Rensselaer Polytechnic Institute

Computer Hardware Design
ECSE 4770

Report

Lab Three
Protoboard Richards Controller and Logic Analyzer using Quartus

Group:
Gideon Bartlett
Joseph Catalano
Kristian Gutenmann
Nikoleta Koleva
TA: Joe Ebel
Date: 14 October 2019
Instructor: John F. McDonald

Table of Contents

Abstract

This lab examines the functionality of a Richards Controller and tests our abilities to implement it. We examine the usefulness of the Richards Controller for easily switching between several states. We also examine the capabilities of Quartus when it comes to simulating the uses of a Richards Controller. Throughout this lab, we gained a lot of experience in how to implement Quartus circuitry onto an FPGA.

Parts List

Chip	Name	Amount
7408	Quad 2-Input AND Gate	3
7421	Dual 4-Input AND Gate	1
7426	Quad 2-Input NAND Gate	1
7432	Quad 2-Input OR Gate	2
7442	BCD To Decimal Decoder	3
7474	Dual D Flip-Flop	3
74151	8-Input Multiplexer	3
74163	4-Bit Binary Counter, Synchronous Reset	1

**Miscellaneous resistors used when necessary*

Description of Internal Controller Operation

Introduction

The Richards Controller implemented in this lab is designed to operate an electronic stopwatch. It is a primary controller, meaning it first assesses primary conditions, followed by jumping to secondary ones. Once these conditions are evaluated, their respective primary or secondary functions are executed. There were three parts to be designed throughout the duration of the lab: the stopwatch controller, the controller power-up circuitry, as well as the synchronization circuitry for the Start/Stop and Counter/Split switches.

Richards Controller Design

The Richards Controller is set up to allow for the state jumps and behaviors laid out in the Richards Flowchart. The controller does this through several features. First, we had to establish which states simply stepped to the next numerical state and which jumped to a different state that was not the next one numerically. Once this was identified, we had to find which conditions triggered certain state jumps as well as certain state steps. Then we were able to establish our design for the Richards Controller. All Functions that indicated a step from one state to the next and their associated conditions were assigned to the primary multiplexer and decoder. Functions that indicated a step from one state to another were associated with specific pins on the decoder corresponding to their state of origin. The matching pins on the multiplexer were then connected to the associated inputs. Some inputs, such as pins 3 and 6 were wired to ground meaning that they were associated only with jumps and not with steps and so they had to go to the secondary multiplexer and decoder. This decoder and multiplexer were wired similarly to the previous ones, however, the functions on the outputs of this decoder were connected to the load pin of the state counter via an AND gate and the A B and C load values of the state counter. This allowed the jumps to activate when the functions became active and also told the counter what states to jump to. There were a few states such as state 3 and state 6 here that had their conditions wired directly to 5V because they had to jump and there was no other condition. Lastly, state 4 had 3 potential states that it could lead to depending on the inputs. One was a step to state 5 but the others were jumps. As a result of this we had to install an extra multiplexer and decoder pair for the third condition of state 4 to be addressed. This multiplexer and decoder were wired similarly to the secondary multiplexer and decoder. The way these multiplexers and decoders are wired allows for the primary multiplexer and decoder to receive priority, however, if none of the conditions are met then the decoder will say that it is in a state that has not been wired to and not output a function and the multiplexer will let the next multiplexer in the priority to read its inputs to its decoder. This continues until an input is reached that can trigger a state change.

Switch Synchronization

The Switch Debouncing Circuitry relies on the use of D - Flip Flops to stabilize the input and combines them with a logic AND gate in order to prevent a single long button press being counted as multiple button presses. First, the input must be stabilized in order to debounce the switch input. In order to do this the switch input was connected to the input of a single D - Flip Flop. This first D - Flip Flop was also wired to the clock. This first D - Flip Flop served the purpose of initial debouncing. By feeding the signal through the Flip Flop, any signal impurities from the switch bouncing could be minimized as the input was synced to the clock cycle. This still left the issue of one long button press being read as multiple button presses. This is where the second D - Flip Flop came into the debouncing circuit. This D - Flip Flop took its input from the output of the first D - Flip Flop. It is also synced to the same clock as the first D - Flip Flop. This means that the output of the second Flip Flop is the same as the output of the first Flip Flop with a delay of 1 clock cycle between them. The output Q of the first Flip Flop of combined with the output Q’ of the second by using an AND gate. This means that if the button is pressed then

the output of the AND gate will be positive for exactly one clock cycle. After this one clock, the output of the AND gate will be 0 as it will functionally be combining the output of the Flip Flop with its own inverse. This guarantees that any press lasting longer than one clock cycle will be limited to one clock cycle.

Power-On Reset

The Power-On Reset Circuitry relies on the use of just one D-Flip Flop to have the stopwatch turn on, as well as having the ability to return the program to a previously known state when pressed. Similarly, Start/Stop and Counter/Split use one D-Flip Flop each for their implementation. It is important to note that the power-on reset button is active low, connecting to ground when pressed and otherwise keeping a high state. Its uses include resetting the state counter to zero, clearing the time counter and setting enable to low, as well as setting the multiplexer to high. The correct implementation and execution of the power-on reset button and circuitry is essential to having the correct outputs when implementing the electronic stopwatch.

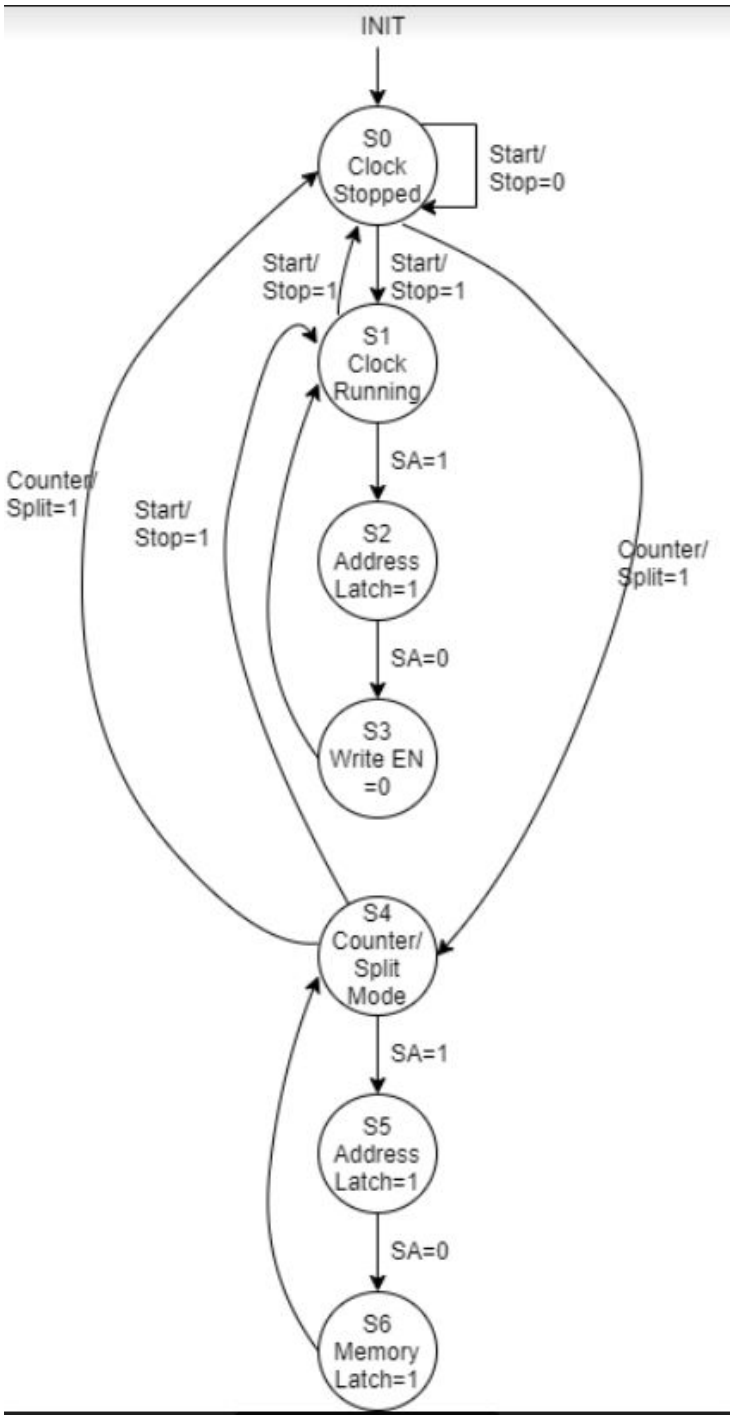
Stopwatch Controller/Timing

The circuit generally had very little to consider in terms of time delays. The delays from chips were roughly on the scale of 1ns while the clock speed was much slower than that. As a result, there were only two instances where an extra state needed to be added to account for timing. These states were the states that were associated with accessing memory. States 2 and 5 were added as intermediary states in order to allow for this timing delay that resulted from the device being linked to the clock pulses. Pin 2 is tied to the Address Latch value. This value is synced with the clock pulses. Pin 5 is associated with Memory Latch, which is also synced with the clock values. Both of these values are needed when the memory is accessed, however, because they are both tied to the clock pulses, the circuit could experience some difficulties in transferring values if both latches were expected to be triggered in the same state. Both latches would perform their tasks but the data from the Address Latch needs time to get to the Memory Latch. Without the state delay, the Memory Latch would output useless values that were unrelated to the current state of the circuit. Adding the one clock cycle delay by adding an extra state means that the circuit will only allow the proper values from the Address Latch to pass through the Memory Latch. This was the biggest timing consideration in the circuit outside of the button debouncing stated above.

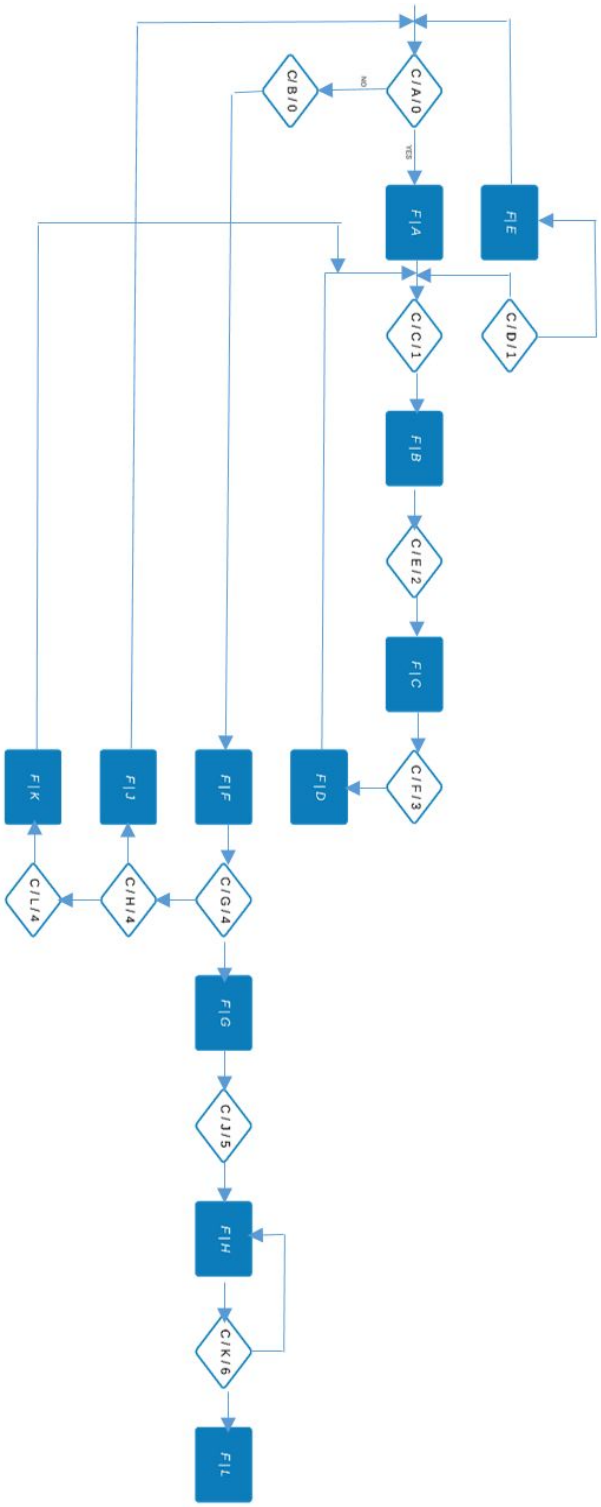
Concise Description of Debugging Process

Initially there were issues with the board not properly responding to button presses. We had made the mistake of assuming that since the board was debounced we did not have to worry about synchronization. We soon realized that the issue we were having with button presses was a result of the lack of clock synchronization to our push buttons. We also had issues with compiling. We found that when copying parts instead of inserting a new part the new part would have the same name and would cause an error when compiling. To fix this error all parts that had been copied instead of placed were changed out for new components and was then able to compile the program.

State Diagram



Richards Flowchart



State Table

	Inputs			Outputs							
State	**C**	**B**	**A**	**ADDRESS LATCH Pin 2**	**MEMORY EN Pin 3**	**READ/ WRITE EN Pin 4**	**MEMORY LATCH Pin 5**	**LOAD CNTRS Pin 6**	**ENABLE CNTRS Pin 7**	**CLEAR CNTRS Pin 8**	**MUXC Pin 10**
0	0	0	0	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	1	1	1
2	0	1	0	1	0	1	0	1	1	1	1
3	0	1	1	0	0	0	1	1	1	1	1
4	1	0	0	0	1	1	0	1	0	1	0
5	1	0	1	1	0	1	0	1	0	1	0
6	1	1	0	0	0	1	1	0	0	1	0
7	1	1	1	x	x	x	x	x	x	x	x

Karnaugh Maps

ADDRESS LATCH (Pin 2)					
A\CB	**00**	**01**	**11**	**10**	**CA + C'BA'**
0	0	1	0	0	
1	0	0	x	1	

MEMORY EN (Pin 3)					
A\CB	**00**	**01**	**11**	**10**	**C'B' + B'A'**
0	1	0	0	1	
1	1	0	x	0	

READ/WRITE EN (Pin 4)

A\CB	**00**	**01**	**11**	**10**	
0	1	1	1	1	**B' + A'**
1	1	0	x	1	

MEMORY LATCH (Pin 5)					
A\CB	**00**	**01**	**11**	**10**	
0	0	0	1	0	**CB + BA**
1	0	1	x	0	

LOAD CNTRS (Pin 6)					
A\CB	**00**	**01**	**11**	**10**	
0	1	1	0	1	**C' + B'**
1	1	1	x	1	

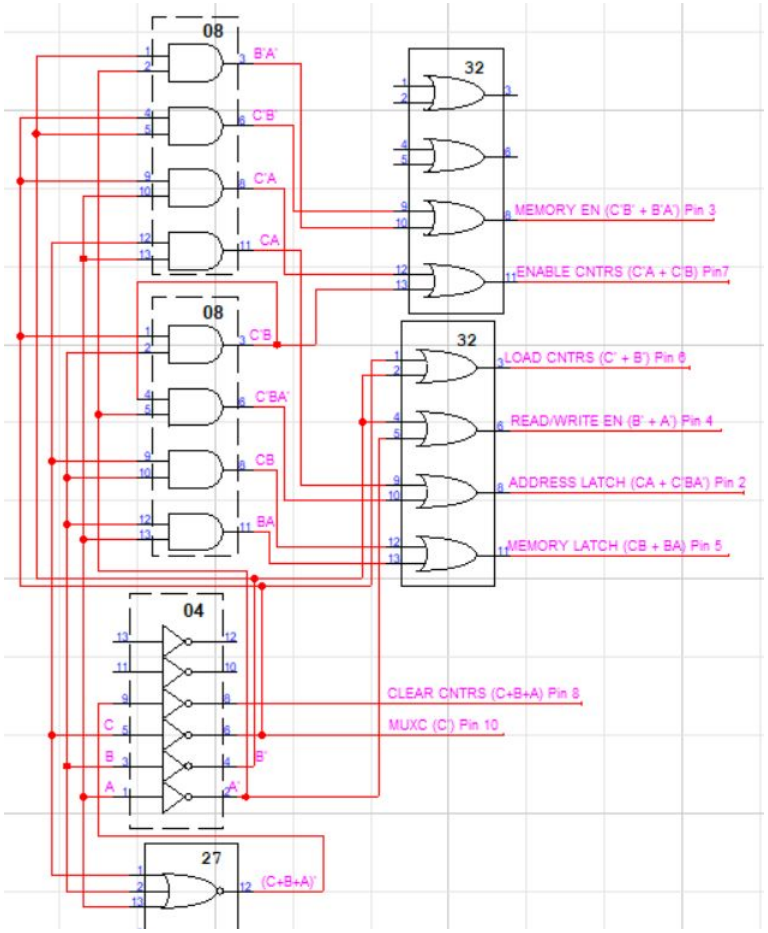
ENABLE CNTRS (Pin 7)					
A\CB	**00**	**01**	**11**	**10**	
0	0	1	0	0	**C'A + C'B**
1	1	1	x	0	

CLEAR CNTRS (Pin 8)					
A\CB	**00**	**01**	**11**	**10**	
0	0	1	1	1	**C + B + A**
1	1	1	x	1	

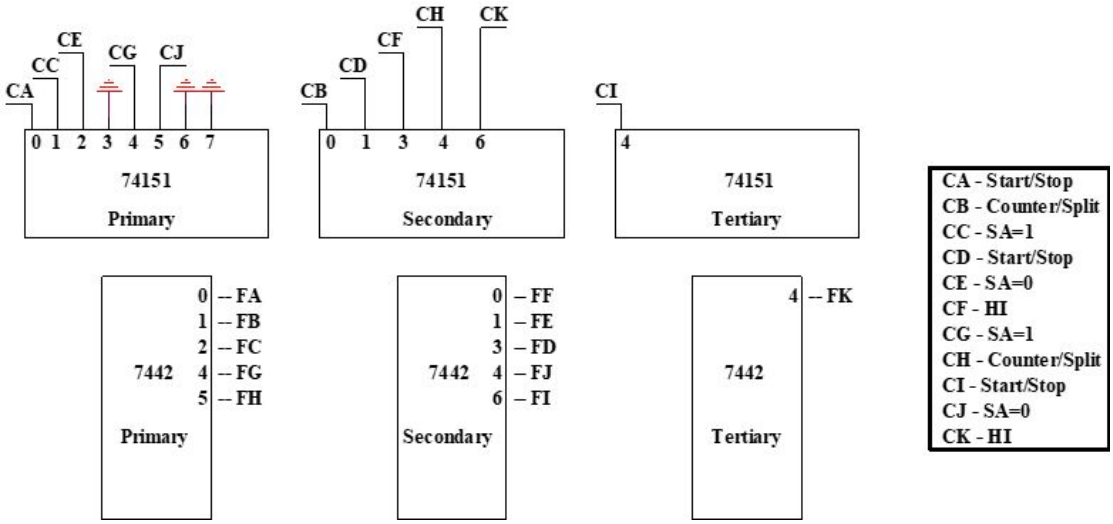
MUXC (Pin 10)					
A\CB	**00**	**01**	**11**	**10**	

0	1	1	0	0	**C’**
1	1	1	x	0	

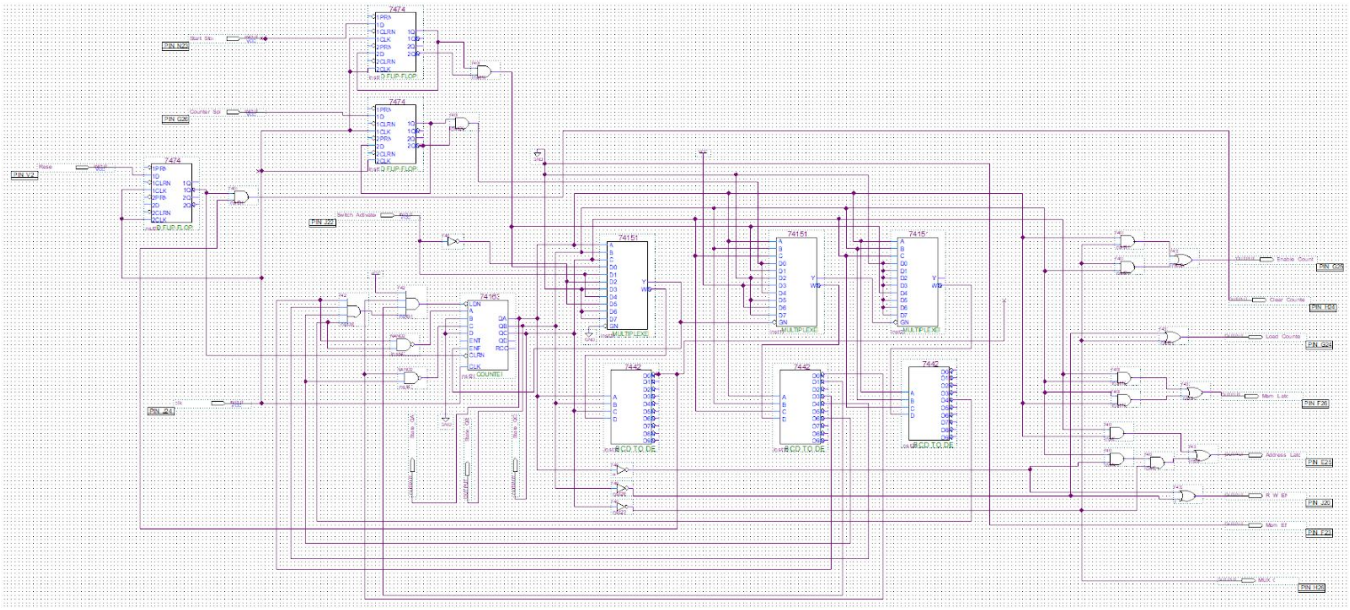
Logic Diagram for Output Circuits



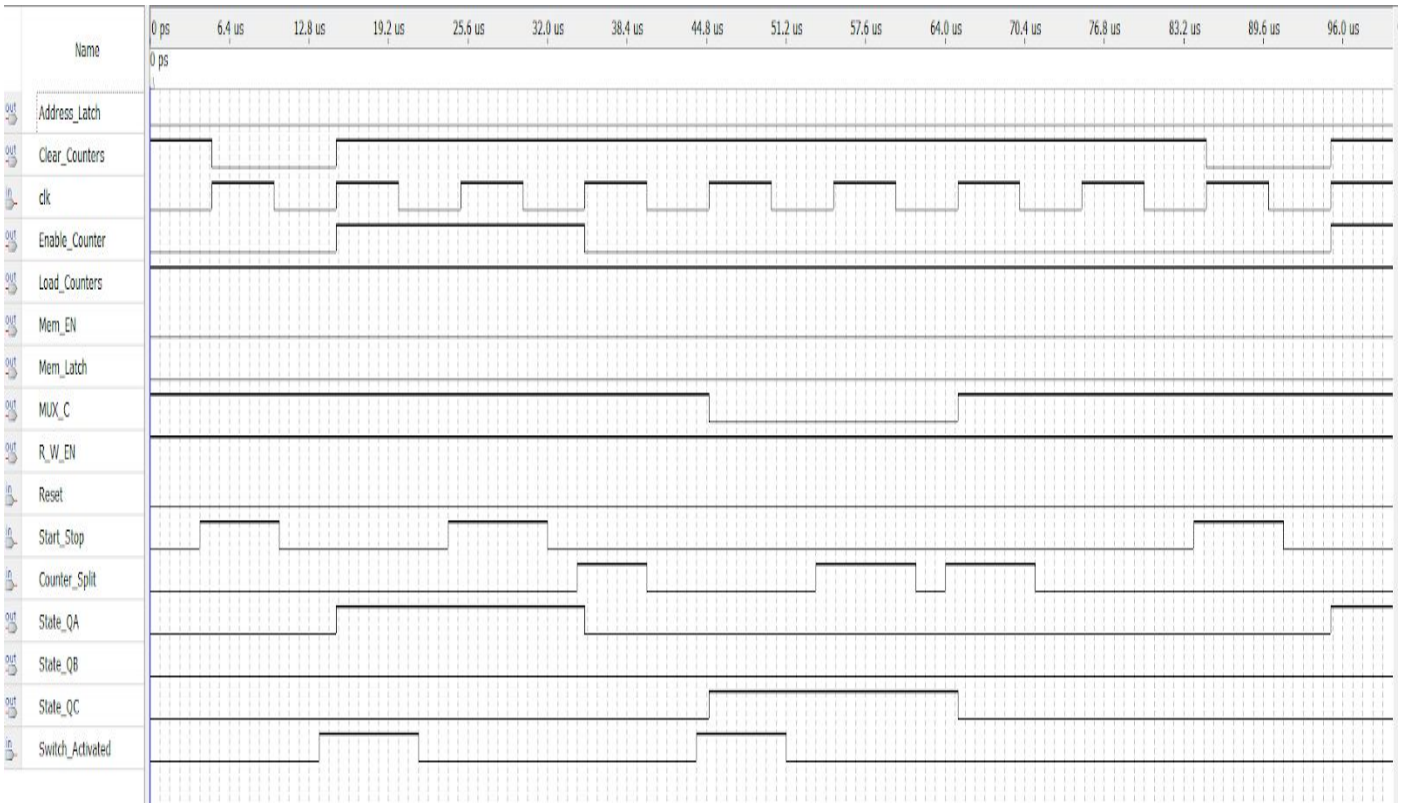
Logic Diagram of 74151 and 7442 Primary, Secondary, and Tertiary Chips



Full Logic Diagram from Quartus



Final Timing Diagrams



Conclusion

The design and building of a digital Richard's controller was both interesting and effective. We learned a great deal from both the physical and Quartus implementations. Our Quartus implementation went smoothly because of everything that we had learned during the physical implementation in Lab 2. As a result we were able to move through this lab very quickly. The end result was a better understanding of Richard's controllers in more ideal situations.