

521 Digital ASIC Design

Title: HW- 2

For

By: Saroj Shah

Date: Feb 18, 2026

HW-2, Professor Seetal Potluri

Introduction and Summary:

In this HW- 2 assignment, we are designing, and simulating fundamental combinational logic circuits using Verilog HDL and Cadence Xcelium. The following digital building blocks were implemented:

- 2-Input XOR Gate
- 1-Bit Full Adder
- 4-Bit Subtractor (using Full Adders)
- 4-Bit Unsigned Multiplier (using Full Adders)

All designs were written in Verilog at gate-level and structural modeling style.

Testbenches were developed for each module to verify correctness through simulation and waveform analysis in SimVision..

Materials:

- Computer system
- Cadence Xcelium (xrun)
- SimVision waveform viewer
- MobaXterm (remote terminal access)

Data:

Please refer to the screenshot provided below:

1. Two in-put XOR Gate:

The XOR gate was implemented using the Boolean equation:

$$Y = A'B + AB'$$

The design uses: 2 NOT gates; 2 AND gates; 1 OR gate

```

ss819183@ceashpc-12.rit.albany.edu
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help

Quick connect...
/network/rit/home/ss819183/cad...

[ss819183@ceashpc-12 ~]$ cd cadence
[ss819183@ceashpc-12 cadence]$ module load cadence/IC618-v2
[ss819183@ceashpc-12 cadence]$ xrun -compile HW2/XOR2.v
TOOL: xrun(64) 19.09-s001: Started on Feb 15, 2026 at 19:53:19 EST
xrun(64): 19.09-s001: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
file: HW2/XOR2.v
module worklib.XOR2.v
errors: 0, warnings: 0
TOOL: xrun(64) 19.09-s001: Exiting on Feb 15, 2026 at 19:53:19 EST (total: 00:00:00)
[ss819183@ceashpc-12 cadence]$ xrun -compile HW2/XOR2_tb.v
TOOL: xrun(64) 19.09-s001: Started on Feb 15, 2026 at 19:53:34 EST
xrun(64): 19.09-s001: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
file: HW2/XOR2_tb.v
module worklib.XOR2_tb.v
errors: 0, warnings: 0
TOOL: xrun(64) 19.09-s001: Exiting on Feb 15, 2026 at 19:53:34 EST (total: 00:00:00)
[ss819183@ceashpc-12 cadence]$ xrun -gui -access rwc HW2/XOR2.v HW2/XOR2_tb.v
TOOL: xrun(64) 19.09-s001: Started on Feb 15, 2026 at 19:53:48 EST
xrun(64): 19.09-s001: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
Caching library 'worklib' ..... Done
Elaborating the design hierarchy:
Top level design units:
XOR2_tb
Building instance overlay tables: ..... Done
Generating native compiled code:
worklib.XOR2_tb.v <8x7becb0e2>
streams: 5, words: 5614
Building instance specific data structures.
Loading native compiled code: ..... Done
Design hierarchy summary:
Instances Unique
Modules: 2 2
Primitives: 5 3
Registers: 2 2
Scalar wires: 2 -
Initial blocks: 2 2
Pseudo assignments: 2 2
Simulation timescale: 1ps
Writing initial simulation snapshot: worklib.XOR2_tb.v
waiting for SimVision/Indago to connect...

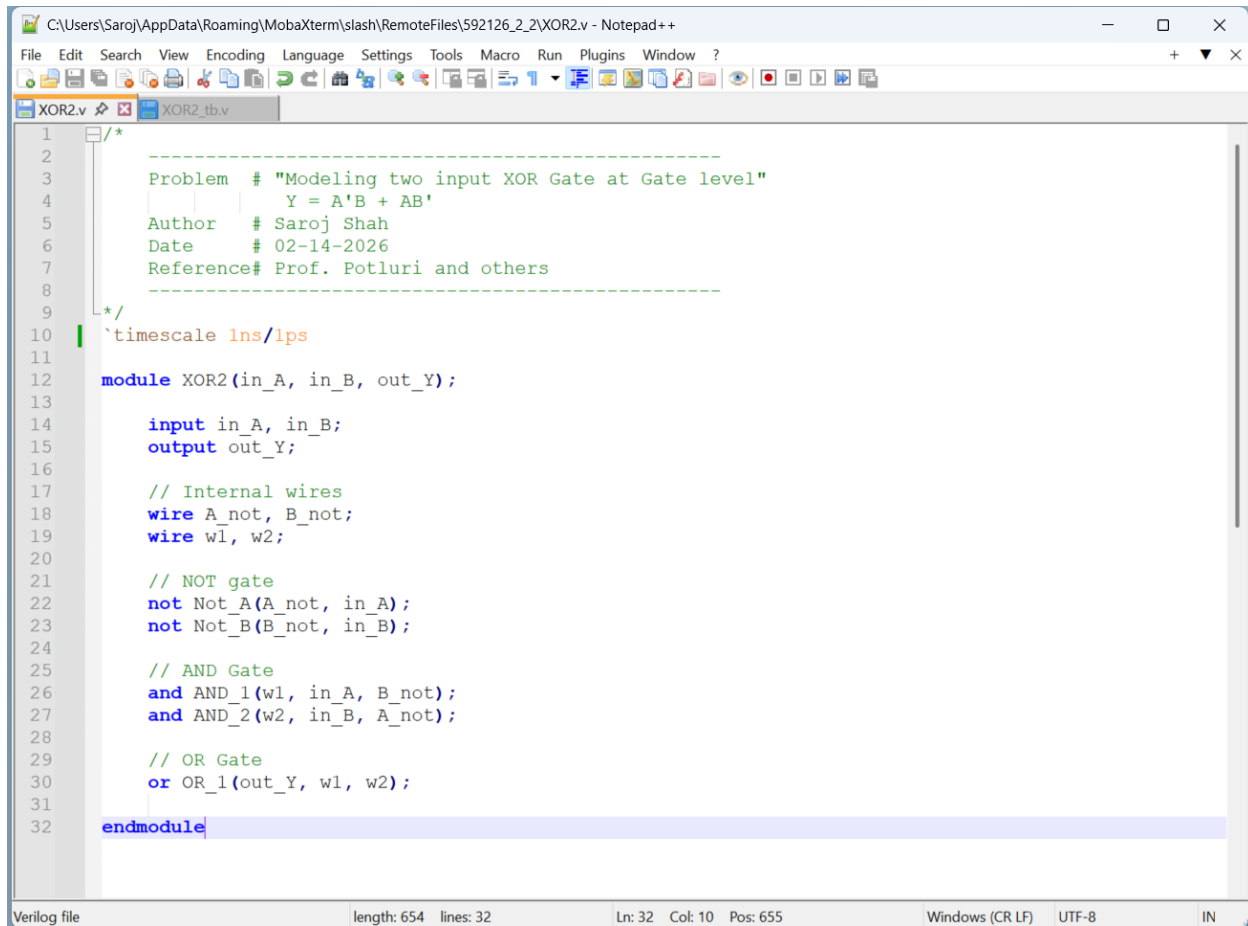
Relinquished control to SimVision...
xcelium> source /network/rit/lab/ceashpc/software/cadence/XCELIUM1909/tools/inca/files/xmsimrc
xcelium> error starting plugin /network/rit/lab/ceashpc/software/cadence/XCELIUM1909/tools/simvision/plugins/64bit/in
voke_ida.so
child process exited abnormally

```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>

Fig: 1 (MobaXterm window)

The above figure shows the code compiling and Cadence Xcelium running without any error.



```
1  /*
2
3  Problem # "Modeling two input XOR Gate at Gate level"
4          Y = A'B + AB'
5  Author  # Saroj Shah
6  Date    # 02-14-2026
7  Reference# Prof. Potluri and others
8  */
9
10 `timescale 1ns/1ps
11
12 module XOR2(in_A, in_B, out_Y);
13
14     input in_A, in_B;
15     output out_Y;
16
17     // Internal wires
18     wire A_not, B_not;
19     wire w1, w2;
20
21     // NOT gate
22     not Not_A(A_not, in_A);
23     not Not_B(B_not, in_B);
24
25     // AND Gate
26     and AND_1(w1, in_A, B_not);
27     and AND_2(w2, in_B, A_not);
28
29     // OR Gate
30     or OR_1(out_Y, w1, w2);
31
32 endmodule
```

Verilog file length: 654 lines: 32 Ln: 32 Col: 10 Pos: 655 Windows (CR LF) UTF-8 IN

Fig: 2A (Code for XOR2 Gate)

```

1  /*
2
3  Problem # "Testbench for two input XOR Gate"
4  Author  # Saroj Shah
5  Date    # 02-14-2026
6  Reference# Prof. Potluri and others
7  */
8
9
10 `timescale 1ns/1ps
11
12 module XOR2_tb;
13
14     reg in_A;
15     reg in_B;
16     wire out_Y;
17
18
19     // Instantiate
20     XOR2 XOR_gate(in_A, in_B, out_Y);
21
22     initial
23     begin
24         // Initializing inputs
25         in_A = 0;
26         in_B = 0;
27
28         #5 in_A = 1; in_B = 0;
29
30         #5 in_A = 0; in_B = 1;
31
32         #5 in_A = 1; in_B = 1;
33
34         #5 $finish; // End simulation
35     end
36
37     initial
38     $monitor("Time=%0t | in_A = %b, in_B = %b, out_Y = %b\n", $time, in_A, in_B, out_Y);
39
40 endmodule

```

Fig: 2B (Code for XOR2 Testbench Gate)

Figure 2A and 2B shows the code for XOR gate and XOR gate testbench. The testbench code is used to run the simulation.

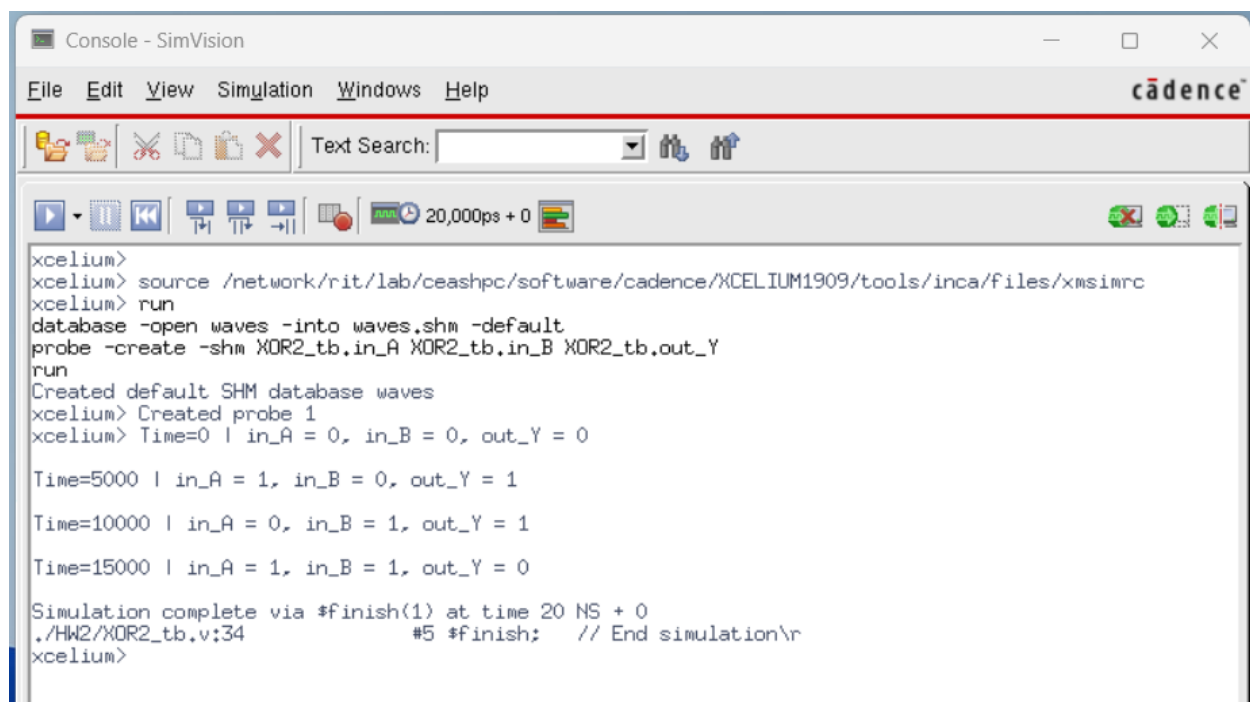
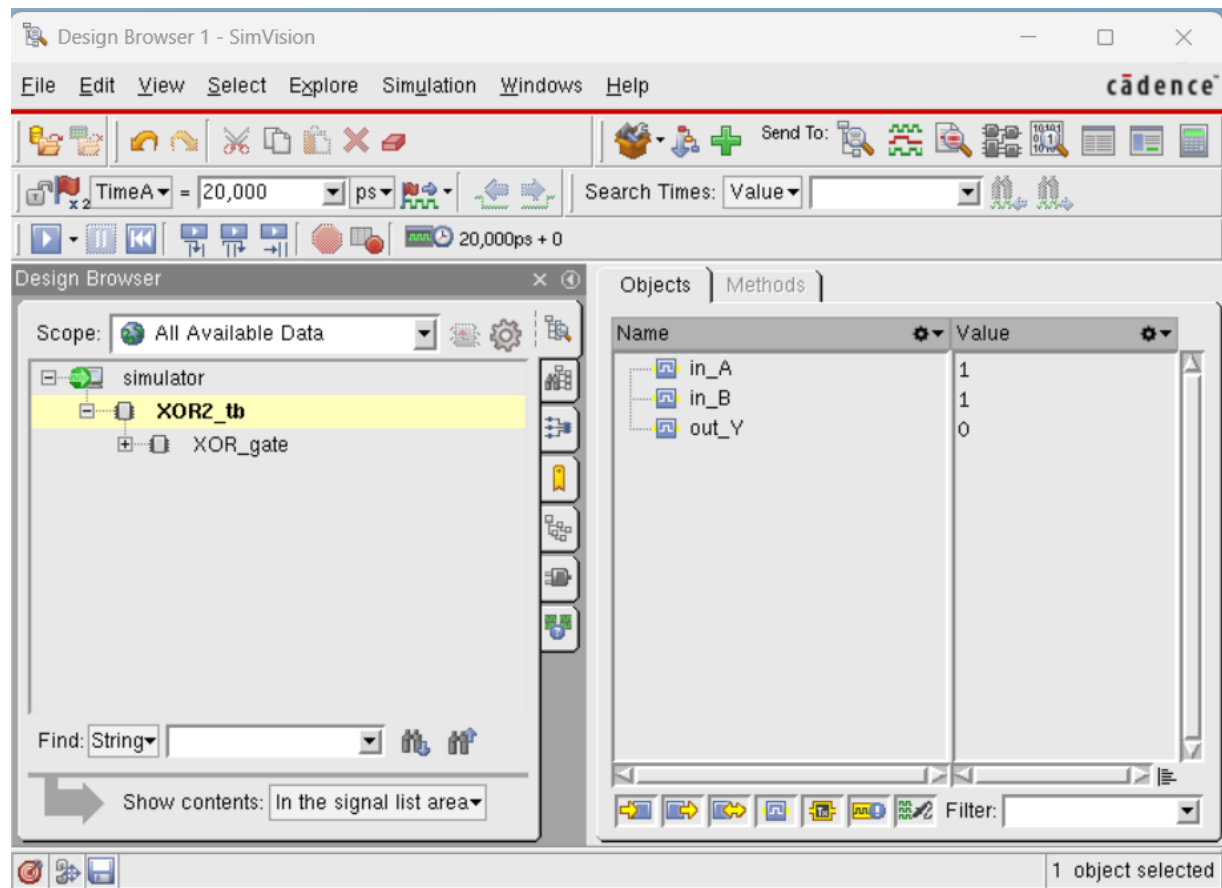


Fig: 3A (Simulation of XOR Gate)

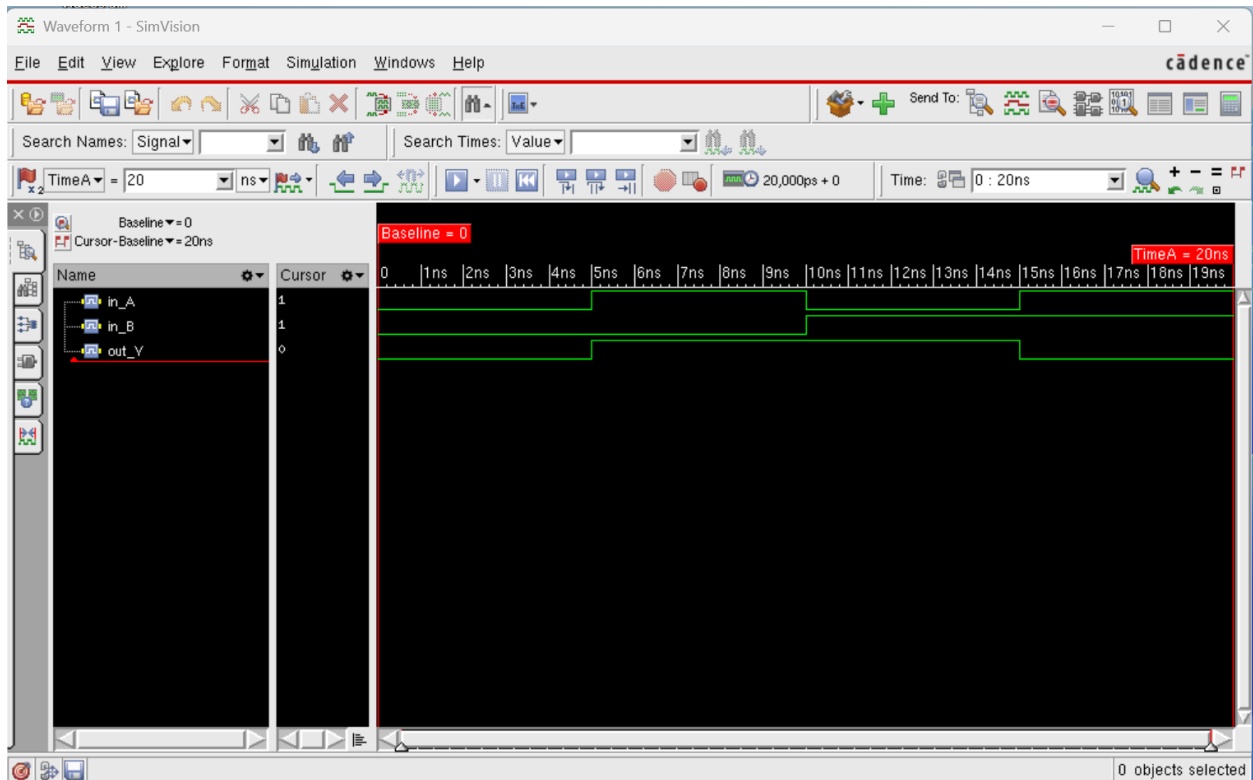


Fig: 3B (Waveform of XOR Gate)

The simulation ran successfully without compilation errors.

Waveform analysis confirms:

- When both inputs are equal (00 or 11), output = 0
- When inputs differ (01 or 10), output = 1

This verifies correct XOR behavior.

2. One-Bit Full adder:

The Full Adder was implemented using:

- $Sum = (A \oplus B) \oplus Cin$
- $Cout = A.B + (A \oplus B).Cin$

The design reuses the XOR2 module and basic AND/OR gates.

```

[ss819183@ceashpc-12 cadence]$ xrun -compile HW2/XOR2.v HW2/FA.v
T00L: xrun(64) 19.09-s001: Started on Feb 15, 2026 at 20:19:37 EST
xrun(64): 19.09-s001: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
file: HW2/FA.v
module worklib.FA:v
errors: 0, warnings: 0
T00L: xrun(64) 19.09-s001: Exiting on Feb 15, 2026 at 20:19:37 EST (total: 00:00:00)
[ss819183@ceashpc-12 cadence]$ xrun -compile HW2/XOR2.v HW2/FA tb.v
T00L: xrun(64) 19.09-s001: Started on Feb 15, 2026 at 20:20:47 EST
xrun(64): 19.09-s001: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
file: HW2/FA_tb.v
module worklib.FA_tb:v
errors: 0, warnings: 0
T00L: xrun(64) 19.09-s001: Exiting on Feb 15, 2026 at 20:20:47 EST (total: 00:00:00)
[ss819183@ceashpc-12 cadence]$ xrun -gui -access rwc HW2/XOR2.v HW2/FA.v HW2/FA_tb.v
T00L: xrun(64) 19.09-s001: Started on Feb 15, 2026 at 20:22:02 EST
xrun(64): 19.09-s001: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
file: HW2/FA_tb.v
module worklib.FA_tb:v
errors: 0, warnings: 0
Caching library 'worklib' ..... Done
Elaborating the design hierarchy:
Top level design units:
FA_tb
Building instance overlay tables: ..... Done
Generating native compiled code:
worklib.FA_tb:v <0x60bbe245>
streams: 6, words: 9146
Building instance specific data structures.
Loading native compiled code: ..... Done
Design hierarchy summary:
Instances Unique
Modules: 4 3
Primitives: 13 3
Registers: 3 3
Scalar wires: 3 -
Initial blocks: 2 2
Pseudo assignments: 3 3
Simulation timescale: 1ps
Writing initial simulation snapshot: worklib.FA_tb:v

```

Fig: 4 (MobaXterm window for running FA)

The above Figure 4 shows the MobaXterm window for running one-bit Full Adder.


```

1  /*
2
3  Problem # "Modeling 1-Bit Full Adder"
4          Sum = (A XOR B) XOR Cin
5          Cout= A.B + (A XOR B).Cin
6  Author  # Saroj Shah
7  Date    # 02-14-2026
8  Reference# Prof. Potluri and others
9
10 */
11 `timescale 1ns/1ps
12
13 module FA(in_A, in_B, in_Cin, out_Sum, out_Cout);
14
15     input in_A, in_B, in_Cin;
16     output out_Sum, out_Cout;
17
18     // Internal wires
19     wire w3;
20     wire w4, w5;
21
22     // XOR - Sum
23     XOR2 xor_a(in_A, in_B, w3);
24     XOR2 xor_b(w3, in_Cin, out_Sum);
25
26     // AND gate
27     and AND_3(w4, in_A, in_B);
28     and AND_4(w5, w3, in_Cin);
29
30     //OR gate => Carry out
31     or OR_2(out_Cout, w4, w5);
32
33 endmodule
34

```

Fig: 5A (Code for 1-Bit Full Adder)

```

1  /*
2
3  -----
4  Problem # Testbench for 1-Bit Full Adder
5  Author  # Saroj Shah
6  Date    # 02-14-2026
7  Reference# Prof. Potluri and others
8  -----
9  */
10 `timescale 1ns/1ps
11
12 module FA_tb;
13
14     reg in_A, in_B, in_Cin;
15     wire out_Sum, out_Cout;
16
17     // Instantiate
18     FA FA_1(in_A, in_B, in_Cin, out_Sum, out_Cout);
19
20     initial
21     begin
22         // All combinations
23         in_A=0; in_B=0; in_Cin=0;
24         #5 in_A=0; in_B=0; in_Cin=1;
25
26         #5 in_A=0; in_B=1; in_Cin=0;
27         #5 in_A=0; in_B=1; in_Cin=1;
28
29         #5 in_A=1; in_B=0; in_Cin=0;
30         #5 in_A=1; in_B=0; in_Cin=1;
31
32         #5 in_A=1; in_B=1; in_Cin=0;
33         #5 in_A=1; in_B=1; in_Cin=1;
34
35         #5 $finish;
36     end
37
38     initial
39     $monitor("\nTime=%0t | A=%b B=%b Cin=%b | Sum=%b Cout=%b",
40             $time, in_A, in_B, in_Cin, out_Sum, out_Cout);
41
42 endmodule
43

```

Fig: 5B (Testbench for 1-Bit Full Adder)

Figure 5A and 5B shows the code for one-bit Full Adder and one-bit Full Adder's testbench. The testbench code is used to running the simulation.

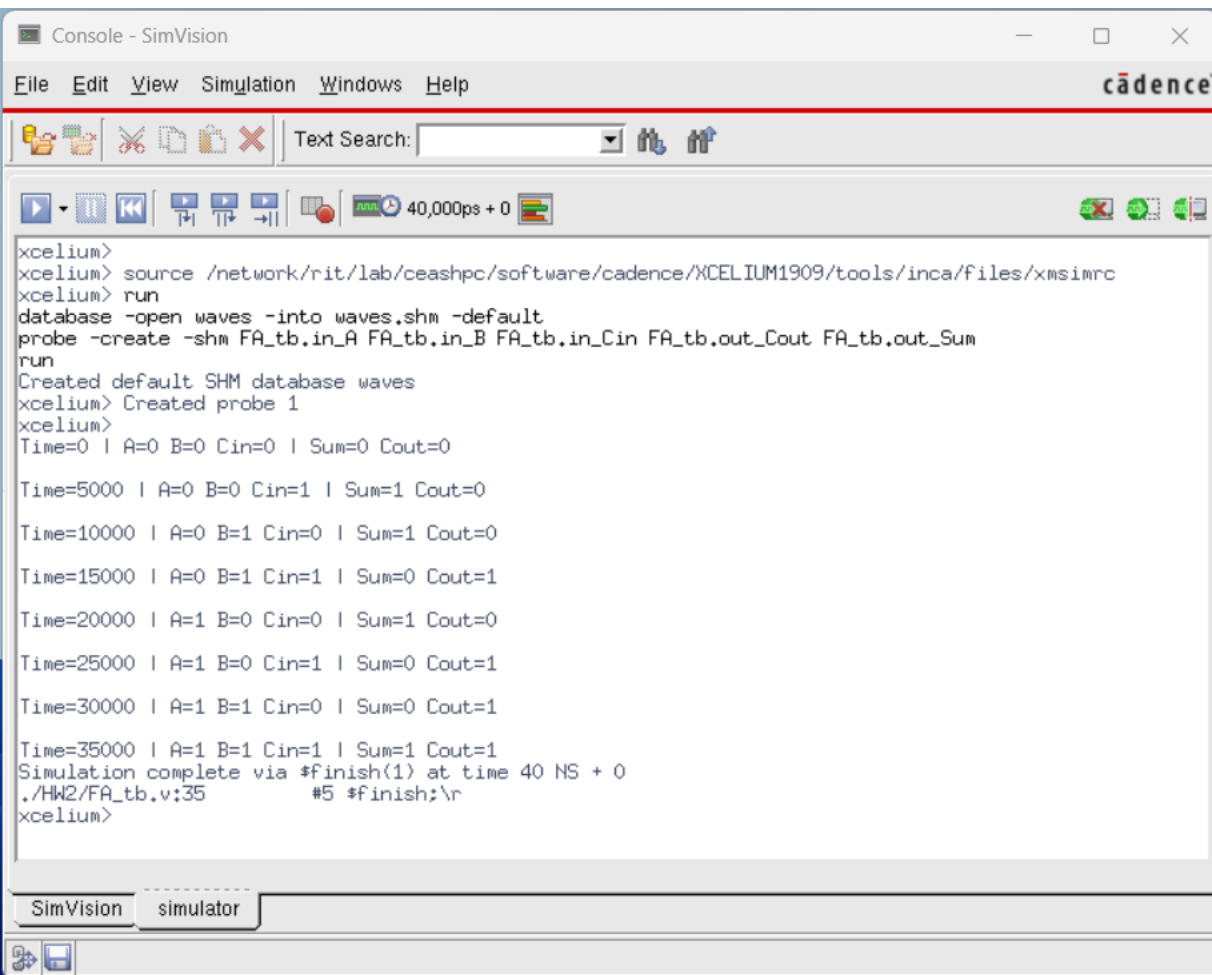
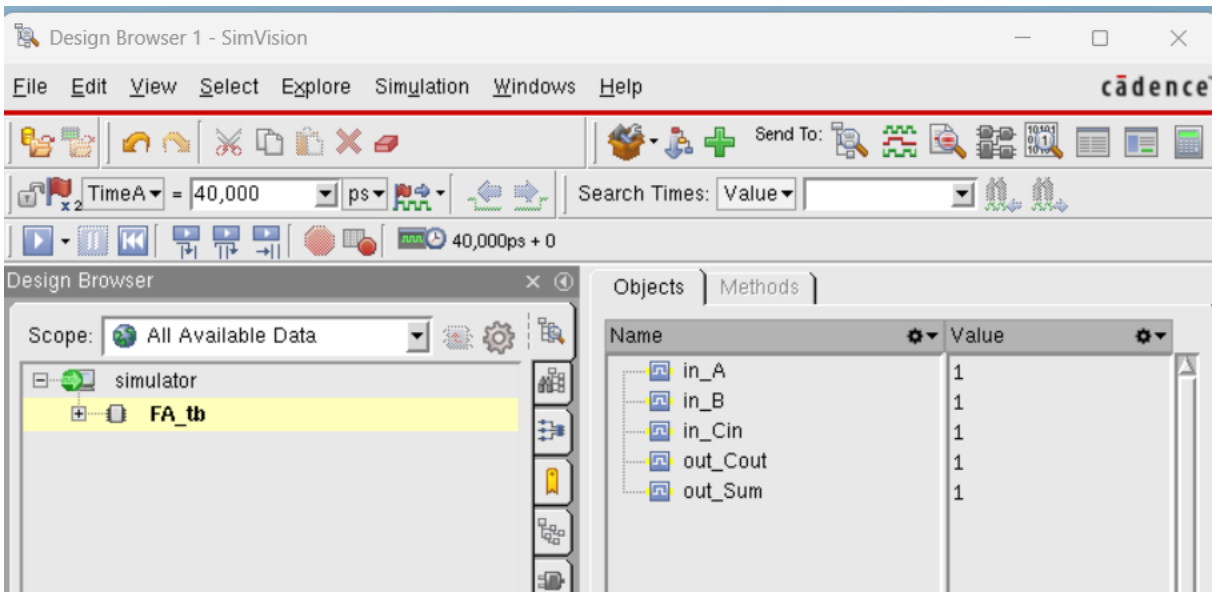


Fig: 6A (Simulation of 1-Bit Full Adder)

```

Time=0 | A=0 B=0 Cin=0 | Sum=0 Cout=0
Time=5000 | A=0 B=0 Cin=1 | Sum=1 Cout=0
Time=10000 | A=0 B=1 Cin=0 | Sum=1 Cout=0
Time=15000 | A=0 B=1 Cin=1 | Sum=0 Cout=1
Time=20000 | A=1 B=0 Cin=0 | Sum=1 Cout=0
Time=25000 | A=1 B=0 Cin=1 | Sum=0 Cout=1
Time=30000 | A=1 B=1 Cin=0 | Sum=0 Cout=1
Time=35000 | A=1 B=1 Cin=1 | Sum=1 Cout=1

```

Fig: 6B (Showing correct calculation)

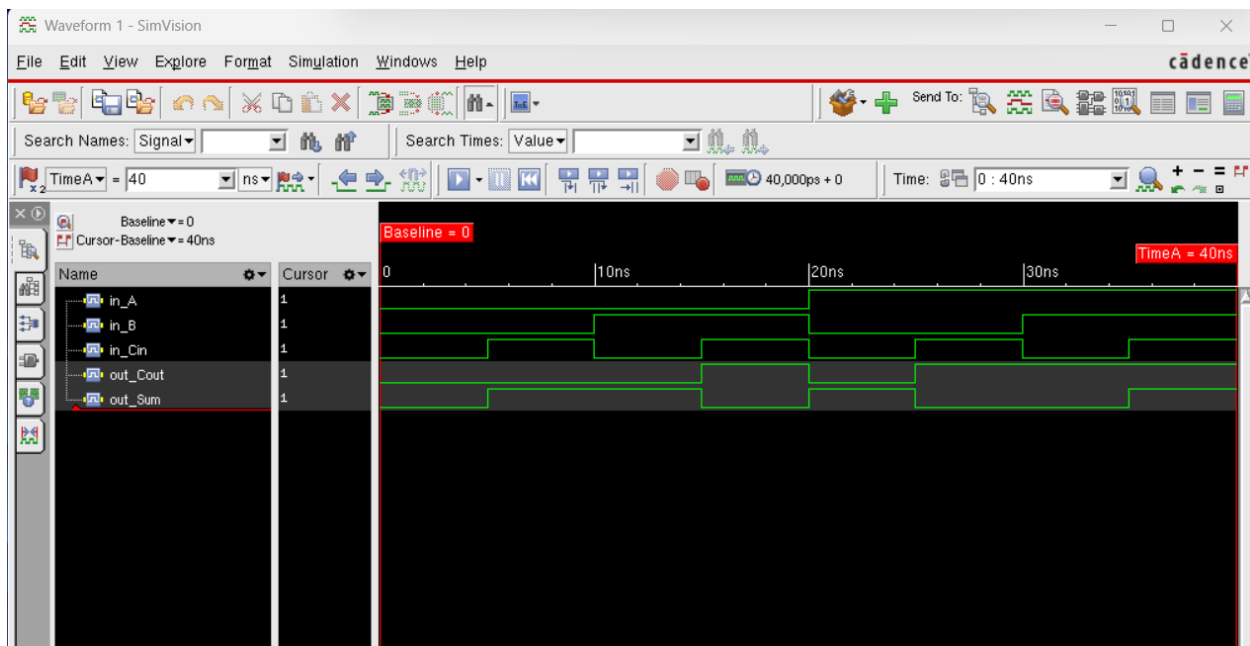


Fig: 6C (Waveform of 1-Bit Full Adder)

The simulation figures shown in figure 6A and 6B verifies correct Full Adder operation. The Sum output follows the XOR relationship of inputs, while the Carry output is asserted whenever at least two inputs are high. Same way the figure 6C, the waveforms clearly show correct propagation of carry and sum bits for all input combinations.

3. Four-Bit Full Subtractor (Using Full Adder):

Subtraction was implemented using 2's complement logic:

$$A - B = A + (B' + 1)$$

Implementation steps:

- Each bit of B is XORed with Bin (control signal).
- Bin is also connected as the initial carry-in.
- Four Full Adders are connected in ripple-carry fashion.
- Final borrow is obtained from the inverted final carry

```

[ss819183@ceashpc-12.rti.albany.edu]
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
/network/rit/home/ss819183/cad...
Name
..
.cadence
.simvision
.tmp_ss819183
HW2
logs_ss819183
sandbox
TechLibs
Test_DE4_1
Test_DE5_1
Test_DE5_2
Test_DE5_3
Test_DE5_4
Test_DE5_5
Test_DE5_6
waves.shm
Wed_class_prof
xcellium.d
.cdseiv
.cdseiv
basic_or.v
basic_or_tb.v
Cadence_bkup.tar.gz
cds.lib.disabled
libManager.log
pegasus_ui.log
pegasus_ui_log
Result_Test_DE5_4.grf
TechLibs_bkup.tar.gz

[ss819183@ceashpc-12 cadence]$ xrun -gui -access rwc HW2/XOR2.v HW2/FA.v HW2/Sub_4bit.v HW2/Sub_4bit_tb.v
TOOL: xrun(64) 19.09-s001: Started on Feb 15, 2026 at 22:19:11 EST
xrun(64): 19.09-s001: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
Recompiling... reason: file './HW2/Sub_4bit_tb.v' is newer than expected.
expected: Sun Feb 15 21:58:42 2026
actual: Sun Feb 15 22:19:00 2026
file: HW2/Sub_4bit_tb.v
module worklib.Sub_4bit_tb:v
errors: 0, warnings: 0
Caching library 'worklib' ..... Done
Elaborating the design hierarchy:
Top level design units:
Sub_4bit_tb
Building instance overlay tables: ..... Done
Generating native compiled code:
worklib.Sub_4bit_tb:v <0x50258ffc>
streams: 6, words: 8346
Building instance specific data structures.
Loading native compiled code: ..... Done
Design hierarchy summary:
Instances Unique
Modules: 18 4
Primitives: 73 3
Registers: 3 3
Scalar wires: 1 -
Expanded wires: 8 2
Initial blocks: 2 2
Pseudo assignments: 3 3
Simulation timescale: 1ps
Writing initial simulation snapshot: worklib.Sub_4bit_tb:v
waiting for SimVision/Indago to connect...

```

Fig: 7 (MobaXterm window)

The above Figure 4 shows the MobaXterm window for running Four-bit Subtractor.

```

1  /*
2  3  -----
3  4  Problem # "Modeling 4-Bit subtractor using FA"
4  5  Sum= (A XOR B) XOR Bin
5  6  Bout= A'.B + (A XOR B)'.Bin
6  7  Author # Saroj Shah
7  8  Date # 02-15-2026
8  9  Reference# Prof. Potluri and others
9  10 -----
10 11 */
11 12 `timescale 1ns/1ps
12 13 module Sub_4bit(in_A, in_B, in_Bin, out_D, out_Bout);
13 14     input [3:0] in_A;
14 15     input [3:0] in_B;
15 16     input      in_Bin;
16 17
17 18     output [3:0] out_D;
18 19     output      out_Bout;
19 20
20 21     // Internal wires
21 22     wire [3:0] B_xor;
22 23     wire c1, c2, c3, c4;
23 24
24 25     // XOR stage
25 26     XOR2 x0(in_B[0], in_Bin, B_xor[0]);
26 27     XOR2 x1(in_B[1], in_Bin, B_xor[1]);
27 28     XOR2 x2(in_B[2], in_Bin, B_xor[2]);
28 29     XOR2 x3(in_B[3], in_Bin, B_xor[3]);
29 30
30 31     // Ripple-carry Full Adders
31 32     FA FA0(in_A[0], B_xor[0], in_Bin, out_D[0], c1);
32 33     FA FA1(in_A[1], B_xor[1], c1, out_D[1], c2);
33 34     FA FA2(in_A[2], B_xor[2], c2, out_D[2], c3);
34 35     FA FA3(in_A[3], B_xor[3], c3, out_D[3], c4);
35 36
36 37     // Borrow output
37 38     not n1(out_Bout, c4);
38 39
39 40 endmodule

```

Fig: 8A (Code for 4-Bit Subtractor)

```

1  /*
2
3  -----
4  Problem   # "Testbench for 4-Bit subtractor"
5  Author    # Saroj Shah
6  Date      # 02-15-2026
7  Reference # Prof. Potluri and others
8  -----
9  */
10
11 `timescale 1ns/1ps
12
13 module Sub_4bit_tb;
14
15     reg [3:0] in_A;
16     reg [3:0] in_B;
17     reg      in_Bin;
18
19     wire [3:0] out_D;
20     wire      out_Bout;
21
22     // Instantiating
23     Sub_4bit sub1(in_A, in_B, in_Bin, out_D, out_Bout);
24
25     initial begin
26
27         /*
28         // When in_Bin = 0, it will do ADDITION
29         in_Bin = 0;
30
31         #10 in_A = 4'b0101; in_B = 4'b0011; // 5 + 3 = 8
32         #10 in_A = 4'b1001; in_B = 4'b0110; // 9 + 6 = 15
33         #10 in_A = 4'b0111; in_B = 4'b1000; // 7 + 8 = 15
34         */
35
36         // When in_Bin = 1, it will do SUBTRACTION
37         in_Bin = 1;
38
39         #10 in_A = 4'b1001; in_B = 4'b0011; // 9 - 3 = 6
40         #10 in_A = 4'b0101; in_B = 4'b0111; // 5 - 7 = -2 (Borrow=1)
41         #10 in_A = 4'b1000; in_B = 4'b1000; // 8 - 8 = 0
42         #10 in_A = 4'b0000; in_B = 4'b0001; // 0 - 1 = -1 (Borrow=1)
43         #10 $finish;
44     end
45
46     initial begin
47         $monitor("\nTime=%0t | Mode=%b | A=%0d(%b) B=%0d(%b) | Result=%0d(%b) Borrow=%b",
48             $time, in_Bin, in_A, in_A, in_B, in_B, out_D, out_D, out_Bout);
49     end
50 endmodule

```

Fig: 8B (Testbench code for 4-Bit Subtractor)

Figure 8A and 8B shows the code for 4-bit subtractor and 4-bit subtractor's testbench.

The testbench code is used to running the simulation. The figures are shown below.

The image displays two windows from the Cadence SimVision software interface.

Design Browser 1 - SimVision: This window shows the project hierarchy on the left with 'Sub_4bit_tb' selected under the 'simulator' scope. The right pane, titled 'Objects', lists the following signals and their current values:

Name	Value
in_A[3:0]	'h 0
in_B[3:0]	'h 1
in_Bin	1
out_Bout	1
out_D[3:0]	'h F

The bottom status bar indicates '1 object selected'.

Console - SimVision: This window shows the command-line interface of the xcelium simulator. The commands and their outputs are as follows:

```
xcelium>
xcelium> source /network/rit/lab/ceashpc/software/cadence/XCELIUM1909/tools/inca/files/xmsimrc
xcelium> run
database -open waves -into waves.shm -default
probe -create -shm Sub_4bit_tb,in_A Sub_4bit_tb,in_B Sub_4bit_tb,in_Bin Sub_4bit_tb,out_Bout Sub_4bit_tb,out_D
run
Created default SHM database waves
xcelium> Created probe 1
xcelium>
Time=0 | Mode=1 | A=x(0000) B=x(0000) | Result=x(0000) Borrow=x
Time=10000 | Mode=1 | A=9(1001) B=3(0011) | Result=6(0110) Borrow=0
Time=20000 | Mode=1 | A=5(0101) B=7(0111) | Result=14(1110) Borrow=1
Time=30000 | Mode=1 | A=8(1000) B=8(1000) | Result=0(0000) Borrow=0
Time=40000 | Mode=1 | A=0(0000) B=1(0001) | Result=15(1111) Borrow=1
Simulation complete via #finish(1) at time 50 NS + 0
./HW2/Sub_4bit_tb.v:42          #10 #finish:\r
xcelium>
```

Fig: 9A (Simulation of 4-Bit Subtractor)


```

Time=0 | Mode=1 | A=x(xxxx) B=x(xxxx) | Result=x(xxxx) Borrow=x
Time=10000 | Mode=1 | A=9(1001) B=3(0011) | Result=6(0110) Borrow=0
Time=20000 | Mode=1 | A=5(0101) B=7(0111) | Result=14(1110) Borrow=1
Time=30000 | Mode=1 | A=8(1000) B=8(1000) | Result=0(0000) Borrow=0
Time=40000 | Mode=1 | A=0(0000) B=1(0001) | Result=15(1111) Borrow=1

```

Fig: 9B (Result of 4-Bit Subtractor)

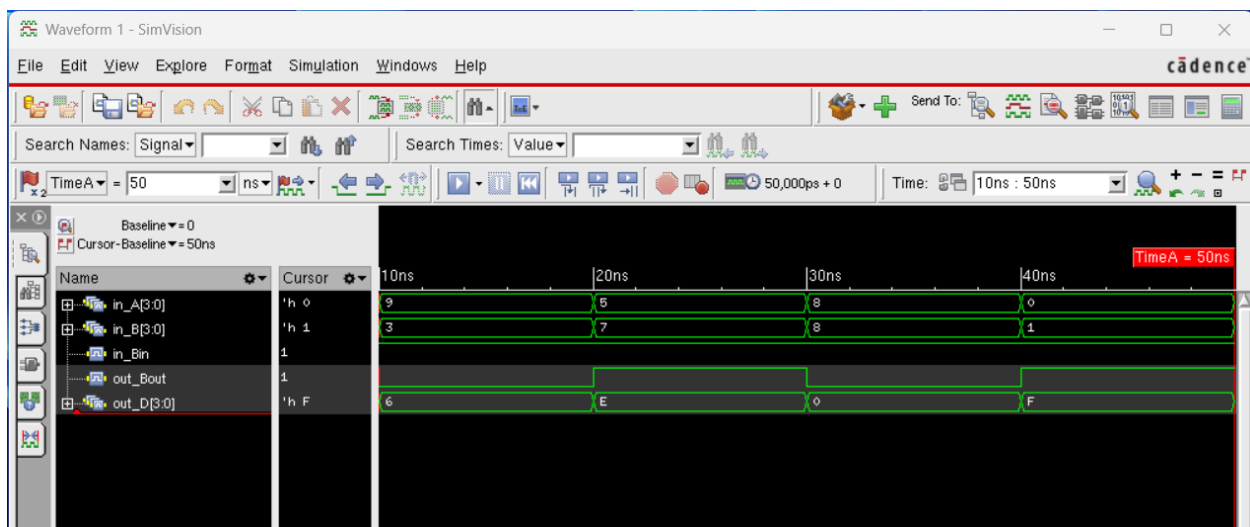


Fig: 9C (Waveform of 4-Bit Subtractor)

Test cases verified:

- $9 - 3 = 6$
- $5 - 7 = -2$ (Borrow = 1)
- $8 - 8 = 0$
- $0 - 1 = -1$ (Borrow = 1)

The simulation figures and waveform shown in figure 9A, 9B and 9C verifies correct operation and result of the Subtractor. The waveforms show correct difference bits and proper borrow generation. The subtractor correctly performs subtraction when Bin = 1.

4. Four-Bit Multiplier (extra credit):

The multiplier was implemented using:

- Partial products generated with AND gates
- Ripple addition using multiple Full Adders
- Column-wise addition of partial products

The output is an 8-bit product:

$$P = A \times B$$

```

T00L:  xrun(64)          19.09-s001: Started on Feb 16, 2026 at 00:14:00 EST
xrun(64): 19.09-s001: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
T00L:  xrun(64)          19.09-s001: Exiting on Feb 16, 2026 at 00:14:00 EST (total: 00:00:00)
[ss819183@ceashpc-12 cadence]$ xrun -gui -access rwc HW2/XOR2.v HW2/FA.v HW2/MULT4.v HW2/MULT4_tb.v
T00L:  xrun(64)          19.09-s001: Started on Feb 16, 2026 at 00:15:29 EST
xrun(64): 19.09-s001: (c) Copyright 1995-2019 Cadence Design Systems, Inc.
          Caching library 'worklib' ..... Done
          Elaborating the design hierarchy:
          Top level design units:
              MULT4_tb
          Building instance overlay tables: ..... Done
          Generating native compiled code:
              worklib.MULT4:v <0x6f56b050>
                  streams: 1, words: 368
              worklib.MULT4_tb:v <0x71d4cc8f>
                  streams: 5, words: 7366
          Building instance specific data structures.
          Loading native compiled code: ..... Done
          Design hierarchy summary:
              Instances  Unique
          Modules:      38      4
          Primitives:   172     3
          Registers:    2       2
          Scalar wires: 5       -
          Expanded wires: 8       2
          Initial blocks: 2       2
          Cont. assignments: 0     1
          Pseudo assignments: 3    3
          Simulation timescale: 1ps
          Writing initial simulation snapshot: worklib.MULT4_tb:v
          waiting for SimVision/Indago to connect...
  
```

Fig: 10 (MobaXterm window for 4-Bit Multiplier)

The above Figure 4 shows the MobaXterm window for running Four-bit Multiplier

```

C:\Users\Saroj\AppData\Roaming\MobaXterm\slash\RemoteFiles\985264_2_6\MULT4.v - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
MULT4.v MULT4_tb.v
10
11 module MULT4(in_A, in_B, out_P);
12     input  [3:0] in_A;
13     input  [3:0] in_B;
14     output [7:0] out_P;
15
16     // Internal Wires
17     wire c1_1;
18     wire s2_1, c2_1, c2_2;
19     wire s3_1, s3_2;
20     wire c3_1, c3_2, c3_3;
21     wire s4_1, s4_2;
22     wire c4_1, c4_2, c4_3;
23     wire s5_1;
24     wire c5_1, c5_2;
25
26     // Partial Products
27     wire p00,p01,p02,p03;
28     wire p10,p11,p12,p13;
29     wire p20,p21,p22,p23;
30     wire p30,p31,p32,p33;
31
32     and (p00, in_A[0], in_B[0]);
33     and (p01, in_A[1], in_B[0]);
34     and (p02, in_A[2], in_B[0]);
35     and (p03, in_A[3], in_B[0]);
36
37     and (p10, in_A[0], in_B[1]);
38     and (p11, in_A[1], in_B[1]);
39     and (p12, in_A[2], in_B[1]);
40     and (p13, in_A[3], in_B[1]);
41
42     and (p20, in_A[0], in_B[2]);
43     and (p21, in_A[1], in_B[2]);
44     and (p22, in_A[2], in_B[2]);
45     and (p23, in_A[3], in_B[2]);
46
47     and (p30, in_A[0], in_B[3]);
48     and (p31, in_A[1], in_B[3]);
49     and (p32, in_A[2], in_B[3]);
50     and (p33, in_A[3], in_B[3]);
51
52     // Output bit 0
53     assign out_P[0] = p00;
54
55     // Column 1
56     FA FA1(p01, p10, 1'b0, out_P[1], c1_1);
57
58     // Column 2
59     FA FA2(p02, p11, c1_1, s2_1, c2_1);
60     FA FA3(s2_1, p20, 1'b0, out_P[2], c2_2);
61
62     // Column 3
63     FA FA4(p03, p12, c2_1, s3_1, c3_1);
64     FA FA5(s3_1, p21, c2_2, s3_2, c3_2);
65     FA FA6(s3_2, p30, 1'b0, out_P[3], c3_3);
66
67     // Column 4
68     FA FA7(p13, p22, c3_1, s4_1, c4_1);
69     FA FA8(s4_1, p31, c3_2, s4_2, c4_2);
70     FA FA9(s4_2, c3_3, 1'b0, out_P[4], c4_3);
71
72     // Column 5
73     FA FA10(p23, p32, c4_1, s5_1, c5_1);
74     FA FA11(s5_1, c4_2, c4_3, out_P[5], c5_2);
75
76     // Column 6
77     FA FA12(p33, c5_1, c5_2, out_P[6], out_P[7]);
78
79 endmodule

```

Fig: 11A (Code for 4-Bit Multiplier)

```

1  /*
2  -----
3  Problem # Testbench for 4-bit Unsigned Multiplier using Full Adders
4  Author  # Saroj Shah
5  Date    # 02-15-2026
6  Reference# Prof. Potluri and others
7  -----
8  */
9  `timescale 1ns/1ps
10
11 module MULT4_tb;
12
13     reg [3:0] in_A;
14     reg [3:0] in_B;
15     wire [7:0] out_P;
16
17     MULT4 mult_1(in_A, in_B, out_P);
18
19     initial
20
21 begin
22     in_A = 0; in_B = 0;
23     #10 in_A = 4'b0011; in_B = 4'b0010; // 3 * 2 = 6
24     #10 in_A = 4'b0101; in_B = 4'b0011; // 5 * 3 = 15
25     #10 in_A = 4'b0111; in_B = 4'b0100; // 7 * 4 = 28
26     #10 in_A = 4'b1001; in_B = 4'b1001; // 9 * 9 = 81
27     #10 in_A = 4'b1111; in_B = 4'b1111; // 15 * 15 = 225
28     #10 $finish;
29 end
30
31     initial
32         $monitor("\nTime=%0t | A=%0d B=%0d | Product=%0d (%b)\n",
33                 $time, in_A, in_B, out_P, out_P);
34
35 endmodule
36

```

Fig: 11B (Testbench code for 4-Bit Multiplier)

Figure 11A shows the code for multiplier where and 11B shows code for simulation (i.e., its testbench code).

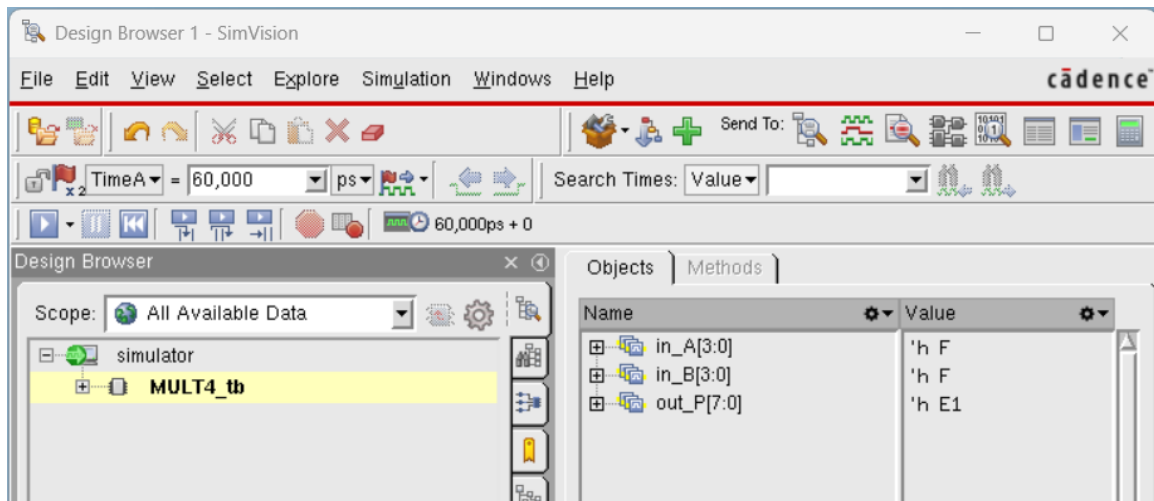
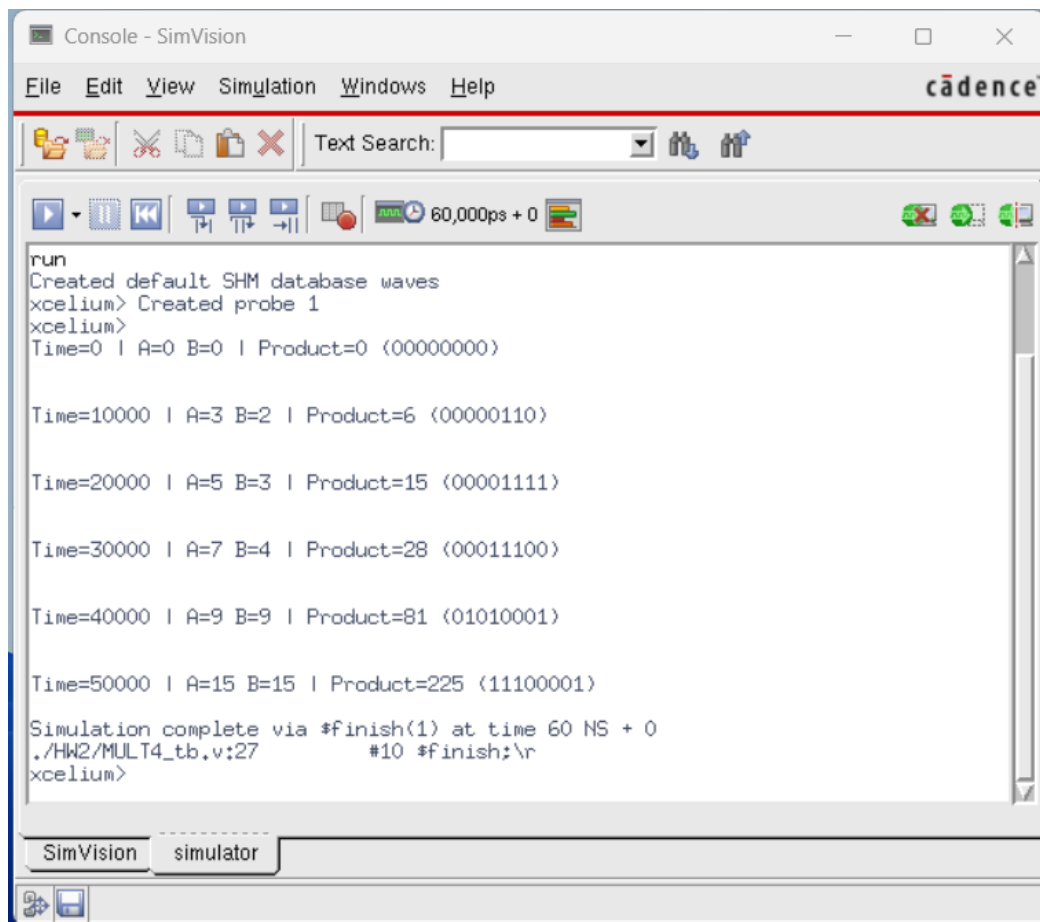


Fig: 12A (Simulation of 4-Bit Multiplier)



```

Time=0 | A=0 B=0 | Product=0 (00000000)

Time=10000 | A=3 B=2 | Product=6 (00000110)

Time=20000 | A=5 B=3 | Product=15 (00001111)

Time=30000 | A=7 B=4 | Product=28 (00011100)

Time=40000 | A=9 B=9 | Product=81 (01010001)

Time=50000 | A=15 B=15 | Product=225 (11100001)

```

Fig: 12B (Result of 4-Bit Multiplier)

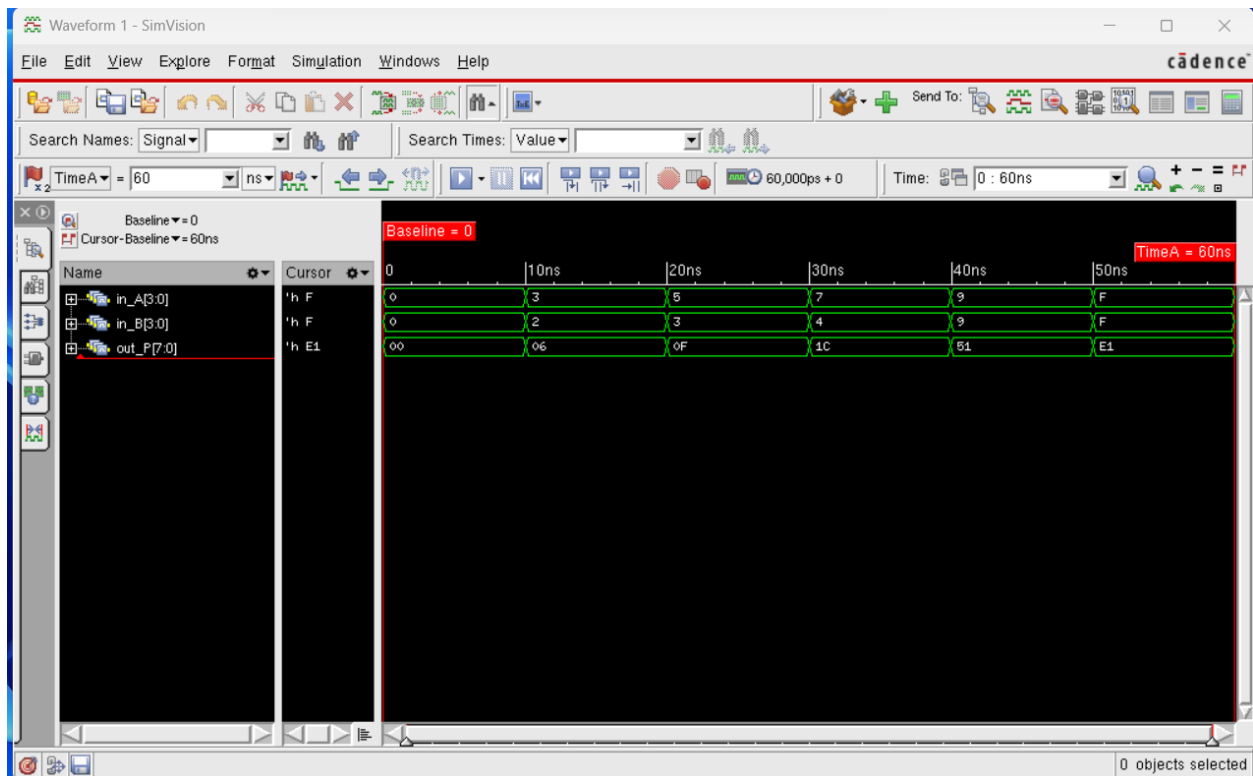


Fig: 12C (Waveform of 4-Bit Multiplier)

Verified test cases:

A	B	Product
3	2	6
5	3	15
7	4	28
9	9	81
15	15	225

The simulation figures and waveform shown in figure 12A, 12B and 12C verifies correct operation and result of the Multiplier. The waveforms also confirm correct bit propagation and addition across all columns. Also note the multiplier correctly generates an 8-bit unsigned result.

Conclusion:

In this assignment, several fundamental combinational logic circuits were successfully designed and simulated using structural Verilog modeling in Cadence Xcelium. The simulation waveforms verified the correct functional behavior for all modules.

Works Cited

Potluri, Seetal. Class Notes, PDF and videos, Spring 2026.