EE2016 - Experiment 1

Nishanth Senthil Kumar(EE23B049)
Gopalkrishna Ramanujan (EE23B029)
Arjun Krishnaswamy (EE23B009)

7 August 2024

1 Aim of Experiment

The aim of the experiment is to simulate and realise a half adder, full adder, and a 4 bit ripple adder using Xilinx Vivado and implementing it on a FPGA board.

2 Half Adder

2.1 Overview

- A half adder takes in two bits as input, and outputs the bitwise sum and the carry of the two bits.
- The sum is the bitwise XOR of the two given bits, and the carry is the bitwise AND of the two bits.

2.2 Source Code

```
module half_adder(
    input a,b,
    output sum,carry

);
    assign sum=(a^b);
    assign carry=(a&b);

endmodule
```

3 Full Adder

3.1 Overview

- A full adder finds the sum and carry of 3 1-bit inputs.
- The simplified boolean logic for the sum is

3.2 Source Code

```
module full_adder(
    input a,b,c,
    output sum,carry

);
    assign sum=(a^b)^(c);
```

```
    assign carry=(a&b)| c&(a^b);

endmodule
```

4 Ripple Adder

4.1 Overview

- A ripple adder is a combination of 4 full adders
- It is used to add two 4-bit binary numbers and it returns the sum (4-bit) and carry(1-bit)
- The inputs are two 4 bit numbers and a 1-bit input carry

4.2 Source Code

```
module fulladder(
    input a,b,c,
    output sum,carry

);
    assign sum=(a^b)^(c);
    assign carry=(a&b)|c&(a^b);

endmodule

module ripple_adder(
    input [3:0]a,
    input [3:0]b,
    input c,
    output s1,s2,s3,s4,
    output carry



);

    wire first_carry,second_carry,third_carry,fourth_carry;
    wire first_sum,second_sum,third_sum,fourth_sum;

    fulladder one(.a(a[0]),.b(b[0]),.c(c),.sum(first_sum),.carry(first_carry));
    fulladder two(.a(a[1]),.b(b[1]),.c(first_carry),.sum(second_sum),
    .carry(second_carry));
    fulladder three(.a(a[2]),.b(b[2]),.c(second_carry),.sum(third_sum),
    .carry(third_carry));
    fulladder four(.a(a[3]),.b(b[3]),.c(third_carry),.sum(fourth_sum),
    .carry(fourth_carry));
    assign s1=first_sum;
    assign s2=second_sum;
    assign s3=third_sum;
    assign s4=fourth_sum;
    assign carry=fourth_carry;


endmodule
```

5 Uploading to FPGA

5.1 '.xdc' Constraints file

5.1.1 For Half Adder

```
set_property -dict { PACKAGE_PIN L5    IOSTANDARD LVCMOS33 } [get_ports { a }];
set_property -dict { PACKAGE_PIN L4    IOSTANDARD LVCMOS33 } [get_ports { b }];
set_property -dict { PACKAGE_PIN T5    IOSTANDARD LVCMOS33 } [get_ports { sum }];
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { carry }];
```

5.1.2 For Full Adder

```
set_property -dict { PACKAGE_PIN L5    IOSTANDARD LVCMOS33 } [get_ports { a }];
set_property -dict { PACKAGE_PIN L4    IOSTANDARD LVCMOS33 } [get_ports { b }];
set_property -dict { PACKAGE_PIN M4    IOSTANDARD LVCMOS33 } [get_ports { c }];
set_property -dict { PACKAGE_PIN T5    IOSTANDARD LVCMOS33 } [get_ports { sum }];
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { carry }];
```

5.1.3 For Ripple Adder

```
set_property -dict { PACKAGE_PIN L5    IOSTANDARD LVCMOS33 } [get_ports { a[0] }];
set_property -dict { PACKAGE_PIN L4    IOSTANDARD LVCMOS33 } [get_ports { a[1] }];
set_property -dict { PACKAGE_PIN M4    IOSTANDARD LVCMOS33 } [get_ports { a[2] }];
set_property -dict { PACKAGE_PIN M2    IOSTANDARD LVCMOS33 } [get_ports { a[3] }]
set_property -dict { PACKAGE_PIN M1    IOSTANDARD LVCMOS33 } [get_ports { b[0] }];
set_property -dict { PACKAGE_PIN N3    IOSTANDARD LVCMOS33 } [get_ports { b[1] }];
set_property -dict { PACKAGE_PIN N2    IOSTANDARD LVCMOS33 } [get_ports { b[2] }];
set_property -dict { PACKAGE_PIN N1    IOSTANDARD LVCMOS33 } [get_ports { b[3] }];
set_property -dict { PACKAGE_PIN P1    IOSTANDARD LVCMOS33 } [get_ports { c }];

set_property -dict { PACKAGE_PIN R6    IOSTANDARD LVCMOS33 } [get_ports { s1 }];
set_property -dict { PACKAGE_PIN T5    IOSTANDARD LVCMOS33 } [get_ports { s2 }];
set_property -dict { PACKAGE_PIN R5    IOSTANDARD LVCMOS33 } [get_ports { s3 }];
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { s4 }];
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { carry }];
```

6 Learnings and Observations

- After writing the Verilog code and testing the code using simulation using a test bench and configuring the ".xdc" file, we started the procedure to upload the code to the FPGA.
- We first synthesised the code. Synthesis refers to the process of converting a high-level hardware description(RTL) into a lower-level, gate-level representation that can be physically implemented on hardware like FPGA. The Synthesis process converts the RTL(Register Transfer Lever) into a lower gate level netlist.
- A netlist is a textual representation of an electronic circuit, specifically detailing the components and the connections between them
- Implementation phase is responsible for translating this netlist into a physical design that can be loaded onto the FPGA.
- The implementation process consists of numerous sub processes.

  - Placement :Assign each logic element from the netlist to specific physical locations on the FPGA.
  - Routing :Connecting the different logic elements that we placed using the FPGA's routing mechanism as per the netlist that was generated.
  - Timing Analysis and Optimisation: Verify that the design meets all the timing constraints and improves the design so as the optimise the space and power taken up by the working components. It also ensures that the design adheres to all physical and logical constraints.

- After the implementation process, Vivado generates a bitstream that can be uploaded into the FPGA using a USB cable.
- We open up the hardware manager and make sure that the port through which we send the bitstream is recognised.
- We showed the full adder and the ripple adder to the TA's. We changed the inputs using switches on the FPGA board and showed ouput using the LED's of the FPGA board.
- The Vivado software generates a large number of reports upon running the code on the FPGA. Some of these include Synthesis Report, Placement design Report, Synthesis Report etc.
- The reports say that the synthesis period had a maximum data usage of 1294.773 MB from our computer. The report also checks if the synthesis is completed properly using the common error checking protocol "checksum" and it provides you with the checksum key of the synthesis.
- It also contains information on the time taken for the system to synthesis this piece of code.
- The place design report contains information on the different types of memory elements that are present on the FPGA board, and how much memory they store, and which part of it is assigned to some part of the netlist.

7 Output Waveforms

7.1 Waveform output Full Adder

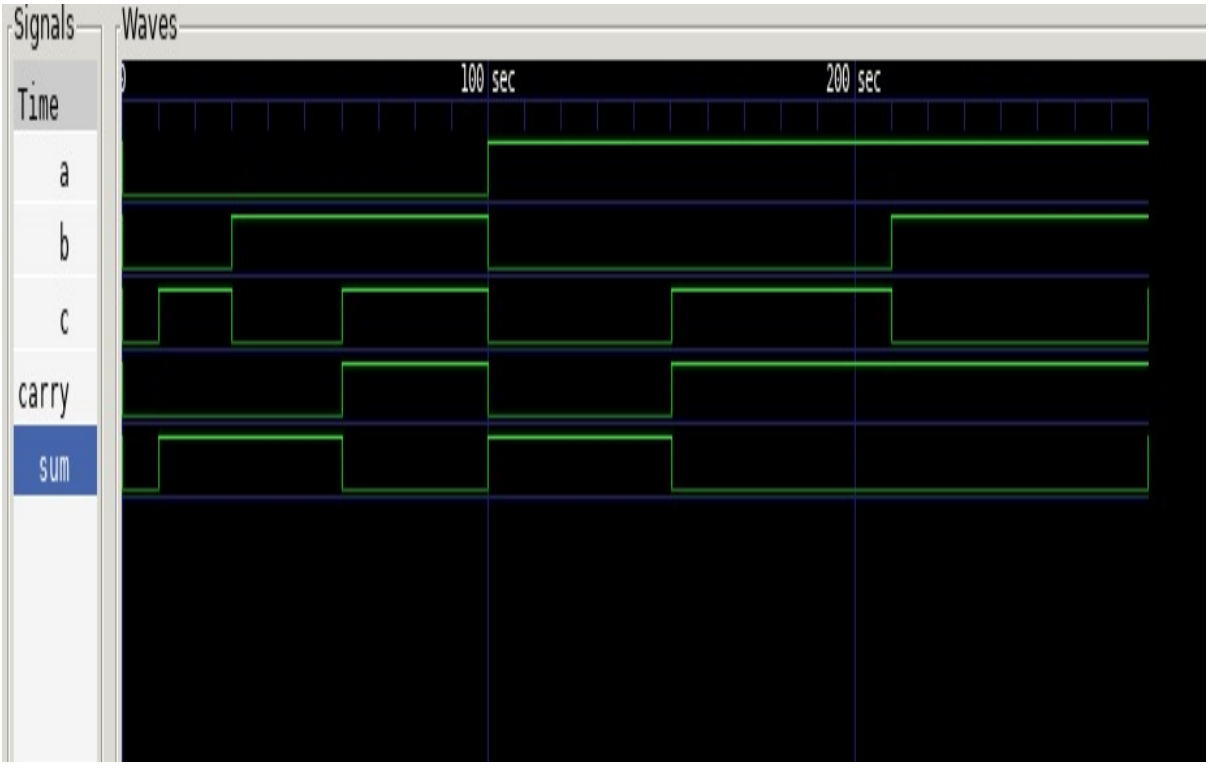

Figure 1: Full Adder Waveforms

7.2 Waveform output Ripple Adder

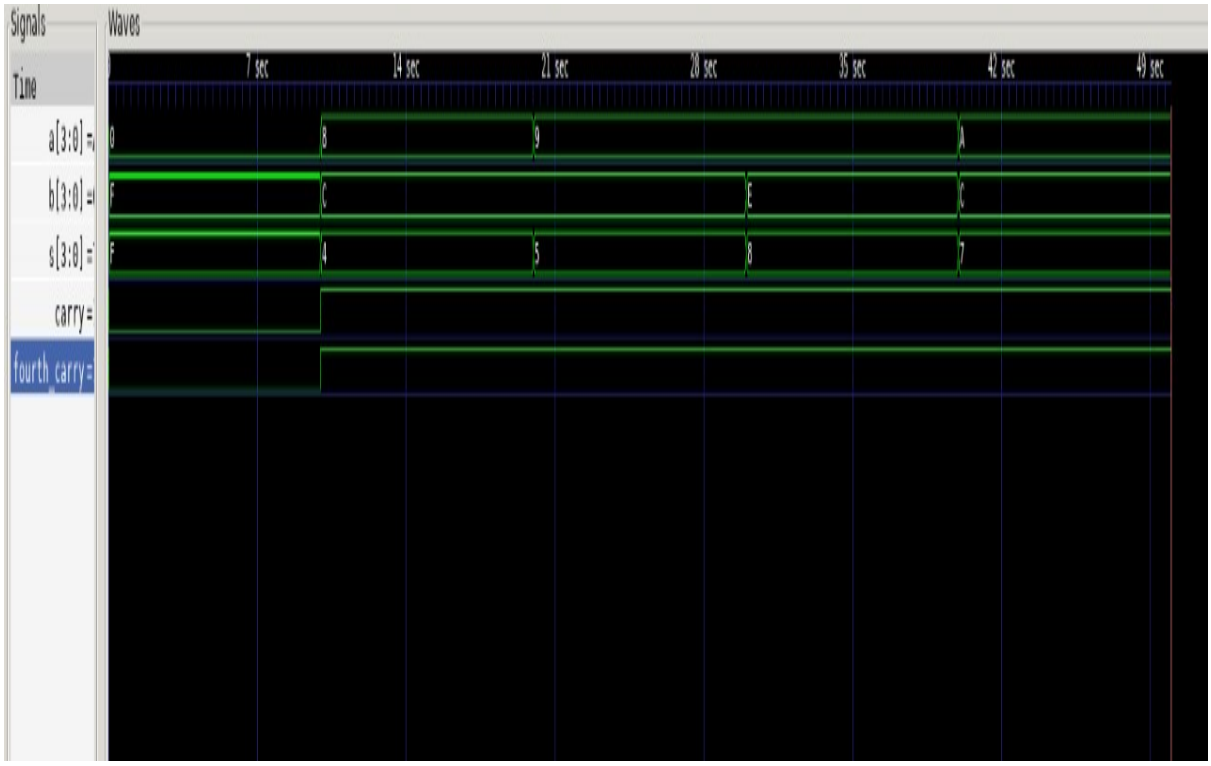

Figure 2: Ripple Adder Waveforms

8 Programming and Debugging experience

- Helped us get familiarised with the foundations of verilog such as creating modules and testbenches.
- Dataflow modeling was used to implement the circuits
- Introduced us to the Xilinx vivado software and the edge-A7 board
- We ran into issues while trying to perform the simulation on the board since we selected some input and output pins that were not on the board
- We also learnt that the testbench file is not necessary when we were trying to simulate since we give the inputs on the board directly.
- Another thing we learnt during the debugging process is that we need to comment out all the unused pins in the constraints file

9 References

- https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Introduction?tocId=CADL3ahSZQVONijyAN534g
- https://docs.amd.com/r/en-US/ug904-vivado-implementation/Preparing-for-Implementation
- http://vlabs.iitkgp.ernet.in/coa/exp1/index.html