**Digital System design Processing Lab**

Lab Journal: 03



Name: M.OMAR

Class: BCE-6

Enrollment No: 01-132182-024

**Lab # 03**

**LAB NO. 3  
Implementing HA, FA and 4-bits Ripple Carry Adder design in Verilog**

**Objective:**

* **The objective of this lab is to give a comprehensive tutorial of HA,FA,RCA in Verilog Hardware Descriptive Language (HDL).**

**Procedure:**

* Open eda playground Software.
* Make a new script and name it on the name of your lab.
* Make your tasks in the same folder and keep them at the same directory.
* Make functions and name them same as your script name.
* Make sure that the main function and the other functions are in the same directory.
* Run the given code and see the output.

**INTRODUCTION:**

**Introduction:**

A Ripple Carry Adder is made of a number of full-adders cascaded together. It is used to add together two binary numbers using only simple logic gates. The figure below shows 4 full-adders connected together to produce a 4-bit ripple carry adder.

**Purpose :** Familiarization with VHSIC Hardware Description Language (VHDL) and with VHDL design tools. VHDL is an increasingly important tool in digital design used for automated specification and testing of digital systems. In this exercise, you will write a VHDL specification for a full adder and use this full adder component to create a 4-bit ripple-carry (RCA) adder. More specifically, you will be instantiating four full adder components to structurally model the 4-bit RCA. You will also simulate and test your VHDL “code” using ModelSim.

**Background information**

There are two types of circuit Sequential circuit and Combinational circuit. Sometimes we have to make the circuit designs that are very big and complex, So it is better to first test that design through coding in Verilog language. By this technique we can save our time and resources. We know the how Half adder circuit and full adder circuit works. But in this lab, we are using Verilog language to make Adder circuit.

**Procedure:**

Full adder is a circuit that is made of 2 half adder and is use to sum three-bit number. In this Lab we are using three methods to write full adder circuit code in Verilog.

1. **Data flow level**

This level of abstraction is higher than the gate level. Expressions, operands and operators characterize this level. Most of the operators used in dataflow modeling are common to software programmers, but there are a few others that are specific to HW design. Operators that are used in expressions for dataflow modeling are given in Table 2.3. At this level every expression starts with the keyword assign. Here is a simple example where two variables a and bare added to produce c: assign c a + b; The value on wire c is continuously driven by the result of the arithmetic operation. This assignment statement is also called ‘continuous assignment’. In this statement the right-hand side must be a variable of type wire, whereas the operands on the left-hand side may be of type wire or reg.

1. **Gate level**

The code at gate level is built from Verilog primitives. These primitives are built in gate level models of basic functions, including nand, nor, and, or, xor, buf and not. Modeling at this level requires describing the circuit using logic gates. This description looks much like an implementation of a circuit in a basic logic design course. Delays can also be modeled at this level.

1. **Behavioral**

The behavioral level is the highest level of abstraction in Verilog. This level provides high level language constructs like for, while, repeat, if else and case. Designers with a software programming background already know these constructs

**Task 01**

**Can you make 8 to 1 mux utilizing 4to 1 mux and 2to1 mux**

|  |  |
| --- | --- |
| **Task 1** | **Making 8x1 Multiplexer using switch statement** |

Code

module Mux(

input I1,

input I2,

input I3,

input I4,

input I5,

input I6,

input I7,

input I8,

input [2:0]S ,

output reg Out

);

always@(\*)

begin

case (S)

3'b000 : Out= I1;

3'b001 : Out=I2;

3'b010 : Out=I3;

3'b011 : Out=I4;

3'b100 : Out=I5;

3'b101 : Out=I6;

3'b110 : Out=I7;

3'b111 : Out=I8;

endcase

end

endmodule

**Setting Timing**

module Timing;

// Inputs

reg I1;

reg I2;

reg I3;

reg I4;

reg I5;

reg I6;

reg I7;

reg I8;

reg [2:0] S;

// Outputs

wire Out;

// Instantiate the Unit Under Test (UUT)

Mux uut (

.I1(I1),

.I2(I2),

.I3(I3),

.I4(I4),

.I5(I5),

.I6(I6),

.I7(I7),

.I8(I8),

.S(S),

.Out(Out)

);

initial begin

// Initialize Inputs

I1 = 0;

I2 = 1;

I3 = 0;

I4 = 1;

I5 = 0;

I6 = 1;

I7 = 0;

I8 = 1;

S = 0;

// Wait 100 ns for global reset to finish

#100;

S = 1;

#100;

S = 0;

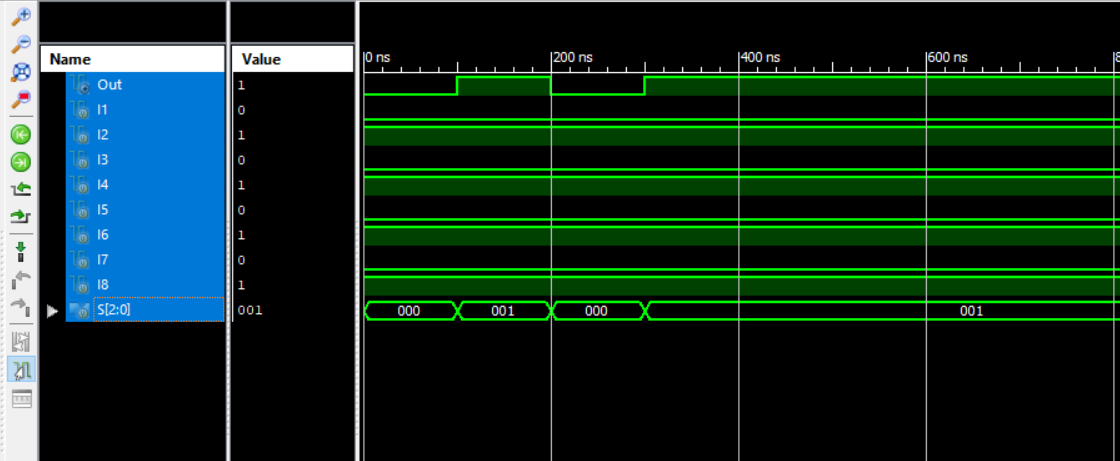
#100;

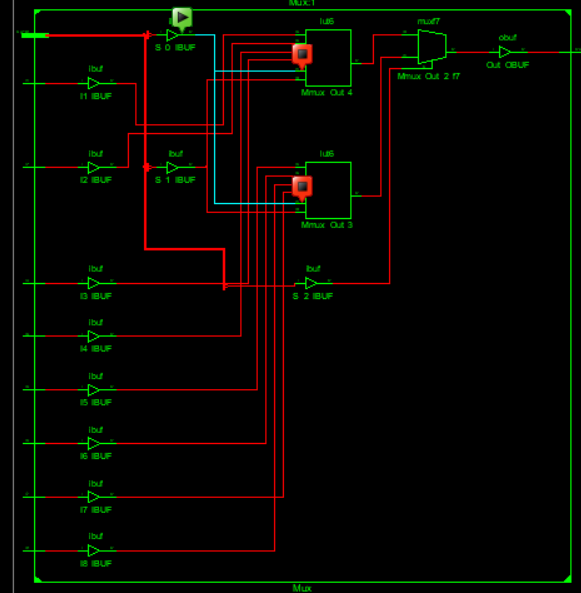
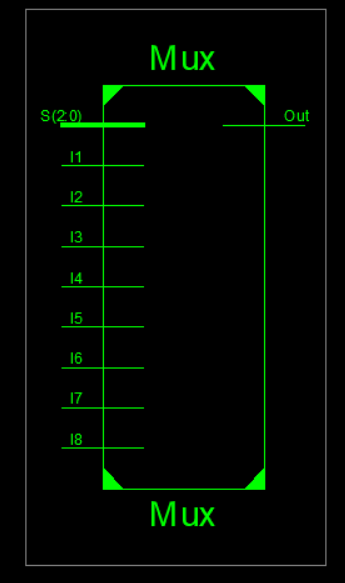
S = 1;

// Add stimulus here

end

endmodule





**Task 02**

* **Perform the following addition in Verilog.**

1. **4’d5 + 4’d2**
2. **4’d6 + 4’d7**

Code:

module addop(C, O, A, B);

input [3:0] A;

input [3:0] B;

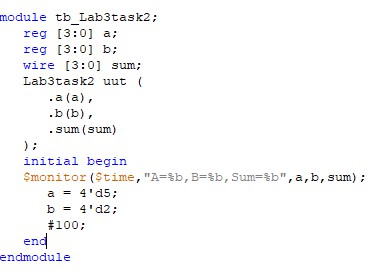
output [3:0] O;

output C;

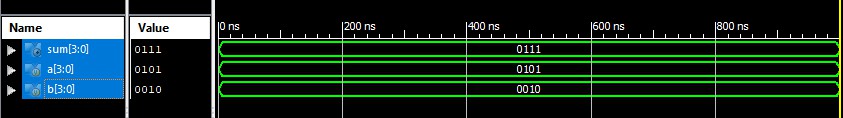
assign {C, O} = A + B;

endmodule

**Test\_bench:**

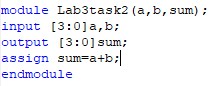


**Output**

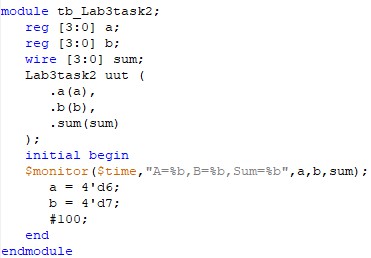


**b. 4’d6 + 4’d7**

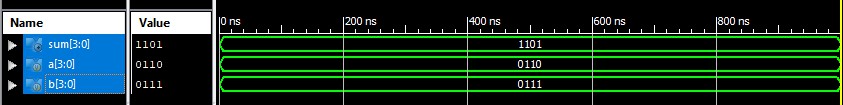
**Code:**



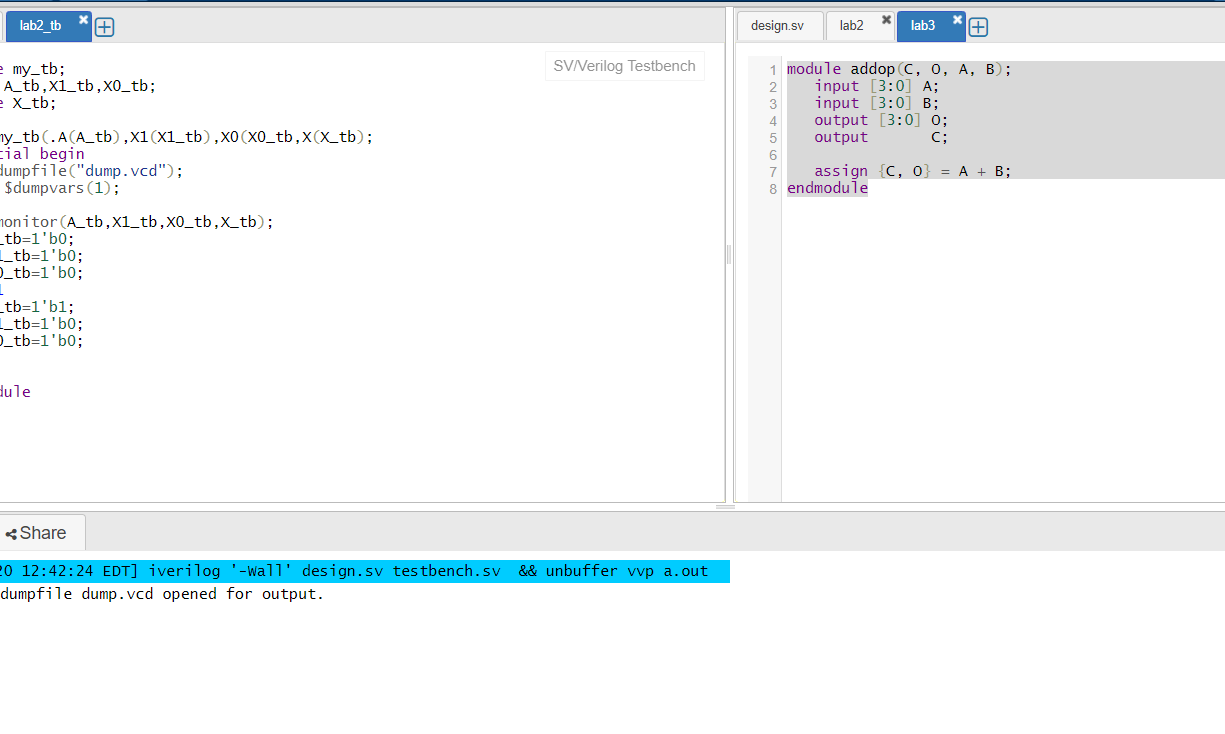
**Test\_bench:**



**Output:**



Output:



**Task 03**

**Implement 1-bit Half Adder in Verilog**

**Write a Verilog code for Half Adder along its stimulus module**

**Half adder Truth table**

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **carry** | **sum** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

module HA(d , e , sum , carry);

input d,e;

output sum , carry;

xor (sum , d , e);

and (carry , d , e);

endmodule

module FA(X , Y , Z , S , C);

input X , Y , Z;

output S , C;

wire s1 , d1 , d2;

HA HA1(X , Y , s1 , d1),

HA2(Z , s1 , s , d2);

or g1 (C , d2 , d1);

endmodule

**Setting Timing of Inputs:**

module fulladdertiming;

// Inputs

reg X;

reg Y;

reg Z;

// Outputs

wire S;

wire C;

// Instantiate the Unit Under Test (UUT)

FA uut (

.X(X),

.Y(Y),

.Z(Z),

.S(S),

.C(C)

);

initial begin

// Initialize Inputs

X = 0;

Y = 0;

Z = 0;

// Wait 100 ns for global reset to finish

#100;

X = 1;

Y = 0;

Z = 0;

#100;

X = 1;

Y = 1;

Z = 0;

// Add stimulus here end

endmodule

**Simulation Output:**



**Task 04**

* **Design Full Adder by using two half adders and code it in verilog**
* **Write a Verilog code for Full Adder along its stimulus module.**
* **Full adder Truth table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **carry** | **sum** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**Program logic code:**

module Fulladder(

input a,

input b,

input cin,

output sum,

output carry);

assign {carry,sum}=a+b+cin;

endmodule

**Setting Timing of Inputs:**

module fulladdertiming;

// Inputs

reg a;

reg b;

reg cin;

// Outputs

wire sum;

wire cout;

initial begin

// Initialize Inputs

a = 0;

b = 0;

cin = 0;

// Wait 100 ns for global reset to finish

#100;

a =1;

b=0;

#50;

a=1;

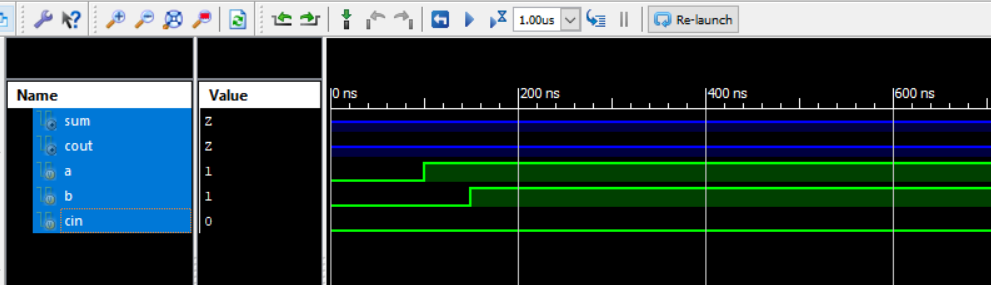
b=1;

// Add stimulus here

end

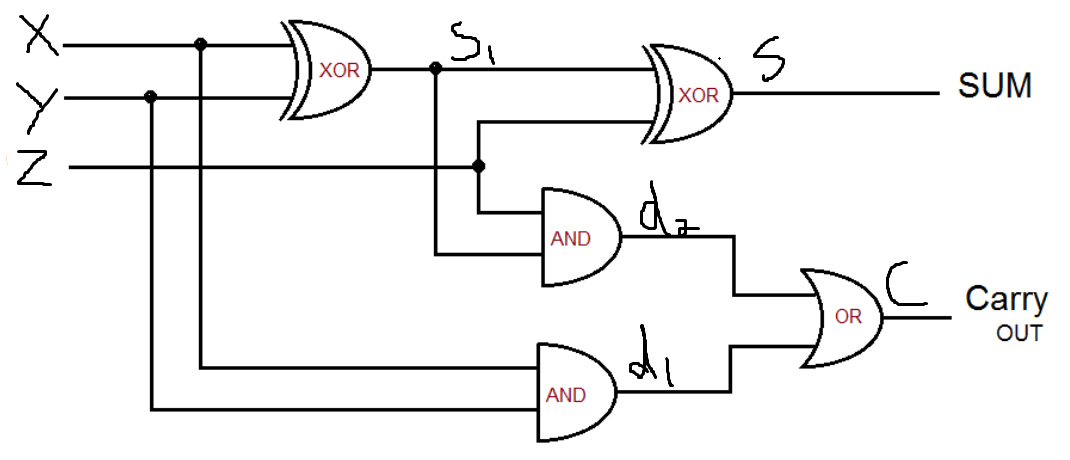
endmodule

**Simulation Output:**



**Gate flow level:**

**Program logic code:**



|  |  |
| --- | --- |
| **Task** | **4 bit Ripple Carry Adder (RCA) design in Verilog HDL** |

**Code :**

`timescale 1ns / 1ps

module HA(A,B,s,c);

input A,B;

output s;

output c;

assign {c,s}=A+B;

endmodule

module FA(A,B,cin,sum,cout);

input A,B;

input cin;

output sum;

output cout;

wire w1,w3;

wire w2;

HA n1(A,B,w2,w1);

HA n2(cin,w2,sum,w3);

assign cout=w1|w3;

endmodule

module RCA3\_bit(input[2:0]a,b,

input cin,

output [2:0] sum,

output cout);

wire c4,c5;

FA F1(a[0],b[0],cin,sum[0],c4),

F2(a[1],b[1],c4,sum[1],c5),

F3(a[2],b[2],c5,sum[2],cout);

endmodule

module RCA\_12bit(a,b,cin,sum,cout);

input[11:0]a;

input [11:0]b;

input cin;

output [11:0]sum;

output cout;

wire c1,c2,c3;

RCA3\_bit f1(a[2:0],b[2:0],cin,sum[2:0],c1),

f2(a[5:3],b[5:3],c1,sum[5:3],c2),

f3(a[8:6],b[8:6],c2,sum[8:6],c3),

f4(a[11:9],b[11:9],c3,sum[11:9],cout);

endmodule

**Test banch**

module RCATEST;

// Inputs

reg [11:0] a;

reg [11:0] b;

reg cin;

// Outputs

wire [11:0] sum;

wire cout;

// Instantiate the Unit Under Test (UUT)

RCA\_12bit uut (

.a(a),

.b(b),

.cin(cin),

.sum(sum),

.cout(cout)

);

initial begin

// Initialize Inputs

a = 520;

b = 110;

cin = 0;

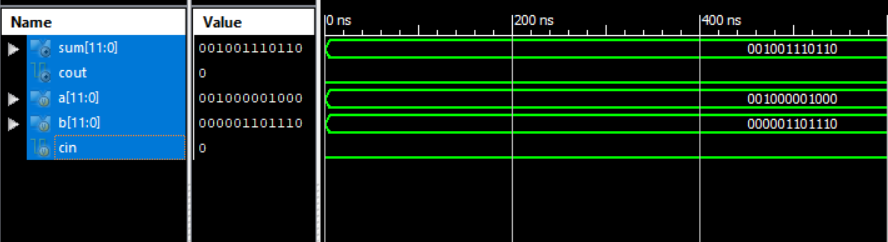
// Wait 100 ns for global reset to finish

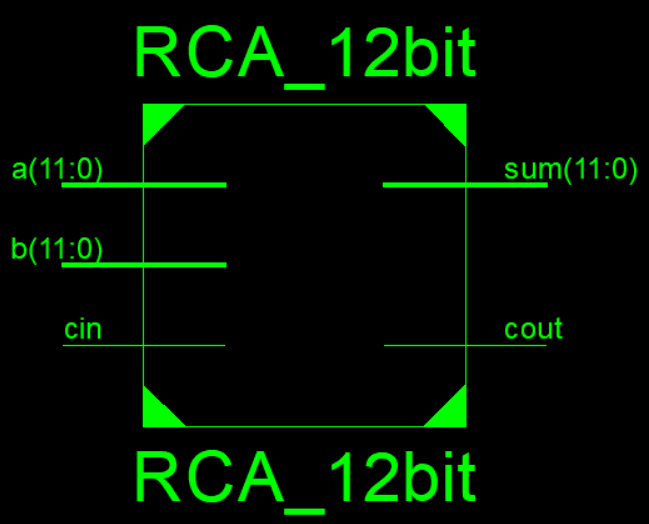
#100;

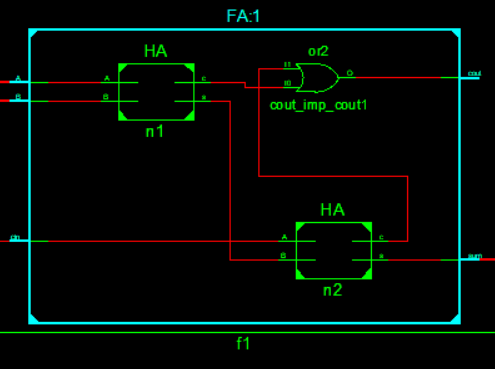
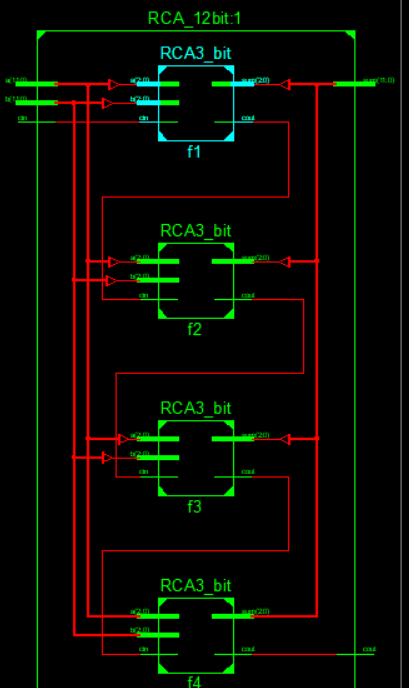
end

endmodule

**Simulation:**



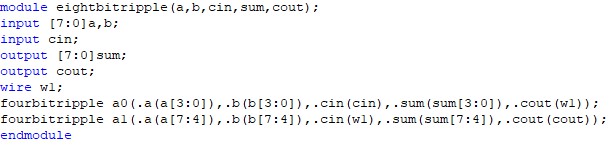




**Task 6:**

**Can we make 8 bit ripple carry adder using 4 bit ripple carry adder.**

**Code:**



**Test\_bench:**

module tb\_eightbitripple;

reg [7:0]a,b;

reg cin;

wire [7:0]sum; wire cout; eightbitripple uut (

.a(a),

.b(b),

.cin(cin),

.sum(sum),

.cout(cout)

);

initial begin

$monitor($time,"a=%b,b=%b,cin=%b,sum=%b,cout=%b",a,b,cin,sum,cout);

a = 8'd0; b = 8'd0; cin = 1'b0;

#100;

a = 8'd255;

b = 8'd255;

#100;

a = 8'd20;

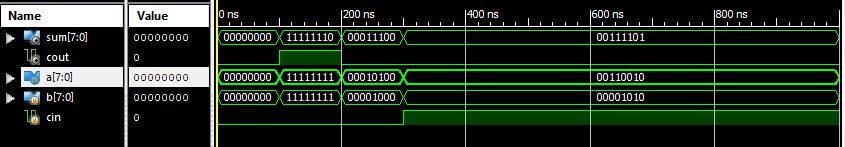
b = 8'd8;

#100;

a = 8'd50; b = 8'd10; cin = 1'b1;

#100; end endmodule

**Output:**



**Conclusion: Today we learned about Half Adder, Full Adder and Ripple Carry Adder.**