[1. Full Adder]

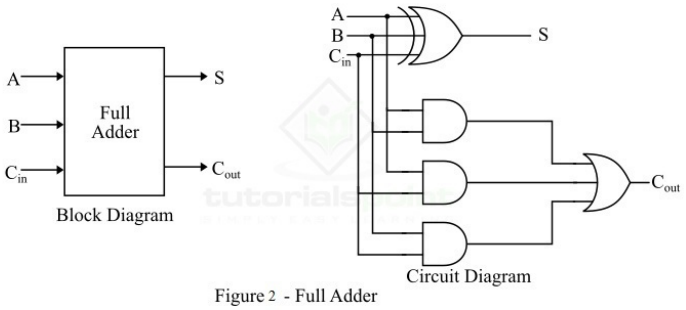


Figure 2 - Full Adder

$$\text{Sum, } S = A \oplus B \oplus C_{in}$$

$$\text{Carry, } C_{out} = Ab + AC_{in} + BC_{in}$$

Implementation of Full Adder using Half Adder

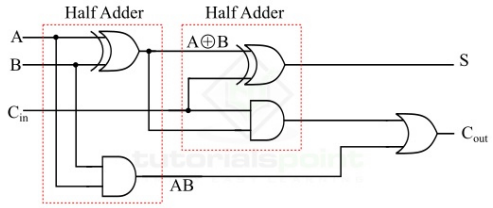


Figure 3 - Logic Diagram of Full Adder using Half Adder

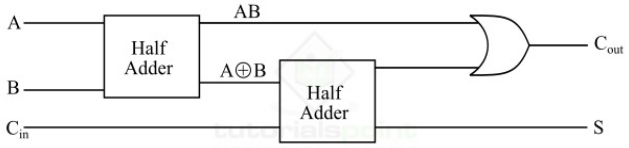


Figure 4 - Block Diagram of Full Adder using Half Adder

RTL

```
// ================================
// Full Adder Module 2 (using 2 Half Adders)
// This module implements a full adder using two half adder instances,
// taking three 1-bit bin inputs (a, b, cin) and producing a 1-bit sum and cout.
// ================================
module full_adder(a, b, cin, sum, cout);
  input a, b, cin;   // Input: a (First input bit), b (Second input bit), cin (Carry-in input)
  output sum, cout;  // Output: sum (Final sum), cout (Final carry-out)

  wire I1, I2, I3;    // Intermediate wires: I1 (sum from ha1), I2 (carry from ha1), I3 (carry from ha2)

  // Instantiate first Half Adder for a and b
  half_adder ha1(.A(a), .B(b), .S(I1), .C(I2));
  // Instantiate second Half Adder for I1 and cin
  half_adder ha2(.A(I1), .B(cin), .S(sum), .C(I3));

  // Final carry-out is OR of both carries
  assign cout = I2 | I3;
endmodule

// ================================
// Half Adder Module
// This module performs the basic half-adder logic,
// which computes the sum and carry-out of two 1-bit binary inputs.
// ================================
module half_adder (A, B, S, C);
  input A,B;          // A: First input bit, B: Second input bit
  output S,C;         // S(sum): XOR of A and B, C(carry-out): AND of A and B

  // Logic for sum and carry
  assign {C, S} = {A & B, A ^ B};
```

endmodule

```
// ================================
// Full Adder Module 2
// This module implements a full adder using two half adder instances,
// taking three 1-bit bin inputs (a, b, cin) and producing a 1-bit sum and cout.
// ================================
module full_adder (a, b, cin, sum, cout);
  input a, b, cin;      // Input: a (First input bit), b (Second input bit), cin (Carry-in input)
  output sum, cout;  // Output: sum (Final sum), cout (Final carry-out)

  // Logic for sum and carry
  // assign {cout, sum} = a + b + cin;
  // assign {cout, sum} = {((a & b) | (b & cin) | (a & cin)), (a ^ b ^ cin)};
  assign sum = a ^ b ^ cin;
  assign cout = (a & b) | (b & cin)  | (cin & a) ;
endmodule
```

TB

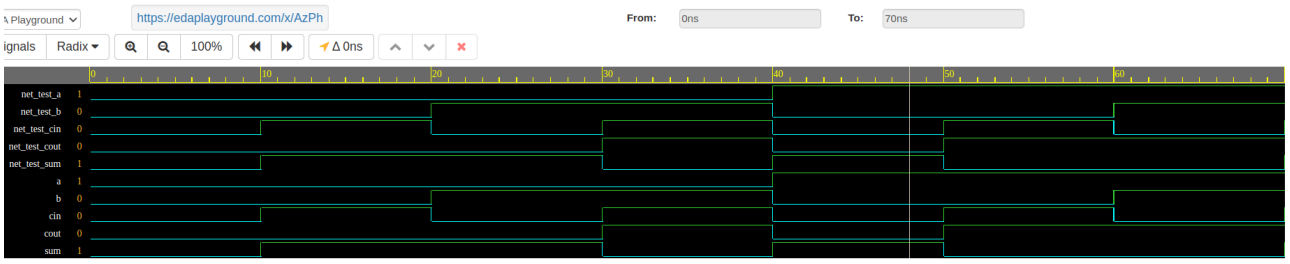
```
// ========================================
// Testbench for Full Adder Module
// This testbench verifies the behavior of the full_adder module
// by applying all possible 1-bit input combinations (2^3 = 8 cases) and monitoring outputs.
// ========================================
module tb_full_adder;
  reg net_test_a;        // Input a for testing: First input bit
  reg net_test_b;        // Input b for testing: Second input bit
  reg net_test_cin;      // Input cin for testing: Carry-in bit
  wire net_test_sum;  // Output sum for testing: Result of the addition
  wire net_test_cout;  // Output cout for testing: Carry-out bit

  // Instantiate Full Adder module
  full_adder fa_test (
    .a(net_test_a),           // Connect test input a to module input a
    .b(net_test_b),           // Connect test input b to module input b
    .cin(net_test_cin),      // Connect test input cin to module input cin
    .sum(net_test_sum),  // Connect module output sum to test wire
    .cout(net_test_cout)   // Connect module output cout to test wire
  );

  // Initial block to perform test cases
  initial begin
    // Create VCD file for waveform analysis
    $dumpfile("dump.vcd");
    $dumpvars;

    $display("FA - Full Adder Test Results");
    $display("-------------------------------");

    // Display signal values at every change
    $monitor("Time=%0t, a=%b, b=%b, cin = %b, sum=%b, carry=%b", $time, net_test_a, net_test_b, net_test_cin,
net_test_sum, net_test_cout);

    // Apply 8 input combinations
    net_test_a = 1'b0; net_test_b = 1'b0; net_test_cin = 1'b0; #10
    net_test_a = 1'b0; net_test_b = 1'b0; net_test_cin = 1'b1; #10

    net_test_a = 1'b0; net_test_b = 1'b1; net_test_cin = 1'b0; #10
    net_test_a = 1'b0; net_test_b = 1'b1; net_test_cin = 1'b1; #10
```

```
    net_test_a = 1'b1; net_test_b = 1'b0; net_test_cin = 1'b0; #10
    net_test_a = 1'b1; net_test_b = 1'b0; net_test_cin = 1'b1; #10

    net_test_a = 1'b1; net_test_b = 1'b1; net_test_cin = 1'b0; #10
    net_test_a = 1'b1; net_test_b = 1'b1; net_test_cin = 1'b1; #10

    // End simulation
    $finish;
  end
endmodule
```

Simulation Result

Log Share

```
# run 100 ns
# FA - Full Adder Test Results
# -------------------------------
# Time=0, a=0, b=0, cin = 0, sum=0, cout=0
# Time=10, a=0, b=0, cin = 1, sum=1, cout=0
# Time=20, a=0, b=1, cin = 0, sum=1, cout=0
# Time=30, a=0, b=1, cin = 1, sum=0, cout=1
# Time=40, a=1, b=0, cin = 0, sum=1, cout=0
# Time=50, a=1, b=0, cin = 1, sum=0, cout=1
# Time=60, a=1, b=1, cin = 0, sum=0, cout=1
# Time=70, a=1, b=1, cin = 1, sum=1, cout=1
# ** Note: $finish    : testbench.sv(48)
#    Time: 80 ns  Iteration: 0  Instance: /tb_full_adder
# End time: 10:28:54 on Jun 13,2025, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
End time: 10:28:54 on Jun 13,2025, Elapsed time: 0:00:01
```



[2. 16-bit full adder]

RTL

```
// ================================================================
// 16-bit Ripple Carry Adder Module
// This parameterized module constructs an N-bit ripple carry adder using N instances of the full_adder module.
// It takes two N-bit inputs (a and b), a carry-in (cin), and produces an N-bit sum and a final carry-out (cout).
// ================================================================
module _16bit_full_adder #(parameter N=16) (a, b, cin, sum, cout);
  input [N-1:0] a, b;  // Input: Two N-bit binary numbers (a and b)
  input cin;                    // Input: Carry-in bit
  output [N-1:0] sum;  // Output: N-bit sum of a and b with carry-in
  output cout;                  // Output: Final carry-out bit

  wire [N:0] c;         // Internal carry wires
  assign c[0] = cin;   // Initial carry-in

  genvar i;
```

```
  // Generate N full adder instances for each bit
  generate
    for (i = 0; i < N ; i = i + 1) begin : gen_fa
      full_adder fa (
        .a(a[i]),
        .b(b[i]),
        .cin(c[i]),
        .sum(sum[i]),
        .cout(c[i+1])
      );
    end
  endgenerate

  assign cout = c[N];  // Assign final carry-out
endmodule

// ================================
// Full Adder Module 2
// This module implements a full adder using two half adder instances,
// taking three 1-bit bin inputs (a, b, cin) and producing a 1-bit sum and cout.
// ================================
module full_adder (a,b,cin, sum,cout);
  input a,b,cin;
  output sum,cout;

  assign {cout, sum} = {((a & b) | (b & cin) | (a & cin)), (a ^ b ^ cin)};
endmodule
```

TB

```
// ============================================
// Testbench for 16-bit Full Adder Module
// This testbench verifies the behavior of the 16bit_full_adder module
// by applying multiple input combinations across 16 bits and monitoring outputs.
// ============================================
module tb_16bit_full_adder;
    reg [15:0] net_test_a;     // Input a for testing: 16-bit first input
    reg [15:0] net_test_b;     // Input b for testing: 16-bit second input
    reg net_test_cin;          // Input cin for testing: Carry-in bit
    wire [15:0] net_test_sum;  // Output sum for testing: 16-bit result of the addition
    wire net_test_cout;        // Output cout for testing: Carry-out bit

    // Instantiate 16-bit Full Adder module
    _16bit_full_adder _16b_fa_test (
        .a(net_test_a),         // Connect test input a to module input a
        .b(net_test_b),         // Connect test input b to module input b
        .cin(net_test_cin),     // Connect test input cin to module input cin
        .sum(net_test_sum),     // Connect module output sum to test wire
        .cout(net_test_cout)    // Connect module output cout to test wire
    );

    // Initial block to perform test cases
    initial begin
        // Create VCD file for waveform analysis
        $dumpfile("dump_16bit.vcd");
        $dumpvars;

        $display("16-bit FA - 16-bit Full Adder Test Results");
        $display("-------------------------------------------");

        // Display signal values at every change
```

```
        $monitor("Time=%0t, a=%h, b=%h, cin=%b, sum=%h, cout=%b", $time, net_test_a, net_test_b, net_test_cin,
net_test_sum, net_test_cout);

        // Apply multiple input combinations
        net_test_a = 16'h0000; net_test_b = 16'h0000; net_test_cin = 1'b0; #10;
        net_test_a = 16'h0001; net_test_b = 16'h0002; net_test_cin = 1'b1; #10;
        net_test_a = 16'hABCD; net_test_b = 16'hEF01; net_test_cin = 1'b0; #10;
        net_test_a = 16'hFFFF; net_test_b = 16'hFFFF; net_test_cin = 1'b1; #10;
        net_test_a = 16'h1234; net_test_b = 16'h5678; net_test_cin = 1'b0; #10;
        net_test_a = 16'h5555; net_test_b = 16'h5555; net_test_cin = 1'b1; #10;

        // End simulation
        $finish;
    end
endmodule
```

Simulation Result

Log | Share

```
# run 100 ns
# 16-bit FA - 16-bit Full Adder Test Results
# ------------------------------------------
# Time=0, a=0000, b=0000, cin=0, sum=0000, cout=0
# Time=10, a=0001, b=0002, cin=1, sum=0004, cout=0
# Time=20, a=abcd, b=ef01, cin=0, sum=9ace, cout=1
# Time=30, a=ffff, b=ffff, cin=1, sum=ffff, cout=1
# Time=40, a=1234, b=5678, cin=0, sum=68ac, cout=0
# Time=50, a=5555, b=5555, cin=1, sum=aaab, cout=0
# ** Note: $finish    : testbench.sv(43)
#    Time: 60 ns  Iteration: 0  Instance: /tb_16bit_full_adder
# End time: 11:17:23 on Jun 13,2025, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
End time: 11:17:23 on Jun 13,2025, Elapsed time: 0:00:01
```