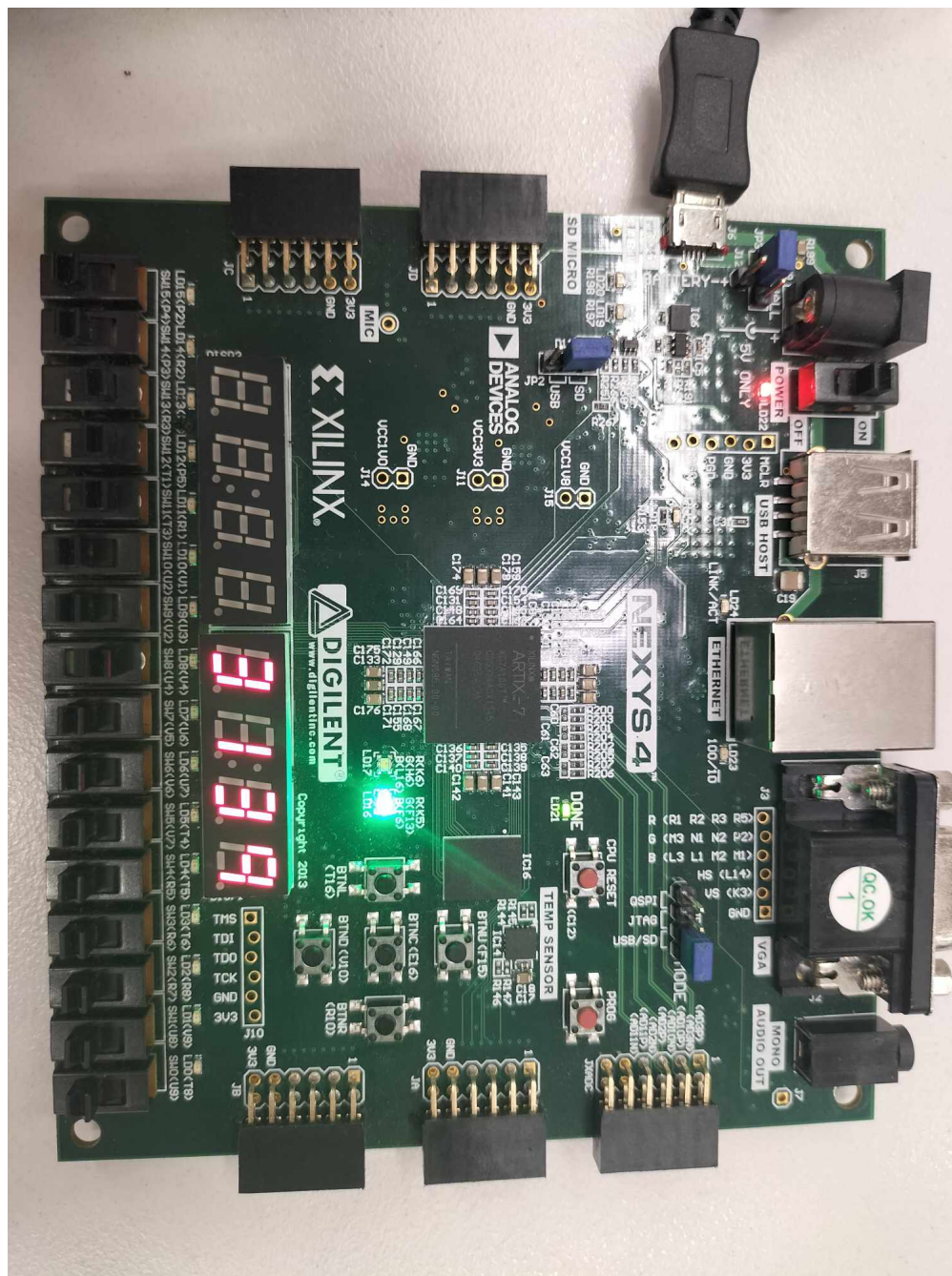


Figure 11: FPGA with correct combination. Notice that the LED has switched from locked to unlocked state

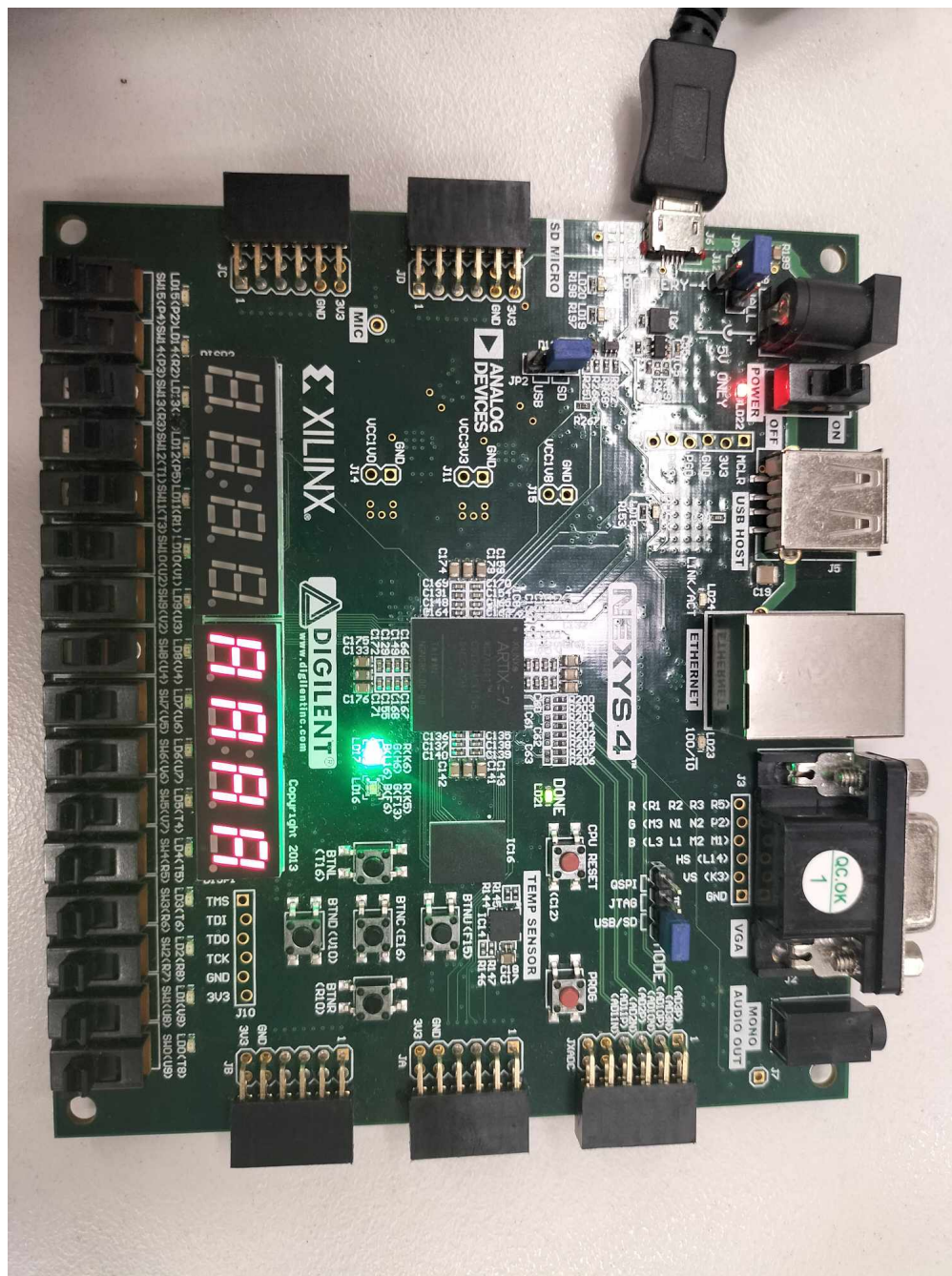


Figure 12: FPGA after selecting the reset switch

As can be seen, total usability has been obtained and the system operates as per the design requirements.

5 Register Transfer Level Schematic

Figure 13 shows the top level design of the RTL schematic. This is going to be explained in greater detail in this segment.

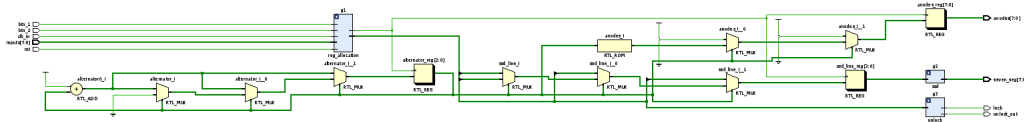


Figure 13: Register Transfer Level top level view

Xilinx is a bit like fearless leader Ian Clough. It's been around a lot longer than I have, and it knows how to optimize a circuit schematic far more efficiently. For this reason, the logic behind RTL designs often look different from the ones that are implemented in code; they do the same thing, just far faster. The areas in blue represent the subsystems for the locking mechanism, SSD and register allocation. Everything around them is simply multiplexers and look-up tables.

And that is an important distinction to make. On the implementation level, Xilinx does EVERYTHING with look-up tables. That's because the Nexys FPGAs are designed using LUTs, which can be seen upon close inspection of the design implementation. Everything else, such as the AND gates that will be seen later are abstract concepts that appear purely in the RTL.

In investigating the register allocation subsystem closely in figure 14, the multiplexing and LUT subsystems come into heavy play again. The presence of the buttons and clock as inputs requires the use of AND gates along with adders and subtractors to detect which button presses are present during the rising clock edges. The storing of the input values themselves are operated by the multiplexers connected to the LUT on the far right of the diagram.

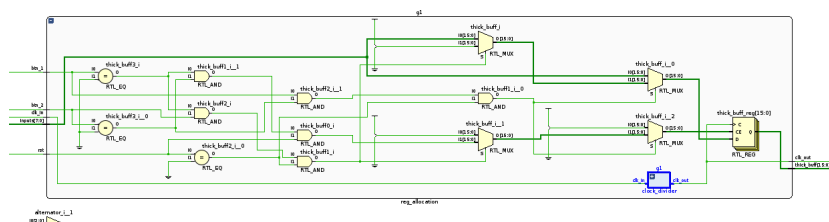


Figure 14: Register Transfer Level view of the register allocation subsystem

The other subsystems in which data is stored are the locking and ssd mechanisms. These registers can also be found within figure 15

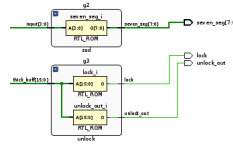


Figure 15: Register Transfer Level view of the SSD and locking subsystems

6 Synthesis Results

There were several key features shown in the synthesis of the model. The first of these was the memory usage of the entire design. As can be seen in figure 16, less than 1% of the entire system's memory was used, both in look-up tables and registers. The majority of memory usage came from the register-allocating system, which used 19 of the 26 LUTs and 31 of the 41 registers.

Tcl Console

Messages

Log

Reports

Design Runs

Methodology

Utilization

Clock Networks

Timing

Q

≡

⦿

⏮

⏭

Q

≡

⦿

%

Hierarchy

Hierarchy

Summary

▼

Slice Logic

▼

Slice LUTs (<1%)

LUT as Logic (<1%)

▼

Slice Registers (<1%)

Register as Flip Flop (1%)

Memory

DSP

▼

IO and GT Specific

▼

Bonded IOB (14%)

IOB Master Pads

IOB Slave Pads

▼

Clocking

▼

BUFCTRL (3%)

Specific Feature

Primitives

Black Boxes

Instantiated Netlists

Name	Slice LUTs (63400)	Slice Registers (126800)	Bonded IOB (210)	BUFCTRL (32)
full_combination	26	41	30	1
g1 (reg_allocation)	19	31	0	0
g2 (ssd)	4	0	0	0

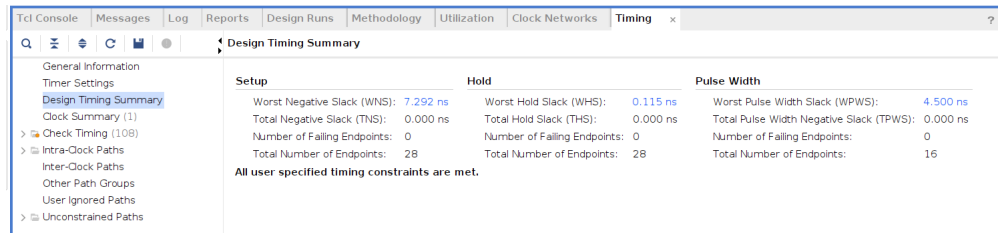
Figure 16: Memory utilisation following synthesis

In order to show the synchronicity of the system, it was also necessary to show the clock network. Figure 17 displays the network of connections to the single 100MHz internal clock. Note that there are 27 variables that are not directly connected to the clock. These are internal signals, and were controlled via processes that were reliant upon the clock, thus maintaining system synchronicity. However, meshing the system to be completely connected to the 100MHz clock could be a suggested improvement, but it's not necessary for functionality.

Tcl Console	Messages	Log	Reports	Design Runs	Methodology	Utilization	Clock Networks
<p>> sys_clk_pin (100.00 MHz) (drives 15 loads)</p> <p>> Unconstrained (27 loads)</p>							

Figure 17: Synthesis showing the network of connections aligned with the 100MHz clock

The other issue worth considering for synchronicity is the device's timing. Figure 18 shows the synthesis results for timing, and indicates that there are no failings throughout the entire system.



Design Timing Summary		
Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 7.292 ns	Worst Hold Slack (WHS): 0.115 ns	Worst Pulse Width Slack (WPWS): 4.500 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 28	Total Number of Endpoints: 28	Total Number of Endpoints: 16
All user specified timing constraints are met.		

Figure 18: Timing of the synthesis showing that there are no failings with timing during implementation

7 Conclusion

In conclusion the system works as intended. The entire design criteria was satisfied. The system locks, unlocks and resets when prompted. Lastly, it does so in a synchronous fashion.

As a future improvement on the design, it could be possible to implement it in such a way that the 100MHz clock is able to connect to every signal in the program, thus eliminating any error warnings associated with it. As discussed in the synthesis section, however, this will not impact the usability of the design.