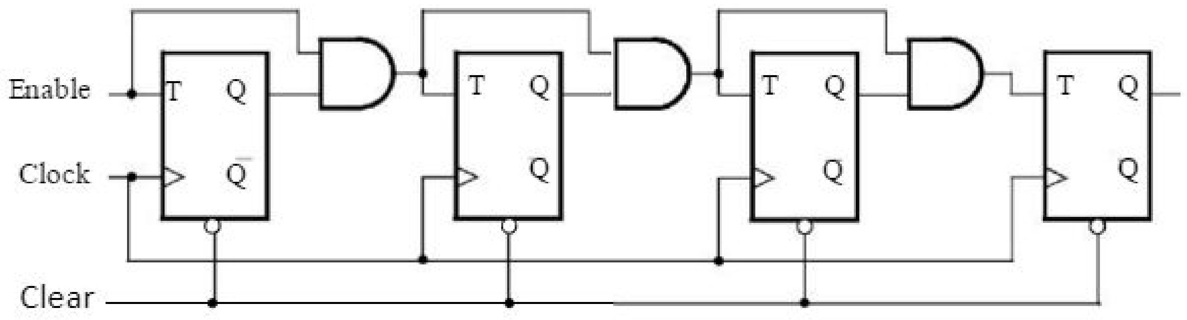
Laboratory Exercise A
Counters

TCES 330 Digital Systems Design

4/20/2016

Authors: Vladislav Psarev, Brandon Watt and Andrew Gates



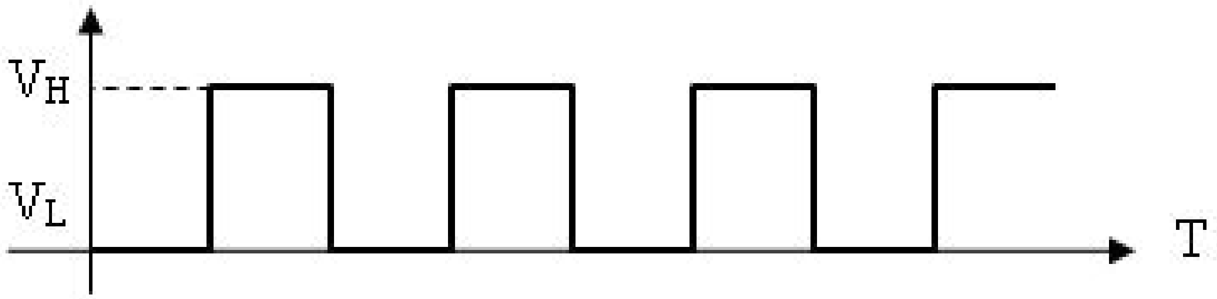


Table of Contents

Laboratory Assignment A

The purpose of this lab is to understand how to implement counters using different methods, such as a T Flip Flop based counter, a counter using verilog code, and a counter using a pre-defined LPM counter. After we experiment with different types of counters we are then to use the 50 MHz clock to display a number going from 0 to 9 on the 7-segment display. After that we are then to take this clock one step further and have it work with our HELLO display from Lab 1. We will use KEY0 as the clock for parts 1, 2, and 3, with SW0 being the Enable signal and SW1 being the clear signal. The output of the counting will display on HEX3, HEX2, HEX1, and HEX0.

Requirements:

The objectives of each part of the lab assignment are described in this section of the report.

Part 1

In Part 1 we are given all the parts we need to make a 16-bit counter. We just had to put the parts together and see how it ran. The button KEY0 will act as a clock, SW0 is the enable signal, SW1 is the reset and the 16-bit output, Q, will go to the four hex displays HEX3-0. For testing in Model Sim we are given an outline of the test and told to modify it so that it could test reset and enable functionalities.

1. Write a Verilog file that defines a 16-bit counter by using the structure depicted in Figure 1. Your code should include a T flip-flop module that is instantiated 16 times to create the counter. Compile the circuit. How many logic elements (LEs) are used to implement your circuit? What is the maximum frequency, Fmax, at which your circuit can be operated?

2. Simulate your circuit to verify its correctness using ModelSim. In your report, figure out some convincing way to demonstrate correct circuit operation without including all 65,536 lines of output!

3. Augment your project with a top-level Verilog file that uses the pushbutton KEY0 as the Clock input, switch SW1 as an active high Reset and switch SW0 as an active high Enable, and 7-segment displays HEX3-0 to display the hexadecimal count as your circuit operates. Make the necessary pin assignments needed to implement the circuit on the DE2 board, and compile the circuit.

4. Download your circuit into the FPGA chip and test its functionality by operating the implemented switches.

5. Use the Quartus II RTL Viewer to see how Quartus II software synthesized your circuit. What are the differences in comparison with Figure 1?

Part 2

In this part we have to replace the counter we were given with one that uses the function Q <= Q + 1. The top level module and tests will remain mostly the same, they will simply call the new counter. Once again the button KEY0 will act as a clock, SW0 is the enable signal, SW1 is the reset and the 16-bit output, Q, will go to the four hex displays HEX3-0.

1. Write a Verilog file that defines a 16-bit counter by using this approach. Compile the circuit. How many logic elements (LEs) are used to implement your circuit? What is the maximum frequency, Fmax, at which your circuit can be operated? Comment on the differences between this and Part I.

2. Simulate your circuit to verify its correctness using ModelSim.

3. Augment your project with a top-level Verilog file that uses the pushbutton KEY0 as the Clock input, switch SW1 as an active high Reset and switch SW0 as an active high Enable, and 7-segment displays HEX3-0 to display the hexadecimal count as your circuit operates. Make the necessary pin assignments needed to implement the circuit on the DE2 board, and compile the circuit.

4. Download your circuit into the FPGA chip and test its functionality by operating the implemented switches.

5. Use the RTL Viewer to see the structure of this implementation and comment on the differences with the design from Part I.

Part 3

With two versions of the counter made using T flip-flops and a simplified verilog statement, the basics of building a counter are established. Now we'll use the Verilog libraries to build the 16-bit counter and compare it to the previous two counters. As with parts 1 and 2, the button KEY0 will act as a clock, SW0 is the enable signal, SW1 is the reset and the 16-bit output, Q, will go to the four hex displays HEX3-0.

1. Write a Verilog file that implements a 16-bit counter by using this approach. Compile the circuit. How many logic elements (LEs) are used to implement your circuit? What is the maximum frequency, Fmax, at which your circuit can be operated? Comment on the differences between this and Part I and Part II.

2. (no ModelSim for this part).

3. Augment your Verilog file to use the pushbutton KEY0 as the Clock input, switches SW1 and SW0 as Enable and Reset inputs, and 7-segment displays HEX3-0 to display the hexadecimal count as your circuit operates. Make the necessary pin assignments needed to implement the circuit on the DE2 board, and 3 compile the circuit.

4. Download your circuit into the FPGA chip and test its functionality by operating the implemented switches.

5. Use the RTL Viewer to see the structure of this implementation and comment on the differences with the design from Parts I and II.

Part 4

In part 4 we were to build a counter that would count from 0 to 9 on HEX0 display, and then reset back to 0 once the counter reaches 9. We needed to use the 50 MHz clock which is CLOCK_50 in Quartus. Since the 50 MHz clock is too fast to see any changes, we needed to decrease this to a 1 Hz clock by making a counter that would only increment the HEX0 display when the count reaches 50,000,000. This way the 50 MHz clock would work as a 1 Hz clock for our counting module.

Part 5

In part 5, we were to design and implement a circuit that displays the word HELLO, in ticker tape fashion, on the eight 7-segment displays HEX7-0. The letters are to move right to left in intervals of about a second. Refer to figure 16 in Appendix A for the patterns that should be displayed in successive clock intervals.

Design:

This section of the report describes our analysis of the requirements for this laboratory exercise and the resulting project design.

Part 1

We have no contributions on this design as all files are given to us. This design is that of 16 T flip-flops connected together with AND gates to act as a 16-bit counter. This counter can take in an enable and reset input. This design of counter is the most complex in terms of verilog code because it requires the generation of 16 T flip-flops and other gates.

Part 2

The design of this counter is heavily dependent on Quartus' ability to interpret and simplify circuits. We give quartus the simple incrementing equation, Q = Q +1. Quartus then creates an optimized circuit that satisfies that equation. Refer to Figure 6 in appendix A for a diagram of the the circuit created by Quartus. The resulting circuit was simpler than the strictly defined Part 1.

Part 3

This part gave all the work over to the Quartus library. We simply went into the Quartus library, chose the kind of function we wished to use, sorted out a few design details and let the library build it for us. Since this counter was designed entirely by Quartus it is highly optimized, the resulting circuit being slightly simpler than the counter in Part 2. An issue with this is the internal workings are not clearly displayed, refer to Figure 9 in appendix A for an image of the circuit.

Part 4

For part 4 we decided to use our counter from part 2, that way we were able to just wrap our incrementing of Q within the counter for our 50 Mhz clock counting from 0 to 50,000,000. Besides this we didn't need to change any other code, besides sending the CLOCK_50 into our CountNG instead of the pseudo clock for KEY0 like we used before.

Part 5

For this part of the Lab, we used our part 6 code from HW1, since it is performing the same display logic. The difference is that we are no longer using SW17-15 as the selection signals, but instead using the CountNG clock from part 2 of this lab modified to be a 3-bit counter. This clocks' outputs are used as the selection signals to the multiplexers.

Test Procedures:

The following test procedures will be used to verify that each part of this laboratory exercise satisfies the requirements given in the Requirements section, above.

Part 1

Testing on the DE2 board:

1. Compile the Part 1 project in Quartus and load it onto the DE2 board.

2. Make sure that the four 7 segment displays (HEX3-0) are displaying zeros.
3. Press KEY0 a few times. The display should remain the same. (Enable test)
4. Now switch SW0 to the on position.
5. Press KEY0 approximately 20 times. As you do this make sure that the displays are showing the appropriate values. Be sure to take into account occasional number skipping due to button bounce. (Refer to the second paragraph in part 1 observations for more details on button bounce.)
6. Once you are certain that the displays can show all 16 digits and will properly increment the next display, put SW1 to the on position.
7. Press KEY0 once. This should reset all displays to 0. (Clear test)
8. Keeping SW1 at the on position, press KEY0 a few more times. The display should continue to display all zeros.
9. Move SW0 to the off position and press KEY0 a few times. The displays should not change.
10. Now move SW1 to the off position and SW0 to the on position.
11. Press KEY0 a few times. The displays should count upwards.
12. Move SW1 to the on position and SW0 to the off position.
13. Press KEY0, the displays should reset to all zeros.

Testing in Module Sim:

1. Load the Part 1 test file into Module Sim and compile it.
2. Run the simulation. The simulation should display the values of the count and run until the count is at 0xffff.
3. Check a few of the values displayed to ensure that it is counting properly.
4. Reset the simulation.
5. Uncomment the clear test section in the Part 1 test file, recompile, reset and run the simulation.
6. The simulation will display values as before but will reach the value 0xaaab and the reset to 0x0000, the simulation will stop at this point. (Clear test.)
7. Stop and reset the simulation.
8. Comment out the clear test and uncomment the enable test section.
9. Recompile, restart and run the simulation.
10. The simulation should stop counting at 0xaaab. (Enable test)
11. Stop the simulation. This completes the Model Sim tests.

Part 2

Testing on the DE2 board:

1. Compile the Part 2 project in Quartus and load it onto the DE2 board.
2. Make sure that the four 7 segment displays (HEX3-0) are displaying zeros.
3. Press KEY0 a few times. The display should remain the same. (Enable test)
4. Now switch SW0 to the on position.
5. Press KEY0 approximately 20 times. As you do this make sure that the displays are showing the appropriate values. Be sure to take into account occasional number skipping due to button bounce. (Refer to the second paragraph in part 1 observations for more details on button bounce.)
6. Once you are certain that the displays can show all 16 digits and will properly increment the next display, put SW1 to the on position.
7. Press KEY0 once. This should reset all displays to 0. (Clear test)
8. Keeping SW1 at the on position, press KEY0 a few more times. The display should continue to display all zeros.
9. Move SW0 to the off position and press KEY0 a few times. The displays should not change.
10. Now move SW1 to the off position and SW0 to the on position.
11. Press KEY0 a few times. The displays should count upwards.
12. Move SW1 to the on position and SW0 to the off position.
13. Press KEY0, the displays should reset to all zeros.

Testing in Module Sim:

1. Load the Part 2 test file into Module Sim and compile it.
2. Run the simulation. The simulation should display the values of the count and run until the count is at 0xffff.
3. Check a few of the values displayed to ensure that it is counting properly.
4. Reset the simulation.
5. Uncomment the clear test section in the Part 1 test file, recompile, reset and run the simulation.
6. The simulation will display values as before but will reach the value 0xaaab and the reset to 0x0000, the simulation will stop at this point. (Clear test.)
7. Stop and reset the simulation.
8. Comment out the clear test and uncomment the enable test section.
9. Recompile, restart and run the simulation.
10. The simulation should stop counting at 0xaaab. (Enable test)
11. Stop the simulation. This completes the Model Sim tests.

Part 3

Testing on the DE2 board:

1. Compile the Part 3 project in Quartus and load it onto the DE2 board.
2. Make sure that the four 7 segment displays (HEX3-0) are displaying zeros.
3. Press KEY0 a few times. The display should remain the same. (Enable test)
4. Now switch SW0 to the on position.
5. Press KEY0 approximately 20 times. As you do this make sure that the displays are showing the appropriate values. Be sure to take into account occasional number skipping due to button bounce. (Refer to the second paragraph in part 1 observations for more details on button bounce.)
6. Once you are certain that the displays can show all 16 digits and will properly increment the next display, put SW1 to the on position.
7. Press KEY0 once. This should reset all displays to 0. (Clear test)
8. Keeping SW1 at the on position, press KEY0 a few more times. The display should continue to display all zeros.
9. Move SW0 to the off position and press KEY0 a few times. The displays should not change.
10. Now move SW1 to the off position and SW0 to the on position.
11. Press KEY0 a few times. The displays should count upwards.
12. Move SW1 to the on position and SW0 to the off position.
13. Press KEY0, the displays should reset to all zeros.

Testing in Model Sim:

1. No test required.

Part 4

Testing on the DE2 board:

1. Compile the Part 4 project in Quartus and load it onto the DE2 board.
2. Make sure that the one 7 segment displays (HEX0) starts displaying zero.
3. Monitor the HEX0 display, the number should go from 0 to 9 and then reset back to 0.

Testing in Model Sim:

1. No test required.

Part 5

Testing on the DE2 board:

1. Compile the Part 5 project in Quartus and load it onto the DE2 board.
2. Initially set all switches to off (all segments HEX7 - HEX0 should display H)
3. Set switches 14, 13, 12, 6, 4, 1, 0 to on position.
4. The display should now read HELLO.
5. Monitor the HEX displays, the word HELLO should move from right to left at 1 second intervals. Refer to figure 16 in Appendix A

Testing in Model Sim:

1. No test required.

Test Results:

The following test results were what we achieved in each part of the lab.

Part 1

ModelSim

1. The simulation using ModelSim produced the output shown in Figures 17 and 18 in Appendix A. This output is expected and the values started from 0000 and ended at ffff.

DE2-115

1. Compilation by Quartus was successful, with the Flow Summary shown in Figure 1 in Appendix A.
2. The RTL View produced the results shown in Figures 2 and 3 in Appendix A . This is expected since it shows the output from CountNG as the input into Hex7seg.
3. The project was uploaded to the DE2-115 board without errors.
4. The numbers in displays HEX3-0 rotated from 0000 up to ffff and then reset back to 0000 by using the KEY0 as the clock.
5. SW[0] was the enable. When this was on the counter incremented, and when this was off the counter did not increment.
6. SW[1] was the clear. When this was on the counter would reset back to 0000 while clear was on.

Part 2

ModelSim

1. The simulation using ModelSim produced the output shown in Figure 19 and 20, in Appendix A. This output is expected and the values started from 0000 and ended at ffff.

DE2-115

1. Compilation by Quartus was successful, with the Flow Summary shown in Figure 4 in Appendix A.
2. The RTL View produced the results shown in Figures 5 and 6 in Appendix A . This is expected since it shows the output from CountNG as the input into Hex7seg.
3. The project was uploaded to the DE2-115 board without errors.
4. The numbers in displays HEX3-0 rotated from 0000 up to ffff and then reset back to 0000 by using the KEY0 as the clock.
5. SW[0] was the enable. When this was on the counter incremented, and when this was off the counter did not increment.
6. SW[1] was the clear. When this was on the counter would reset back to 0000 while clear was on.

Part 3

ModelSim

1. No ModelSim for this part.

DE2-115

1. Compilation by Quartus was successful, with the Flow Summary shown in Figure 7 in Appendix A.
2. The RTL View produced the results shown in Figures 8 and 9 in Appendix A . This is expected since it shows the output from CountNG as the input into Hex7seg.
3. The project was uploaded to the DE2-115 board without errors.
4. The numbers in displays HEX3-0 rotated from 0000 up to ffff and then reset back to 0000 by using the KEY0 as the clock.
5. SW[0] was the enable. When this was on the counter incremented, and when this was off the counter did not increment.
6. SW[1] was the clear. When this was on the counter would reset back to 0000 while clear was on.

Part 4

ModelSim

1. No ModelSim for this part.

DE2-115

1. Compilation by Quartus was successful, with the Flow Summary shown in Figure 10 in Appendix A.
2. The RTL View produced the result shown in Figures 11 and 12 in Appendix A . This is expected since it shows the output from CountNG as the input into Hex7seg.
3. The project was uploaded to the DE2-115 board without errors.
4. The number in the display HEX0 rotated from 0 to 9 and then reset back to 0 in intervals of 1 second like expected.

Part 5

ModelSim

1. No ModelSim for this part.

DE2-115

1. Compilation by Quartus was successful, with the Flow Summary shown in Figure 13 in Appendix A.
2. The RTL View produced the result shown in Figures 14 and 15 in Appendix A . This is expected since it shows the Part5 module connecting to the various counters and multiplexers.
3. The project was uploaded to the DE2-115 board without errors.
4. The word HELLO scrolled to the left across our HEX displays and after it reached it's initial position it restarted like expected.
5. The correct sequence was produced as shown in figure 16 in Appendix A.

Observations:

The following observations are points of interest to be noted in each part of the lab.

Part 1

The design of the counter in part one resulted in 50 logic units and Fmax of 184.7 MHz (Refer to Figure 1). Considering that we will be using a 50 MHz clock for this lab, this circuit is easily fast enough to handle that kind of input. Unsurprisingly, there is no real noticeable delay between when the button is released and the number on the displays increments. The enable and clear switches work as they are suppose to, though there was a short moment where we forgot that the button has to be pressed after the clear switch is set for the counter to clear.

A bizarre behavior we encountered while testing the counter on the DE2 board was that occasionally when pressing the button a value would be skipped over. For example the counter would be at 6, we would press the button and the new value displayed would be 8, when it should be 7. This confused us for a little while but once we asked if it was an error in the code, it was explained to us that it is a flaw in the button where the mechanism in the button sometimes "bounces" enough to create an additional voltage jump that is read as another pulse of the clock. This additional jump causes the counter to increment twice for a single press of the button. We will refer to this as "button bounce" in the rest of the document.

Part 2

With 45 logic elements, versus part one's 50, part two is simpler than part one (Refer to Figure 4). Since the statement is fairly simple itself, there is probably a lot of backend simplification that Verilog performs. Far more surprising was part two's Fmax of 386.1 MHz, which is limited at 250 MHz. If it was not limited it would be more than twice as fast as part one.
It's behavior on the DE2 board parallelled the behavior of part one with no real issues outside of the same button bounce that was present in part one.

One of the difficulties we had while attempting to test this part was that Model Sim didn't seem to work. Whenever we would run the test the counter output would be xxxx and never change. The module worked fine on the DE2 board so we were unsure what was wrong. After an hour or so of modifying the test file we looked into the TFFx file and compared it to our counter and found that we weren't initializing Q to zero. Once that was fixed Model Sim ran without issues.

Part 3

The reports for the compilation of part 3 reported that it had a total of 46 logic units and a restricted Fmax of 250 MHz (Refer to Figure 7). According to Quartus the Fmax is restricted due to I/O toggle rate and unrestricted would be 386.7 MHz. Whether restricted or not this much faster than part 1's Fmax of 184.7 MHz, and is probably due to it's 4 fewer logic elements and additional optimization provided by it being a library construct.

Outside of its superior performance the counter behaves just like parts 1 & 2 on the DE2 board. It even suffers from the button bounce issue that would cause the counter to "skip" values.

Part 4

For part 4 the compilation report showed us using 58 logic elements, refer to figure 10 in Appendix A. This was fairly surprising that it used only a few more than our previous adders that used 4 HEX displays, and counted up to ffff, whereas this counter only counts to 9 and then resets. The gain in logic elements is most likely due to using the actual clock and not a pseudo clock with KEY0. But still we expected the increase in logic elements to be a lot larger.

We noticed at first that when trying to use the binary value for 50,000,000 as the counter for CLOCK_50 that the HEX display would increase every .1 second or so and not 1 second. When we changed the value to 50_000_000 instead of the binary value it fixed it. We were unsure if we had the number of bits wrong or the binary value for 50_000_000 wrong but this issue was fixed by just putting in the actual value of 50,000,000.

Part 5

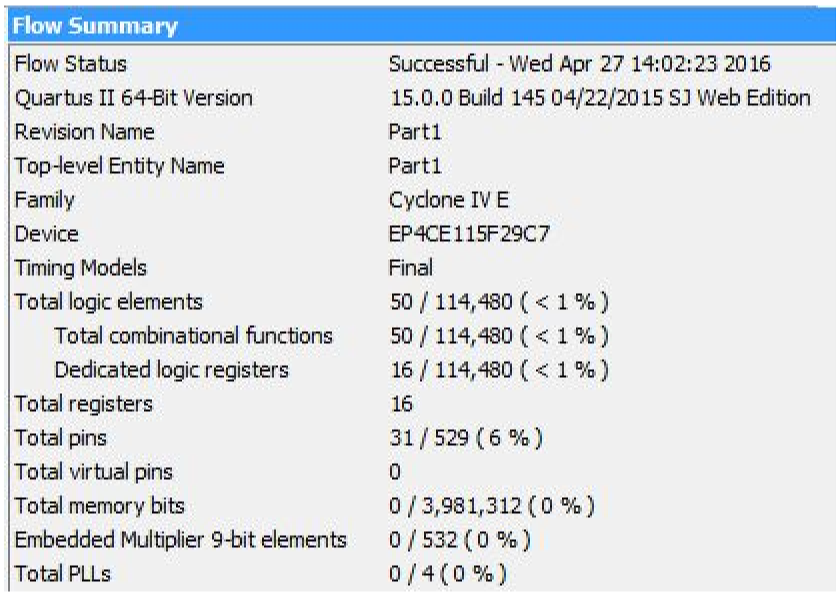
For part 5 the compilation report showed us using 145 logic elements, refer to figure 13 in Appendix A. This was fairly surprising since it was about 3 times the amount of logic elements of our other four parts. The RTL view was fairly expansive and was hard to evaluate due to how large and how many logic elements were included, refer to figure 14 in Appendix A.

We had an issue initially with all of the letters not being H when the switches were off. We found out that this was due to us using a constant value to control the blanks and not the switches. Once we fixed this and recompiled our module to the board then all of the letters were H when the switches were off and all of them were blank when the switches were on.

Conclusion:

In this lab we went over three different ways to build a 16-bit counter, one using T flip-flops, the second using a simple incrementing equation and the last taking from the Quartus library. Each had different numbers of logic units and Fmaxs displaying various advantages and disadvantages of their designs. We also went over how to utilize and manipulate clocks so that we can have a variety of clocks with varying speeds. Once we verified that these counters worked we took them and implemented them into modules that utilized the 50 MHz clock on the DE2 board as the clock input to the counters. We used this clock to count from 0-9 and displayed this on one of the HEX displays. After that we used the clock to rotate the word HELLO across the various HEX displays. This lab was a great way to test various design methods when designing a counter, and to compare and contrast these counters against each other. It was also a great way to actually implement a real clock and to use a real clock to drive the HEX displays.

Appendix A:



Flow Summary	
Flow Status	Successful - Wed Apr 27 14:02:23 2016
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Web Edition
Revision Name	Part1
Top-level Entity Name	Part1
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	50 / 114,480 (< 1 %)
Total combinational functions	50 / 114,480 (< 1 %)
Dedicated logic registers	16 / 114,480 (< 1 %)
Total registers	16
Total pins	31 / 529 (6 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 1. Summary for Part 1

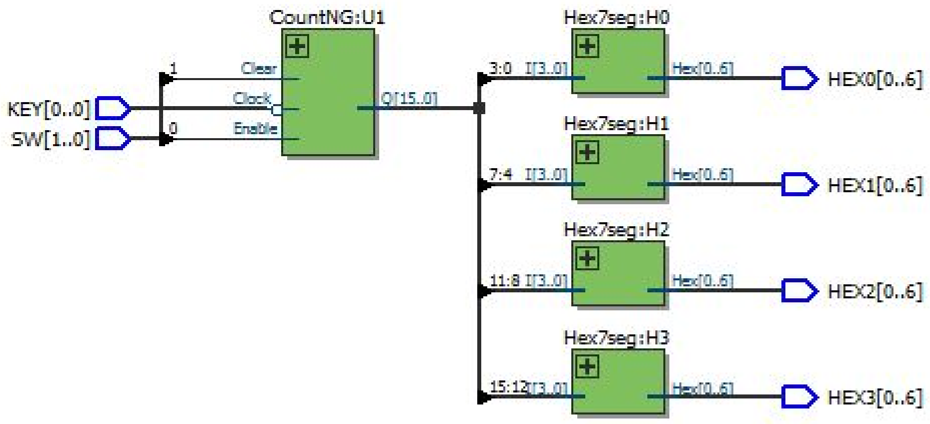


Figure 2. RTL for Part 1

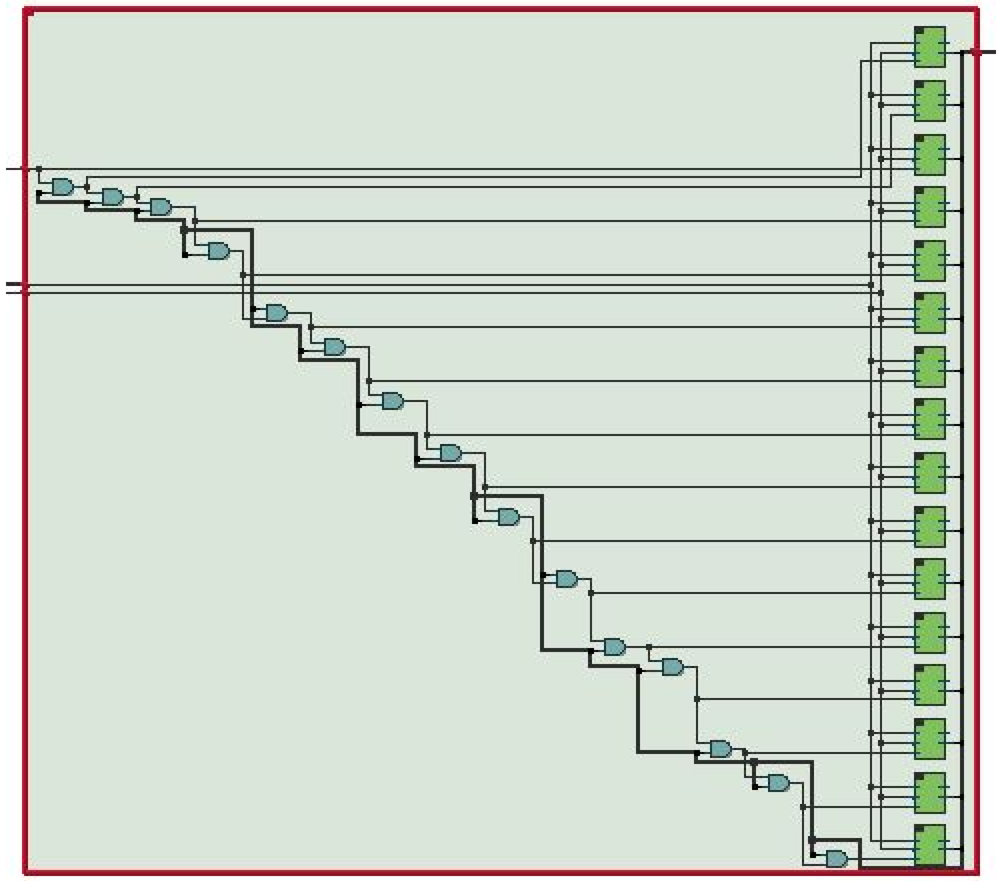
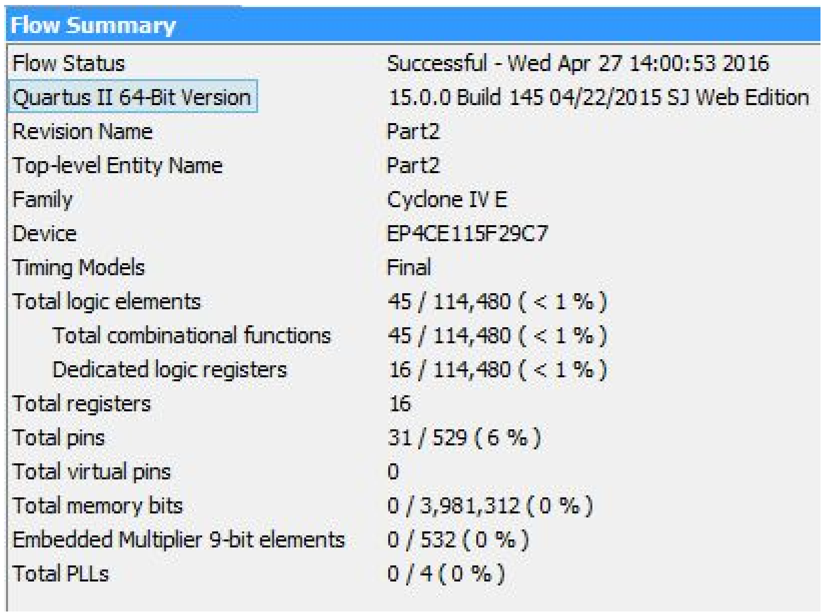


Figure 3. Expanded RTL view for CountNG for Part 1



Flow Summary	
Flow Status	Successful - Wed Apr 27 14:00:53 2016
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Web Edition
Revision Name	Part2
Top-level Entity Name	Part2
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	45 / 114,480 (< 1 %)
Total combinational functions	45 / 114,480 (< 1 %)
Dedicated logic registers	16 / 114,480 (< 1 %)
Total registers	16
Total pins	31 / 529 (6 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 4. Summary for Part 2

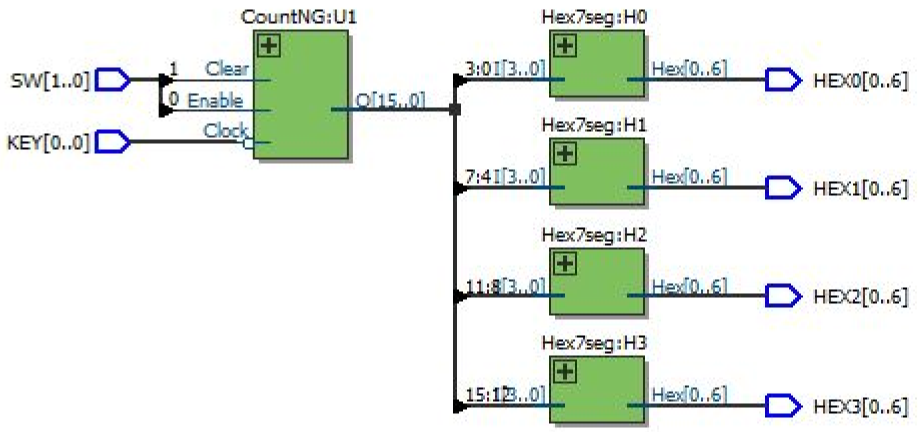


Figure 5. RTL for Part 2

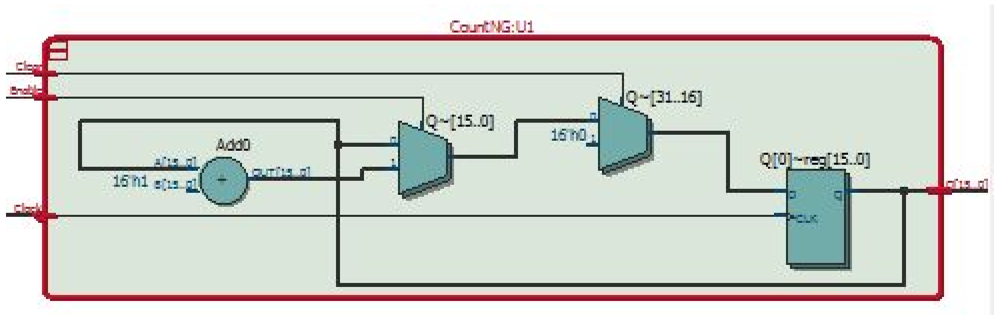
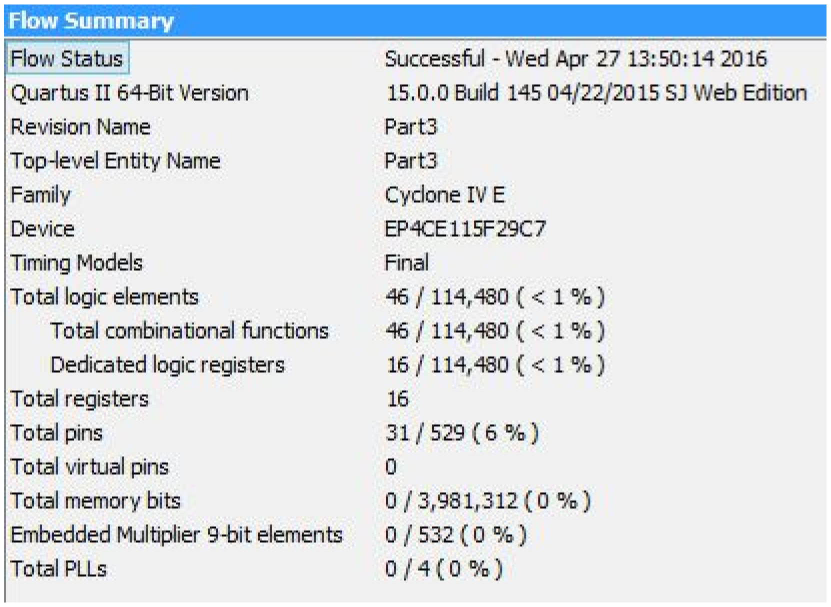


Figure 6. Expanded RTL view for CountNG for Part 2



Flow Summary	
Flow Status	Successful - Wed Apr 27 13:50:14 2016
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Web Edition
Revision Name	Part3
Top-level Entity Name	Part3
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	46 / 114,480 (< 1 %)
Total combinational functions	46 / 114,480 (< 1 %)
Dedicated logic registers	16 / 114,480 (< 1 %)
Total registers	16
Total pins	31 / 529 (6 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 7. Summary for Part 3

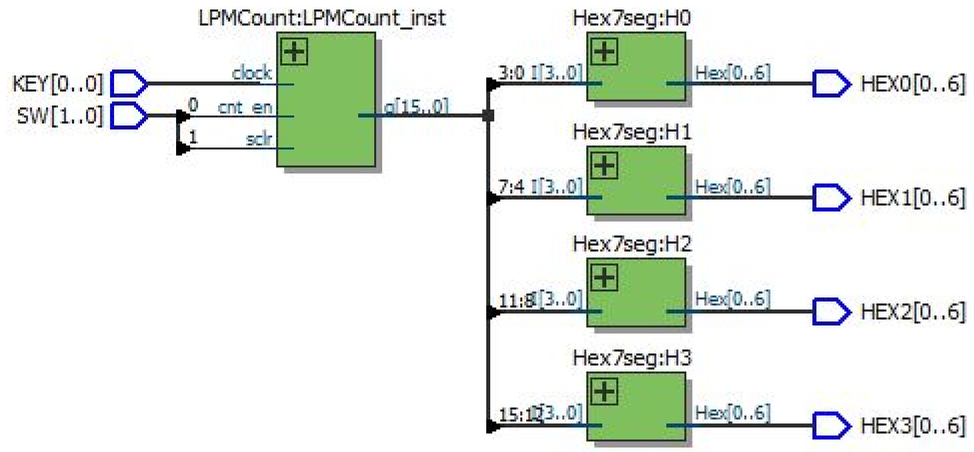


Figure 8. RTL for Part 3

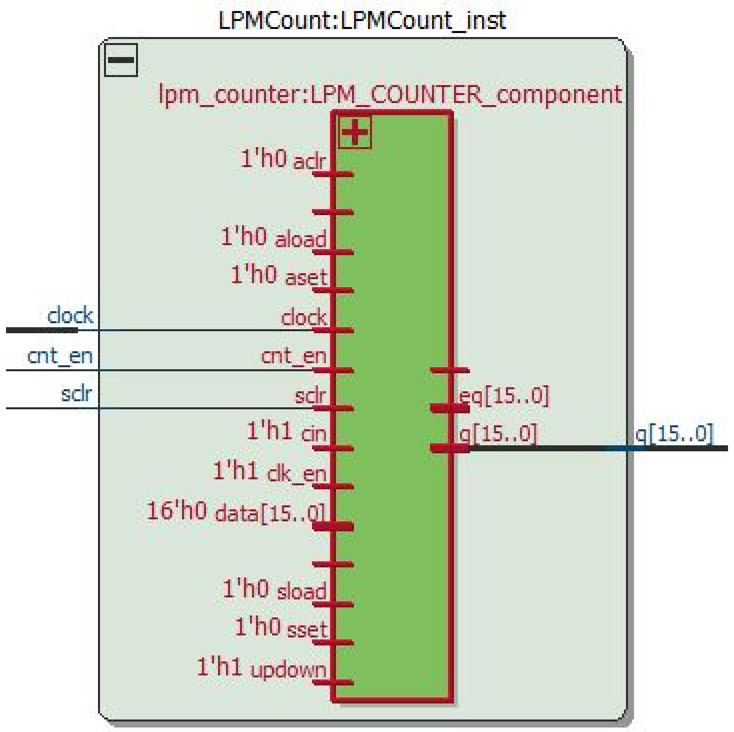
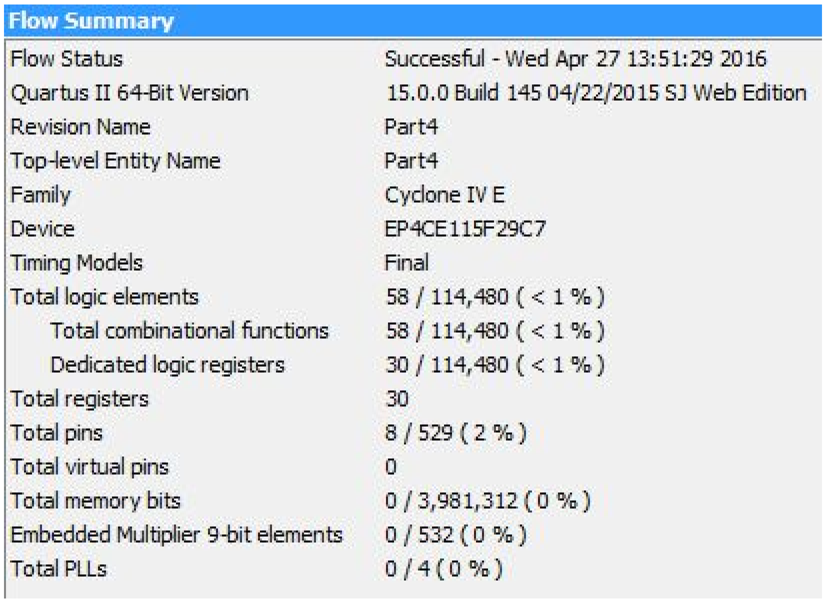


Figure 9. Expanded RTL view for LPMCount for Part 3



Flow Summary	
Flow Status	Successful - Wed Apr 27 13:51:29 2016
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Web Edition
Revision Name	Part4
Top-level Entity Name	Part4
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	58 / 114,480 (< 1 %)
Total combinational functions	58 / 114,480 (< 1 %)
Dedicated logic registers	30 / 114,480 (< 1 %)
Total registers	30
Total pins	8 / 529 (2 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 10. Summary for Part 4

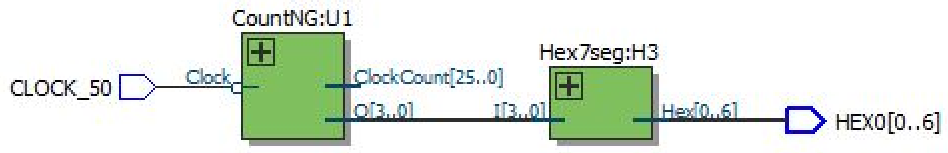


Figure 11. RTL for Part 4

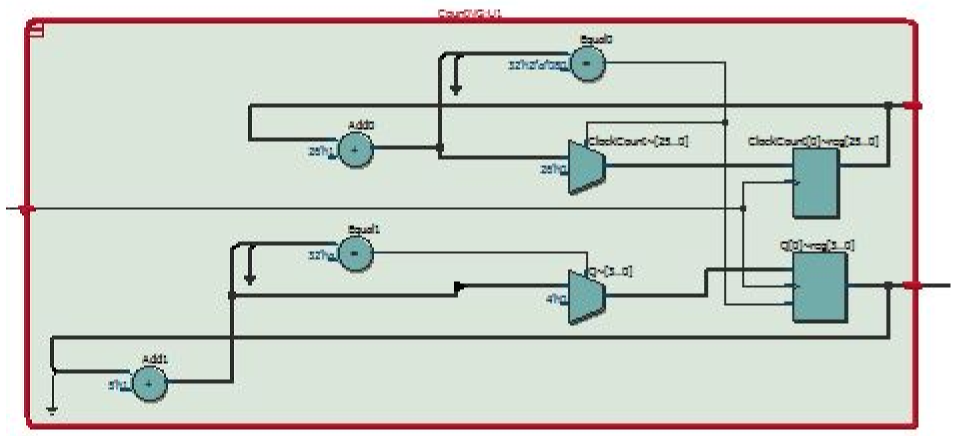
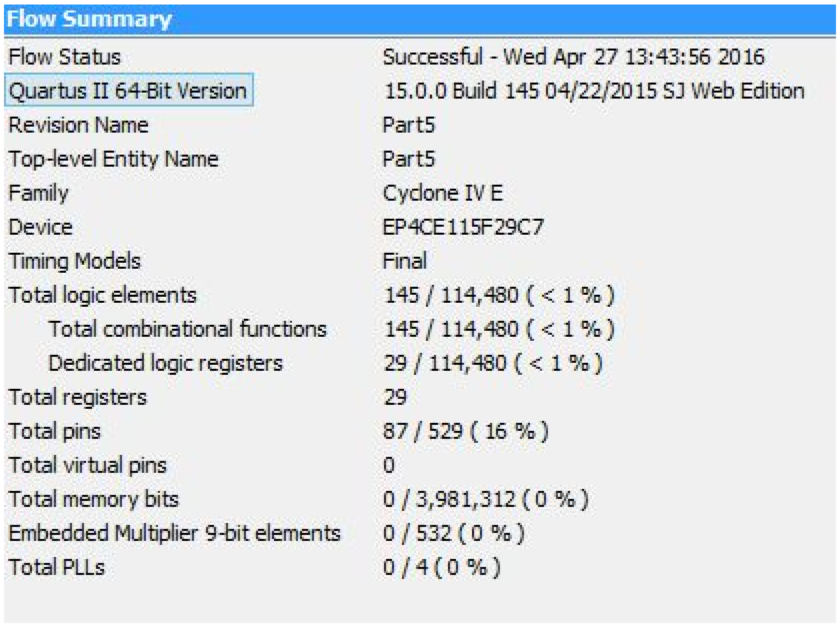


Figure 12. Expanded RTL view for CountNG for Part 4



Flow Summary	
Flow Status	Successful - Wed Apr 27 13:43:56 2016
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Web Edition
Revision Name	Part5
Top-level Entity Name	Part5
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	145 / 114,480 (< 1 %)
Total combinational functions	145 / 114,480 (< 1 %)
Dedicated logic registers	29 / 114,480 (< 1 %)
Total registers	29
Total pins	87 / 529 (16 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 13. Summary for Part 5

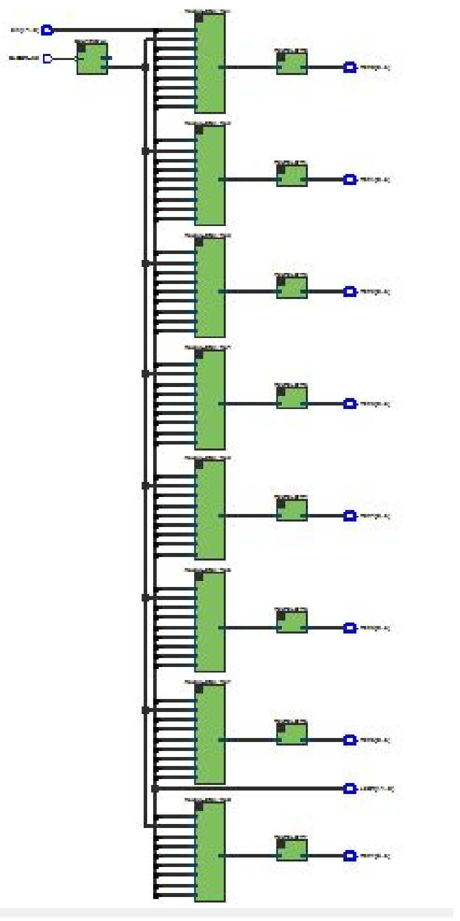


Figure 14. RTL for Part 5

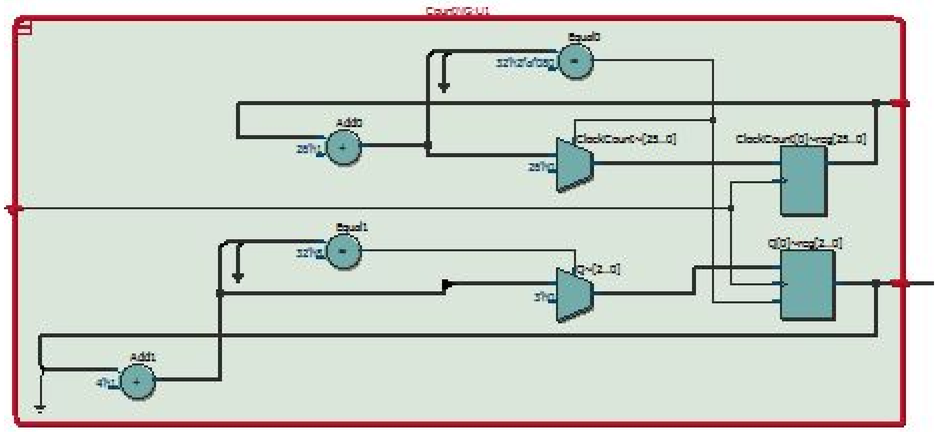
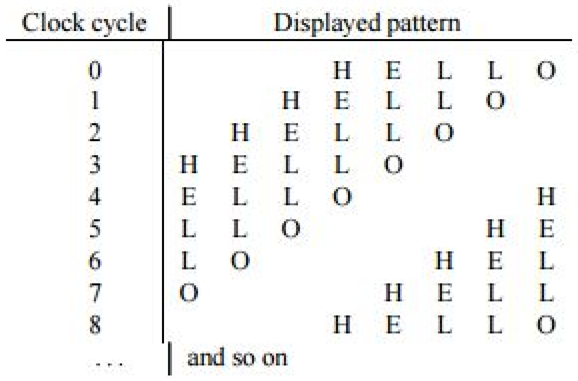
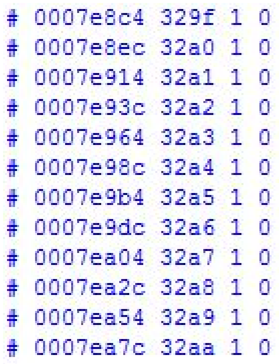


Figure 15. Expanded RTL view for CountNG for Part 5



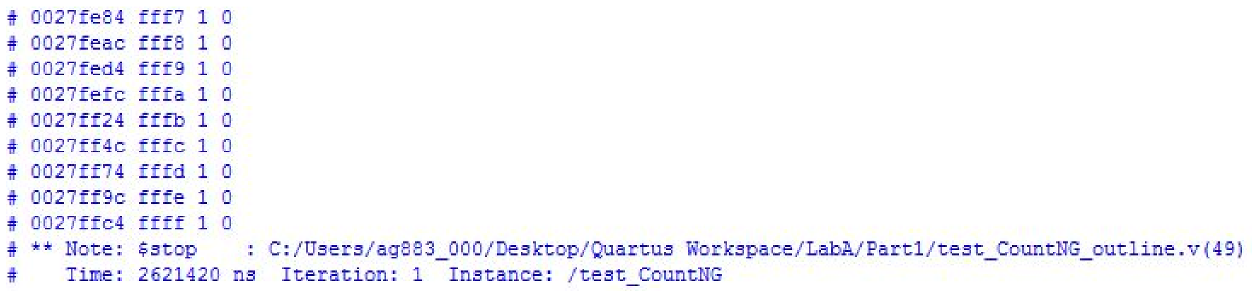
Clock cycle	Displayed pattern							
0				H	E	L	L	O
1			H	E	L	L	O	
2		H	E	L	L	O		
3	H	E	L	L	O			
4	E	L	L	O				H
5	L	L	O				H	E
6	L	O				H	E	L
7	O				H	E	L	L
8				H	E	L	L	O
. . .	and so on							

Figure 16. HEX HELLO display pattern for Part 5



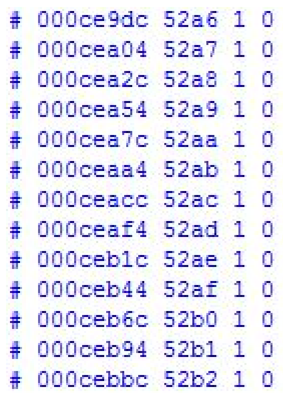
```
# 0007e8c4 329f 1 0
# 0007e8ec 32a0 1 0
# 0007e914 32a1 1 0
# 0007e93c 32a2 1 0
# 0007e964 32a3 1 0
# 0007e98c 32a4 1 0
# 0007e9b4 32a5 1 0
# 0007e9dc 32a6 1 0
# 0007ea04 32a7 1 0
# 0007ea2c 32a8 1 0
# 0007ea54 32a9 1 0
# 0007ea7c 32aa 1 0
```

Figure 17. ModelSim for Part 1 (Middle)



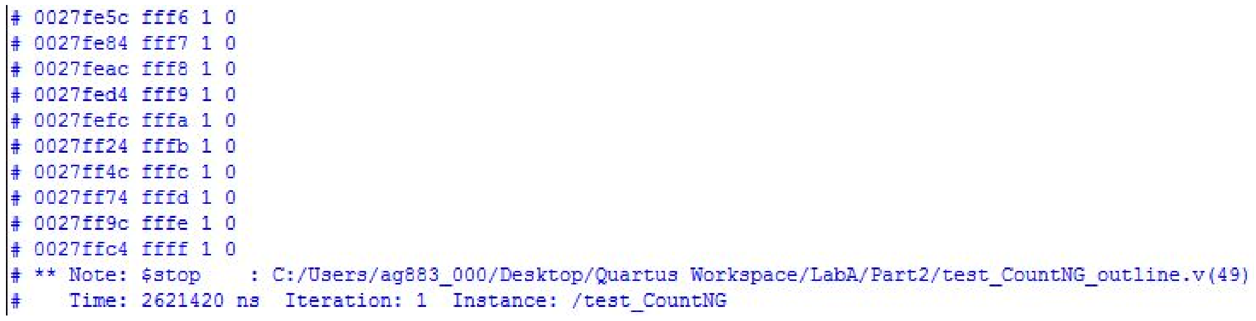
```
# 0027fe84 fff7 1 0
# 0027feac fff8 1 0
# 0027fed4 fff9 1 0
# 0027fefc fffa 1 0
# 0027ff24 fffb 1 0
# 0027ff4c fffc 1 0
# 0027ff74 fffd 1 0
# 0027ff9c fffe 1 0
# 0027ffc4 ffff 1 0
# ** Note: $stop    : C:/Users/ag883_000/Desktop/Quartus Workspace/LabA/Part1/test_CountNG_outline.v(49)
#    Time: 2621420 ns  Iteration: 1  Instance: /test_CountNG
```

Figure 18. ModelSim for Part 1 (End)



```
# 000ce9dc 52a6 1 0
# 000cea04 52a7 1 0
# 000cea2c 52a8 1 0
# 000cea54 52a9 1 0
# 000cea7c 52aa 1 0
# 000ceaa4 52ab 1 0
# 000ceacc 52ac 1 0
# 000ceaf4 52ad 1 0
# 000ceb1c 52ae 1 0
# 000ceb44 52af 1 0
# 000ceb6c 52b0 1 0
# 000ceb94 52b1 1 0
# 000cebbc 52b2 1 0
```

Figure 19. ModelSim for Part 2 (Middle)



```
# 0027fe5c fff6 1 0
# 0027fe84 fff7 1 0
# 0027feac fff8 1 0
# 0027fed4 fff9 1 0
# 0027fefc fffa 1 0
# 0027ff24 fffb 1 0
# 0027ff4c fffc 1 0
# 0027ff74 fffd 1 0
# 0027ff9c fffe 1 0
# 0027ffc4 ffff 1 0
# ** Note: $stop    : C:/Users/ag883_000/Desktop/Quartus Workspace/LabA/Part2/test_CountNG_outline.v(49)
#    Time: 2621420 ns  Iteration: 1  Instance: /test_CountNG
```

Figure 20. ModelSim for Part 2 (End)