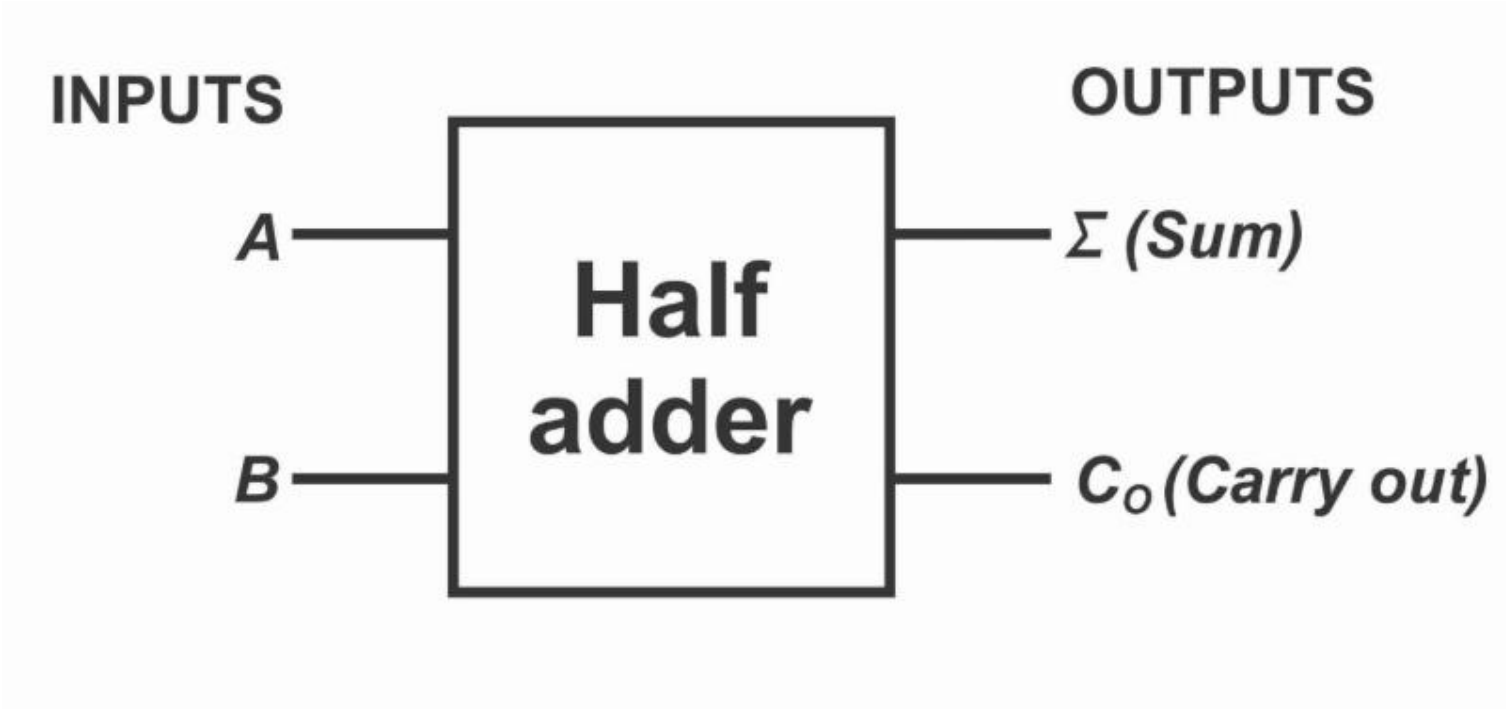
Lab3

Half Adder & Full Adder

Half Adder



Half Adder Design in SystemVerilog

- A **Half Adder** is the simplest type of digital adder that takes two input values and produces a **sum output** and a **carry output**. Unlike a **Full Adder**, a Half Adder does not have a **carry-in** input.

Truth Table for Half Adder

a	b	Sum (sum)	Carry (carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

SystemVerilog Code for Half Adder

C: > Users > Maryam > Desktop > Lab > fulladder > halfadder.sv > ...

```
1   module halfadder(
2       input  logic a, b,   // 1-bit inputs
3       output logic sum,    // 1-bit sum output
4       output logic carry   // 1-bit carry output
5   );
6
7       // Sum is XOR of inputs
8       assign sum = a ^ b;
9
10      // Carry is AND of inputs
11      assign carry = a & b;
12
13  endmodule
14
```

Inputs:

`a, b` → These are **single-bit binary numbers** that will be added together.

Outputs:

`sum` → Stores the **sum** of the two input bits.

`carry` → Stores the **carry-out** generated when both inputs are `1`.

✅ **Sum Calculation (`sum`)**

- The `sum` is calculated using the **XOR (`^`) operator**:

$$sum = a \oplus b$$

✅ **Carry Calculation (`carry`)**

- The `carry` is calculated using the **AND (`&`) operator**:

$$carry = a \cdot b$$

C: > Users > Maryam > Desktop > Lab > fulladder > halfadder.sv > ...

```
1  `timescale 1ns / 1ps
2  module tb();
3
4      // Input and Output Variables
5      logic a, b;       // Inputs
6      logic sum, carry; // Outputs
7
8      // Instantiate the Half Adder module
9      halfadder dut (
10         .a(a),
11         .b(b),
12         .sum(sum),
13         .carry(carry)
14     );
15
16     // Monitor signal values during execution
17     initial begin
18         $monitor("Time: %0t | a = %b, b = %b | sum = %b, carry = %b",
19                  $time, a, b, sum, carry);
20     end
21
22     // Apply all possible input combinations
23     initial begin
24         a = 0; b = 0; #10;
25         a = 0; b = 1; #10;
26         a = 1; b = 0; #10;
27         a = 1; b = 1; #10;
28
29         // End of simulation
30         $finish;
31     end
32
33 endmodule
34
```

Testbench Code

Explanation of the Testbench

✅ **Variable Definitions:**

- `a, b` are **single-bit inputs**.
- `sum, carry` are the **outputs computed by the Half Adder**.

✅ **Instantiation of the Half Adder Module:**

- `halfadder dut (...)` creates an instance of the **main module** and connects its inputs and outputs.

✅ **Displaying Values During Execution:**

- `$monitor(...)` prints the values of `a, b, sum, carry` whenever a change occurs.

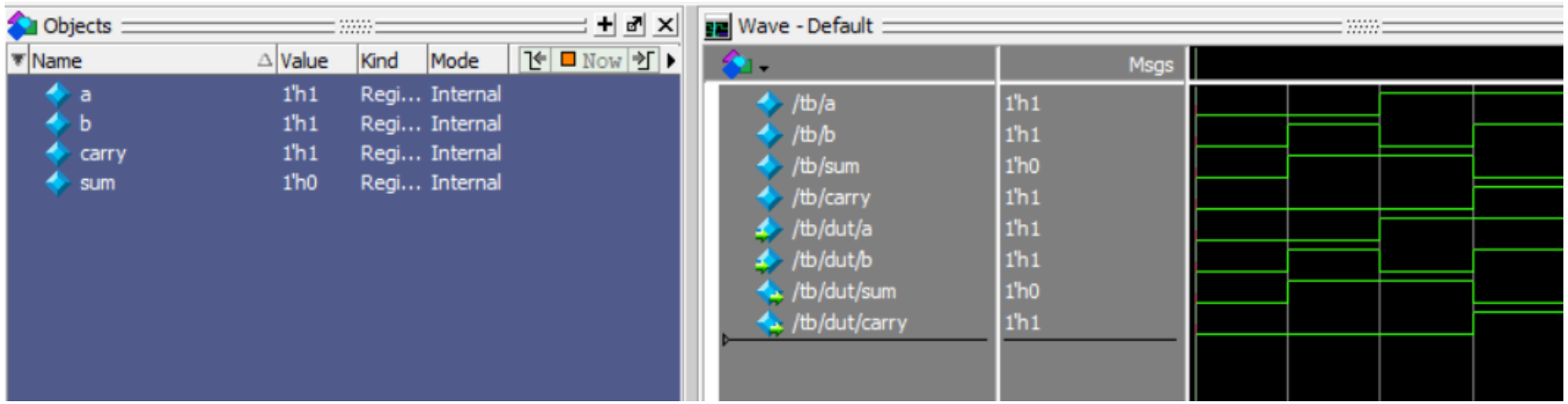
✅ **Applying Test Cases for All Possible Conditions:**

- The four possible input combinations `(a, b)` are tested:
  - `(0,0)`, `(0,1)`, `(1,0)`, `(1,1)`
- A **#10 delay** is added between each change in values.

✅ **Ending the Simulation:**

- After testing all cases, `$finish;` is executed to stop the simulation.

Simulation Waveform of Half Adder in ModelSim



Expected Output (Simulation Log)

bash

```
Time: 0  | a = 0, b = 0 | sum = 0, carry = 0
Time: 10 | a = 0, b = 1 | sum = 1, carry = 0
Time: 20 | a = 1, b = 0 | sum = 1, carry = 0
Time: 30 | a = 1, b = 1 | sum = 0, carry = 1
```

Full Adder

Full Adder Truth Table

Full Adder Truth Table

	a	b	c	Sum (s)	Carry (cout)
1	0	0	0	0	0
2	0	0	1	1	0
3	0	1	0	1	0
4	0	1	1	0	1
5	1	0	0	1	0
6	1	0	1	0	1
7	1	1	0	0	1
8	1	1	1	1	1

SystemVerilog Code for Full Adder

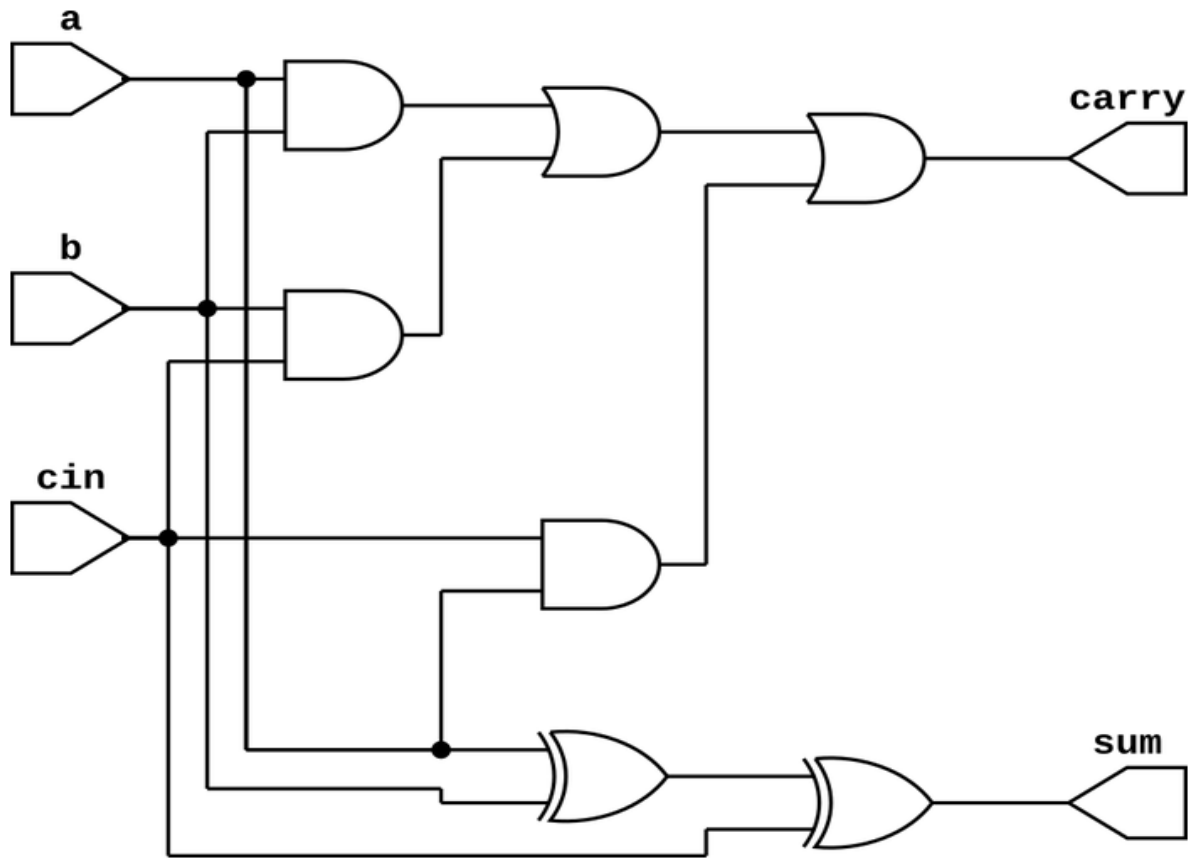
fulladder.sv × Extension: SystemVerilog-1800-2012

C: > Users > Maryam > Desktop > Lab > fulladder > fulladder.sv

```
1   module fulladder(
2       input  logic a, b, c,
3       output logic s, cout
4   );
5
6       // Sum calculation using XOR
7       assign s = a ^ b ^ c;
8
9       // Carry-out calculation using AND & OR gates
10      assign cout = (a & b) | (a & c) | (b & c);
11
12  endmodule
13
```

$$sum = a \oplus b \oplus cin$$

$$carry = a.b + b.cin + cin.a$$



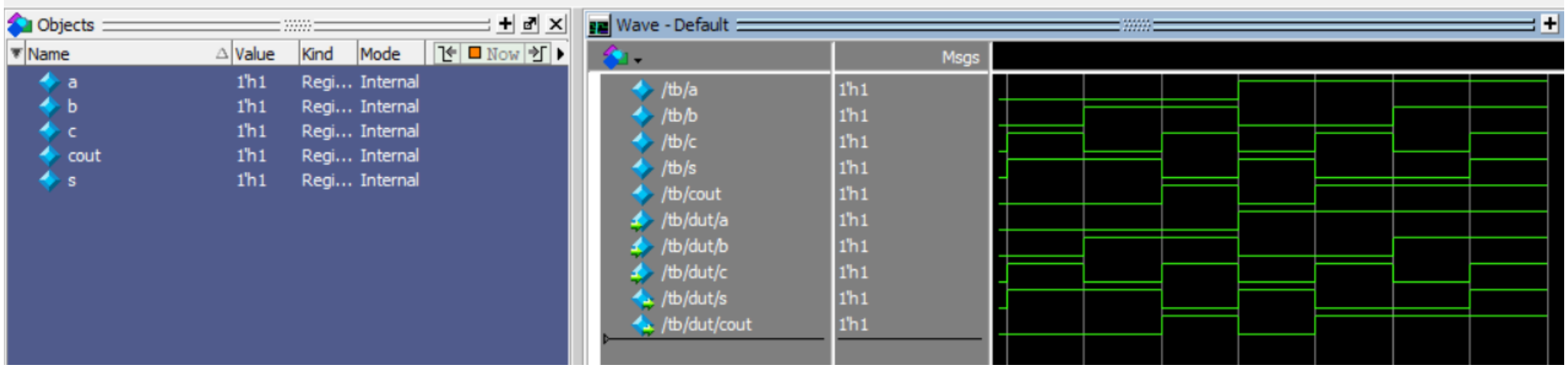
Testbench Code

fulladder_tb.sv

C: > Users > Maryam > Desktop > Lab > fulladder > fulladder_tb.sv

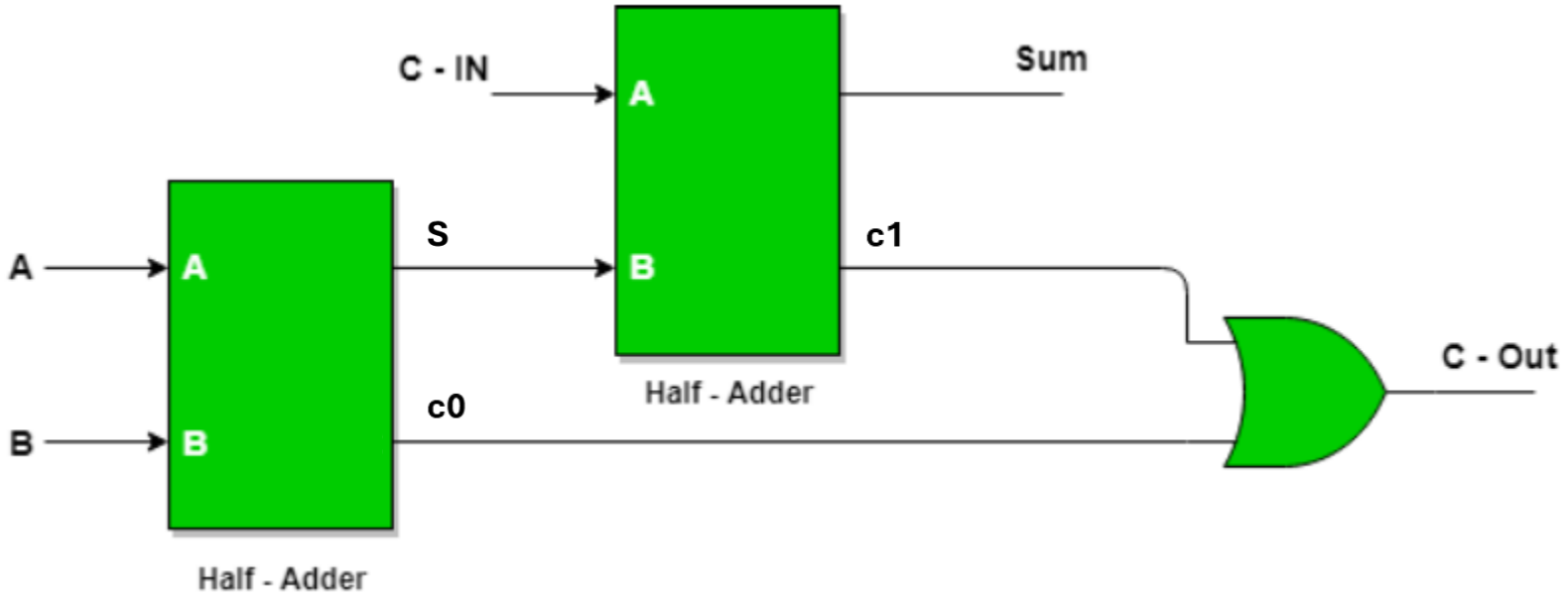
```
1  `timescale 1ns / 1ps
2
3  module tb();
4
5      // Input and Output Variables
6      logic a, b, c;   // Inputs
7      logic s, cout;   // Outputs
8
9      // Instantiating the Full Adder module
10     fulladder dut (
11         .a(a),
12         .b(b),
13         .c(c),
14         .s(s),
15         .cout(cout)
16     );
17
18     // Displaying signal values during execution
19     initial begin
20         $monitor("Time: %0t | a = %b, b = %b, c = %b | s = %b, cout = %b",
21                  $time, a, b, c, s, cout);
22     end
23
24     // Applying all possible input combinations
25     initial begin
26         a = 0; b = 0; c = 0; #10;
27         a = 0; b = 0; c = 1; #10;
28         a = 0; b = 1; c = 0; #10;
29         a = 0; b = 1; c = 1; #10;
30         a = 1; b = 0; c = 0; #10;
31         a = 1; b = 0; c = 1; #10;
32         a = 1; b = 1; c = 0; #10;
33         a = 1; b = 1; c = 1; #10;
34
35         // End of simulation
36         $finish;
37     end
38
39 endmodule
40
```

Full Adder Testbench Waveform Analysis



Implementation of Full Adder using Half Adders

A Full Adder can be implemented using two Half Adders and an OR gate. With this logic circuit, two bits can be added together, taking a carry from the next lower order of magnitude, and sending a carry to the next higher order of magnitude



Assignment: Implementing a Full Adder Using Two Half Adders

For the next assignment, please design a Full Adder using two Half Adders, as shown in the provided diagram.

- ❑ Your implementation should include:
- ❑ SystemVerilog Code for the Full Adder using two Half Adders.
- ❑ Testbench to verify the functionality.
- ❑ Truth Table covering all possible input combinations and expected outputs.
- ❑ Waveform Simulation to confirm correct operation.

Please ensure that your design follows the given structure and provides accurate results.