Laboratory 3 (100 Points)

Assignment 1 (20 Points)

Design a logic diagram and write a Verilog program to implement 4-bit ripple carry adder.

1. Design a logic diagram for a full adder.

Z	0	0	0	0
X	0	0	1	1
+Y	+0	+1	+0	+1
C S	0 0	0 1	0 1	1 0
Z	1	1	1	1
X	0	0	1	1
+Y	+0	+1	+0	+1
C S	0 1	1 0	1 0	1 1

TABLE 3-12
Truth Table of Full Adder

Inputs			Outputs	
X	Y	Z	C	S
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

S		YZ			
		00	01	11	10
X	0		**1**		**1**
	1	**1**		**1**	

C		YZ			
		00	01	11	10
X	0			**1**	
	1		**1**	**1**	**1**

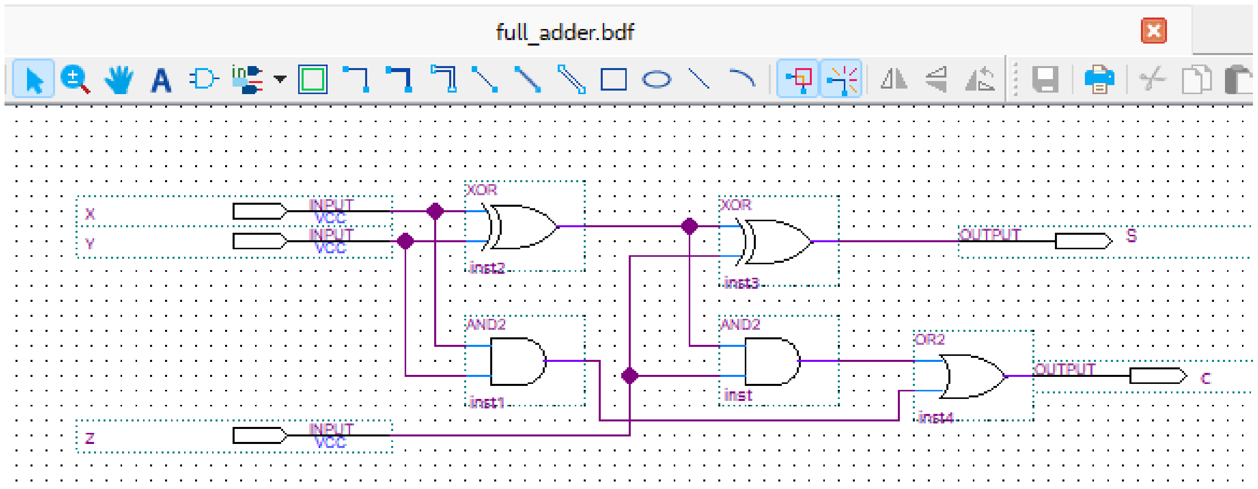
$$S = \overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + XYZ$$
$$= X \oplus Y \oplus Z$$

$$C = XY + XZ + YZ$$
$$= XY + Z(X\overline{Y} + \overline{X}Y)$$
$$= XY + Z(X \oplus Y)$$

35

a. Create a new project named **`Adder`**.

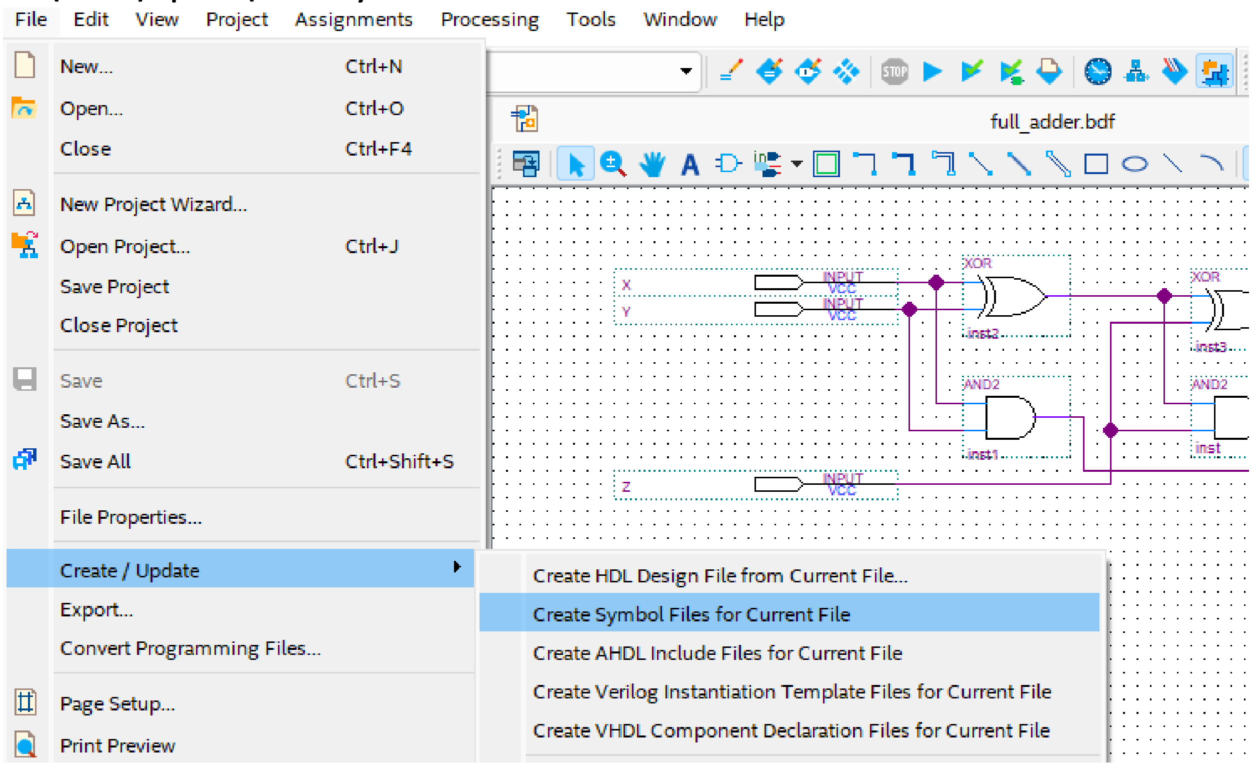
b. Add a new block diagram file (schematic file) named **`full_adder.bdf`** into the project.

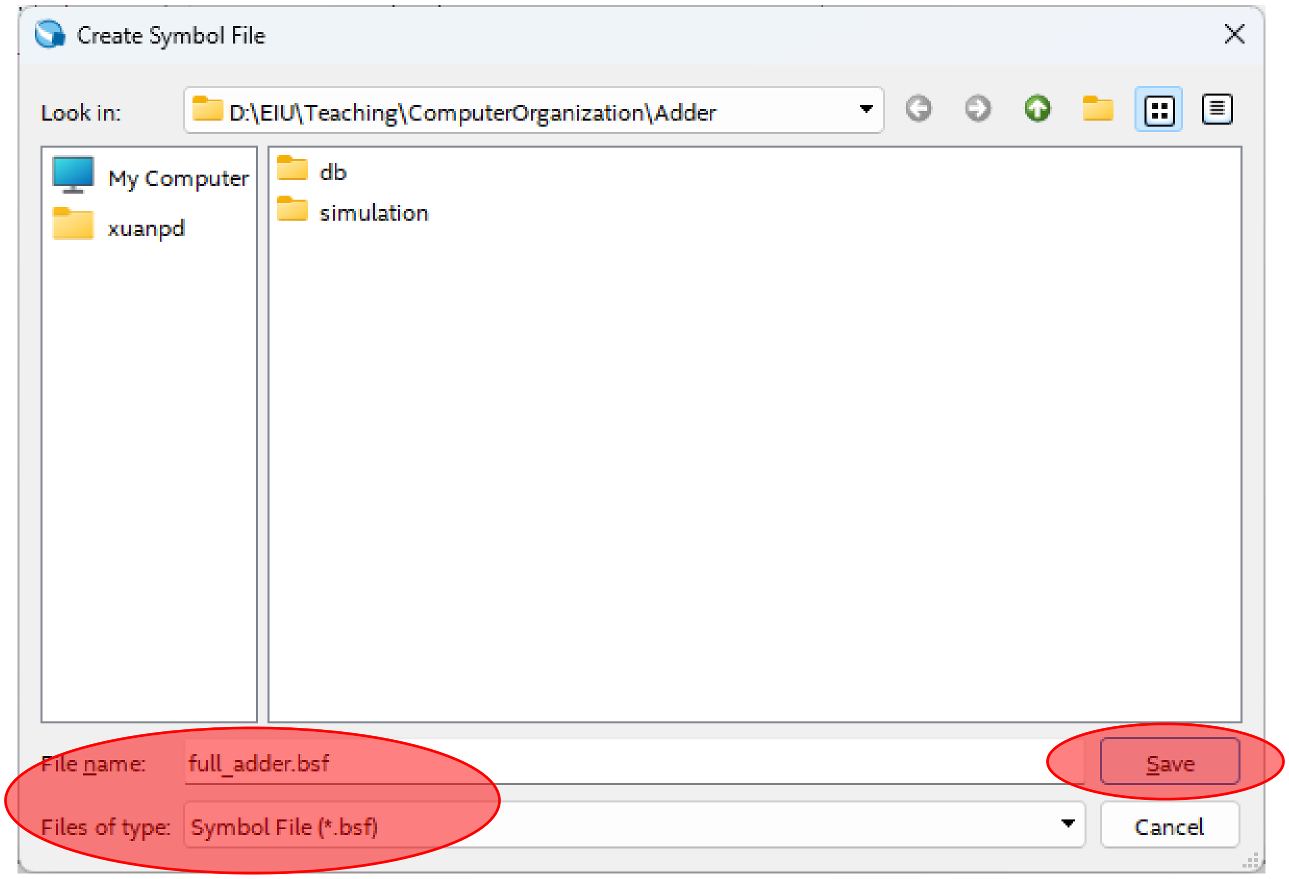


2. Create a symbol for the block diagram **`full_adder`.**

- Make sure that the diagram **`full_adder`** is currently open. Select

File\Create/Update\Create Symbols File for Current File.





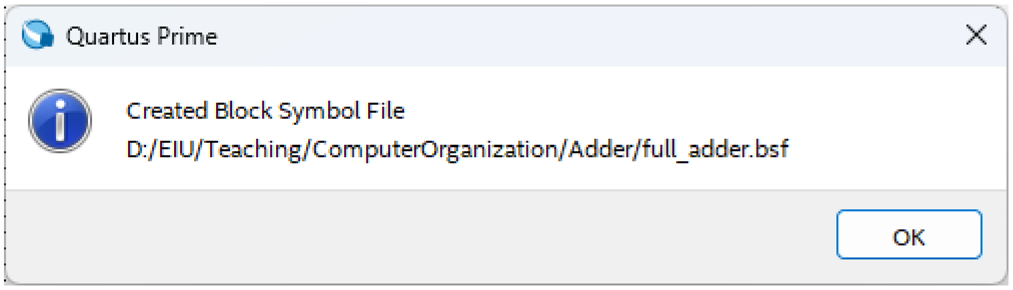
Look in: D:\EIU\Teaching\ComputerOrganization\Adder

db
simulation

File name: full_adder.bsf

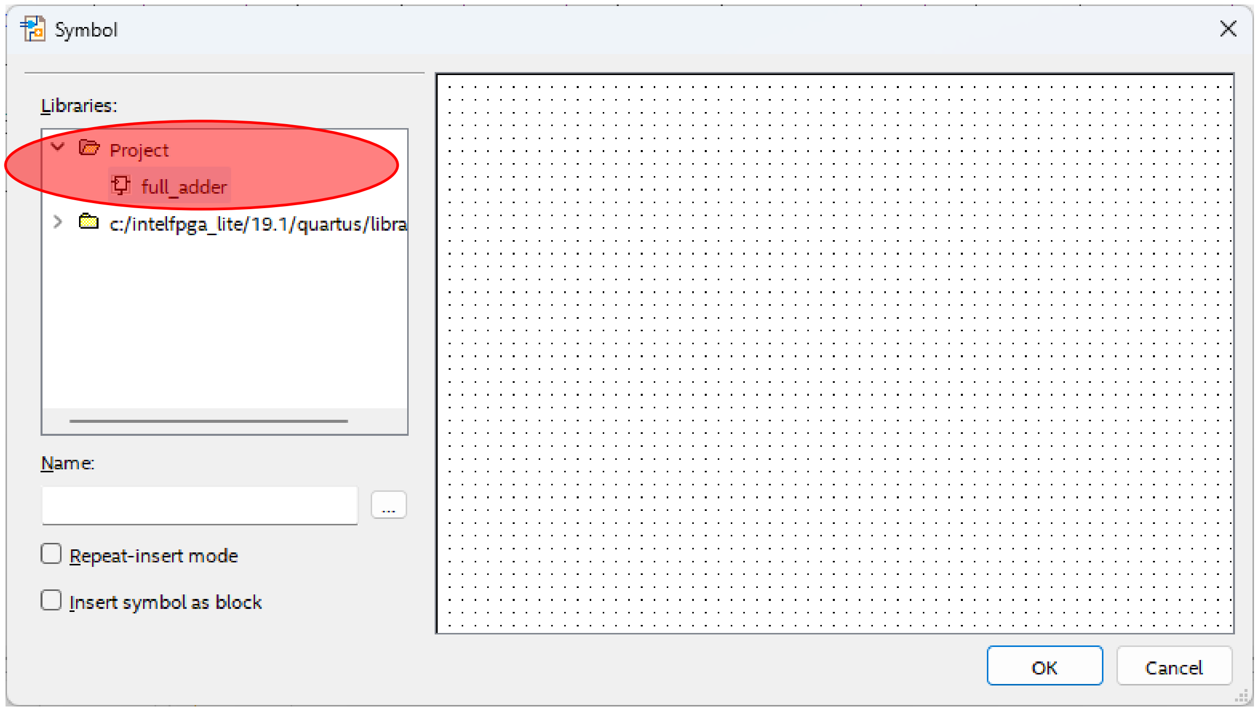
Files of type: Symbol File (*.bsf)

Save

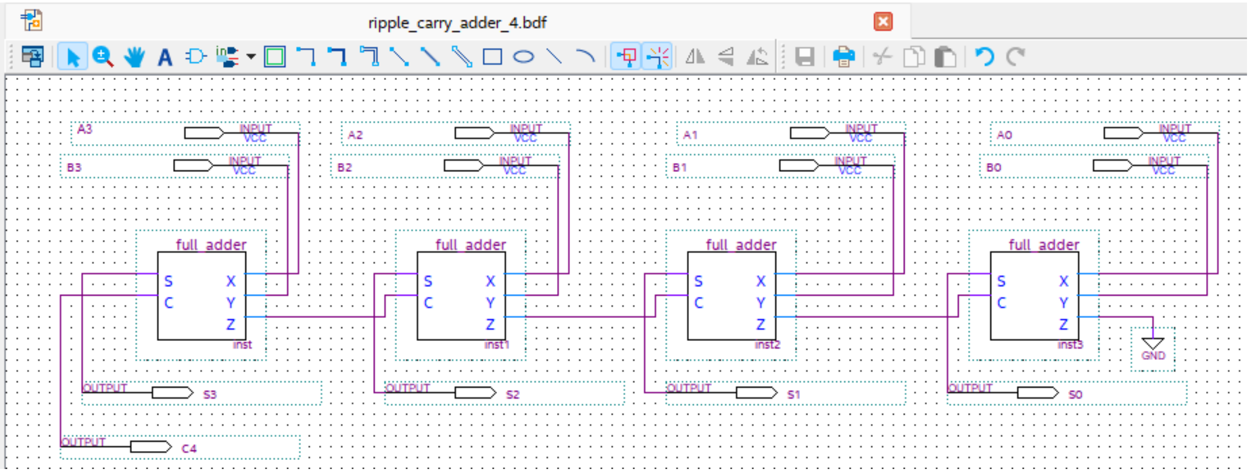


Created Block Symbol File
D:/EIU/Teaching/ComputerOrganization/Adder/full_adder.bsf

3. Design a block diagram of 4-bit ripple carry adder named **ripple_carry_adder_4.bdf** using **`full_adder`** circuit created in the previous step.
- Insert **full_adder** symbol:



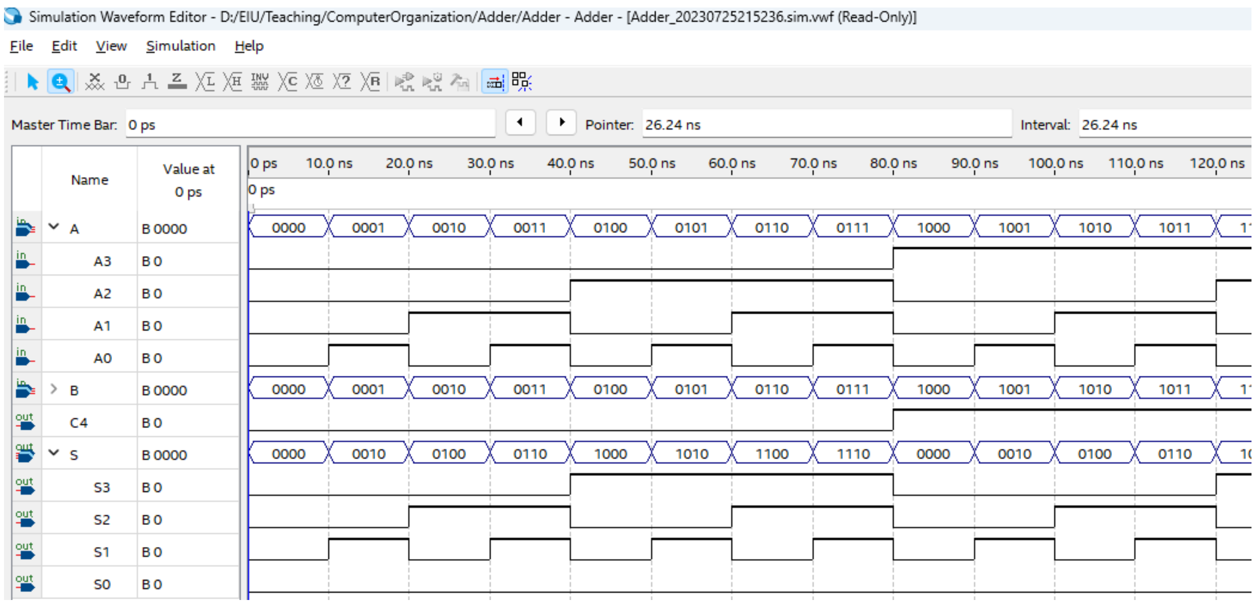
- Design the 4-bit ripple carry adder block diagram



- Compile the design

4. Simulate the 4-bit ripple carry adder using waveform editor tool

Test the adder with some input values.


Simulation Waveform Editor - D:/EIU/Teaching/ComputerOrganization/Adder/Adder - Adder - [Adder_20230725215236.sim.vwf (Read-Only)]

File Edit View Simulation Help

Master Time Bar: 0 ps Pointer: 26.24 ns Interval: 26.24 ns

Name	Value at 0 ps
A	B 0000
A3	B 0
A2	B 0
A1	B 0
A0	B 0
B	B 0000
C4	B 0
S	B 0000
S3	B 0
S2	B 0
S1	B 0
S0	B 0

0 ps 10.0 ns 20.0 ns 30.0 ns 40.0 ns 50.0 ns 60.0 ns 70.0 ns 80.0 ns 90.0 ns 100.0 ns 110.0 ns 120.0 ns
A: 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011
B: 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011
S: 0000 0010 0100 0110 1000 1010 1100 1110 0000 0010 0100 0110

5. Write a program in Verilog to implement 4-bit ripple carry adder:

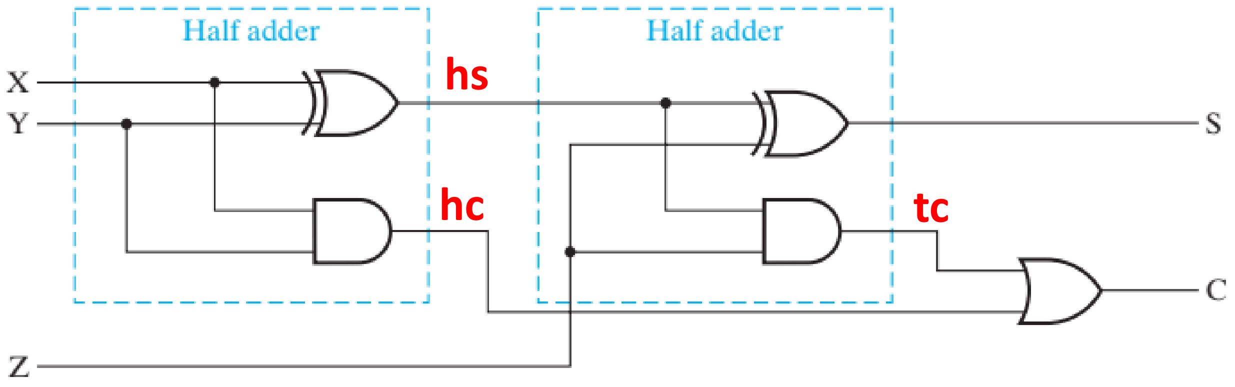


FIGURE 3-42
Logic Diagram of Full Adder

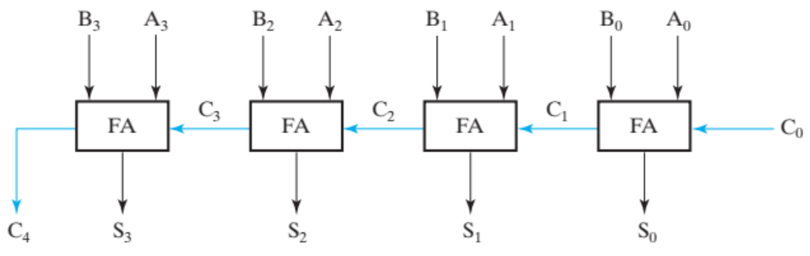
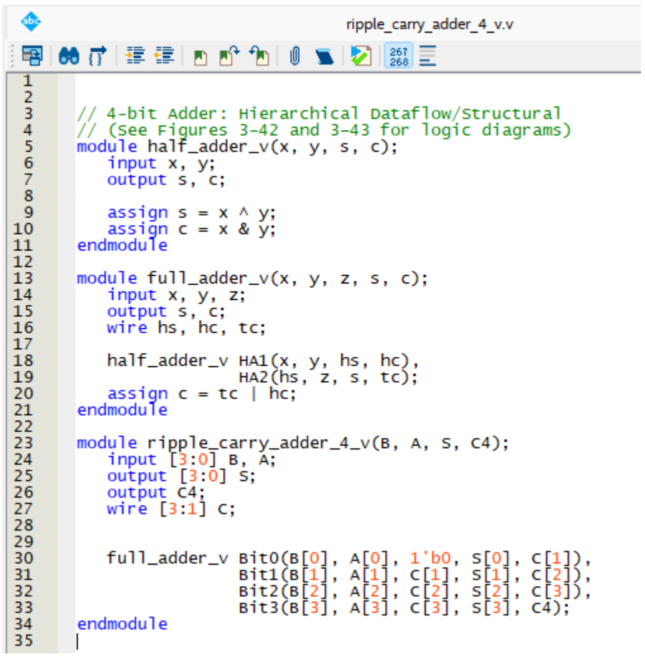

B_3 A_3 B_2 A_2 B_1 A_1 B_0 A_0
FA FA FA FA
C_3 C_2 C_1 C_0
C_4 S_3 S_2 S_1 S_0

FIGURE 3-43
4-Bit Ripple Carry Adder

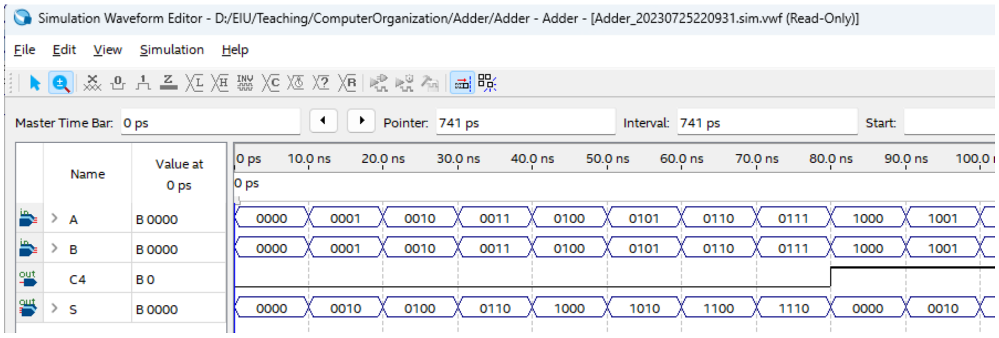


ripple_carry_adder_4_v.v

```verilog

// 4-bit Adder: Hierarchical Dataflow/Structural
// (See Figures 3-42 and 3-43 for logic diagrams)
module half_adder_v(x, y, s, c);
   input x, y;
   output s, c;

   assign s = x ^ y;
   assign c = x & y;
endmodule

module full_adder_v(x, y, z, s, c);
   input x, y, z;
   output s, c;
   wire hs, hc, tc;

   half_adder_v HA1(x, y, hs, hc),
                HA2(hs, z, s, tc);
   assign c = tc | hc;
endmodule

module ripple_carry_adder_4_v(B, A, S, C4);
   input [3:0] B, A;
   output [3:0] S;
   output C4;
   wire [3:1] C;


   full_adder_v Bit0(B[0], A[0], 1'b0, S[0], C[1]),
                Bit1(B[1], A[1], C[1], S[1], C[2]),
                Bit2(B[2], A[2], C[2], S[2], C[3]),
                Bit3(B[3], A[3], C[3], S[3], C4);
endmodule

```

Simulate the 4-bit ripple carry adder using waveform editor tool

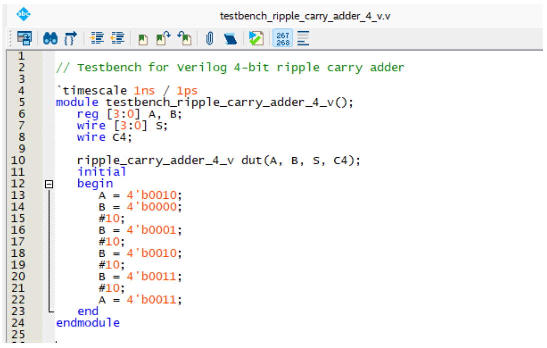
Test the adder with some input values.



Simulation Waveform Editor - D:/EIU/Teaching/ComputerOrganization/Adder/Adder - Adder - [Adder_20230725220931.sim.vwf (Read-Only)]

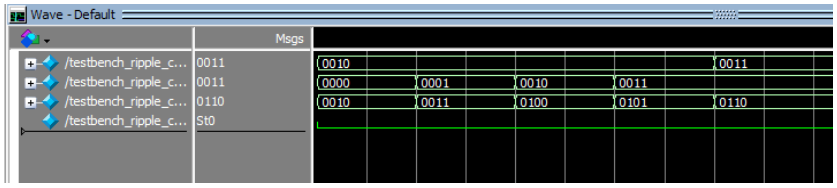
Master Time Bar: 0 ps   Pointer: 741 ps   Interval: 741 ps   Start:

```
Name   Value at 0 ps | 0 ps  10.0 ns  20.0 ns  30.0 ns  40.0 ns  50.0 ns  60.0 ns  70.0 ns  80.0 ns  90.0 ns  100.0
A      B 0000        | 0000   0001     0010     0011     0100     0101     0110     0111     1000     1001
B      B 0000        | 0000   0001     0010     0011     0100     0101     0110     0111     1000     1001
C4     B 0           | 0 until 80.0 ns, then 1
S      B 0000        | 0000   0010     0100     0110     1000     1010     1100     1110     0000     0010
```



testbench_ripple_carry_adder_4_v.v

```verilog

// Testbench for Verilog 4-bit ripple carry adder

`timescale 1ns / 1ps
module testbench_ripple_carry_adder_4_v();
   reg [3:0] A, B;
   wire [3:0] S;
   wire C4;

   ripple_carry_adder_4_v dut(A, B, S, C4);
   initial
   begin
      A = 4'b0010;
      B = 4'b0000;
      #10;
      B = 4'b0001;
      #10;
      B = 4'b0010;
      #10;
      B = 4'b0011;
      #10;
      A = 4'b0011;
   end
endmodule

```



Wave - Default

```
                          Msgs
/testbench_ripple_c...    0011   0010                                    0011
/testbench_ripple_c...    0011   0000       0001       0010       0011
/testbench_ripple_c...    0110   0010       0011       0100       0101       0110
/testbench_ripple_c...    St0
```

Assignment 2 (20 Points)

Design a logic diagram and write a Verilog program to implement 4-bit adder-subtractor.

Assignment 3 (Problem 3.3) (20 Points)

Design a Gray code–to–BCD code converter that gives output code 1111 for all invalid input combinations. Assume that the Gray code sequence for decimal numbers 0 through 9 is 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, and 1101. All other input combinations should be considered to be invalid.

a. Truth table

Decimal	Gray Code (Input)				BCD (Output)			
	A	B	C	D	X	Y	Z	W
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
3	0	0	1	0	0	0	1	1
2	0	0	1	1	0	0	1	0
7	0	1	0	0	0	1	1	1
6	0	1	0	1	0	1	1	0
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
Invalid	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
Invalid	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**
Invalid	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**
Invalid	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
Invalid	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**
Invalid	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Assignment 4 (Problem 3.7) (20 Points)

Modified version

A traffic light control at a simple intersection uses a binary counter to produce the following sequence of combinations on lines A, B, C, and D: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111. After 1111, the sequence repeats, beginning again with 0000, forever. Each combination is present for 5 seconds before the next one appears. These lines drive combinational logic with outputs to lamps RNS (red—north/south), YNS (yellow—north/south), GNS (green—north/south), REW (red—east/west), YEW (yellow—east/west), and GEW (green—east/west). The lamp controlled by each output is ON for a 1 applied and OFF for a 0 applied. **For a given direction, assume that green is on for 30 seconds, yellow for 5 seconds, and red for 45 seconds**. (The red intervals overlap for 5 seconds.) Divide the 80 seconds available for the cycle through the 16 combinations into 16 intervals and determine which lamps should be lit in each interval based on expected driver behavior.

Assume that, for interval 0001, a change has just occurred and that RNS = 1, GEW = 1, and all other outputs are 0 **(this means that the lamp EW begins GREEN for 30 seconds, the lamp NS is RED)**. Design the logic to produce the six outputs using AND, OR gates and inverters.

a. Truth table (modified version – yellow-highlighted text)

Inputs				Outputs					
A	B	C	D	**RNS**	**YNS**	**GNS**	**REW**	**YEW**	**GEW**
0	0	0	0	**1**	**0**	**0**	**1**	**0**	**0**
0	0	0	1	**1**	**0**	**0**	**0**	**0**	**1**
0	0	1	0	**1**	**0**	**0**	**0**	**0**	**1**
0	0	1	1	**1**	**0**	**0**	**0**	**0**	**1**
0	1	0	0	**1**	**0**	**0**	**0**	**0**	**1**
0	1	0	1	**1**	**0**	**0**	**0**	**0**	**1**
0	1	1	0	**1**	**0**	**0**	**0**	**0**	**1**
0	1	1	1	**1**	**0**	**0**	**0**	**1**	**0**
1	0	0	0	**1**	**0**	**0**	**1**	**0**	**0**
1	0	0	1	**0**	**0**	**1**	**1**	**0**	**0**
1	0	1	0	**0**	**0**	**1**	**1**	**0**	**0**
1	0	1	1	**0**	**0**	**1**	**1**	**0**	**0**
1	1	0	0	**0**	**0**	**1**	**1**	**0**	**0**
1	1	0	1	**0**	**0**	**1**	**1**	**0**	**0**
1	1	1	0	**0**	**0**	**1**	**1**	**0**	**0**
1	1	1	1	**0**	**1**	**0**	**1**	**0**	**0**

Assignment 5 (Problem 3.8) (20 Points)

Design a combinational circuit that accepts a 3-bit number and generates a 6-bit binary number output equal to the square of the input

a. Truth table

Inputs			Outputs					
A	B	C	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1