Benjamin Griffiths (160159871)

DIGITAL LOGIC CIRCUITS LAB | REPORT

Introduction:

This laboratory focuses on using simulation tools to design and test a circuit using a Field Programmable Gate Array (FPGA), which allows a circuit to be designed inside software and tested with a range of inputs. An FPGA is essentially a large array of programmable logic blocks that can all be interconnected. It can be programmed to perform any digital function and can be re-programmed at any time, making it ideal in engineering as a rapid prototyping tool, allowing a circuits function to be tested and developed thoroughly before manufacturing it. This in practice can reduce the costs and time during development of a product.

The design problem is to design and test a 2-bit synchronous adder, which will add together two 2-bit numbers using digital logic circuits which will be programmed into the FPGA.

Principles of Design Methodology:

Hierarchical Design:

Hierarchical design refers to dividing up a design task into multiple, independent tasks that are smaller and more manageable to work on. For example, a top-down design approach would involve identifying complex systems and dividing them into successively less complex sub systems.

The advantage of using such a design method is that the complexity of a system can be greatly reduced. For example, in *figure 1*, the total system can be said to be formed of the 3 subsystems, which is less complex than saying its formed of the 6 modules. It also adds a modular aspect of the system, enabling certain areas to be updated or modified easier when looking at an individual module compared to the system as a whole.

The diagram in *figure 1*, can also been seen as a 'top down' design approach, which again involves viewing the large and complex task at the top, then progressively working downwards while expanding each section of the task into smaller and simpler modules.

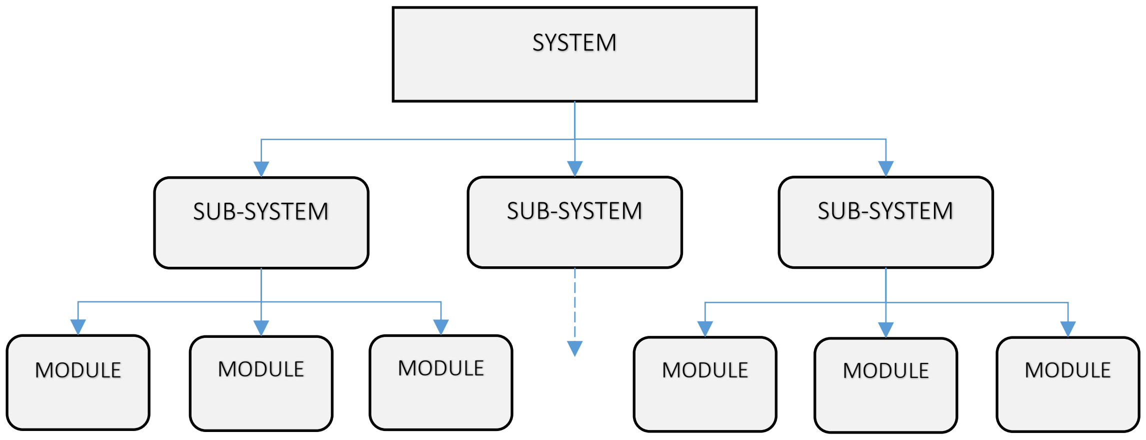

Figure 1: Hierarchical example diagram.

Basic Design Flow:

Below, in *figure 2*, is an example of a typical design flow outlining the steps involved in the process of design a circuit to be tested on an FPGA. Next to the flow-chart is a basic description of what each step involves.

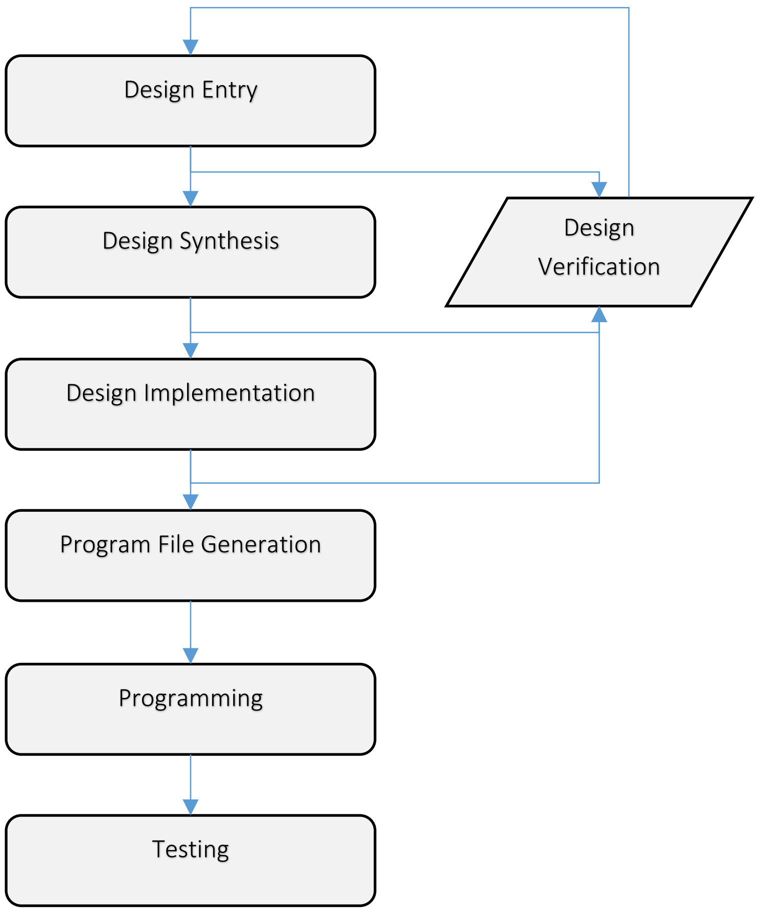

Figure 2: Basic FPGA Design Flow.

Design Entry - Creating circuit design using schematic diagram tools or coding a HDL (hardware description language) model.

Design Synthesis – Converting HDL model into a gate-level netlist (netlist consists of a list of all the terminals of the components within the circuit).

Design Implementation – Translates the netlist into an FPGA design. The Xilinx design flow consists of translating, mapping, placing and routing to achieve implementation. Place and route is the most important stage as it defines the interconnections within the FPGA.

Design Verification – At each stage, Design Verification is carried out, ensuring that the HDL implementation follows the design specification and that there are no errors. The verification process could consist of Behavioural and Timing Simulation. Where Behavioural simulation allows you to test the function of the design and timing simulation looks at propagation delays through various logic components etc.

Program File Generation - Creates a bit stream file that can be transferred to the device.

Programming - Configures device from program file.

Testing - Thoroughly tests devices function.

Design Problem/Discussion:

As described in the introduction, the design is 2-bit synchronous adder that inputs two 2-bit numbers and outputs the sum of them.

This design problem can be simplified by adopting a top-down design approach as shown below in *figure 3.* This allows the lower-level circuits/modules to be designed and tested at each step as the design is built up to its final arrangement.

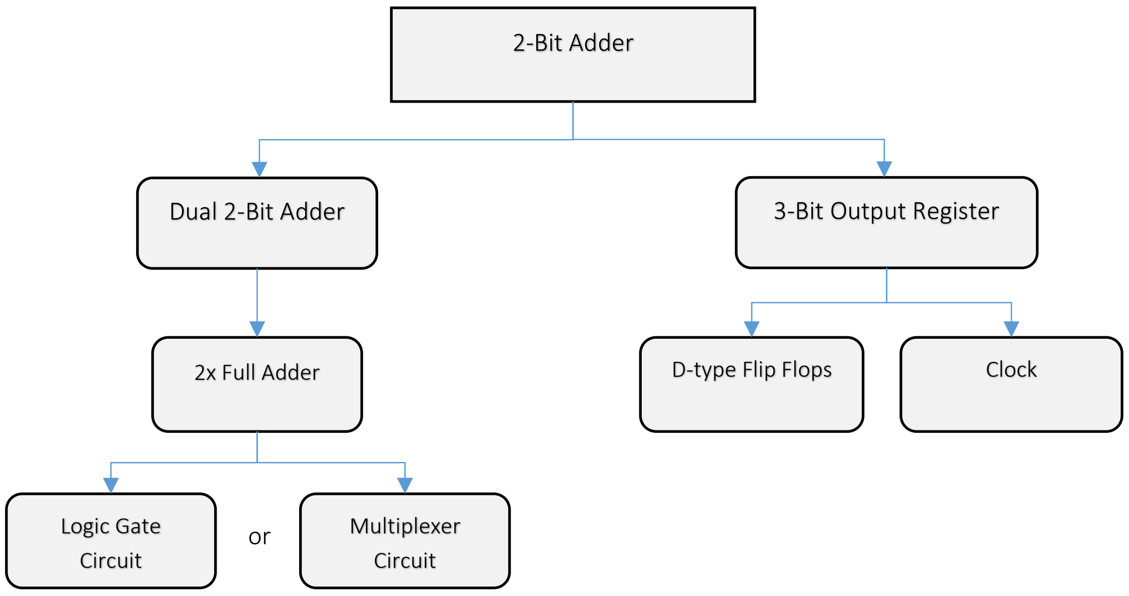

Figure 3: Top-down design approach used to design 2-bit adder.

The first step was to design and test the Logic gate circuit that functions as a 2-Bit full adder. This was designed using a schematic diagram like in *figure 4*, which was then verified using behavioural simulation that tested all the possible input and output signals to ensure the circuits function was correct.

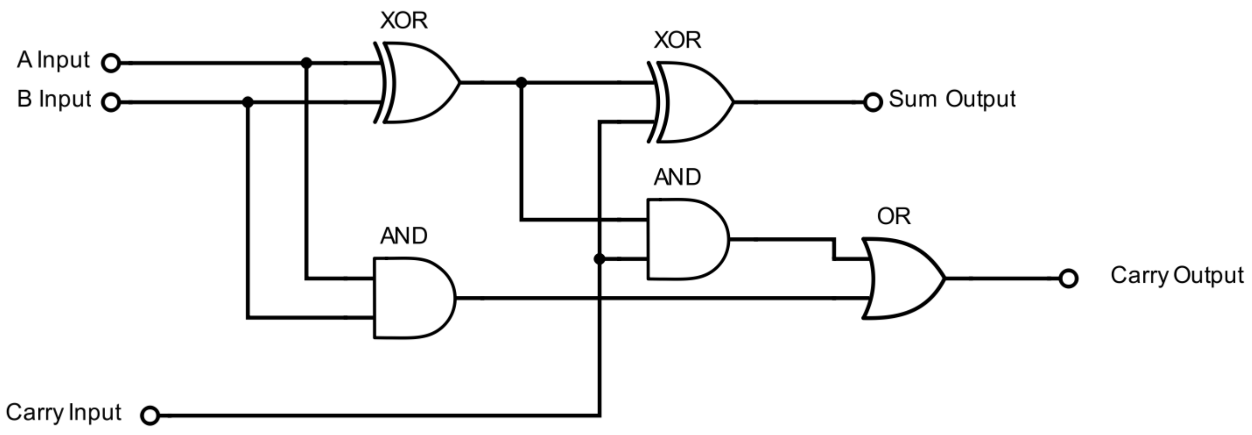

Figure 4: Full adder logic circuit.

To verify the correct function of this circuit, the behavioural simulation produced a timing diagram like in *figure 6* of all the possible input combinations which were compared to the truth table for this circuit in *figure 5.* As seen in the timing diagram, each vertical red line signifies a change in the input combination, with the corresponding change of the output combination.

Input			Output	
A	B	Carry	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 5: Full adder truth table.

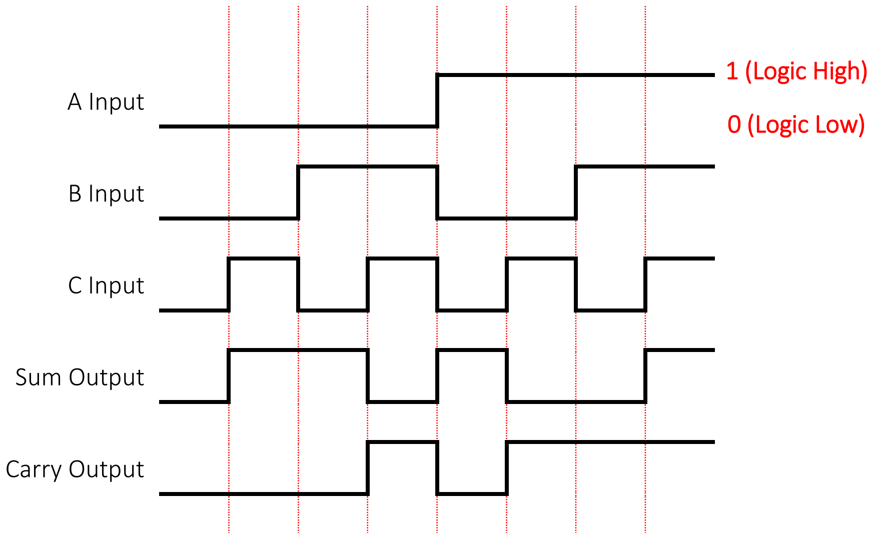

Figure 6: Full adder timing diagram.

An alternative method of designing a full adder is to use multiplexers, as seen in *figure 8.* It has the exact same function as the logic gate based circuit in *figure 4*, but uses two 4-1 multiplexers by manipulating the truth table and using the C input as one of the inputs to the multiplexers with the A & B inputs used as the address/select signals.

Input			Output		
A	B	Carry	Sum	Carry	
0	0	0	0	0	Sum = C
0	0	1	1	0	Carry = 0
0	1	0	1	0	Sum = C’
0	1	1	0	1	Carry = C
1	0	0	1	0	Sum = C’
1	0	1	0	1	Carry = C
1	1	0	0	1	Sum = C
1	1	1	1	1	Carry = 1

Figure 7: Full adder truth table for multiplexers.

[C’ = NOT C]
[1 or 0 = Logic HIGH or LOW]

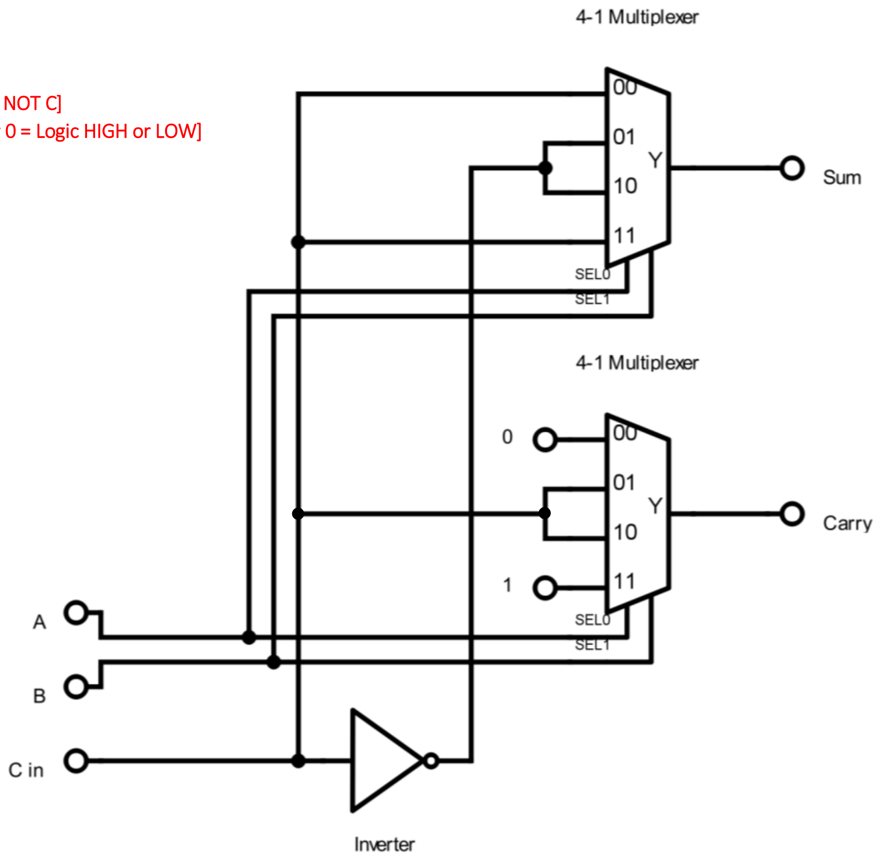

Figure 8: Full adder circuit using multiplexers.

As seen in *figure 7*, the output SUM and CARRY signals can be written in term of the CARRY IN signal, allowing a simpler multiplexer circuit to be designed where only the A and B input signals are required to change the multiplexers position.

Similar to the Logic gate based full adder circuit, behavioural simulation is carried out to verify that the function of the circuit follows the circuits truth table.

The next step is to move up the design flow, which as seen is *figure 3,* is to combine 2 of these together full adders into 1 module which is effectively a dual 2-bit adder, using either the circuit in *figure 4* or *figure 8*. This process is illustrated in the block diagram in *figure 9*.

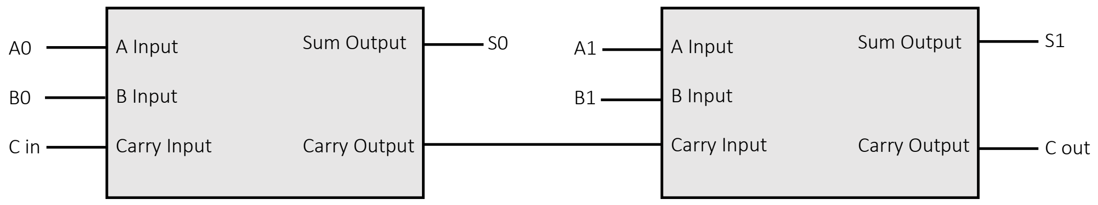

Figure 9: Combined full adders to make dual 2-bit adder.

The other component of the full system in accordance to the top-down design flow is the 3-bit output register, which stores the output of the dual full adder and displays them after a clock pulse (due to being a synchronous adder). As shown in *figure 3*, this register is constructed of 3 D-type flip flops connected together with a common clock. The arrangement of the circuit is shown below in *figure 10.*

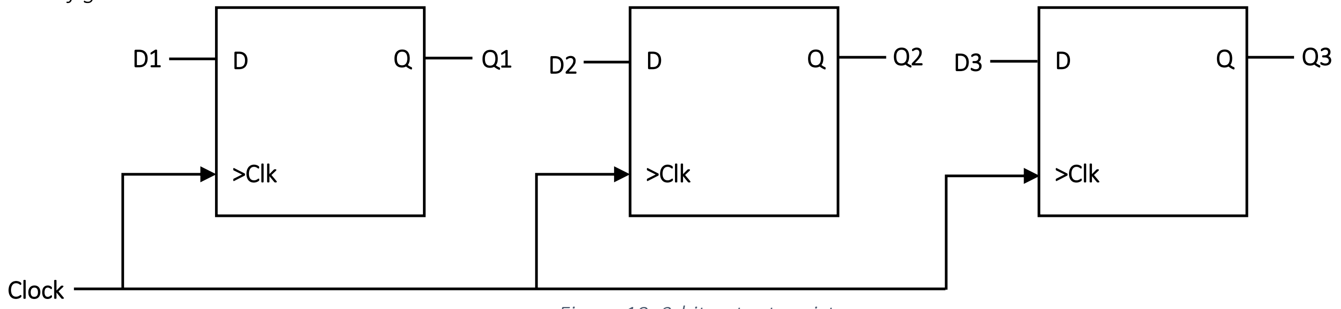

Figure 10: 3-bit output register.

This circuit can now be verified by putting in under a behavioural simulation to ensure it functions correctly. This circuit was tested using every input combination and the simulated output was compared to the expected output. The simulation displayed a timing diagram as seen in *figure 11,* which shows each input combination with its output between each vertical red line. The timing diagram was checked to see that each output is the same as it's corresponding output on the rising edge of each clock pulse.

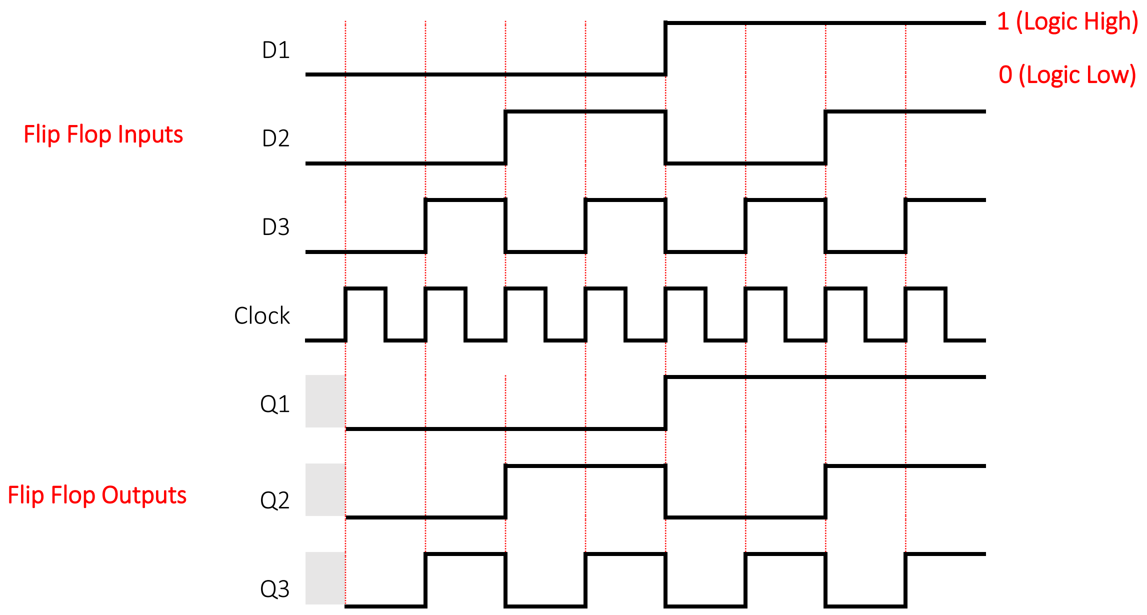

Figure 11: 3-bit register timing diagram.

Now that all the modules below the highest level of the hierarchy design flow diagram have been completed, the 3-bit register can be combined with the dual 2-bit adder. This is shown in *figure 12*, with the register connected to the 3-bit output from the adder.

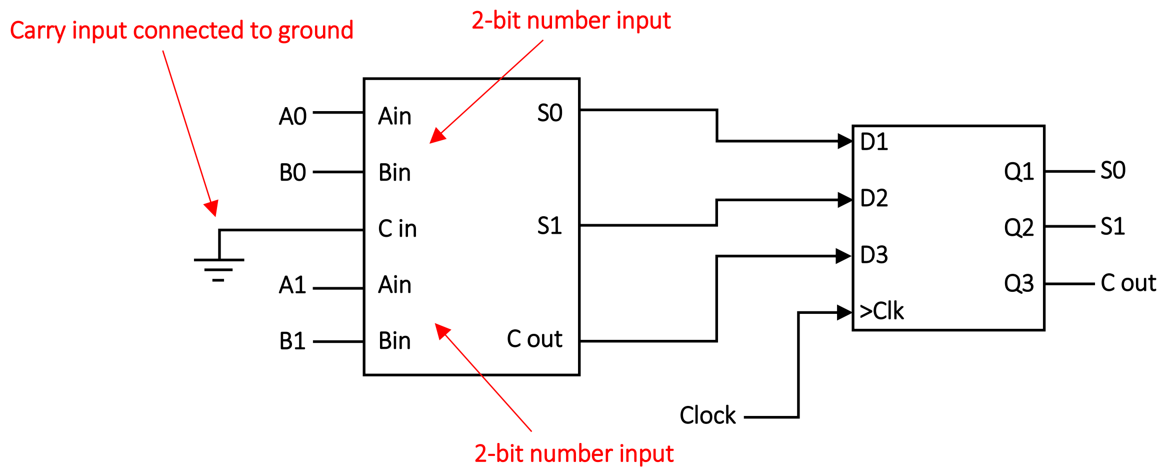

Figure 12: Complete dual 2-bit adder block diagram.

Finally, the complete system is verified using behavioural simulation, where every combination of the inputs is checked to see that the correct sum is calculated at the output according to the systems truth table as seen in *figure 13.* The behavioural simulation produced a timing diagram as seen in *figure 14,* where the output can be seen as the sum of the two 2-bit numbers on the rising edge of each clock pulse.

Input				Output		
B0	A0	B1	A0	C out	S1	S0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Figure 13: Complete dual 2-bit adder truth table.

As seen in *figure 14*, all the possible input combinations between the red vertical lines are summed and transferred to the output on the rising edge of each clock pulse.

The finished circuit was then uploaded to the FPGA and was physically tested to ensure is adds the two 2-bit numbers correctly and displays the result with the clock is pulsed.

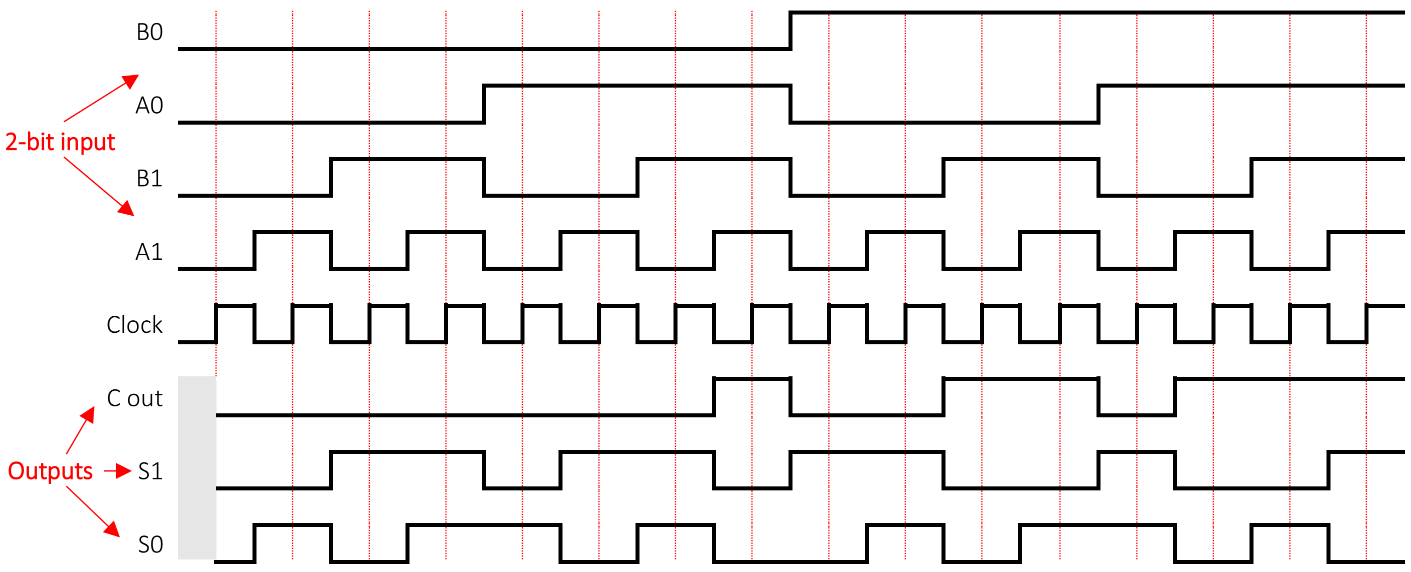

Figure 14: Complete dual 2-bit adder timing diagram.

Overall, the testing for this circuit was very thorough at every stage in the development. For the testing pattern, I chose to simply increment the binary value of the input combination each time, so effectively every possible input combination is tested to ensure that the particalar circuit operates exactly as expected. This testing method was also used to test the top level design, where every combination of each 2-bit number was tested to make sure that both the sum is correct and that the sum is only shown on the rising edge of a clock pulse.

This sort of thorough testing process would be essential if any small changes needed to be made to a particular piece of software code, known as regression testing. This is where after even a slight alteration to an existing piece of software, testing is carried out to ensure that the operation of the software is still as expected and that no bugs or errors have been introduced. Regression testing could be linked with benefits of hierarchy design, where if a slight change is made, only the single module the change is associated with would need to be tested. Whereas if the system was not designed as a hieracrchy model, it would very difficult to find where possible sources or error could come from and potentially all parts of the system would need to be tested. For example, in context of the design problem, by using the hierachy model, changing between the logic and multiplexer based adder circuit was easy as changes only needed to be made and tested to one module.

The benefits in terms of cost and time reduction have also been made clear, as there was no circuit construction that needed to be done, due to the FPGA being reprogrammable. For example, from a development perspective, to make a change to a system, you would potentially need to completely redesign a circuit and produce it, costing money and time, where it’s just the case of changing code and reprogramming for an FPGA.

In conclusion, by undertaking this design task, the key advantages of FPGA circuit design for rapid prototyping have been made clear and the design problem was solved successfully without any errors. For example, by adopting a hieracrchy design flow for the adder, it was possible to break down the circuit into several smaller and more simple modules, which could be easily designed and tested before building up the system. This would make sourcing errors if they were to occur much easier each module could be tested seperately compared to the entire system as a whole. There is not much that could be done to possibly improve this design task.

References:

Unknown Author. (2016, July 20th). *Basic FPGA Tutorial* [Online]. Available: https://www.so-logic.net/pl/knowledgebase/fpga_universe/tutorials/Basic_FPGA_Tutorial_Verilog

Microsoft. (2003). *Regression Testing* [Online]. Available: https://msdn.microsoft.com/en-us/library/aa292167(v=vs.71).aspx

1-CORE Technologies. (2009, February 11th). *FPGA design flow overview* [Online]. Available: http://www.fpgacentral.com/docs/fpga-tutorial/fpga-design-flow-overview