Experiment - 1: Carry Look ahead Adder and Binary Multiplier Simulation Using Quartus Prime

NAME: Jayakrishnan Menon

REG NO: 22BEC1205

DATE: 06/01/2025

Aim:
Write a Verilog RTL code for a Carry Look Ahead and Multiplier. Also, perform the simulation of the Full Adder using Quartus Prime and Model Sim. Finally, implement these circuits on the FPGA kit, "5CSXFC6D6F31C6N"

Software Required: Quartus Prime**,** ModelSim

Haraware Required: Altera Cyclone V 5CSXFC6D6F31C6N

Procedure:

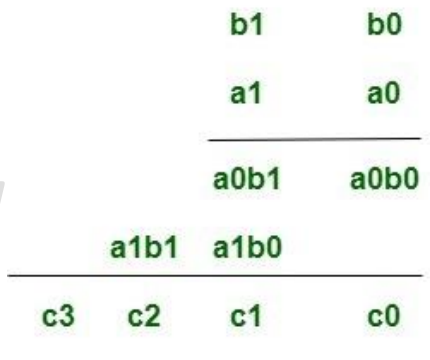
- Open Quartus Prime 21.1.
- Go to File -> New Project wizard -> select Source folder & type file name -> Next.
- Select Empty Project in Project type -> Next.
- Click on Next in Add Files dialog box.
- Select "Cyclone V SX Extended Features" in Device Family.
- In available devices, select the one ending with "31C6" -> Next.
- Select the tool name as "ModelSim" and Format as "VerilogHDL" in simulation.
- Click on Finish. The project is now created.
- In the Task window, select RTL simulation and run, this would open the ModelSim window.
- Simulate the full adder as you would do using ModelSim by forcing the input values.
- In Compilation -> select compile design.
- Go to Assignments tab -> Pin planner -> Give the location for each i/o pin.
- Go to Hardware Setup -> Select USB.
- Change the file to Fulladder.v in the program/configure option and select Start.

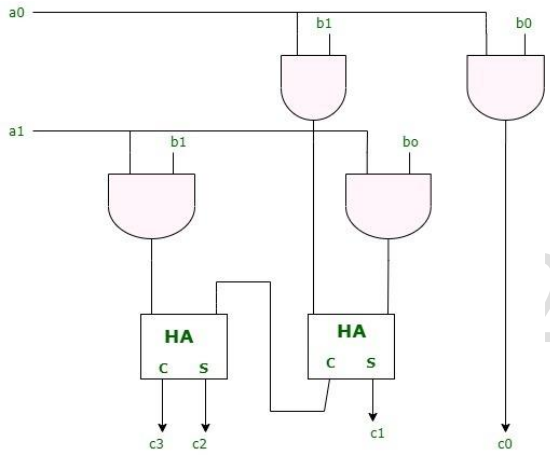
Theory

Multiplier:

A binary array multiplier is a digital combinational circuit used for multiplying two binary numbers. A variety of computer arithmetic techniques can be used to implement a digital multiplier. Most techniques involve computing the set of partial products, which are then summed together using binary adders. This process is similar to long multiplication, except that it uses a base-2 (binary) numeral system.

Here, Multiplication is done by employing an array of full adders and half adders. This array is used for the nearly simultaneous addition of the various product terms involved. To form the various product terms, an array of AND gates is used before the Adder array.





INPUT A		INPUT B		OUTPUT			
A_1	A_0	B_1	B_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Carry Lookahead Adder:

Ripple adders produce some carry propagation delay while performing arithmetic operations like multiplication and divisions as it involves several additions or subtraction steps. This is a major problem for the Ripple adder and hence improving the speed of addition will improve the speed of all other arithmetic operations. Hence reducing the carry propagation delay of adders is of great importance. There are different logic design approaches that have been employed to overcome the carry propagation problem. One widely used approach is to employ a carry look-ahead which solves this problem by calculating the carry signals in advance, based on the input signals. This type of adder circuit is called a carry look-ahead adder.

Consider two variables G_i as "carry generate" and P_i as "carry propagate", where:

$$P_i = A_i \oplus B_i$$

$$G_i = A_i \cdot B_i$$

The sum output and carry output can be expressed in terms of carry generate and carry propagate, where:

$$S_i = P_i \oplus C_i$$

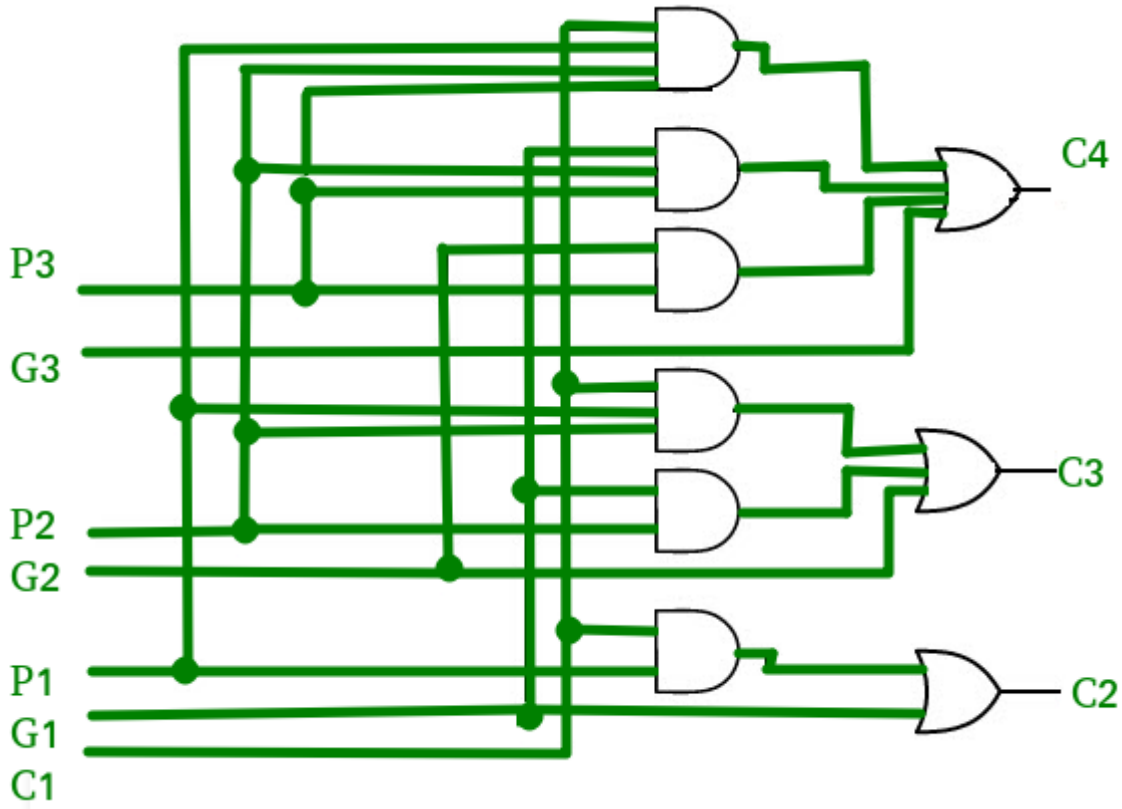
$$C_{i+1} = G_i + (P_i \cdot C_i)$$

Therefore, the carry output Boolean function of each stage in a 4 stage carry look-ahead adder can be expressed as:

$$C_1 = G_0 + P_0 C_{in}$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

From the above Boolean equations we can observe that C_4 does not have to wait for C_3 and C_2 to propagate but actually C_4 is propagated at the same time as C_3 and C_2. Since the Boolean expression for each carry output is the sum of products so these can be implemented with one level of AND gates followed by an OR gate.

The implementation of three Boolean functions for each carry output for a carry look-ahead carry generator shown in below figure.



Advantages of Carry Look-Ahead Adder:

- The propagation delay is reduced.
- It provides the fastest addition logic.

Disadvantages of Carry Look-Ahead Adder:

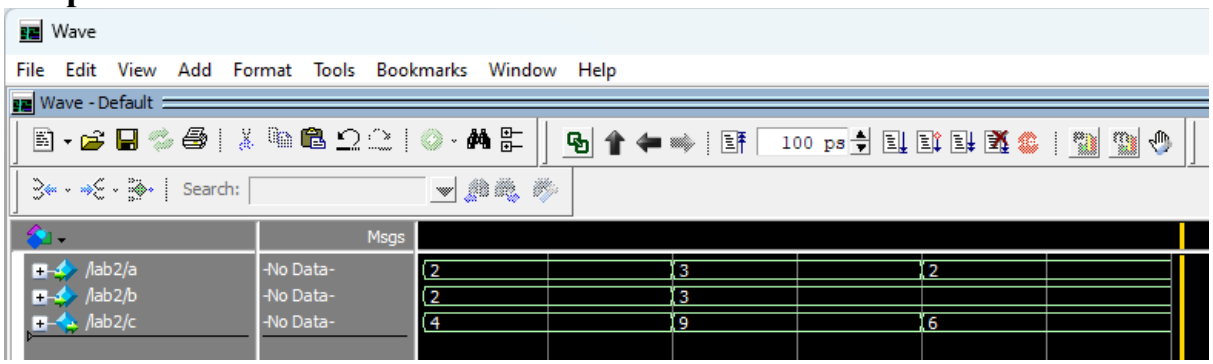
- The Carry Look-ahead adder circuit gets complicated as the number of variables increase.
- The circuit is costlier as it involves more hardware. For n-bit carry lookahead adder to evaluate all the carry bits it requires [n(n + 1)]/2 AND gates and n OR gates.

Source code and Outputs:

1. 2-Bit Multiplier:

```
module twobitmultiplier(input [1:0]a,b, output [3:0]c);

assign c[0] = (a[0]&b[0]);
assign c[1] = (a[0]&b[1]) ^ (a[1]&b[0]);
assign c[2] = (a[1]&b[1]) ^ ((a[0]&b[1]) & (a[1]&b[0]));
assign c[3] = (a[1]&b[1]) & ((a[0]&b[1]) & (a[1]&b[0]));

endmodule
```

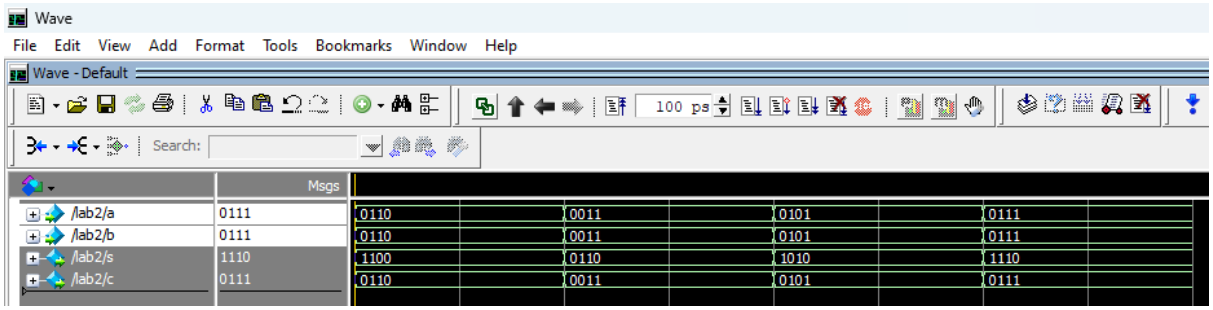
Output:



2. Carry Lookahead Adder (All Outputs Produced):


```
module lab2(input [3:0]a,b, output reg [3:0]c,s);
reg [3:0]p,g;
integer i;
reg cin=1'b0;

always@(*)
begin
s=4'b0000;
p[0] = a[0]^b[0];
g[0] = a[0]&b[0];
s[0] = p[0]^cin;
c[0] = g[0]|(p[0]&cin);
for(i=1; i<4; i=i+1)
begin
p[i] = a[i]^b[i];
g[i] = a[i]&b[i];
s[i] = p[i]^c[i-1];
c[i] = g[i]|(p[i]&c[i-1]);
end
end

endmodule
```

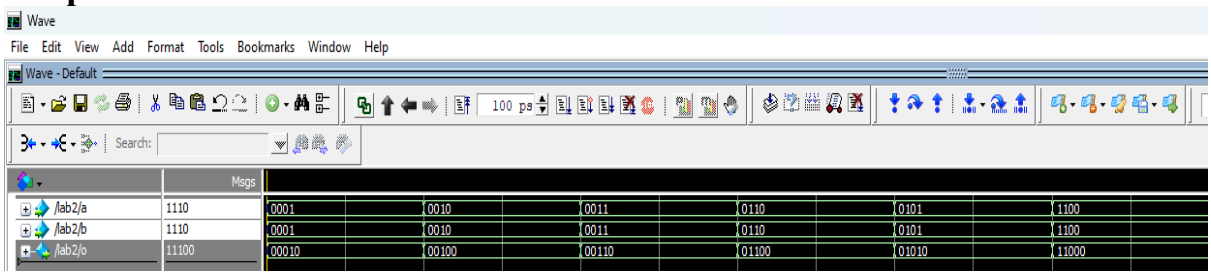
Output:



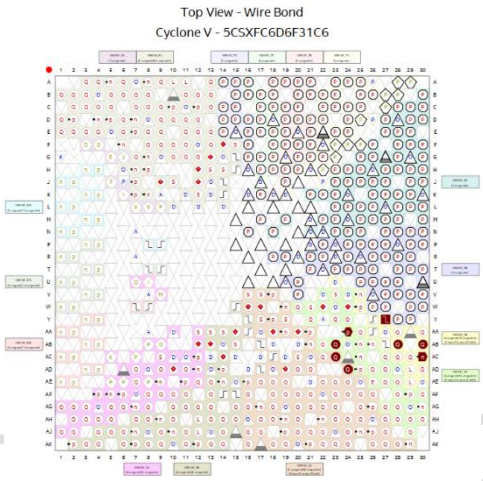
3. Carry Lookahead Adder (Required Outputs Produced for FPGA Kit):

```
module lab2(input [3:0]a,b, output reg [4:0]o);
reg [3:0]p,g,c,s;

integer i;
reg cin=1'b0;

always@(*)
begin
s=4'b0000;
p[0] = a[0]^b[0];
g[0] = a[0]&b[0];
s[0] = p[0]^cin;
c[0] = g[0]|(p[0]&cin);
for(i=1; i<4; i=i+1)
begin
p[i] = a[i]^b[i];
g[i] = a[i]&b[i];
s[i] = p[i]^c[i-1];
c[i] = g[i]|(p[i]&c[i-1]);
end

o[4:0]={c[3],s[3:0]};

end

endmodule
```

Output:



Pin Planning:



Chosen I/O for Multiplier:

Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate
a[1]	Input				PIN_AB30	2.5 V (default)		12mA (default)	
a[0]	Input				PIN_AB28	2.5 V (default)		12mA (default)	
b[1]	Input				PIN_AA28	2.5 V (default)		12mA (default)	
b[0]	Input				PIN_AA30	2.5 V (default)		12mA (default)	
c[3]	Output				PIN_V25	2.5 V (default)		12mA (default)	1 (default)
c[2]	Output				PIN_AC28	2.5 V (default)		12mA (default)	1 (default)
c[1]	Output				PIN_W25	2.5 V (default)		12mA (default)	1 (default)
c[0]	Output				PIN_AC29	2.5 V (default)		12mA (default)	1 (default)
<<new node>>									

All Pins

Chosen I/O for Carry Lookahead Adder:

Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate
a[3]	Input	PIN_AC30	5B	B5B_N0	PIN_AC30	2.5 V		12mA (default)	
a[2]	Input	PIN_AB28	5B	B5B_N0	PIN_AB28	2.5 V		12mA (default)	
a[1]	Input	PIN_Y27	5B	B5B_N0	PIN_Y27	2.5 V		12mA (default)	
a[0]	Input	PIN_AB30	5B	B5B_N0	PIN_AB30	2.5 V		12mA (default)	
b[3]	Input	PIN_AD30	5B	B5B_N0	PIN_AD30	2.5 V		12mA (default)	
b[2]	Input	PIN_AC28	5B	B5B_N0	PIN_AC28	2.5 V		12mA (default)	
b[1]	Input	PIN_V25	5B	B5B_N0	PIN_V25	2.5 V		12mA (default)	
b[0]	Input	PIN_W25	5B	B5B_N0	PIN_W25	2.5 V		12mA (default)	
o[4]	Output	PIN_AG25	4A	B4A_N0	PIN_AG25	2.5 V		12mA (default)	1 (default)
o[3]	Output	PIN_AD24	4A	B4A_N0	PIN_AD24	2.5 V		12mA (default)	1 (default)
o[2]	Output	PIN_AC23	4A	B4A_N0	PIN_AC23	2.5 V		12mA (default)	1 (default)
o[1]	Output	PIN_AB23	5A	B5A_N0	PIN_AB23	2.5 V		12mA (default)	1 (default)
o[0]	Output	PIN_AA24	5A	B5A_N0	PIN_AA24	2.5 V		12mA (default)	1 (default)
<<new node>>									

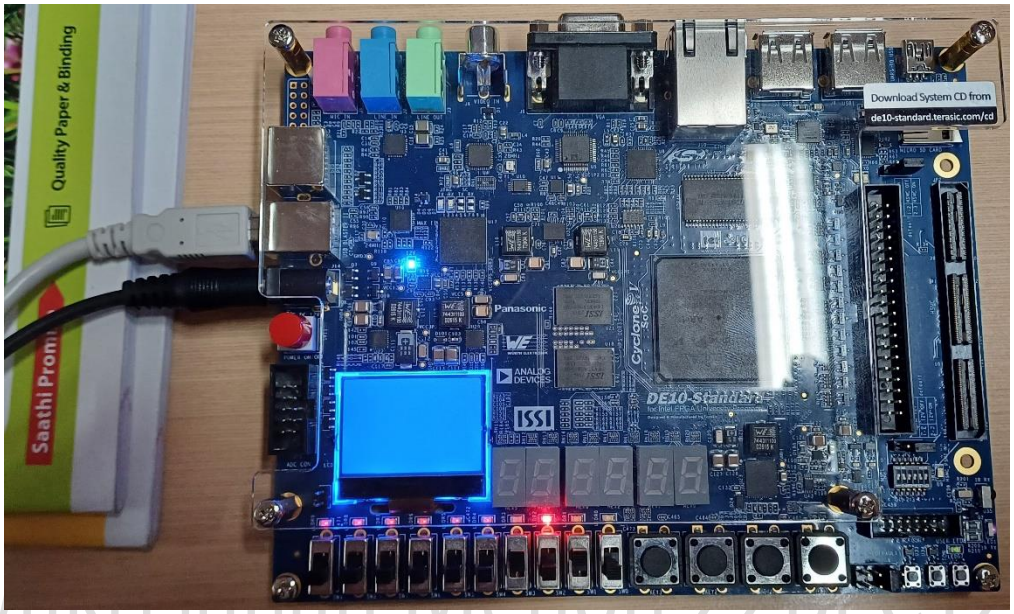
All Pins

Hardware Implementation:

2-Bit Multiplier:

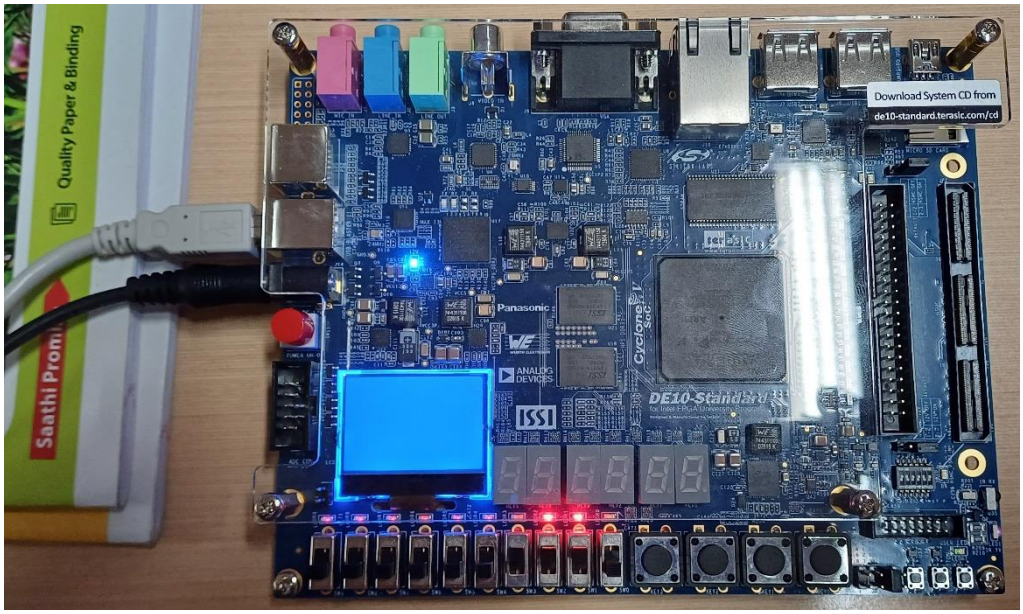
Input: a=10, b=10

Output: c=0100



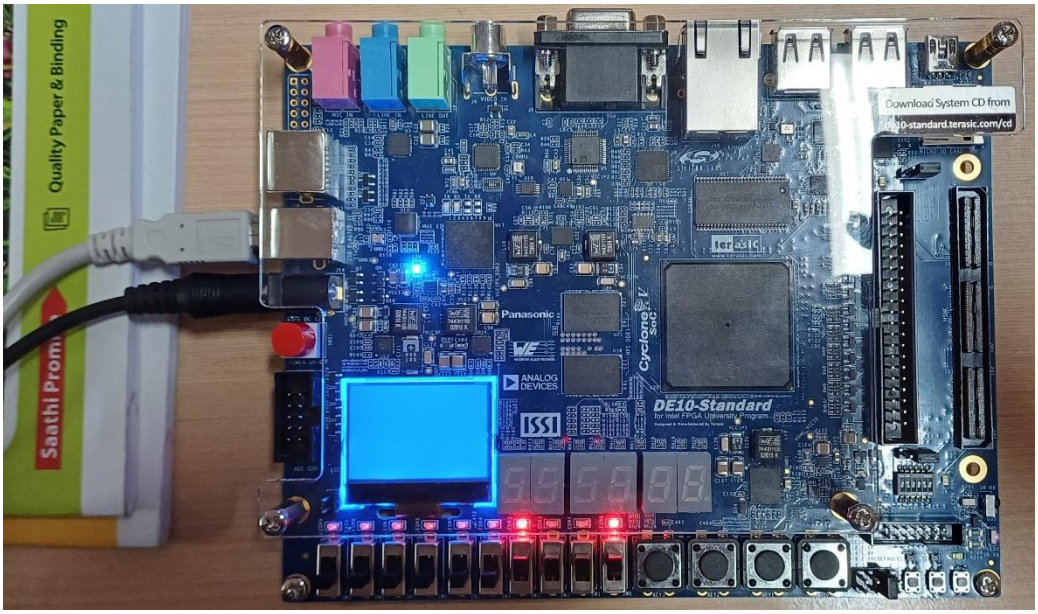
Input: a=10, b=11

Output: c=0110



Input: a=11, b=11

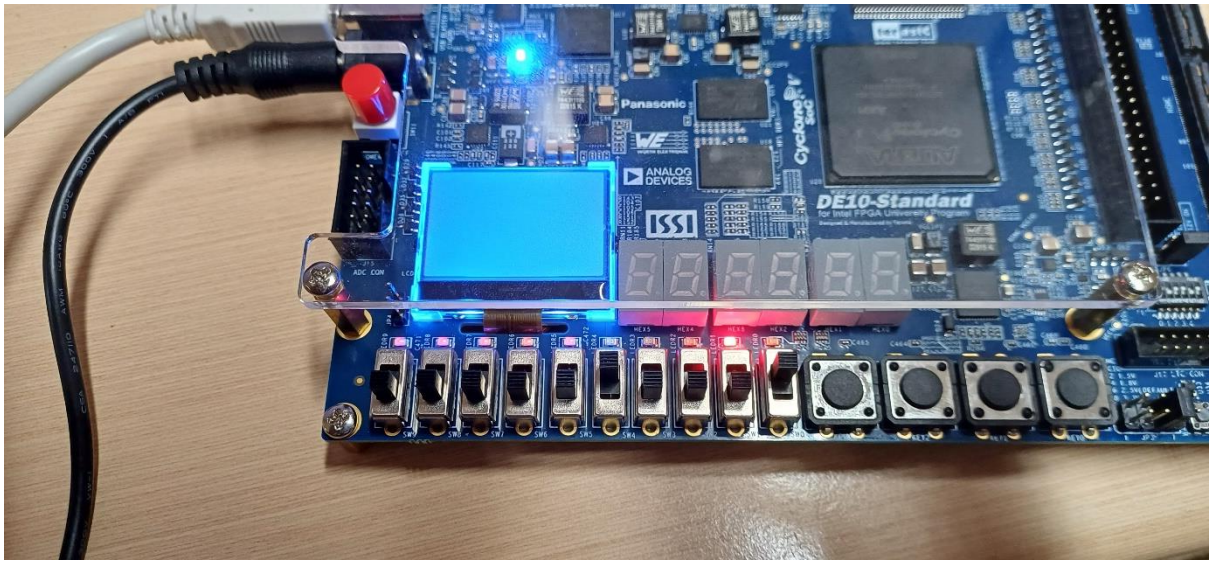
Output: c=1001



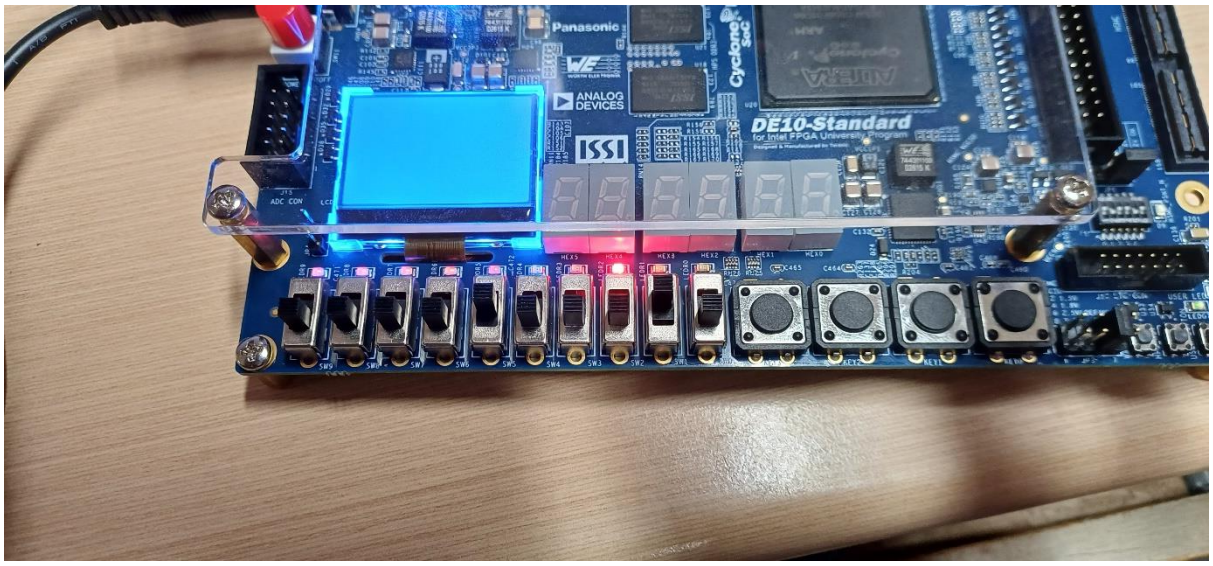
Carry Lookahead Adder:

Input: a=0001, b=0001

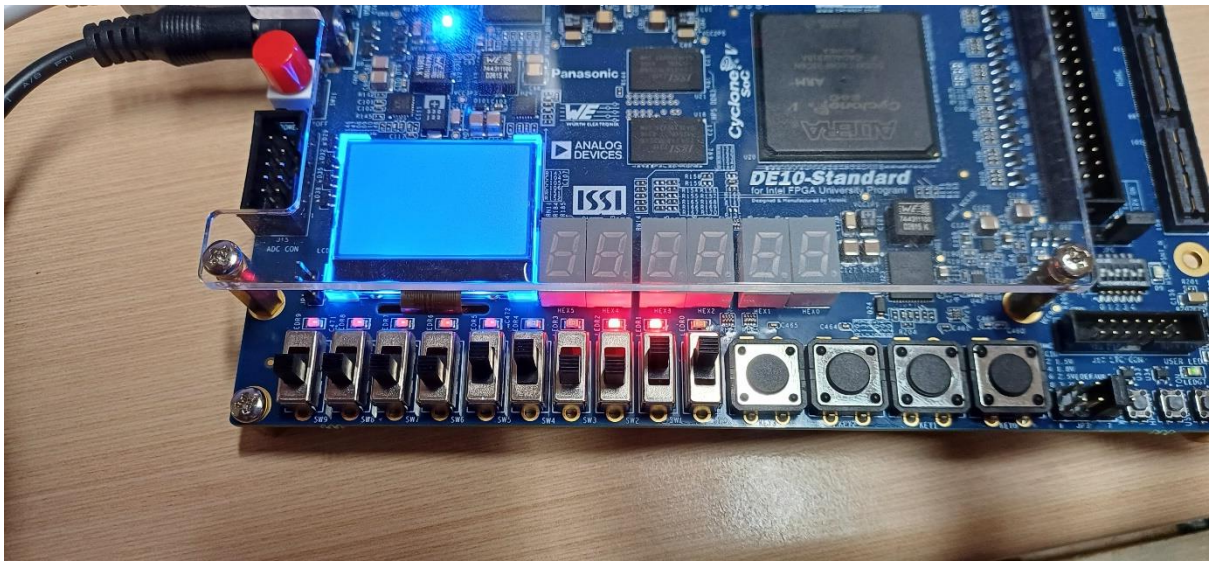
Output: c=00010



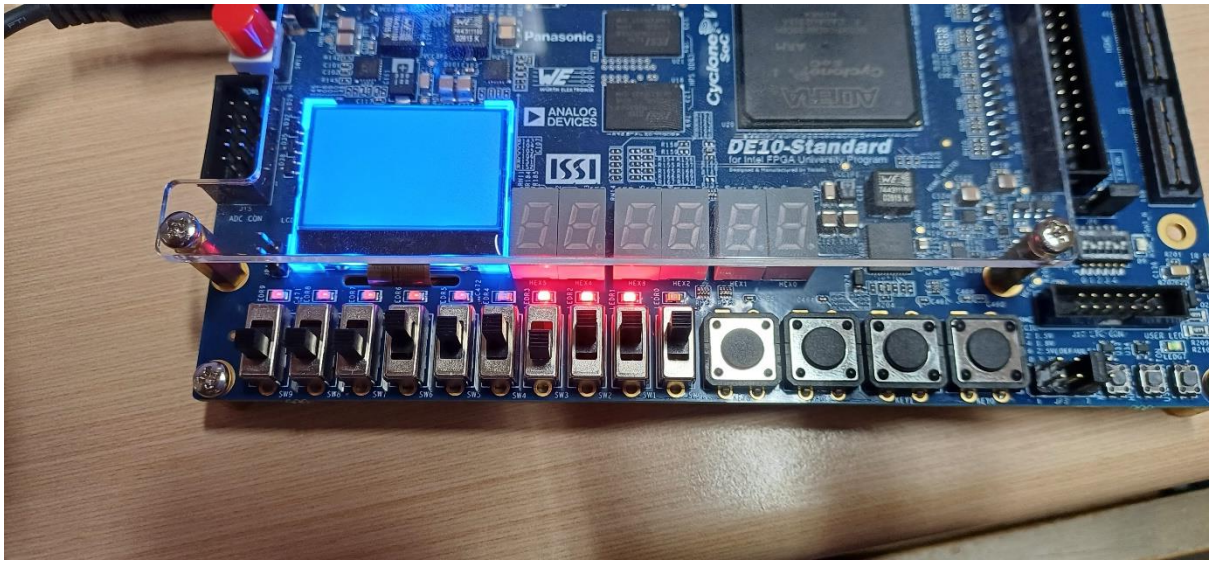
Input: a=0010, b=0010
Output: c=00100



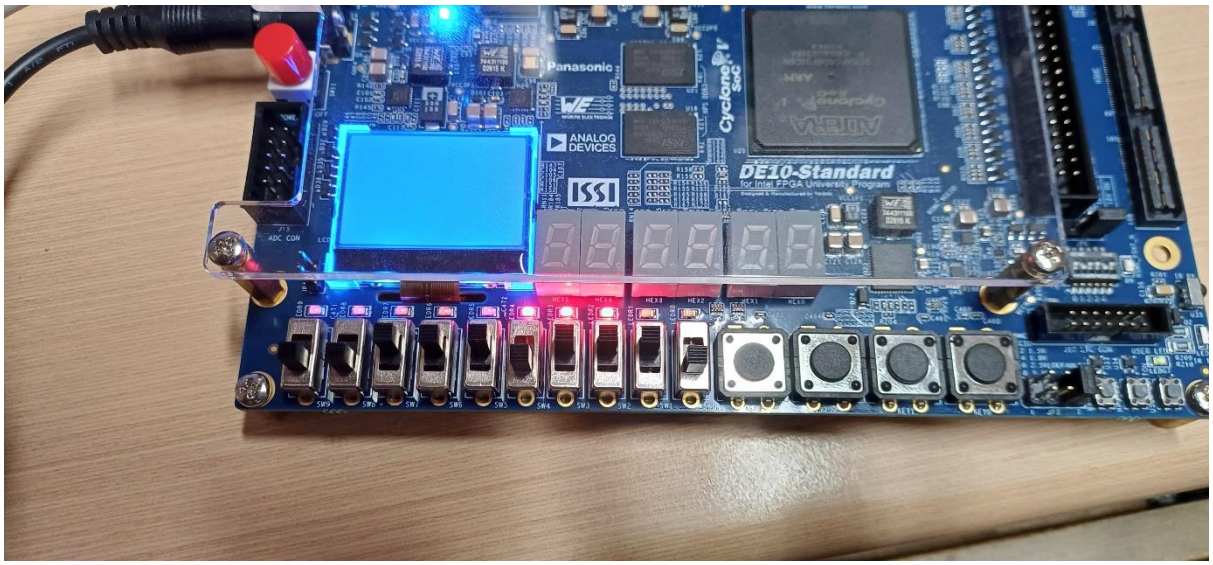
Input: a=0011, b=0011
Output: c=00110



Input: a=0111, b=0111
Output: c=01110



Input: a=1110, b=1110
Output: c=11100



Result:
Hence, Verilog RTL code for a Carry Look Ahead and Multiplier, were successfully verified and implemented using Quartus Prime.