3002CEM Coursework

Hardware Realisation of a computer system

Alexander Moses

# **Table of contents**

# **[Section A](#_Section_A_–) – 4-Bit Down Counter**

# [Page 2: Introduction](#_Introduction), Critical Discussion, Conclusion

# [Page 3: Design Source](#_Design_Source)

# [Page 4: Test Bench](#_Test_Bench)

# [Page 5:](#_Simulation) Constraints, Simulation

# Page 6: Synthesis

# Page 7: Implementation

# [**Section B**](#_Section_B_–) **– 32-Bit Controllable Counter**

# [Page 8: Introduction](#_Introduction_1), Critical Discussion, Conclusion

# [Page 9: Design Source](#_Design_Source_1)

# Page 10: Constraints

# Page 11: Synthesis

# Page 12: implementation

# Page 13 - 17: State Machine

# **Section A – 4-bit Down Counter**

# **Introduction**

In this section I will be creating a 4-bit down counter, counting from 15 to 0, then resets to 15. It displays the values of the counter as binary on 4 LEDs on a FPGA, with a reset switch to reset the value to 15 at any given time.

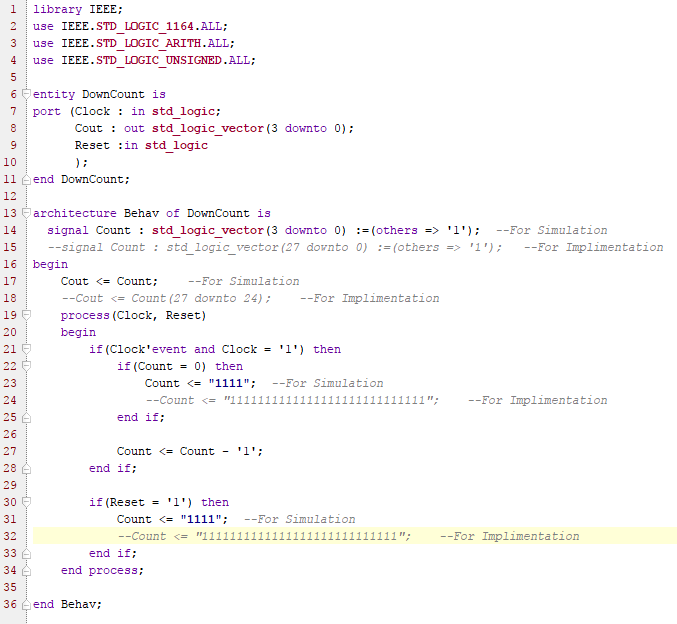
# **Critical Discussion of Steps**

I started this section off by taking the 4-bit up counter created in class during previous tutorials, and proceeded by changing it to count down and also changed aspects in respect to that (reset and the overflow check). The test bench was identical between the up counter and down counter, and so nothing of that had to be changed, and the same goes for the constraints. As the FGPA has a very high clock speed the 4-bit counter had to be changed for it to be able to be perceived by the human eye, and so I changed it to a 28-bit counter and only displayed the 4 most significant bits.

# **Conclusion**

This section was rather straight forward, only changing a few things, thus resulted in no problems or complications.

# **Design Source**



Entity ‘DownCount’ declaration, containing 3 ports: Clock being the clock of the FPGA; Cout being the 4-bit output; and Reset, which resets the counter.

Assigns the value of the counter (Count) to the 4-bit output

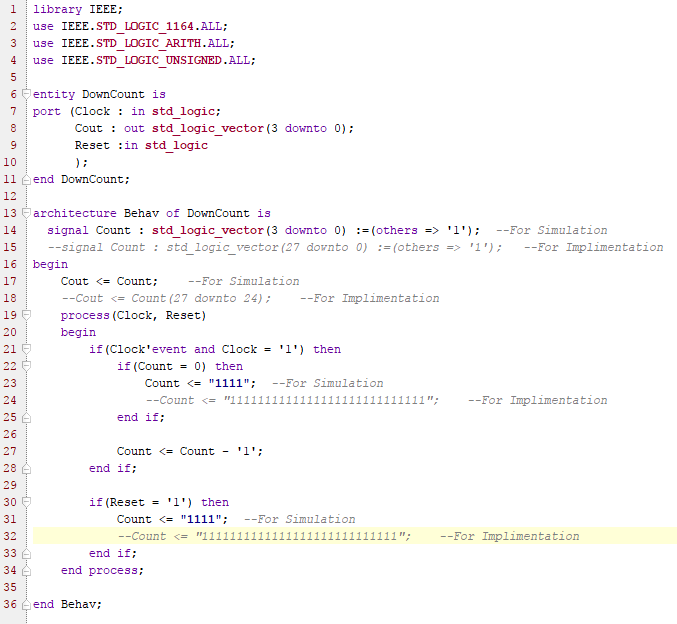
Process that is called when either clock or reset changes

Libraries

Signal that holds the value of the counter. The 4-bit version is for the simulation, the 28-bit version is for implementation as the FPGA’s clock is too fast for a 4-bit counter

Checks if count is equal to 0, then sets it back to its maximum to prevent overflow problems (4-bit for simulation and 28-bit for implementation)

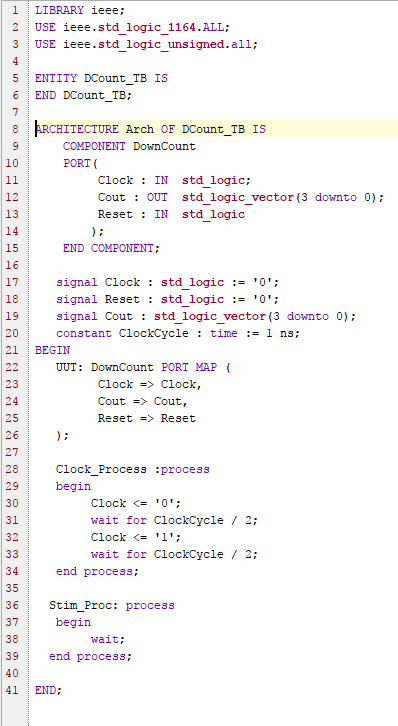
Checks if it was the clock that was changed which activated the process, and also checks if clock is equal to 1 so it changes on rising edge



Decrements the value of Count

Checks if Reset is equal to 1, if so it resets the value of the counter to its maximum. (4-bit for simulation and 28-bit for implementation)

# **Test Bench**



Process that is used to simulate the reset being changed, in this situation it isn’t simulated.

Clock\_process, occurs constantly, it regulates the clock pulse, making sure it changes

Maps the ports to their respective signal

Creates the signals that the testbench is going to simulate

States which component (DownCount) is going to be used, and states the ports that the component uses

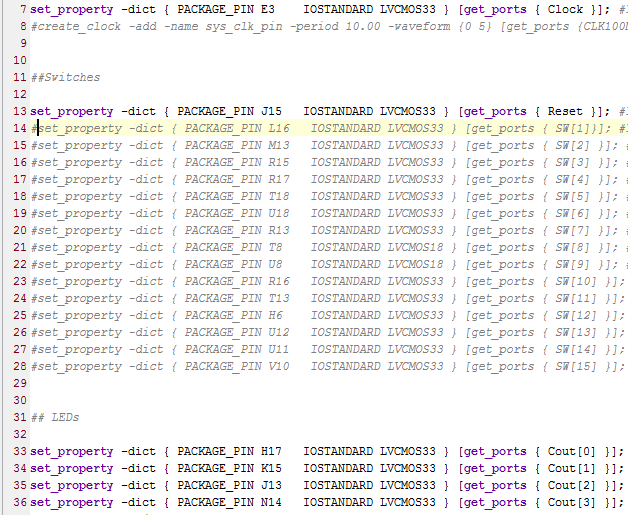
Entity ‘DCount\_TB’ declaration, it requires no ports and so none have been declarared

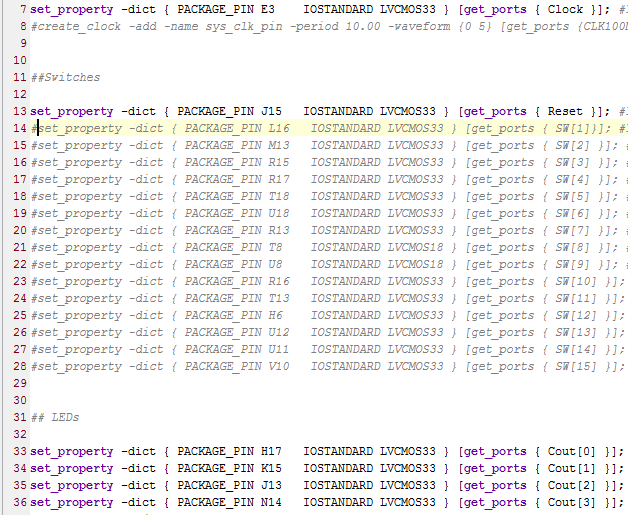
Libraries

# **Constraints**

The constraints used for implementing the design, mapping the ports to their respective hardware on the FPGA

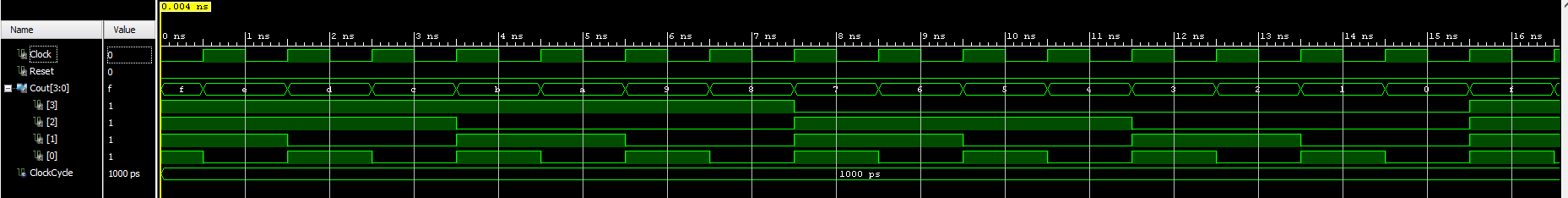
# 

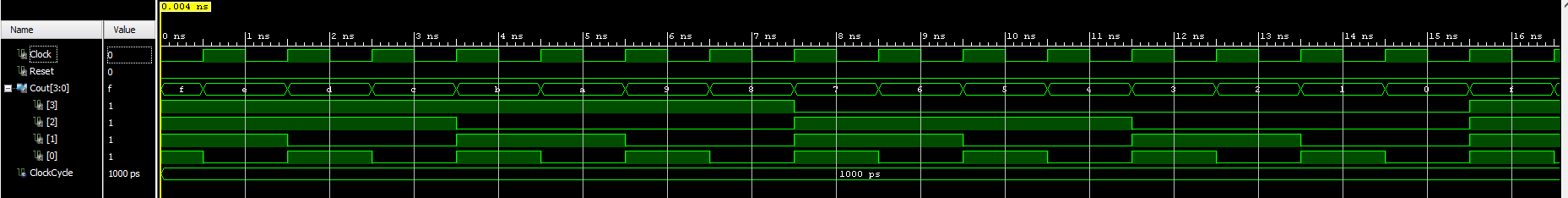
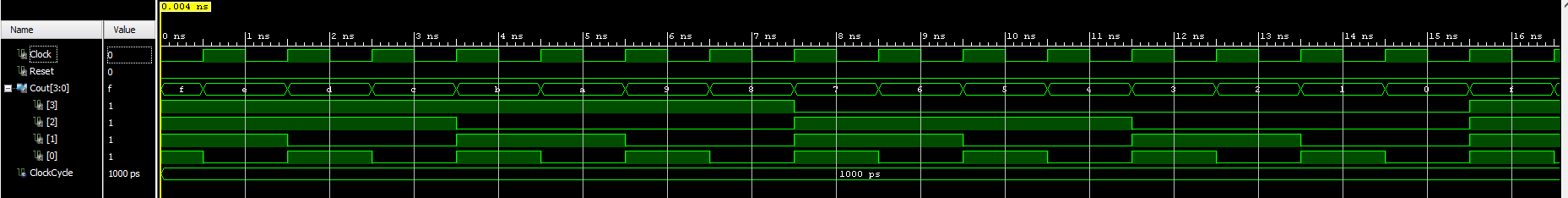


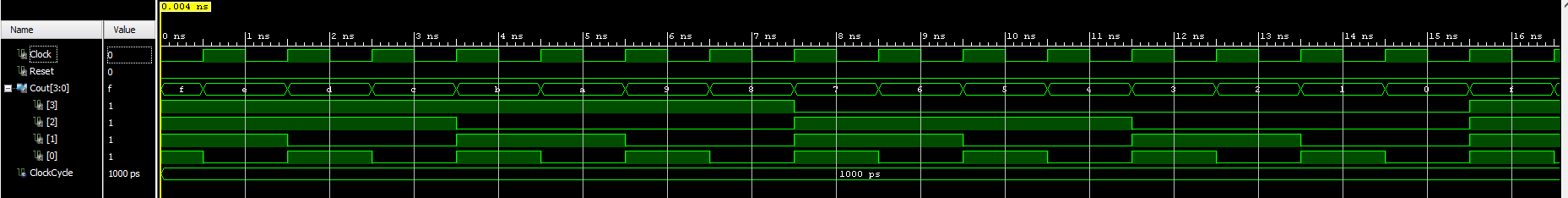


# **Simulation**

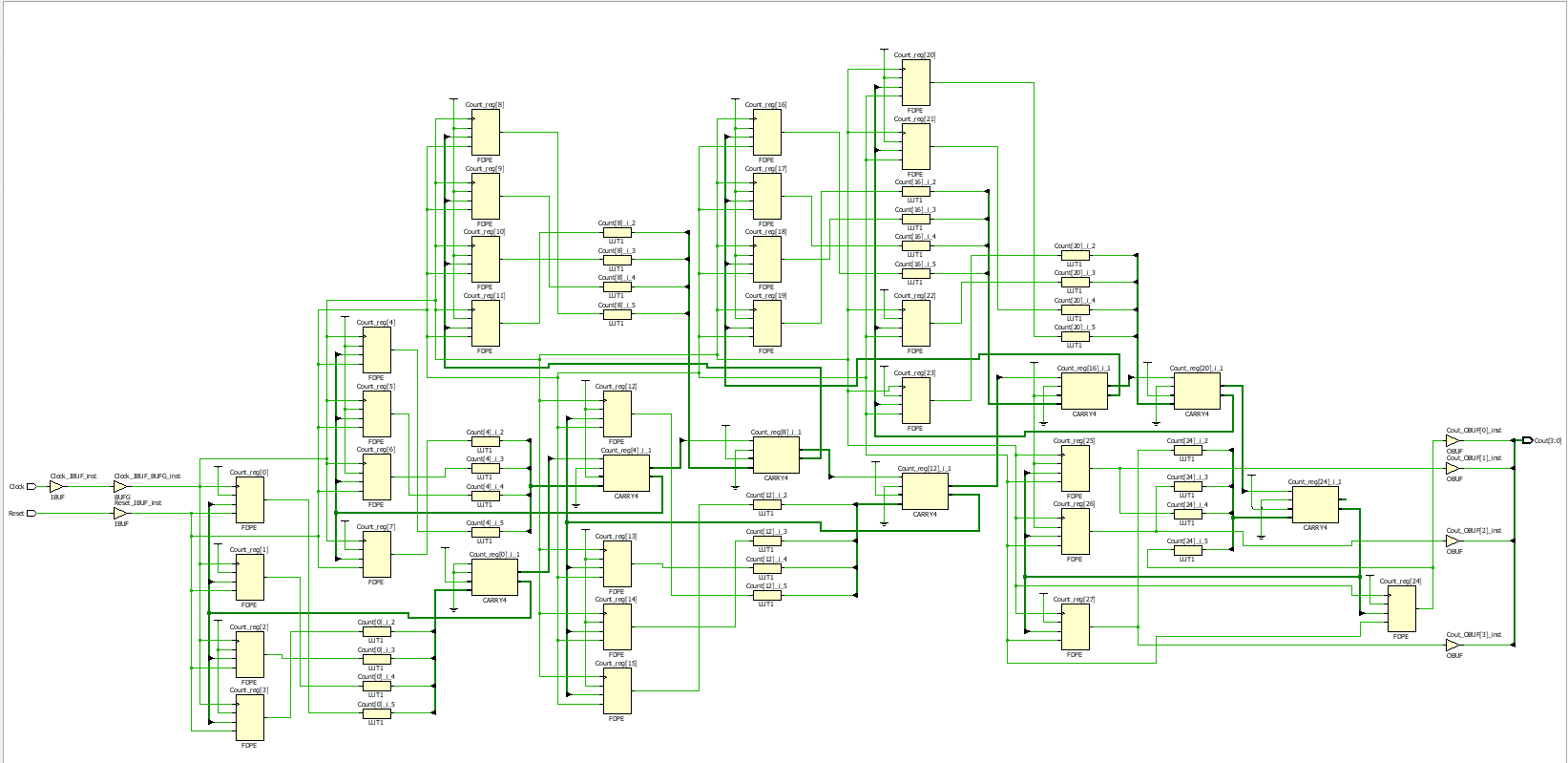
Below is the simulation of the 4-bit down counter. It starts at 15 (F) and counts down to 0 at 1ns intervals. Once it reaches 0, it resets to 15 (F) on the next rising edge clock pulse.





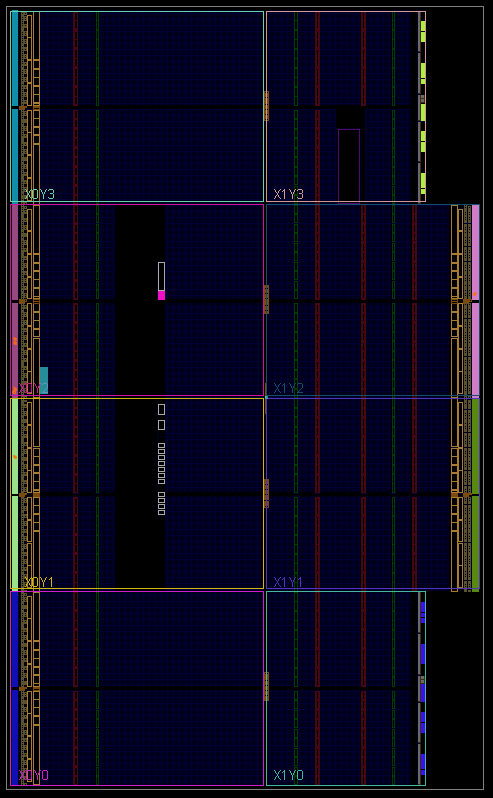


# **Synthesis - Schematic**



This is the schematic created by Vivado when the design source is synthesised, it shows the logic gates, adders etc.

# **Implementation – Die Shot**



This is the die shot of the circuit created by Vivado when the synthesised schematic is implemented.

# **Section B – 32-bit Controllable Counter**

# **Introduction**

In this section I will be creating a 32-bit counter, which has the capacity to count both up and down. It displays the 4 most significant bits of the counter as binary on 4 LEDs on a FPGA. With 2 switches, the first is a reset switch to reset the value to 0 at any given time, the second controls the direction (count up or down). While the counter is counting up, a red LED will be lit indicating its counting up, whilst a green LED will be lit to indicate counting down.

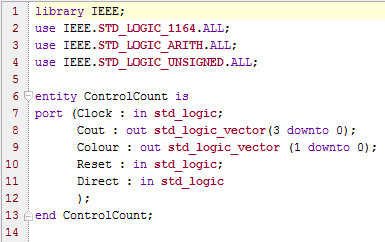
# **Critical Discussion of Steps**

I started this section by adapting the 4-bit counter used in section A, by changing it to 32-bit, and adding in the use of a second switch to control the direction. Once done I then added in the two different counters (up and down) and used the new switch to determine which one is used at a time, with the addition of the RGB LEDs to indicate which direction it is counting. The constraints from the first section were used, with the inclusion of the new switch plus the 2 RGB LEDs to be used.

# **Conclusion**

This section was also straight forward, with only having to use the previous as two different counters and adding in a switch to determine which one counts (Up or down), this resulted in no problems or complications.

# **Design Source**

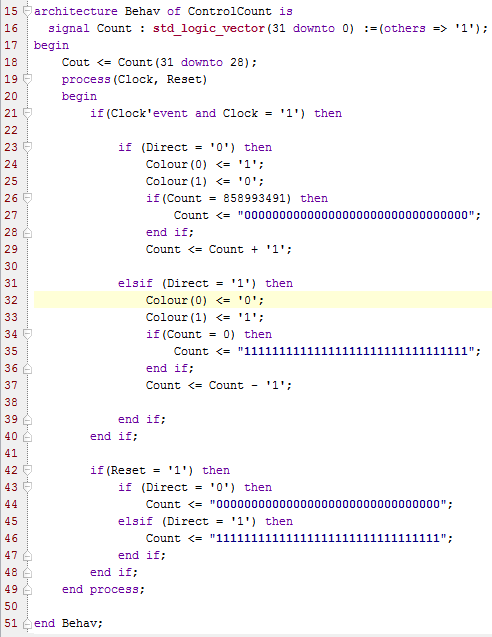


Declaration of a signal “Count”, 32-bits used to store the value of the counter

Libraries

Entity declaration, with 5 ports. First is the clock; second is the 4-bit output; third controls the coloured LEDs; fourth is the reset; and the fifth controls the direction

Assigns the 4 most significant bits of Count into the output (Cout)



Process that activates when clock or reset changes.

Checks if clock was the signal that changed and that Clock is equal to 1 so the counter changes Count on the rising edge

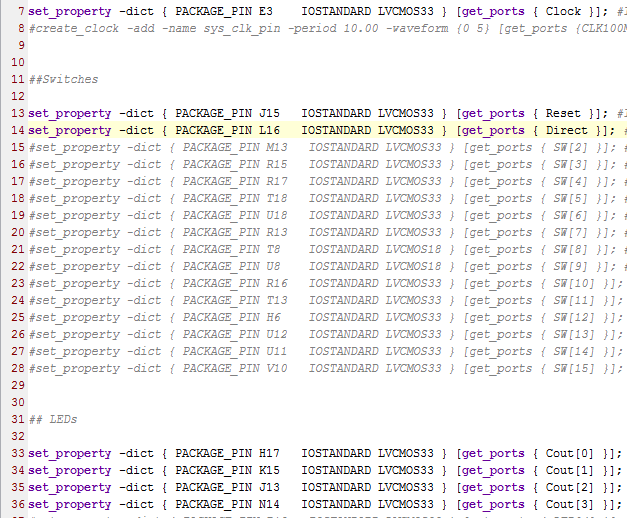
If Direct is 0 (counting up) it increments Count and changes the LEDs so the right-hand LED is red. It checks if the counter has reached it maximum possible value and resets it to the lowest possible value to prevent overflow problems

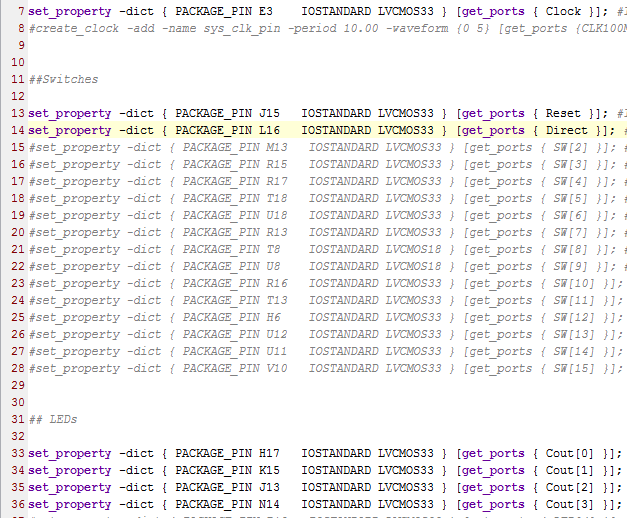
If Direct is 1 (counting down) it decrements Count and changes the LEDs so the left-hand LED is green. It checks if the counter has reached it lowest possible value and resets it to the maximum possible value to prevent underflow problems

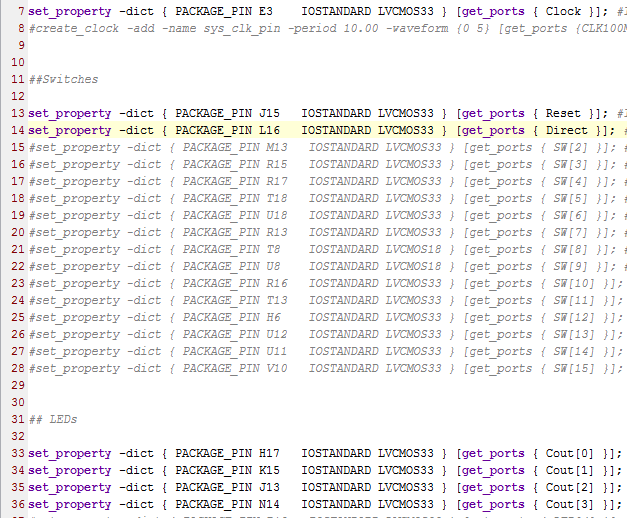
Checks if reset is 1, if so it resets the value of the counter depending on the direction. If its counting up then it resets to the lowest possible value whilst if its counting down it resets to the highest possible value

# **Constraints**

The constraints used for implementing the design, mapping the ports to their respective hardware on the FPGA







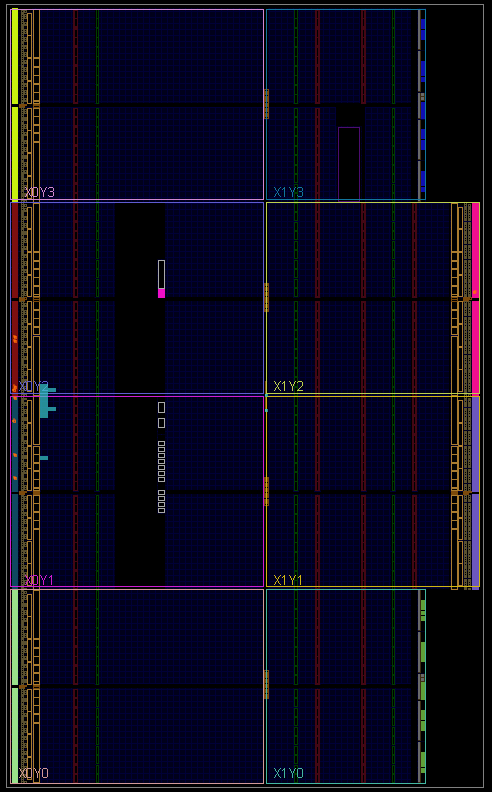


# **Synthesis - Schematic**

# 

This is the schematic created by Vivado when the design source is synthesised, it shows the logic gates, adders etc.

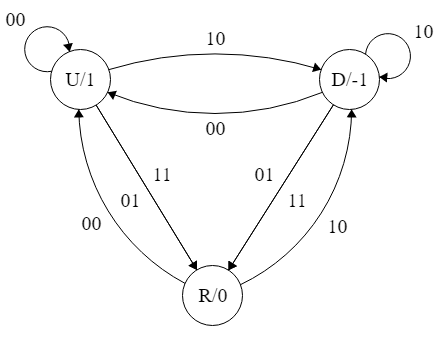
# **Implementation – Die Shot**



This is the die shot of the circuit created by Vivado when the synthesised schematic is implemented.

# **State Machine**

The controllable up down counter could be fashioned to follow a state machine design, with three states (Up, Down, and Reset) with two switches controlling the transition between the states.



11 01

This is the state machine for section B. It consists of three states: Up; Down; and Reset. Up being the state of which it counts up, Down being the state of which it counts down, and Reset being the state of which it resets the counter. The transitions between the three with the use of two switches, the first controlling to and from the reset state, and the second controlling the change of state from up to down or vice versa.

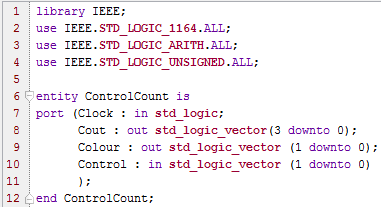
# **Critical Discussion of Steps**

I started by determining the best way to create the states, using processes, procedures or functions, of which I chose to use procedures as processes required the use of extra signals and functions can only return 1 value of which I needed 2 values (Count and Colour). Once done I then had to learn how to use procedures and then take the original design and implement the procedures. I then added in a select case to cover the 4 possible different inputs. The constraints had only been changed slightly, changing the two switches so that they both are under one vector.

# **Conclusion**

This section took the most amount of work, determining the best way to do the state machine, learning new aspects of VHDL and implementing it. However, once complete the code is a lot cleaner than the original design and is also a better way of completing the same task, from the coding point of view and also the schematic (page 16).

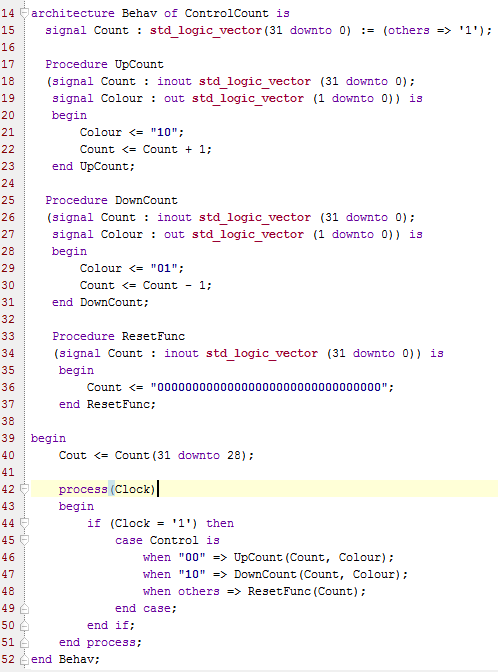
# **State machine - Design Source**



Declaration of a signal “Count”, 32-bits used to store the value of the counter

Entity declaration, with 4 ports. First is the clock; second is the 4-bit output; third controls the coloured LEDs; and fourth is the control, linked to the two switches, one controls the direction, one resets the value of the counter

Libraries



Process that activates when clock changes. Checks if Clock is equal to 1 so the counter changes Count on the rising edge, then calls which ever procedure needs to be called

Assigns the 4 most significant bits of Count into the output (Cout)

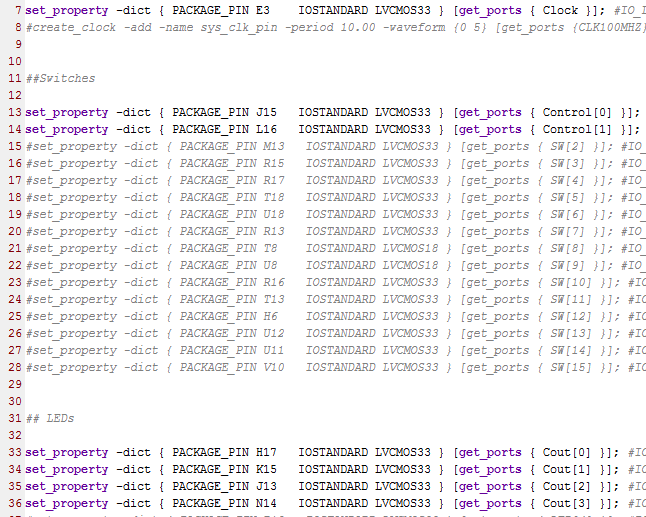
Procedure declaration “ResetFunc”, called when the reset switch is active, it resets the counters value.

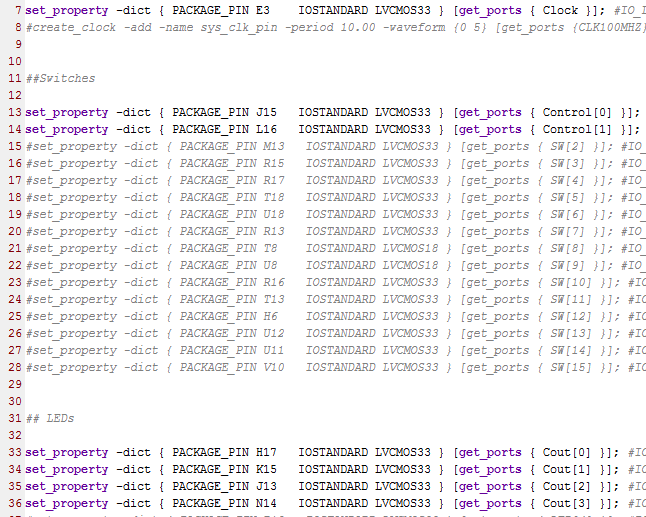
Procedure declaration “DownCount”, called when the counter is counter down, it decrements Count and changes the LEDs so the left-hand LED is green.

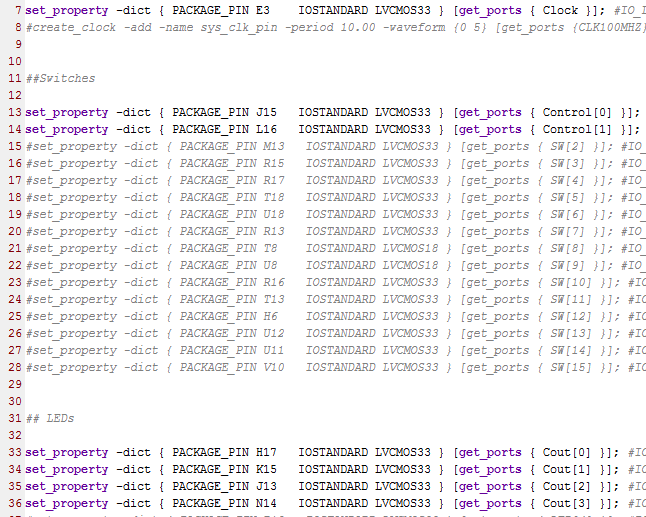
Procedure declaration “UpCount”, called when the counter is counter up, it increments Count and changes the LEDs so the right-hand LED is red.

# **State Machine – Constraints**

The constraints used for implementing the design, mapping the ports to their respective hardware on the FPGA







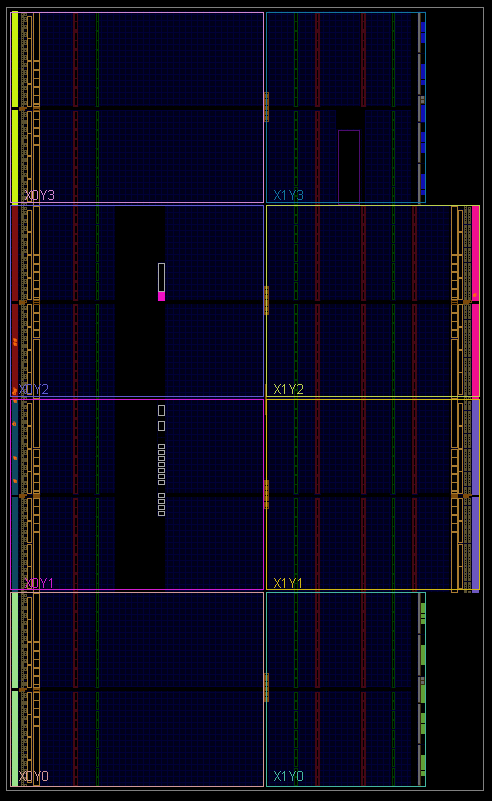


# **State machine – Synthesis - Schematic**

# 

This is the schematic created by Vivado when the design source is synthesised, it shows the logic gates, adders etc. Compared to the original design for section B, the state machine design uses a lot less logic gates, whilst being less complex

# **State machine – Implementation – Die Shot**



This is the die shot of the circuit created by Vivado when the synthesised schematic is implemented.