

“Simulation of 4-bit Ripple Carry Adder”

Assignment-1 Report

*Submitted in Partial Fulfillment of the
Requirements for completion of*

Course:

VL505 - System design with FPGA

By

Aman Prajapati – MT2022501

Nishit Chechani – MT2022511

Raj Kachhadiya – MT2022513



**Department of Electronics and Communication Engineering,
International Institute of Information Technology,
Bangalore - 560100**

August 2022

Abstract

The report demonstrates a 4-bit Ripple Carry Adder with its simulation on Vivado and implementation on the *Basys 3* board. An important idea in digital electronics is the ripple carry adder, which is crucial for developing digital circuits. It also serves as a basic fundamental circuit used in digital electronics. It involves a basic idea of cascading several full adder combinational circuits in parallel fashion to compute the addition. To add a M-bit number M number of full adder circuits need to be cascaded in parallel. The project begins with designing a full adder first and then using that, constructing a 4-bit Ripple Carry Adder. After the Verilog HDL is written, with the help of Test Bench, the RTL simulation is carried out. The entire designing and implementation of the project is done on *Xilinx Vivado*. Next, using VIO (Virtual Input-Output), the inputs are fed to the Basys 3 board and the corresponding output is observed on the board. The carry later is mapped to the carry-chains of the FPGA fabric. Moreover, analysis of delay and resource utilisation after implementation is also carried out successfully. The factors which result in the delay were studied and an attempt to reduce them was made successfully as well.

INDEX

Chapter No.	Title	Page No.
	Index	3
	List of Figures	4
	List of Tables	5
1	Introduction	6
	1.1 Prologue	6
	1.2 Working of 4-bit Ripple Carry Adder	7
	1.3 Objective of the Report	8
2	Details related to the topic 2.1 Vivado 2.2 Basys 3 board 2.3 FPGA Fabric	9
3	Questions	12
	Conclusion	36
	References (as per IEEE format)	37

LIST OF FIGURES

Figure No.	Title	Page No.
1.1	4-bit Ripple Carry Adder block diagram	5
1.2	1-bit Full Adder Truth table & Block diagram	6
2.1	Basys 3 FPGA board with callouts	9
2.2	Sample Carry chain	10
3.1	RTL Simulation of 4-bit RCA	15
3.2	Schematic of 4-bit RCA	15
3.3	LED Output 1	16
3.4	LED Output 2	16
3.5	LED Output 3	17
3.6	LED Output 4	17
3.7	LED Output 5	18
3.8	LED Output 6	18
3.9	LED Output 7	19
3.10	LED Output 8	19
3.11	LED Output 9	20
3.12	LED Output 10	20
3.13	VIO Input 1	22
3.14	VIO - LED Output 1	22
3.15	VIO Input 2	23
3.16	VIO - LED Output 2	23
3.17	VIO Input 3	24
3.18	VIO - LED Output 3	24
3.19	VIO Input 4	25
3.20	VIO - LED Output 4	25
3.21	VIO Input 5	26
3.22	VIO - LED Output 5	26

3.23	VIO Input 6	27
3.24	VIO - LED Output 6	27
3.25	VIO Input 7	28
3.26	VIO - LED Output 7	28
3.27	VIO Input 8	29
3.28	VIO - LED Output 8	29
3.29	VIO Input 9	30
3.30	VIO - LED Output 9	30
3.31	VIO Input 10	31
3.32	VIO - LED Output 10	31
3.33	Carry Chain Mapping	32
3.34	Ports before changes	33
3.35	Timing before changes	33
3.36	Ports after changes	34
3.37	Timing after changes	34
3.38	Timing with negative Slack	35
3.39	Timing with positive Slack	35

Introduction

1.1 Prologue

In the domain of digital circuits, the purpose of addition was carried out using half adders and full adders. However, if there are more bits in the input sequence, a half adder or a full adder are unable to finish the addition operation. The "Ripple Carry Adder" can be used to overcome these disadvantages. It's a special kind of logic circuit used in digital operations to add N-bit values. The results of adding an n-bit binary sequence are produced by cascading a structure of numerous complete adders. The carry will be created at each full adder stage in a ripple-carry adder circuit since this adder's construction comprises cascaded full adders. These carry outputs from each full adder stage are passed on to the following full adder and used as a carry input there. This procedure goes on until the final complete adder stage. In a complete adder, each carry output bit is then rippled to the following stage. Thus, it goes by the term "RIPPLE CARRY ADDER".

Block Diagram:

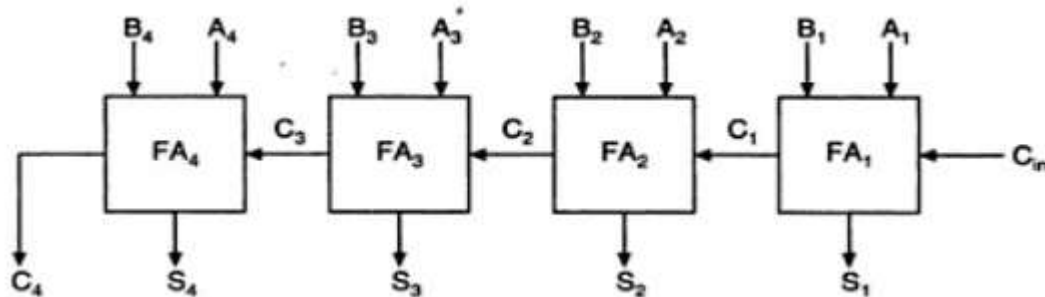


Fig 1.1: 4-bit Ripple Carry Adder Block Diagram

The above block diagram consists of four Full Adders cascaded to each other. The 'Cin' is the input carry which is applied to add the two 4-bit numbers that are $A_4A_3A_2A_1$ and $B_4B_3B_2B_1$. Each Full Adder produces intermediate carry which is C_1 , C_2 and C_3 . The sum produced is $S_4S_3S_2S_1$ and the final carry output is C_4 .

1.2 Working of 4-bit Ripple Carry Adder

- The sum of Full Adder is given as $S = A \oplus B \oplus C_{in}$ and Carry is given as $C = AB + BC_{in} + AC_{in}$. It is shown below in the truth table:

Inputs			Outputs	
A	B	Cin	Cout	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

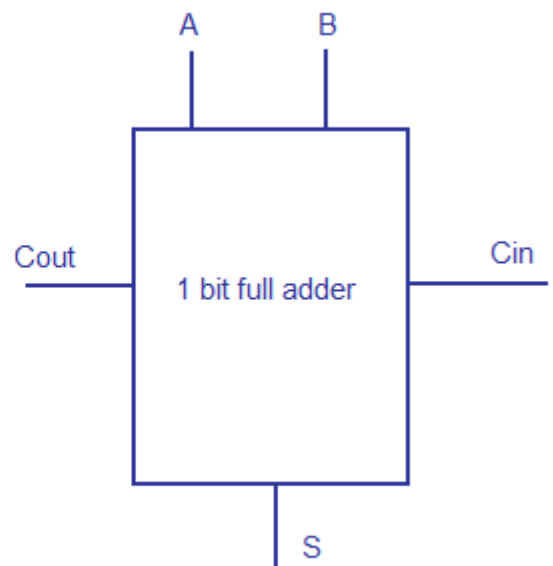


Fig 1.2: 1-bit Full Adder Truth Table & Block Diagram

- Consider two 4-bit numbers, $A_4A_3A_2A_1 = 0101$ and $B_4B_3B_2B_1 = 1010$. Let $C_{in} = 0$.
- The inputs of 1st FA are $A_1=1$ and $B_1=0$ and $C_{in}=0$. This produces sum $S_1=1$ and carry $C_1=0$.
- The inputs of 2nd FA are $A_2=0$ and $B_2=1$ and $C_1=0$. This produces sum $S_2=1$ and carry $C_2=0$.
- The inputs of 3rd FA are $A_3=1$ and $B_3=0$ and $C_2=0$. This produces sum $S_3=1$ and carry $C_3=0$.
- The inputs of 4th FA are $A_4=0$ and $B_4=1$ and $C_3=0$. This produces sum $S_4=1$ and carry $C_4=0$.
- Hence the final sum is given as $S_4S_3S_2S_1 = 1111$ and final carry is $C_4=0$.

1.3 Objective of the Report

The objective of the report is to implement a 4-bit ripple carry adder with its simulation on Vivado IDE and implementation on the *Basys 3* board. The code for the same has to be written in Verilog HDL as well as the corresponding testbench. The outputs that are Sum and Carry are to be observed on the Basys 3 board. Moreover, the inputs are to be given using VIO as well and the outputs are again to be observed on the board. The carry later is to be mapped to the carry-chains of the FPGA fabric. Moreover, analysis of delay and resource utilisation after implementation should also be carried out keeping in mind the slack timings as well . The factors which result in the delay are to be studied and an attempt should be made to reduce the slack timings.

2. Details related to the topic

2.1 Vivado

The Vivado Integrated Design Environment (IDE) offers a simple-to-use GUI and a variety of essential functionalities. Because they are all written in native tool command language (Tcl) format, all of the tools and tool settings may be used in the Vivado IDE or Vivado Design Suite Tcl shell. It is possible to do analysis and constraint assignment along the full method of design. Furthermore, there are several other features including:

- RTL design in Verilog/VHDL or System Verilog
- Behavioral, Timing and Functional simulation
- Vivado Synthesis
- Implementation of design
- Static Timing Analysis
- Detailed placement and routing modification
- Bitstream generation

Apart from above, there are several other features as well like serial I/O and logic analyzer for debugging, floorplanning, Virtual I/O, memory management etc.

2.2 Basys 3 board

The Basys 3 board can be treated as a digital electronic circuit development platform based on the Artix - 7 Field Programmable Gate Array (FPGA) by Xilinx®. The Basys 3 board serves as a host to designs such as combinational circuits as well as complex sequential circuits including processors, embedded SOCs, controllers etc. It has enough switches, LEDs, and other I/O devices to support the implementation of several designs without the need for any further hardware, as well as enough open FPGA I/O pins to enable the expansion of designs.

The pin diagram of Basys 3 board is given below:

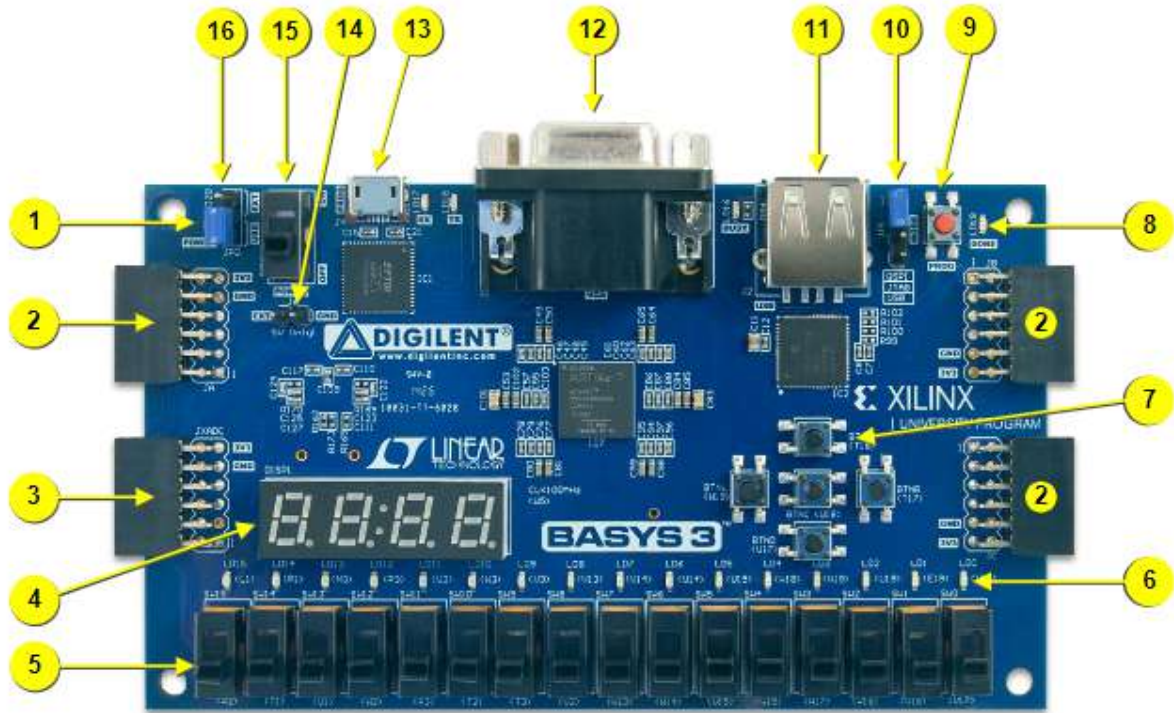


Fig 2.1: Basys 3 FPGA board with callouts

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod port(s)	10	Programming mode jumper
3	Analog signal Pmod port (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

2.3 FPGA Fabric

Carry Chain

FPGAs use a straightforward ripple carry approach to commit a portion of their circuitry to achieve the speed requirements. Some designated resources are present which are specifically

optimized for implementing carry. This is mainly used in Adder and Subtractor circuits for serving the purpose of connecting carry.

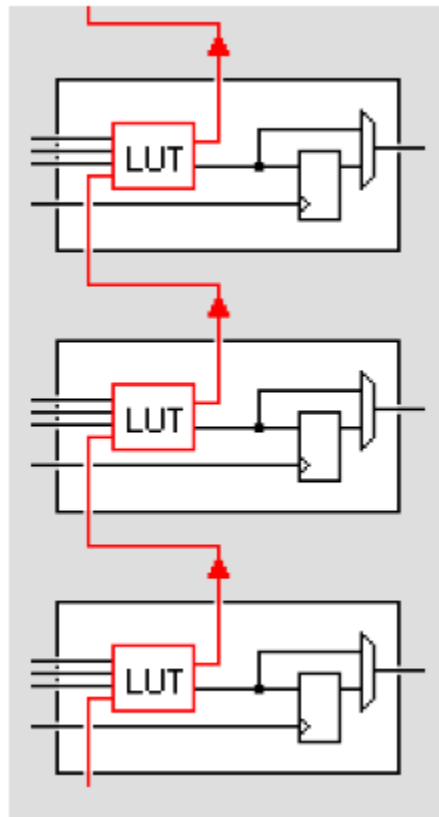


Fig 2.2: Sample Carry chain

3. Questions

Q1) Write a Verilog code for a 4-bit ripple carry adder. Simulate the ripple carry adder design using a test bench and verify that the design works --> Save this waveform in a report. The outputs sum and carry out should be observed on the LEDs on the Basys 3 board --> Save the LED output snapshot (5 marks)

A1)

Verilog Code:

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 21.08.2022 10:11:42
// Design Name:
// Module Name: four_bit_adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

// MT2022501 - Aman Prajapati
// MT2022513 - Raj Kachhadiya
// MT2022511 - Nishit Chechani
```

```

module four_bit_adder(
    input [3:0] a,    //4-bit input a
    input [3:0] b,    //4-bit input b
    input cin,       //carry in
    input clk,       //clk input for the d-flipflops
    output [3:0] sum, // 4-bit output sum
    output cout      //final carry out
);
    wire [3:0] da,db,ds; //inputs for d flipflop
    wire dci,dco;        //inputs for d flipflop
    reg [3:0] qa,qb,qs; //outputs of d flipflop
    reg qci,qco;         //outputs of d flipflop
    wire c1,c2,c3;       //internal carry that transfers from one full adder to another

    //assigning input as input of d flipflops
    assign da=a;
    assign db=b;
    assign dci=cin;
    assign sum=qs;
    assign cout=qco;

    always @(posedge clk)
    begin
        qa<=da;
        qb<=db;
        qs<=ds;
        qci<=dci;
        qco<=dco;
    end

    // instantiation of four full adders and giving carry out of one adder to carry in of the next
    adder a1(qa[0],qb[0],qci,ds[0],c1);
    adder a2(qa[1],qb[1],c1,ds[1],c2);
    adder a3(qa[2],qb[2],c2,ds[2],c3);
    adder a4(qa[3],qb[3],c3,ds[3],dco);

endmodule

```

```
// definition of 1-bit full adder that is instantiated in the top module.

module adder (x,y,c,s,co);
input x,y,c;
output s;
output co;

assign {co,s} = x+y+c; // concatenation as the output can get overflow.

endmodule
```

Testbench:

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 21.08.2022 10:31:28
// Design Name:
// Module Name: tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////

module tb();
```

```

reg [3:0]a,b;
reg cin;
reg clk;

wire[3:0] sum;
wire cout;

four_bit_adder fl(a, b, cin, clk, sum, cout);

always #18 clk=~clk;

initial begin
clk=1'b0;a=4'b0000; b=4'b0000; cin=1'b0;
#100 a=4'b0001; b=4'b0001; cin=1'b0;
#100 a=4'b0010; b=4'b0101; cin=1'b1;
#100 a=4'b1101; b=4'b1001; cin=1'b1;
#100 a=4'b1001; b=4'b1101; cin=1'b0;
#100 a=4'b1101; b=4'b0011; cin=1'b0;
#100 a=4'b1011; b=4'b1100; cin=1'b1;
#100 a=4'b1101; b=4'b1000; cin=1'b1;
#100 a=4'b1001; b=4'b0101; cin=1'b0;
#100 a=4'b0101; b=4'b0011; cin=1'b1;
#100 a=4'b1111; b=4'b1111; cin=1'b0;
$finish;
end

endmodule

```

RTL Simulation:

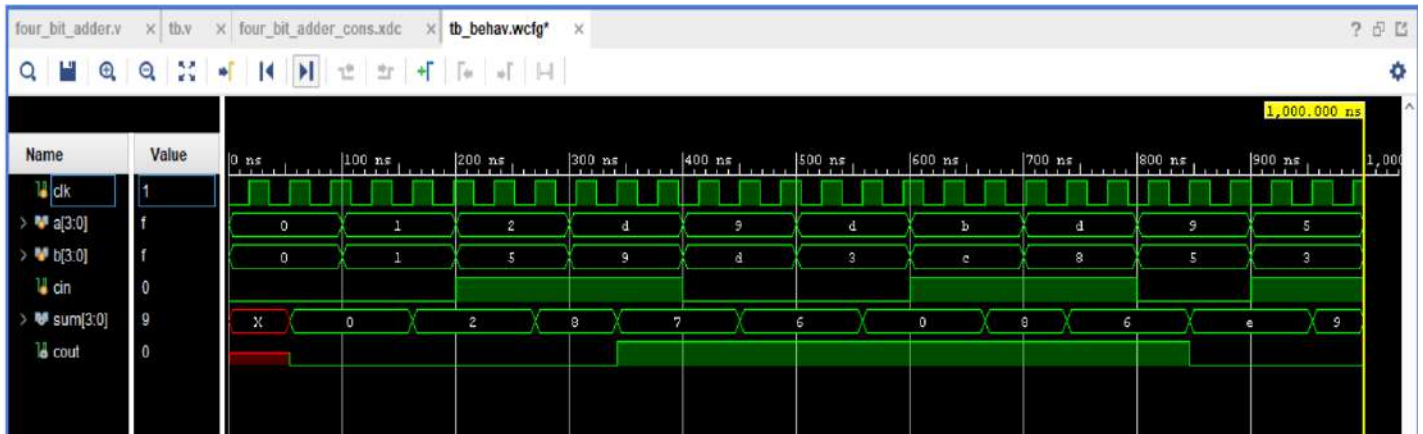


Fig 3.1: RTL Simulatoion of 4-bit RCA

Schematic:

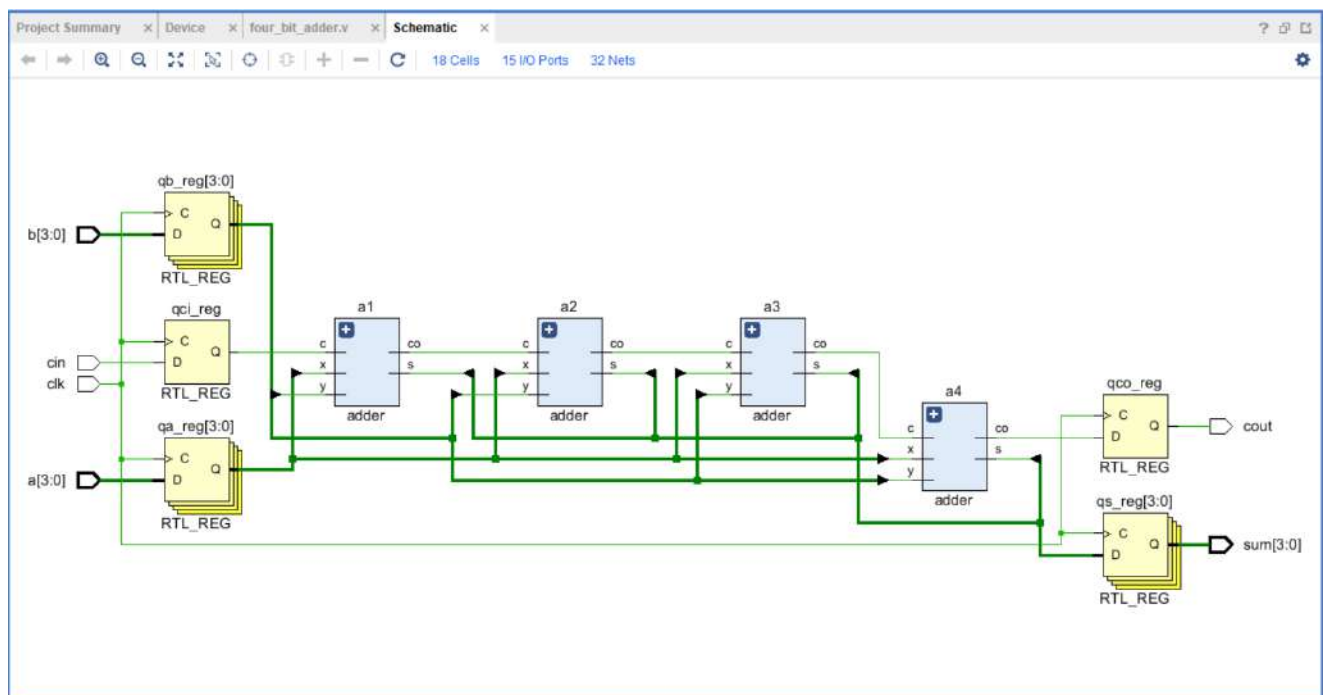


Fig 3.2: Schematic of 4-bit RCA

Output:

1. $A = 0101$; $B = 0011$; $C_{in} = 1$

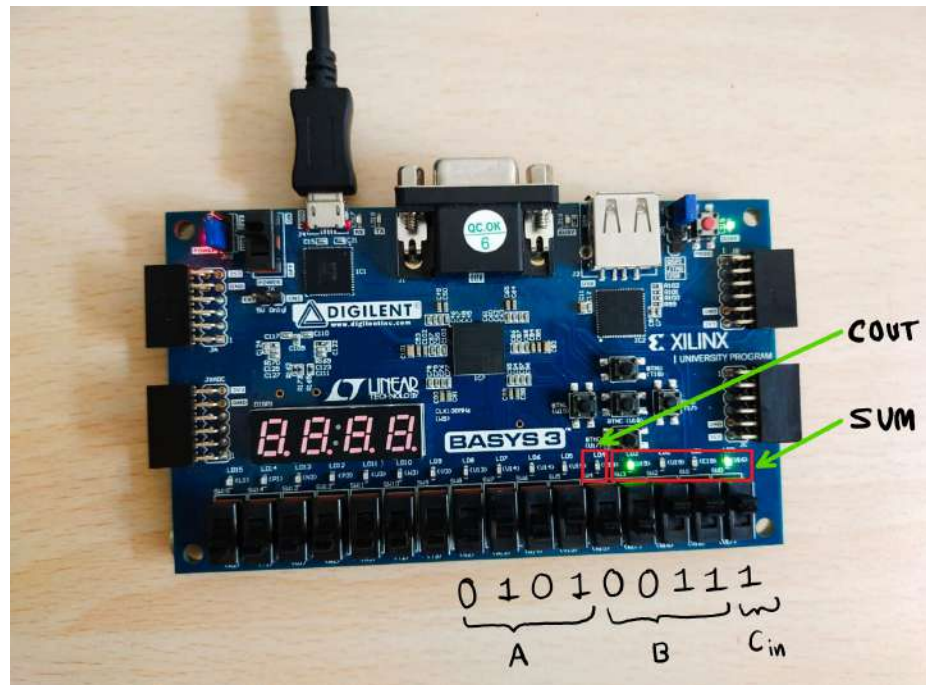


Fig 3.3: LED Output 1

2. $A = 1001$; $B = 0101$; $C_{in} = 0$

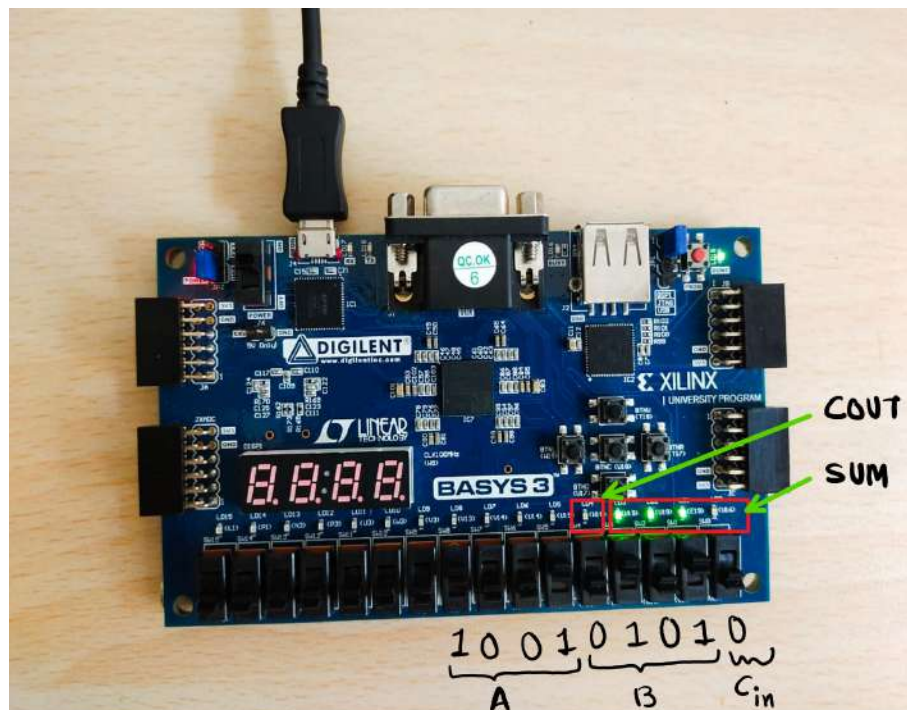


Fig 3.4: LED Output 2

3. $A = 1101$; $B = 1000$; $C_{in} = 1$

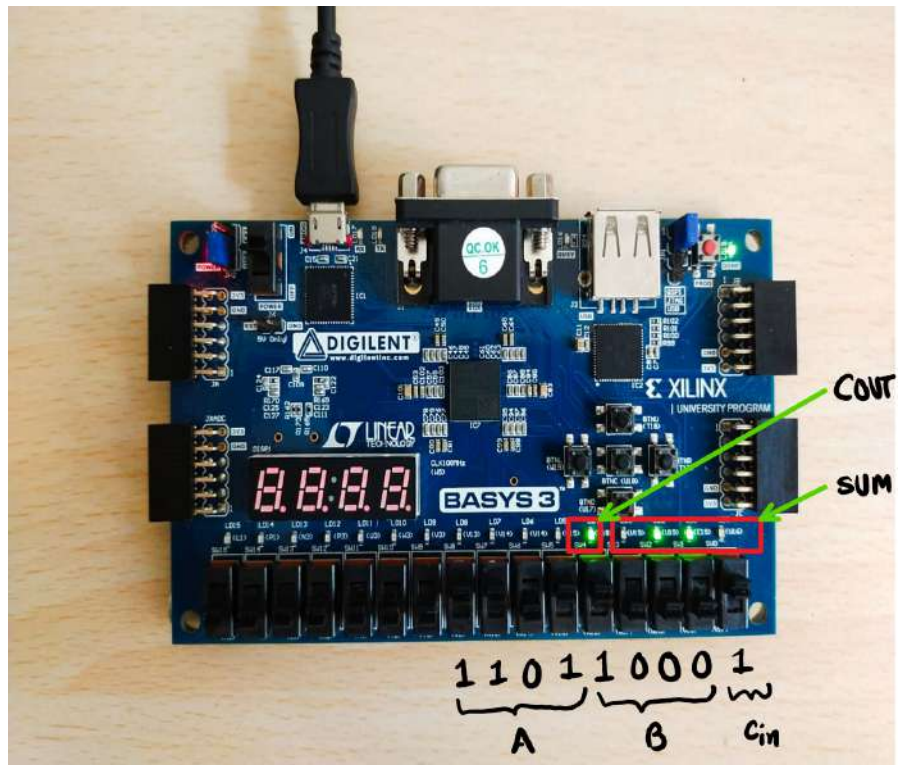


Fig 3.5: LED Output 3

4. $A = 1011$; $B = 1100$; $C_{in} = 1$

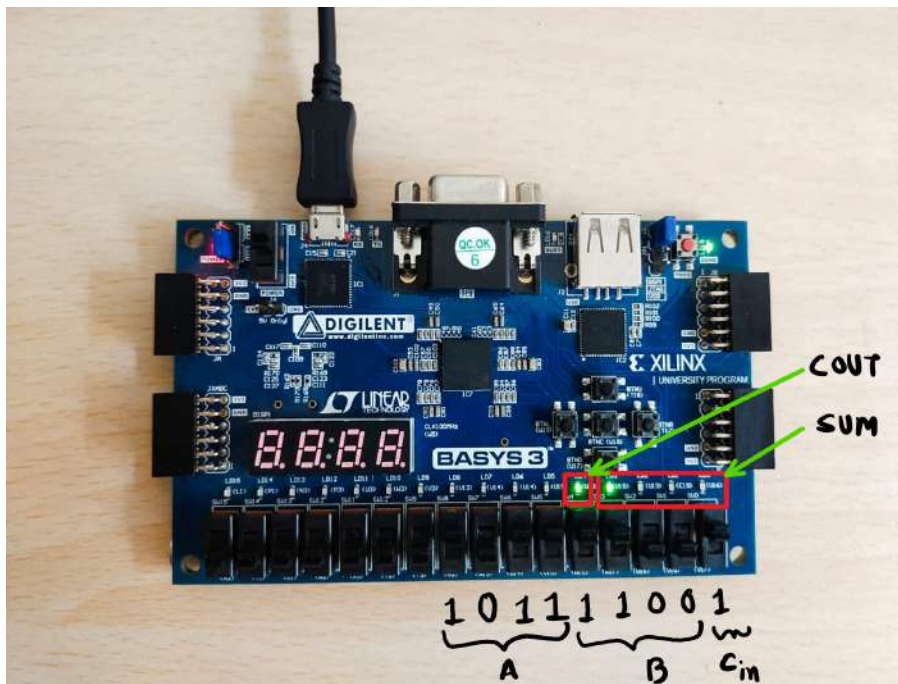


Fig 3.6: LED Output 4

5. $A = 1101$; $B = 0011$; $C_{in} = 0$

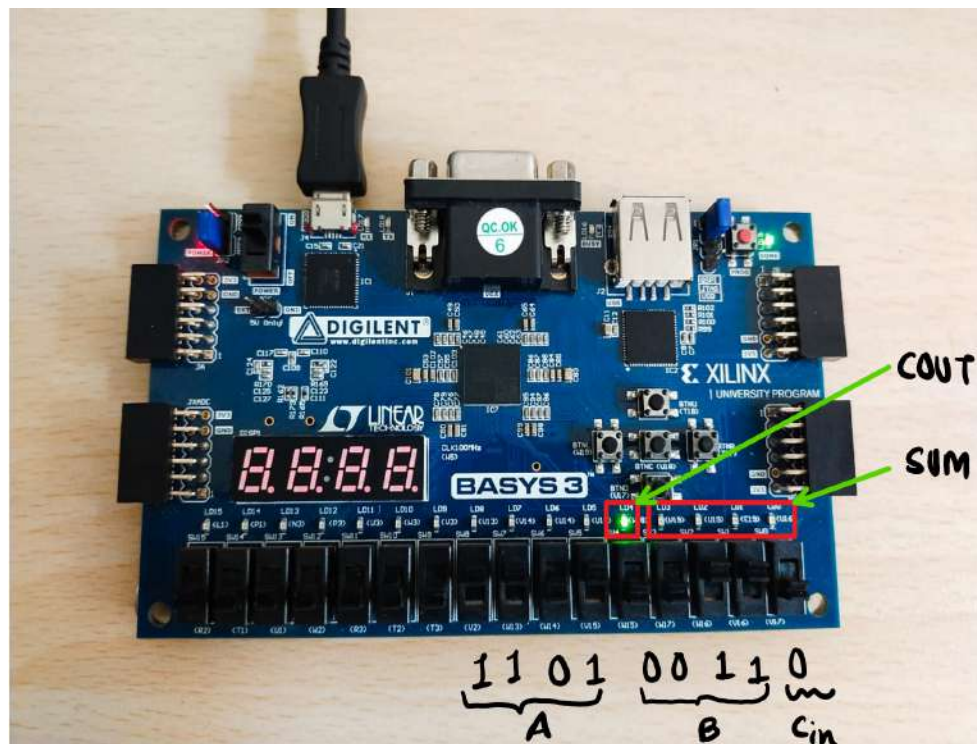


Fig 3.7: LED Output 5

6. $A = 1001$; $B = 1101$; $C_{in} = 0$

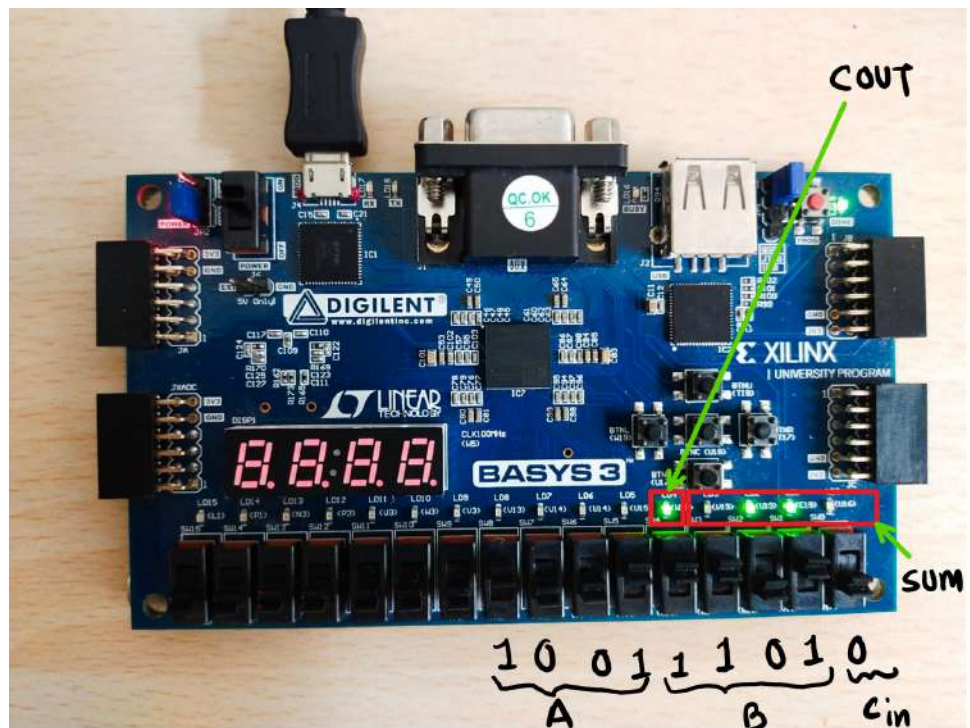


Fig 3.8: LED Output 6

7. $A = 1101$; $B = 1001$; $C_{in} = 1$

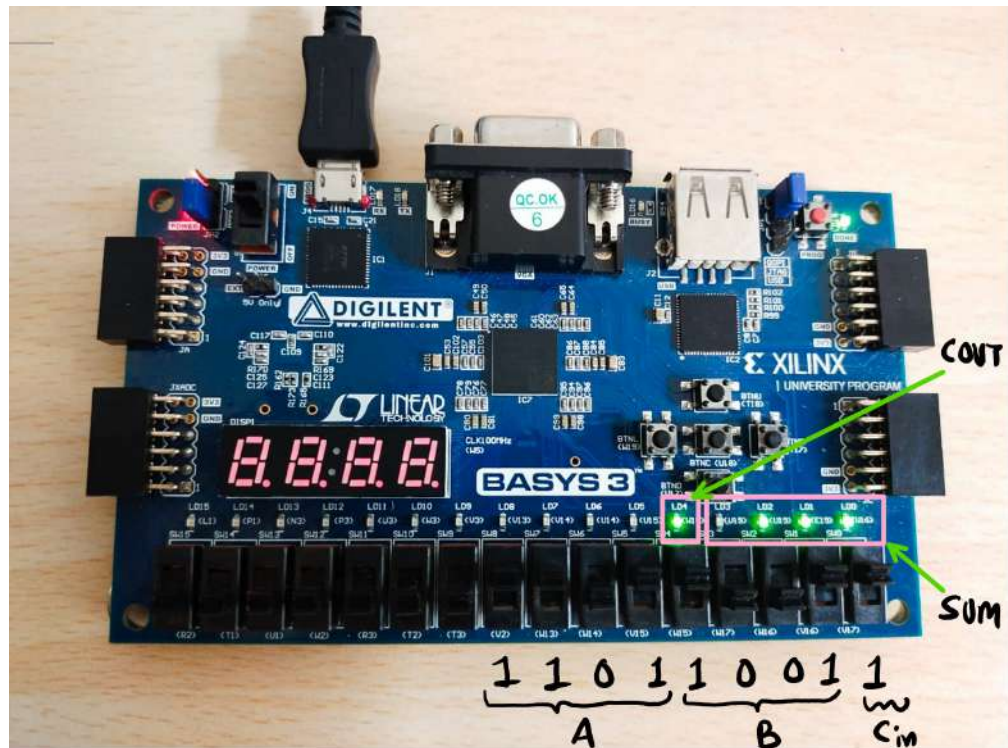


Fig 3.9: LED Output 7

8. $A = 1101$; $B = 0010$; $C_{in} = 1$

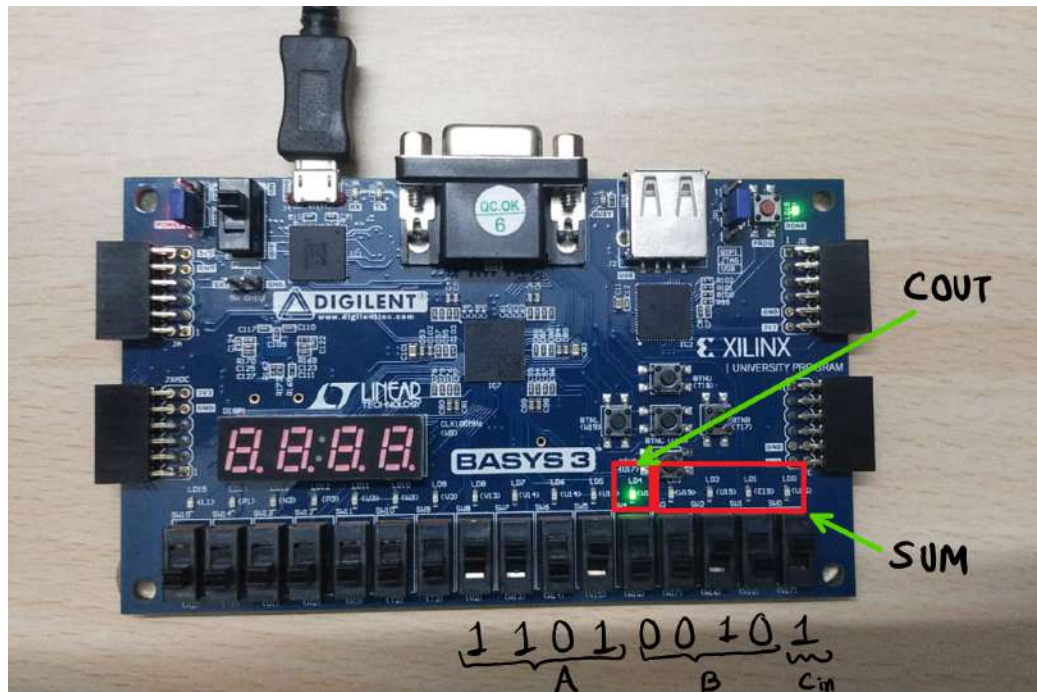


Fig 3.10: LED Output 8

9. $A = 0001$; $B = 0001$; $C_{in} = 0$

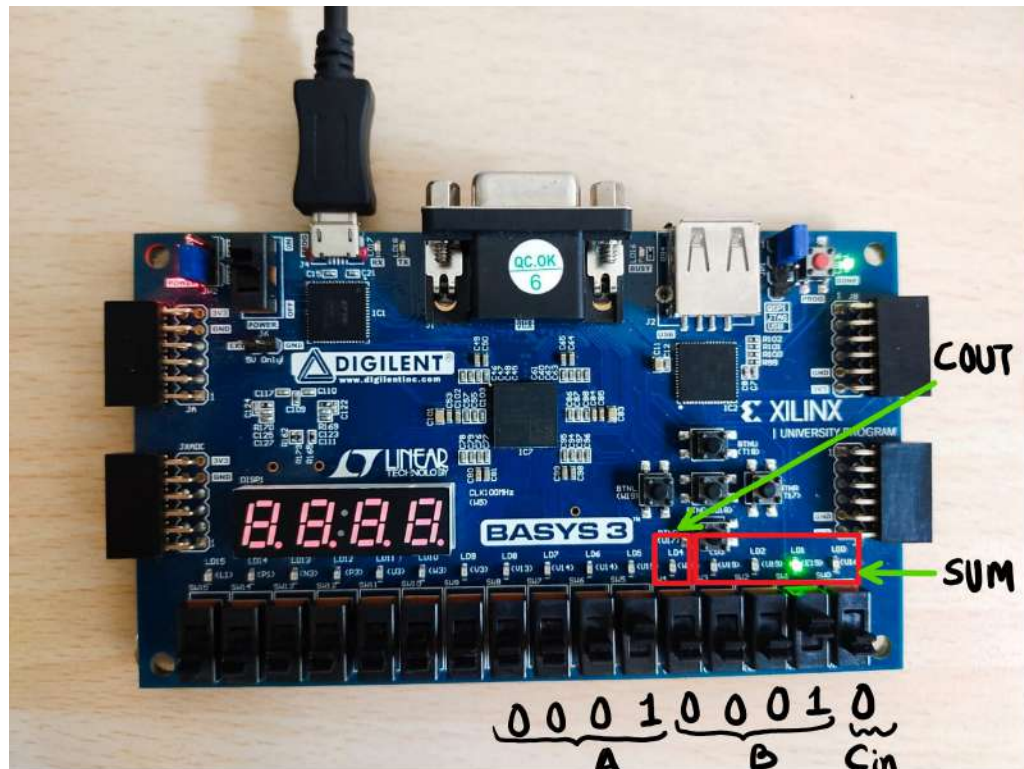


Fig 3.11: LED Output 9

10. $A = 0000$; $B = 0000$; $C_{in} = 0$

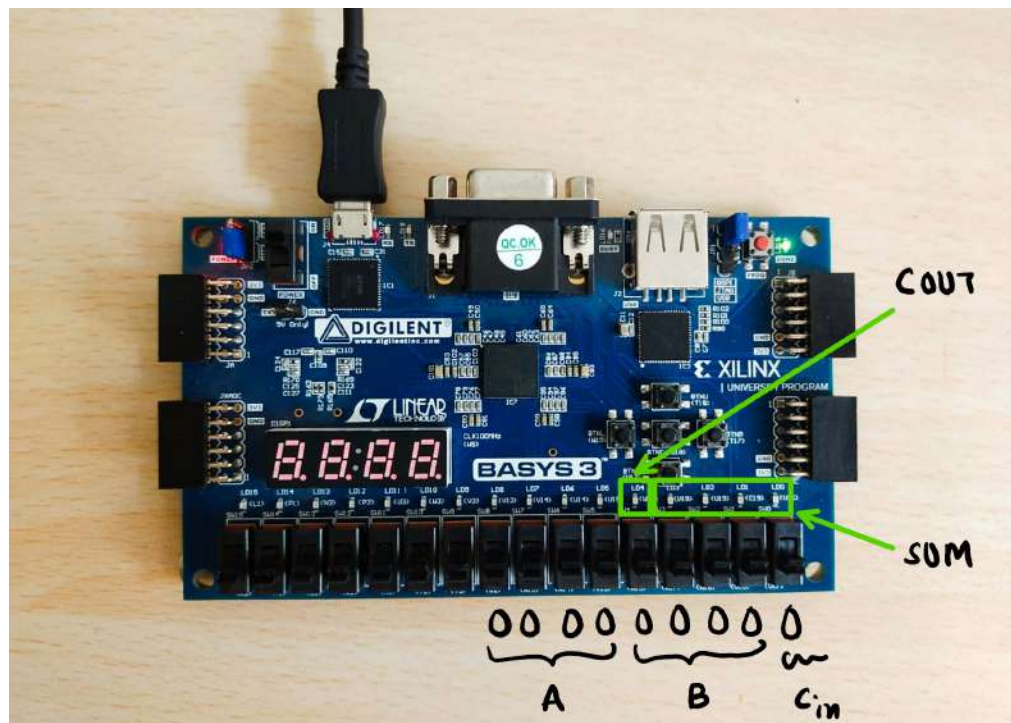


Fig 3.12: LED Output 10

Q2) Assign 'a', 'b' and "cin" from a VIO. Output should be observed on the LEDs on FPGA. Provide snapshots (5 marks)

A2)

1. VIO input: 5+3+1

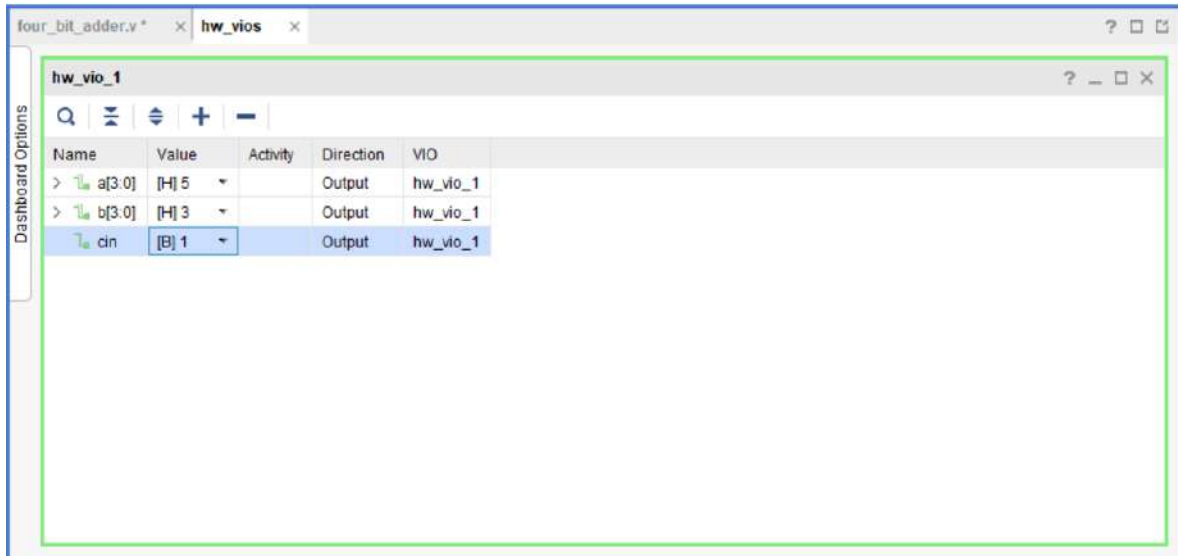


Fig 3.13: VIO Input 1

LED Output: Cout=0, Sum=9

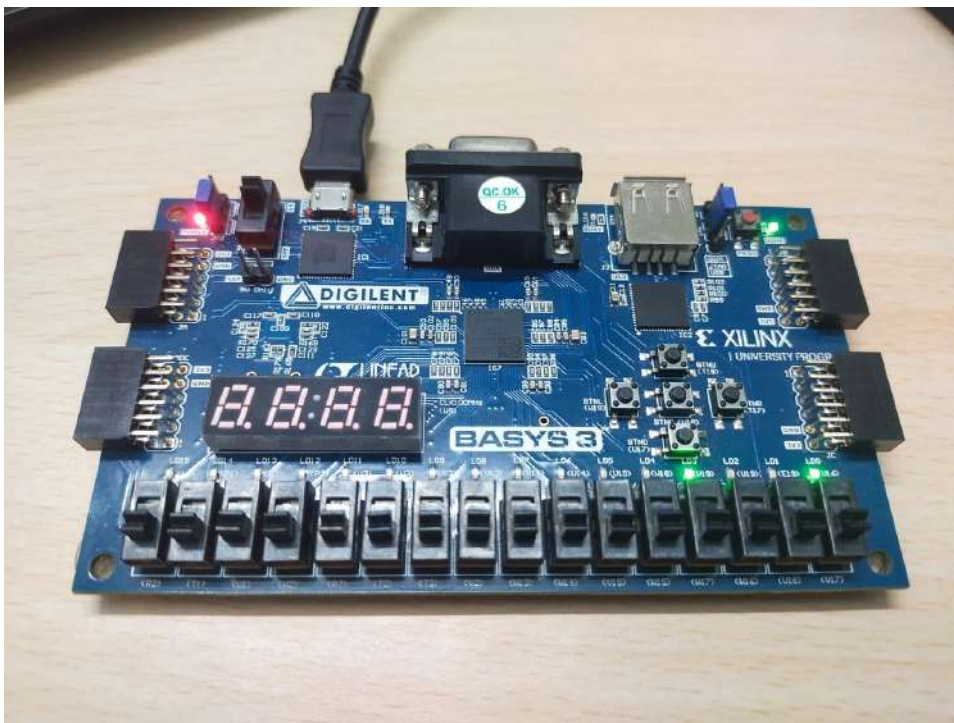


Fig 3.14: VIO - LED Output 1

2. VIO input: $9+5+0$

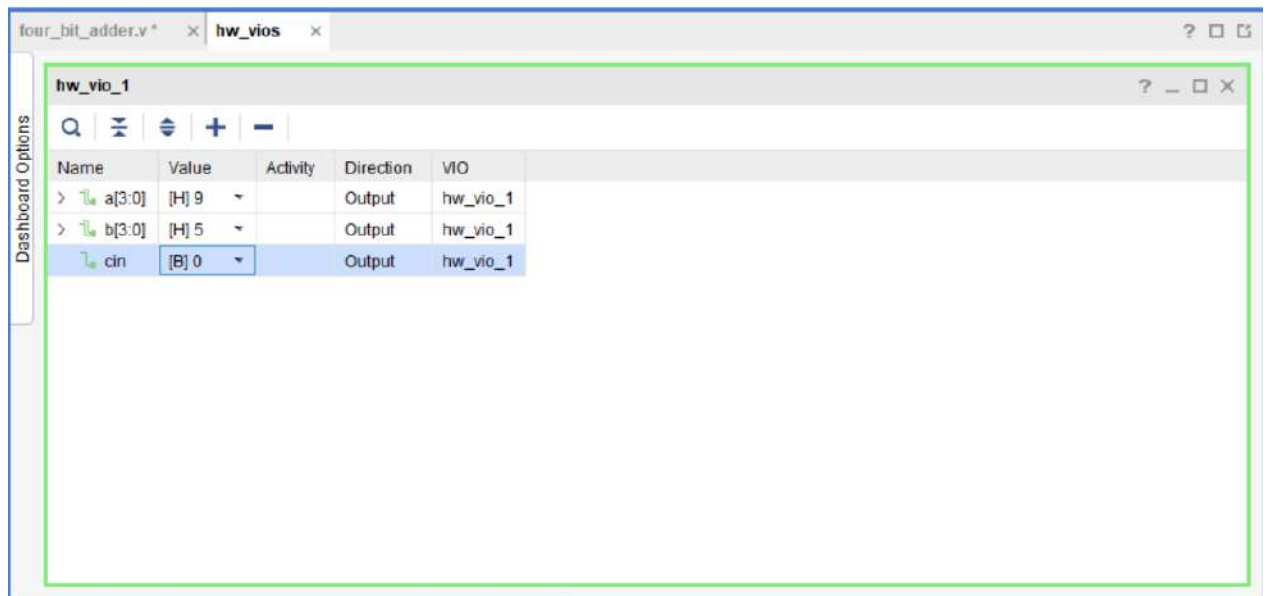


Fig 3.15: VIO Input 2

LED output: Cout=0, Sum=14

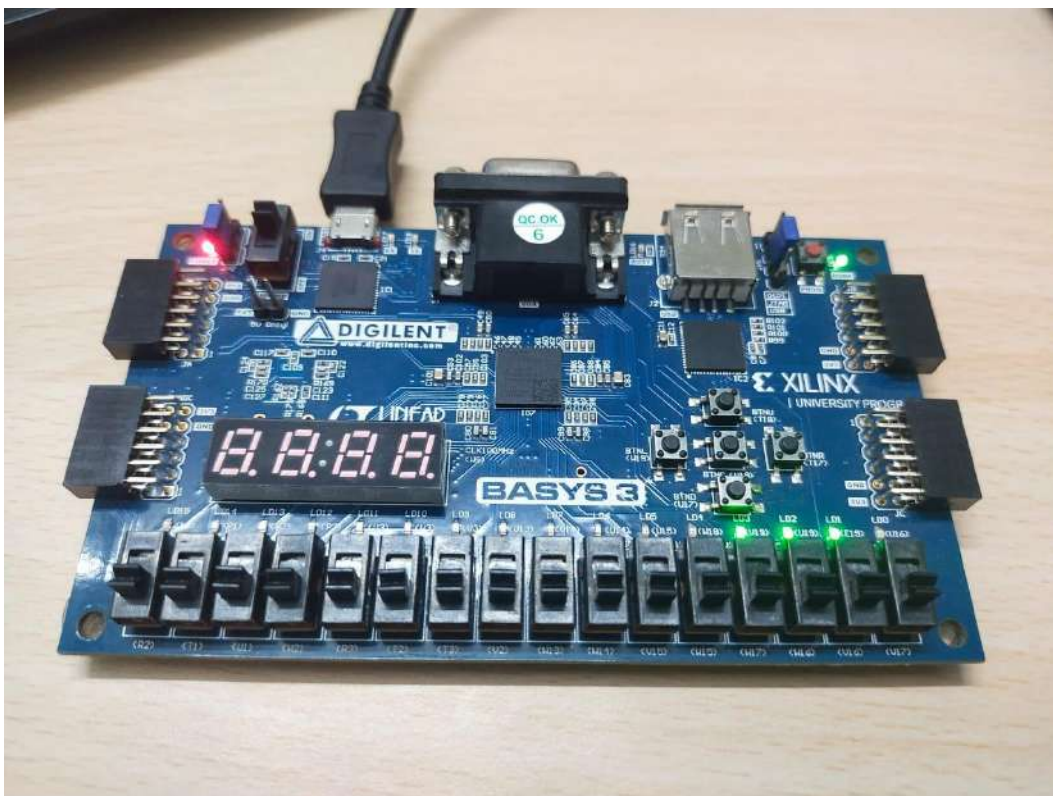


Fig 3.16: VIO - LED Output 2

3. VIO input: $D+8+1$

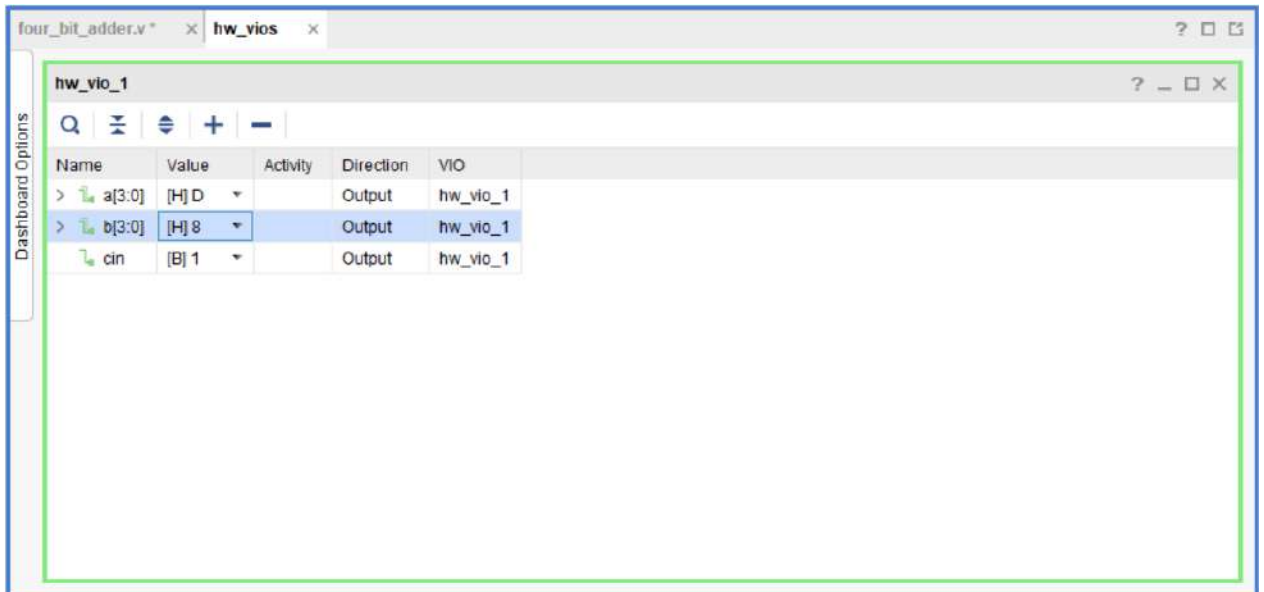


Fig 3.17: VIO Input 3

LED output: Cout=1, Sum=6

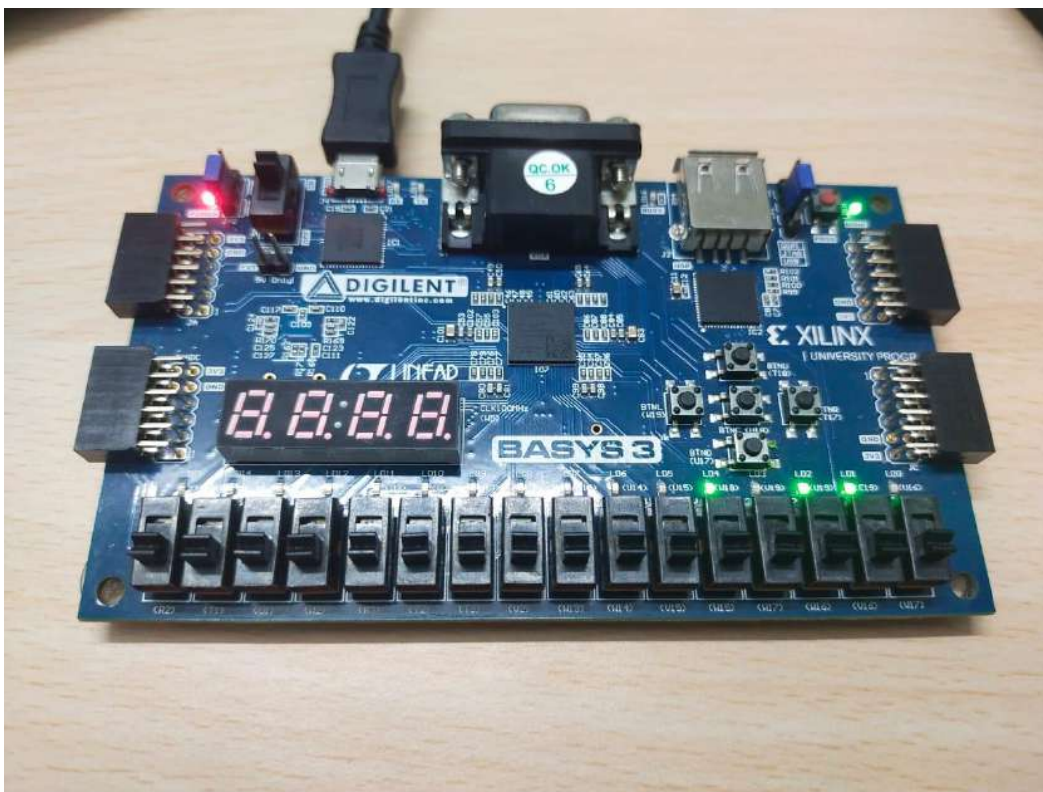


Fig 3.18: VIO - LED Output 3

4. VIO input: $B+C+1$

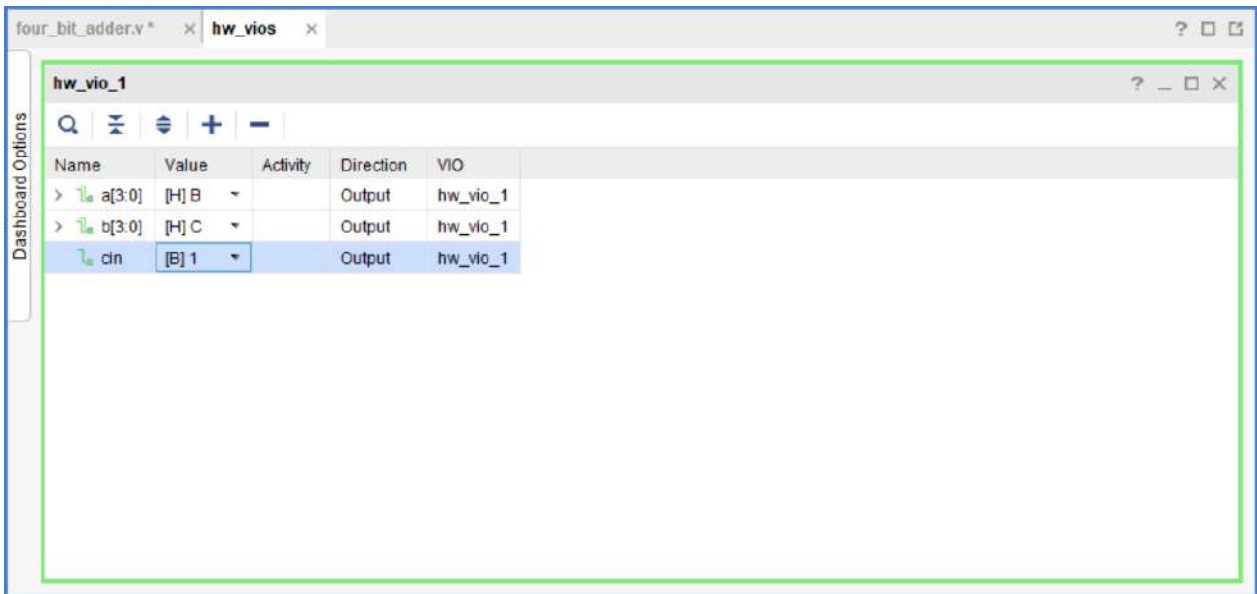


Fig 3.19: VIO Input 4

LED output: Cout=1, Sum=8

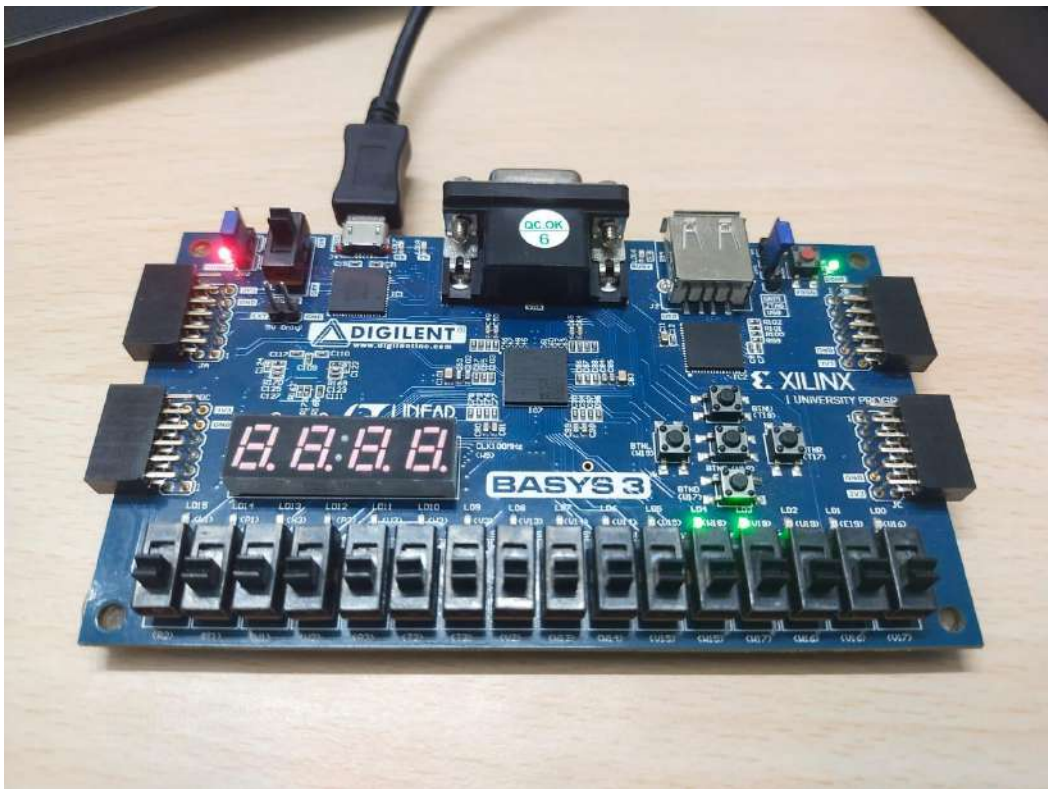
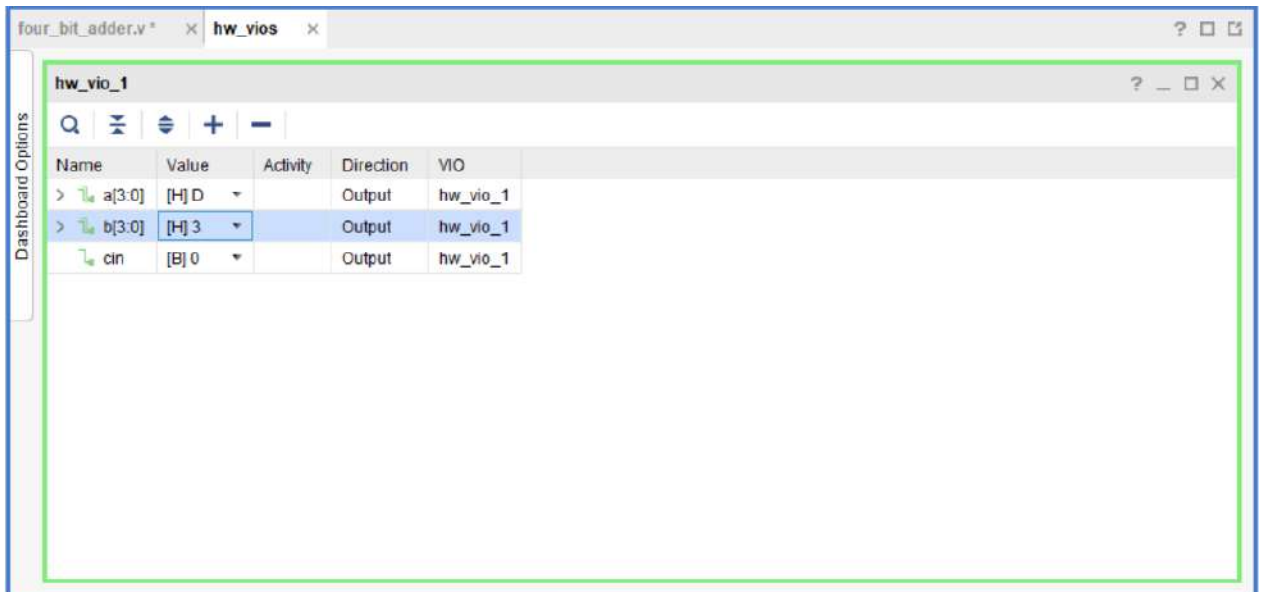


Fig 3.20: VIO - LED Output 4

5. VIO input: $D+3+0$



Name	Value	Activity	Direction	VIO
> a[3:0]	[H] D		Output	hw_vio_1
> b[3:0]	[H] 3		Output	hw_vio_1
cin	[B] 0		Output	hw_vio_1

Fig 3.21: VIO Input 5

LED output: Cout=1, Sum=0

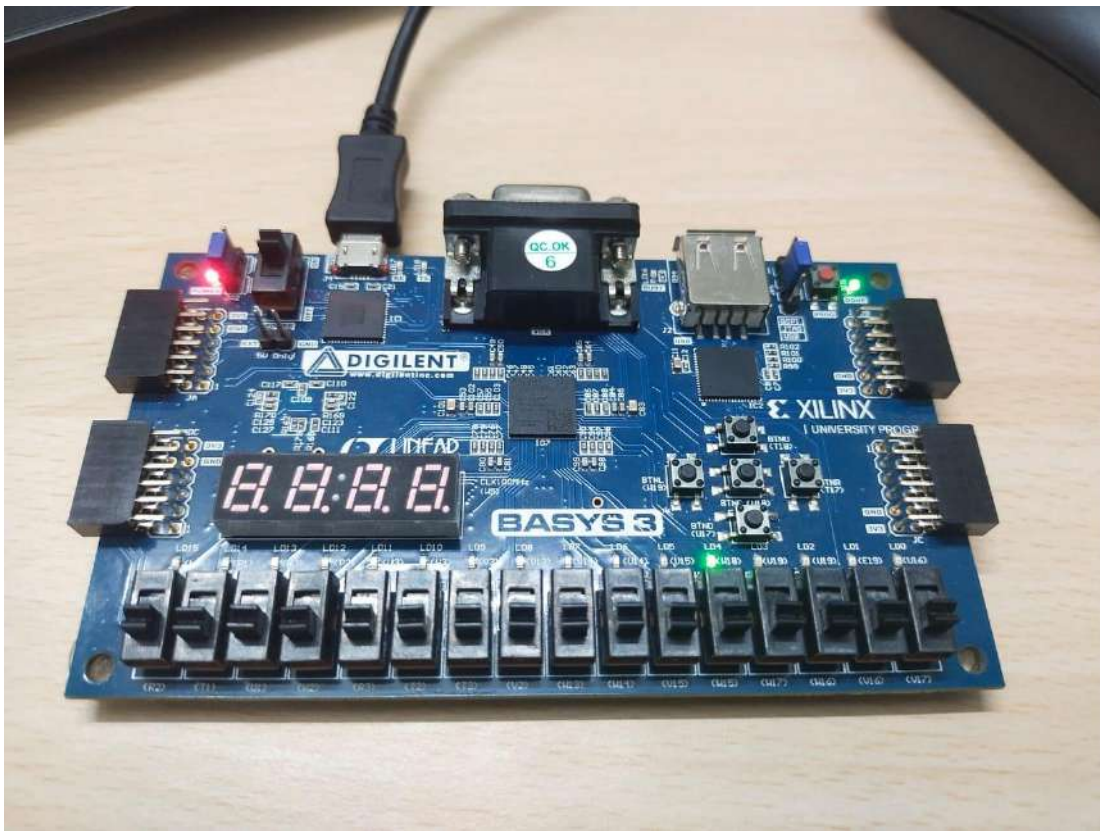


Fig 3.22: VIO - LED Output 5

6. VIO input: $9+D+0$

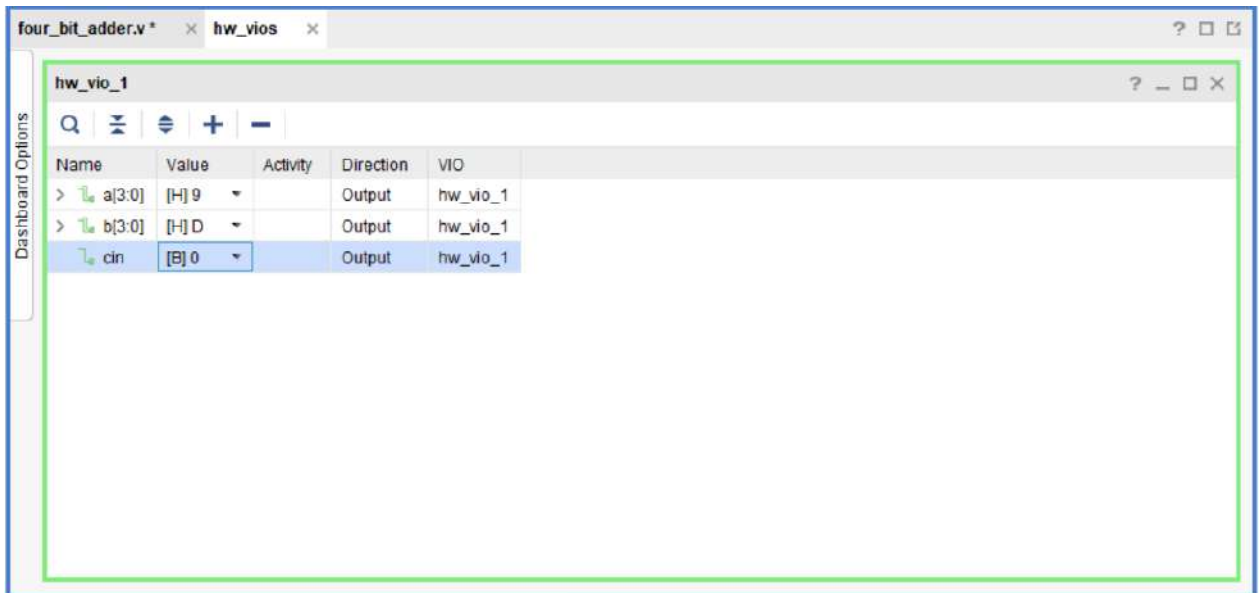


Fig 3.23: VIO Input 6

LED output: Cout=1, Sum= 6

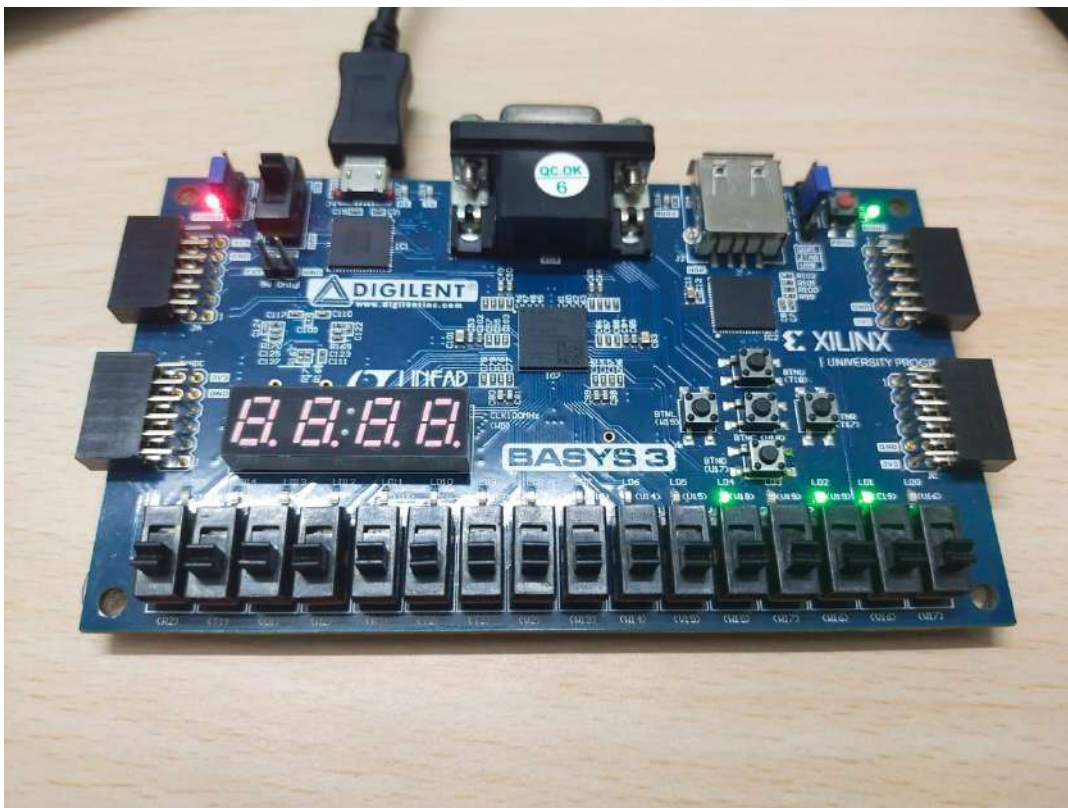
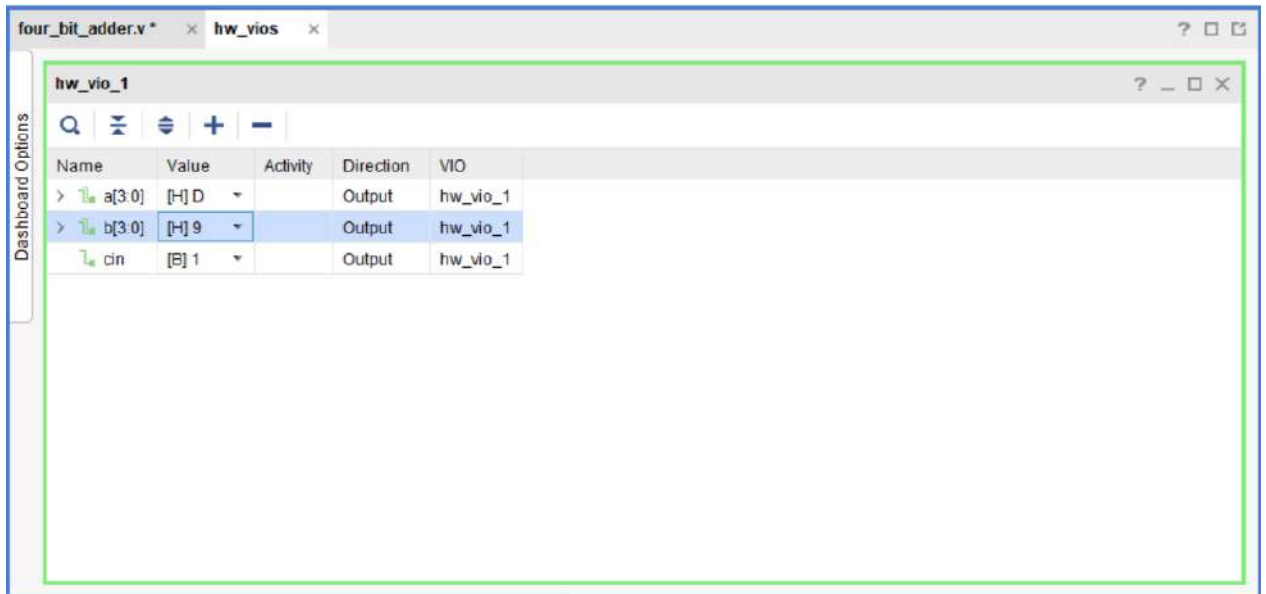


Fig 3.24: VIO - LED Output 6

7. VIO input: D+9+1



Name	Value	Activity	Direction	VIO
> 1. a[3:0]	[H] D		Output	hw_vio_1
> 1. b[3:0]	[H] 9		Output	hw_vio_1
1. cin	[B] 1		Output	hw_vio_1

Fig 3.25: VIO Input 7

LED Output: Cout=1, Sum=7

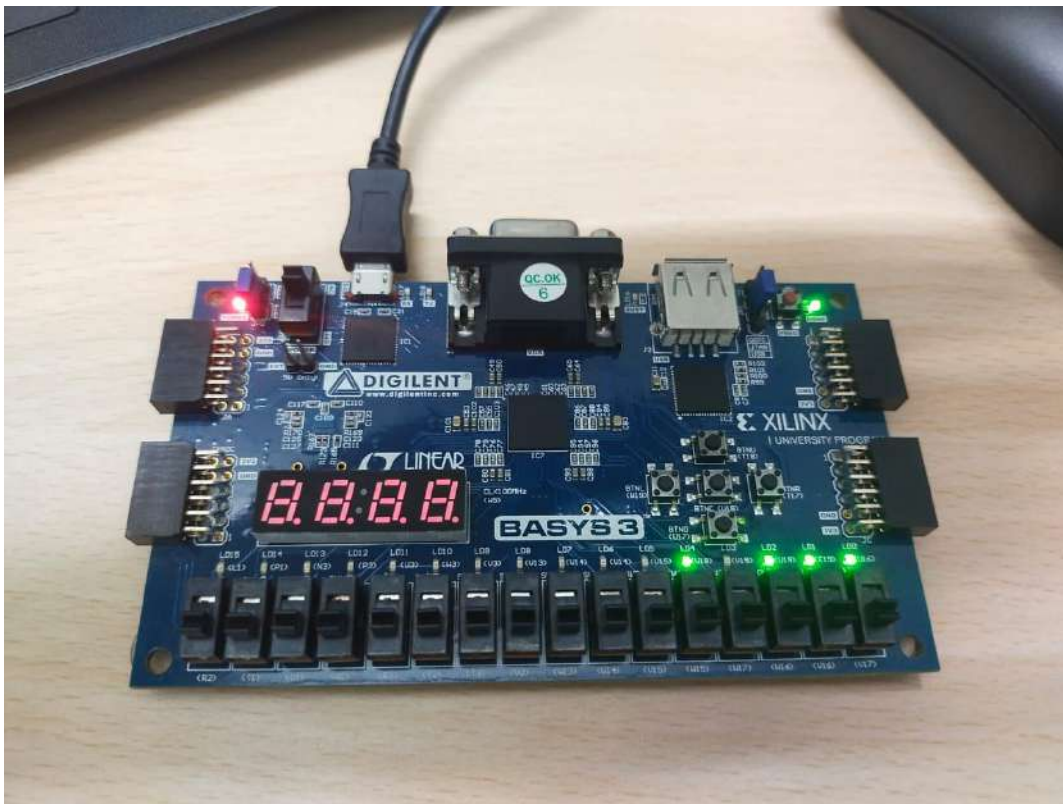


Fig 3.26: VIO - LED Output 7

8. VIO input: $D+2+1$

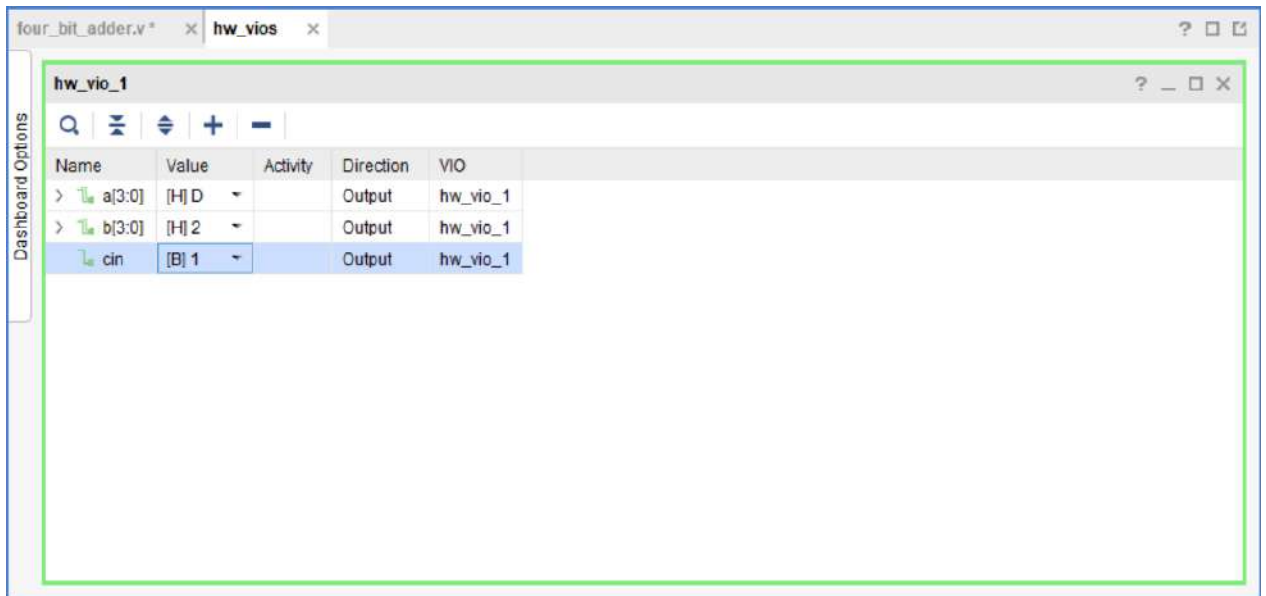


Fig 3.27: VIO Input 8

LED output: Cout=1, Sum=0

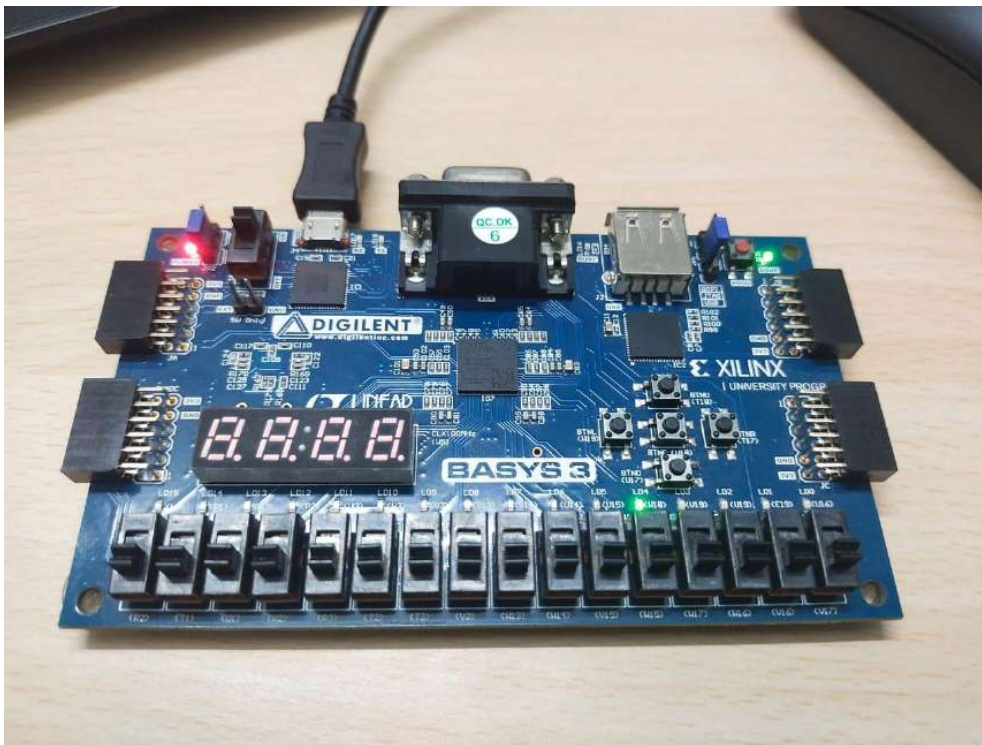


Fig 3.28: VIO - LED Output 8

9. VIO input: 1+1+0

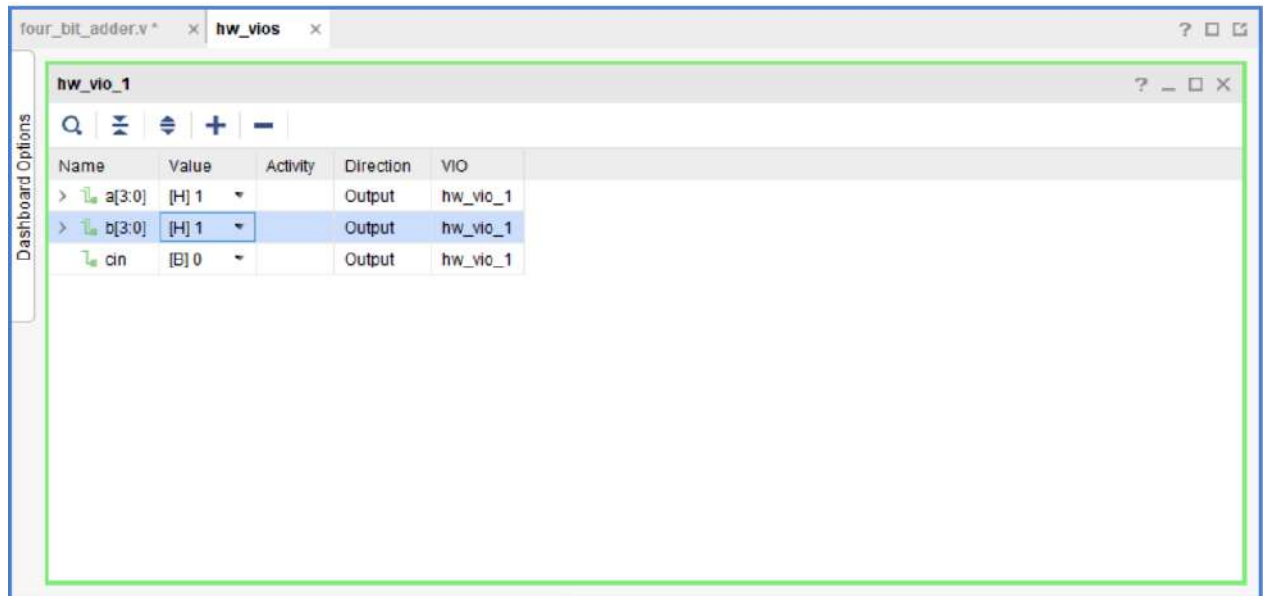


Fig 3.29: VIO Input 9

LED Output: Cout=0, Sum=2

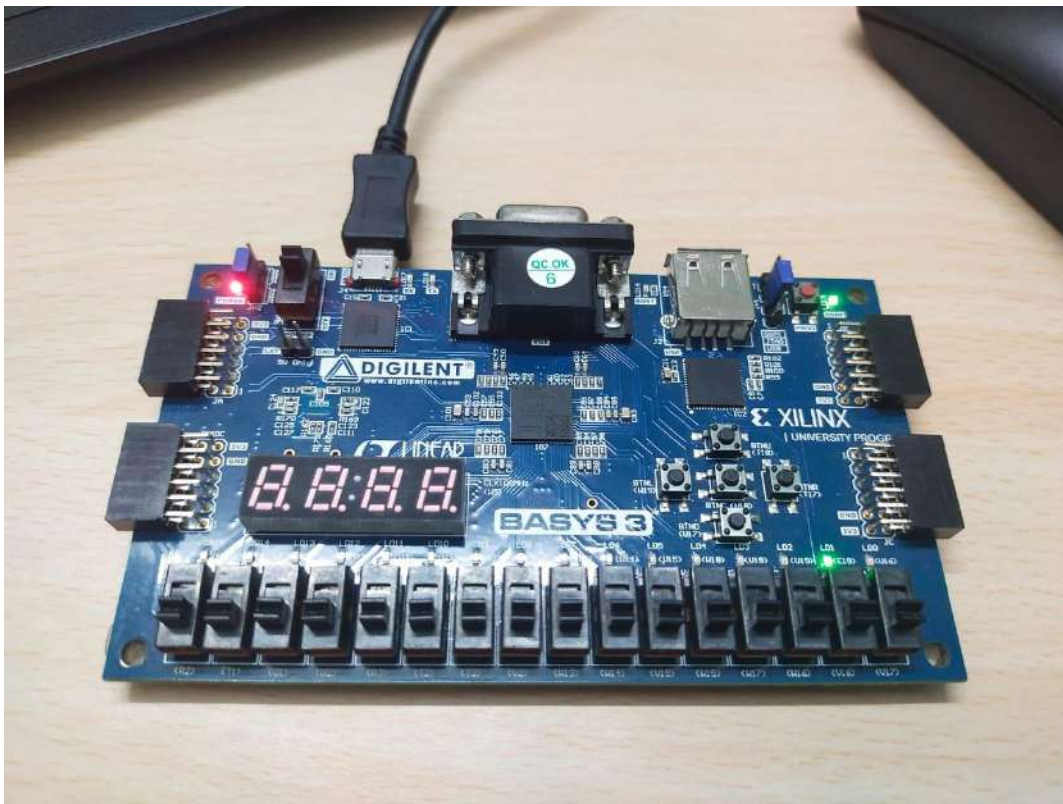


Fig 3.30: VIO - LED Output 9

10. VIO input: 0+0+0

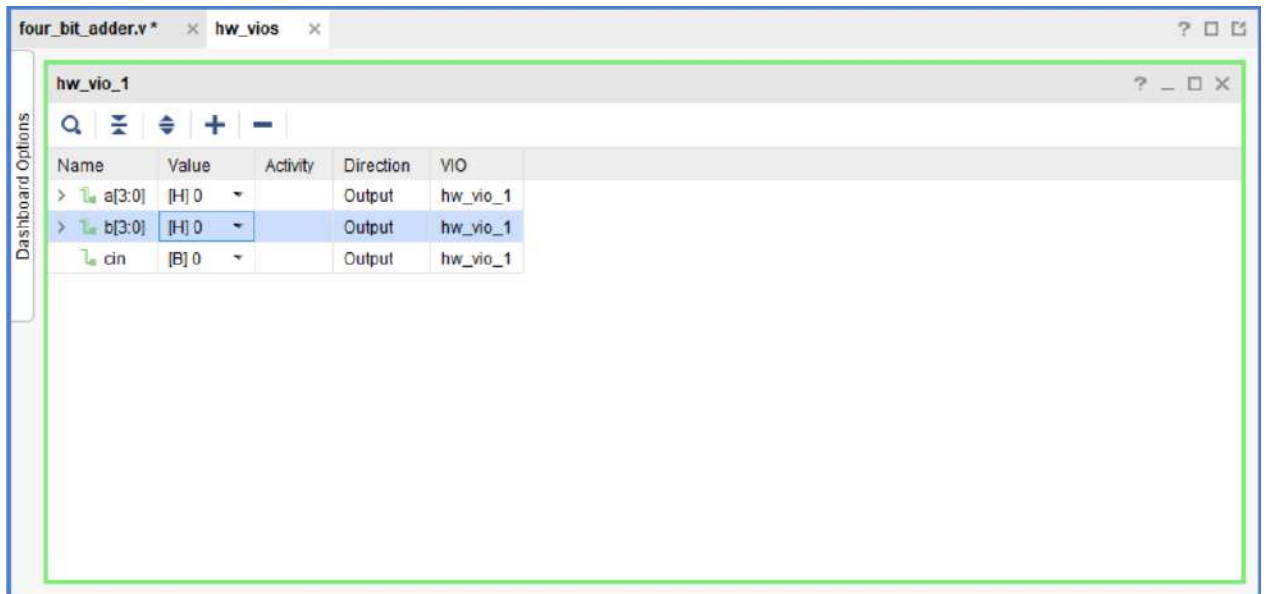


Fig 3.31: VIO Input 10

LED Output: Cout=0, Sum=0

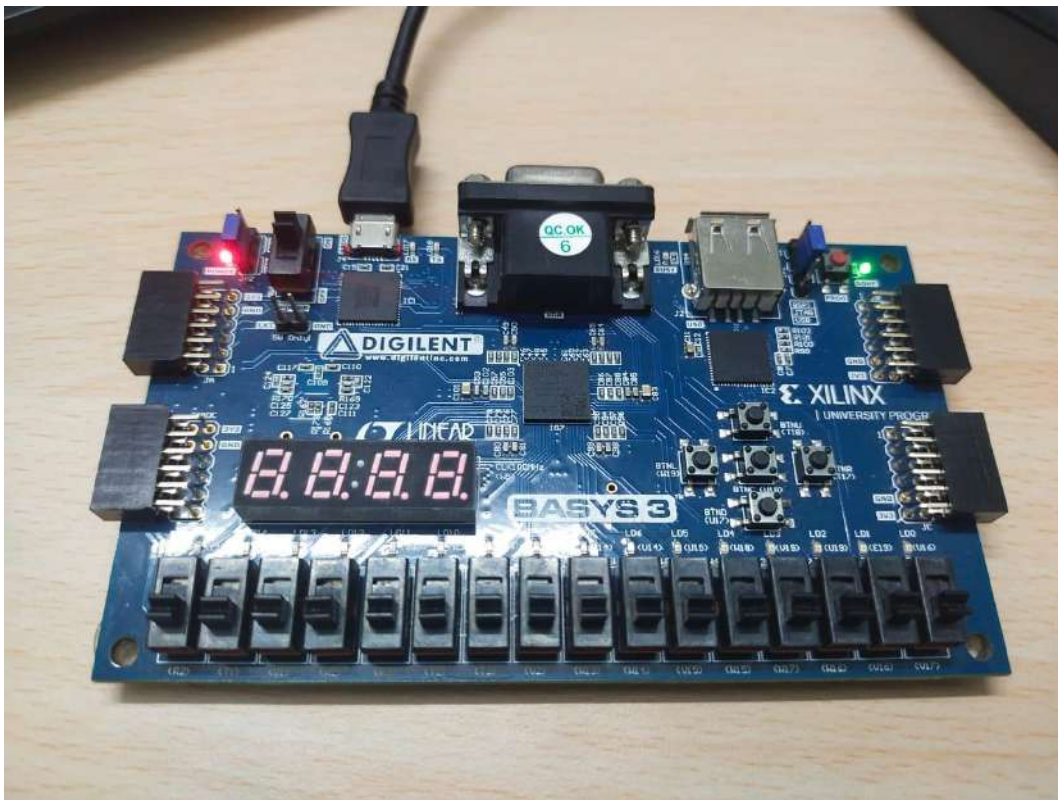


Fig 3.32: VIO - LED Output 10

Q3) Find out if the carry is mapped to the carry-chains of the FPGA fabric. (5 marks) - Provide a snapshot of the Floorplan containing the resource on which the carry is getting mapped - Note the clock region on to which the adders are getting mapped

A3)



Fig 3.33: Carry-Chain Mapping

The entire carry is marked in RED color, also the Cin and Cout can be observed clearly.

The clock regions onto which carry is mapped are :

- Adder1 and Adder2 to X0Y0 clock region.
- Adder 3 and Adder 4 are mapped to X1Y0 clock region

Q4) Delay and resource utilisation after implementation: (5) - What is the logic delay and wire delay? - Make an attempt to reduce either of the delays. Describe the process you followed. - Write the resource utilisation in the report

A4)

Before Changes:

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type
b(4)	IN			<input checked="" type="checkbox"/>	14	LVCNMOS33*	3.300				NONE
sum(4)	OUT			<input checked="" type="checkbox"/>	14	LVCNMOS33*	3.300		12	SLOW	NONE
sum[3]	OUT		V19	<input checked="" type="checkbox"/>	14	LVCNMOS33*	3.300		12	SLOW	NONE
sum[2]	OUT		U19	<input checked="" type="checkbox"/>	14	LVCNMOS33*	3.300		12	SLOW	NONE
sum[1]	OUT		E19	<input checked="" type="checkbox"/>	14	LVCNMOS33*	3.300		12	SLOW	NONE
sum[0]	OUT		U16	<input checked="" type="checkbox"/>	14	LVCNMOS33*	3.300		12	SLOW	NONE

Fig 3.34: Ports before changes

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	3.740	1	1	qs_reg[1]/C	sum[1]	6.094	3.986	2.108	15.0	clk	clk
Path 2	3.846	1	1	qs_reg[3]/C	sum[3]	5.978	4.103	1.875	15.0	clk	clk
Path 3	3.971	1	1	qco_reg/C	cout	5.853	3.965	1.888	15.0	clk	clk
Path 4	3.998	1	1	qs_reg[2]/C	sum[2]	5.826	3.957	1.869	15.0	clk	clk
Path 5	4.002	1	1	qs_reg[0]/C	sum[0]	5.820	3.961	1.860	15.0	clk	clk
Path 6	11.727	1	3	qa_reg[0]/C	qs_reg[1]/D	2.919	0.608	2.311	15.0	clk	clk
Path 7	12.224	2	3	qa_reg[0]/C	qco_reg/D	2.742	0.704	2.038	15.0	clk	clk
Path 8	12.244	2	3	qa_reg[0]/C	qs_reg[3]/D	2.768	0.730	2.038	15.0	clk	clk
Path 9	12.362	2	3	qa_reg[0]/C	qs_reg[2]/D	2.606	0.704	1.902	15.0	clk	clk
Path 10	13.594	1	3	qa_reg[0]/C	qs_reg[0]/D	1.376	0.580	0.796	15.0	clk	clk

Fig 3.35: Timing before changes

After Changes:

All ports (15)														
> a (4)	IN				<input checked="" type="checkbox"/>	(Multiple)	LVC MOS33*	3.300					NONE	▼
> b (4)	IN				<input checked="" type="checkbox"/>	14	LVC MOS33*	3.300					NONE	▼
▼ sum (4)	OUT				<input checked="" type="checkbox"/>	14	LVC MOS33*	3.300	12	▼	SLOW	▼	NONE	▼
sum[3]	OUT	V19	▼	<input checked="" type="checkbox"/>		14	LVC MOS33*	3.300	12	▼	SLOW	▼	NONE	▼
sum[2]	OUT	U19	▼	<input checked="" type="checkbox"/>		14	LVC MOS33*	3.300	12	▼	SLOW	▼	NONE	▼
sum[1]	OUT	U15	▼	<input checked="" type="checkbox"/>		14	LVC MOS33*	3.300	12	▼	SLOW	▼	NONE	▼
sum[0]	OUT	U16	▼	<input checked="" type="checkbox"/>		14	LVC MOS33*	3.300	12	▼	SLOW	▼	NONE	▼
Scalar ports (3)														
cin	IN	V17	▼	<input checked="" type="checkbox"/>		14	LVC MOS33*	3.300					NONE	▼
clk	IN	W5	▼	<input checked="" type="checkbox"/>		34	LVC MOS33*	3.300					NONE	▼
cout	OUT	W18	▼	<input checked="" type="checkbox"/>		14	LVC MOS33*	3.300	12	▼	SLOW	▼	NONE	▼

Fig 3.36: Ports after changes

Tcl Console	Messages	Log	Reports	Design Runs	Timing	Power	Methodology	DRC	I/O Ports	?	—	□
Intra-Clock Paths - clk - Setup												
General Information	Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Timer Settings	Path 1	3.846	1	1	qs_reg[3]/C	sum[3]	5.978	4.103	1.875	15.0	clk	clk
Design Timing Summary	Path 2	3.971	1	1	qco_reg/C	cout	5.853	3.965	1.888	15.0	clk	clk
Clock Summary (2)	Path 3	3.998	1	1	qs_reg[2]/C	sum[2]	5.826	3.957	1.869	15.0	clk	clk
Check Timing (0)	Path 4	4.002	1	1	qs_reg[0]/C	sum[0]	5.820	3.961	1.860	15.0	clk	clk
▼ Intra-Clock Paths	Path 5	4.316	1	1	qs_reg[1]/C	sum[1]	5.504	3.970	1.533	15.0	clk	clk
▼ clk	Path 6	12.324	1	3	qb_reg[0]/C	qs_reg[1]/D	2.351	0.608	1.743	15.0	clk	clk
Setup 3.846 ns (10)	Path 7	12.375	1	1	b[2]	qb_reg[2]/D	6.895	1.448	5.446	15.0	clk	clk
Hold 0.149 ns (10)	Path 8	12.465	1	1	a[3]	qa_reg[3]/D	6.799	1.454	5.344	15.0	clk	clk
Pulse Width 7.000 ns (30)	Path 9	12.531	1	1	b[0]	qb_reg[0]/D	6.719	1.461	5.258	15.0	clk	clk
Inter-Clock Paths	Path 10	12.612	2	3	qb_reg[0]/C	qco_reg/D	2.360	0.704	1.656	15.0	clk	clk
Other Path Groups												
User Ignored Paths												
Unconstrained Paths												

Fig 3.37: Timing after changes

Wire delay or net delay: It is the time required by the signal to propagate through the described path.

Logic delay: It is defined as the time required by the output to get stable from the instance the stable inputs are applied.

Process followed for reducing the delays and resolving timing violations:

- Firstly, the worst negative slack and worst hold slack was coming out to be negative. So, we increased the time period of our clock from 10ns to 15ns in order to remove the Setup violation.
- After the above step, setup violations were resolved, but the hold violations were still there. Hence, we tried to change the input port of sum[1] from E19 to U15. This step resolved the hold violations as well as decreased the delays.

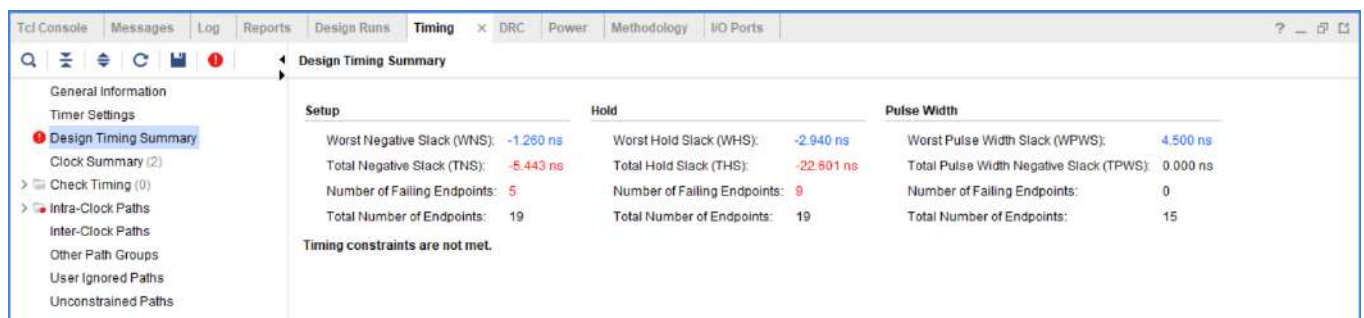


Fig 3.38: Timing with Negative Slack

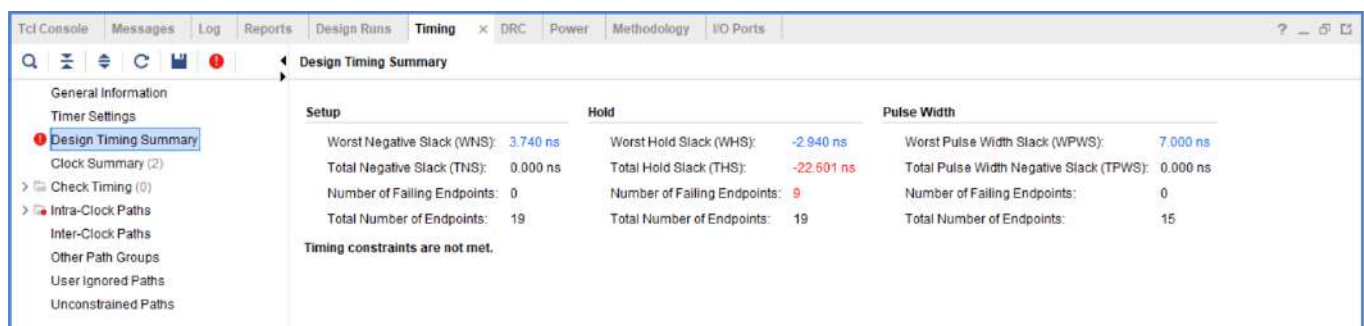


Fig 3.39: Timing with Positive Slack

Conclusion

The project of designing a 4-bit Ripple Carry Adder was carried out successfully with its simulation on Vivado and implementation on the *Basys 3* board. The Verilog HDL for the Ripple Carry Adder was simulated using its testbench successfully and the same was reflected on RTL Simulation. Also, using VIO (Virtual Input-Output), the inputs are fed to the Basys 3 board and the corresponding output is observed on the board. The carry later is mapped to the carry-chains of the FPGA fabric. Moreover, analysis of delay and resource utilisation after implementation is also carried out successfully. The factors which resulted in the delay were studied and a successful attempt to reduce them was made successfully as well.

References

- [1] <https://docs.xilinx.com/v/u/2017.3-English/ug893-vivado-ide>
- [2] https://digilent.com/reference/_media/basys3/basys3_rm.pdf
- [3] H. Parandeh-Afshar, P. Brisk and P. Ienne, "Exploiting fast carry-chains of FPGAs for designing compressor trees," *2009 International Conference on Field Programmable Logic and Applications*, 2009, pp. 242-249, doi: 10.1109/FPL.2009.5272301.