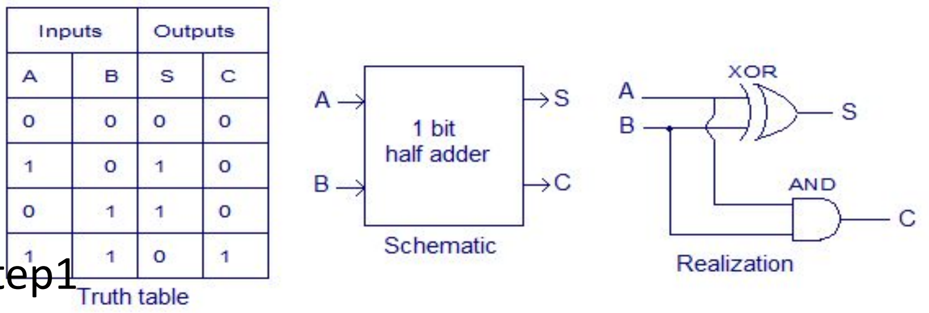
ADDERS

- Arithmetic circuits are the central building blocks of computers. Computers and digital logic perform many arithmetic functions: addition, subtraction, comparisons, shifts, multiplication, and division
- Addition is one of the most common operations in digital systems.
- We first consider how to add two 1-bit binary numbers. We then extend to

  N-bit binary numbers

Half adder

- A half-adder shows how two bits can be added together with a few simple logic gates.
- A single half-adder has two one-bit inputs, a sum output, and a carry-out output.



Inputs		Outputs	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Step1

```
module halfadd(a, b,sum, cout);
input a, b;
output sum, cout;
 xor2 x0(---------);
 and2 a0(--------);
endmodule
```

Half Adder Test Bench halfadder_tb.v

```
module halfadd_tb;
reg aa,bb;
wire ss,cy;
halfadd add1(.a(aa), .b(bb),  .sum(ss), .cout(cy));
initial
begin
$dumpfile("halfadd_test.vcd");
$dumpvars(0, halfadd_tb);
end
```

```
initial
 begin
$monitor($time, "a=%b, b=%b, sum=%b, carry=%b", aa, bb, ss, cy);
 aa = 1'b0;bb = 1'b0;
#5  aa = 1'b0;bb = 1'b1;
#5  aa = 1'b1;bb = 1'b0;
#5  aa = 1'b1;bb = 1'b1;
end
endmodule
```

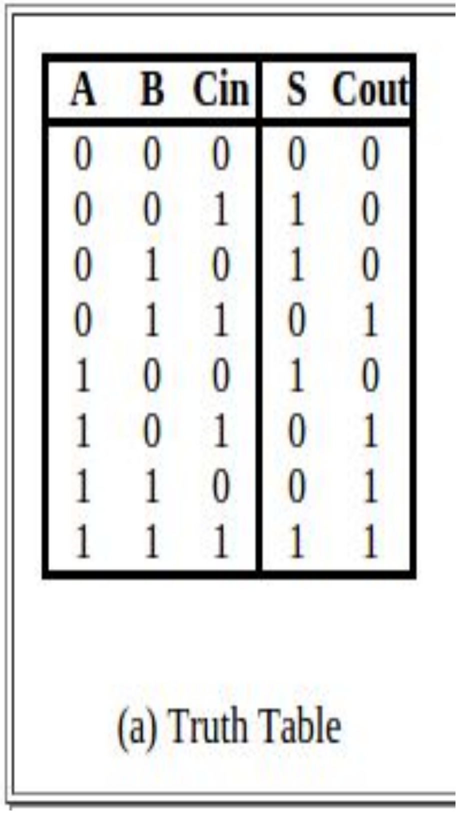
Simulation

Step1:iverilog -o test1 halfadder.v half_adder_tb.v

Step2:vvp test1

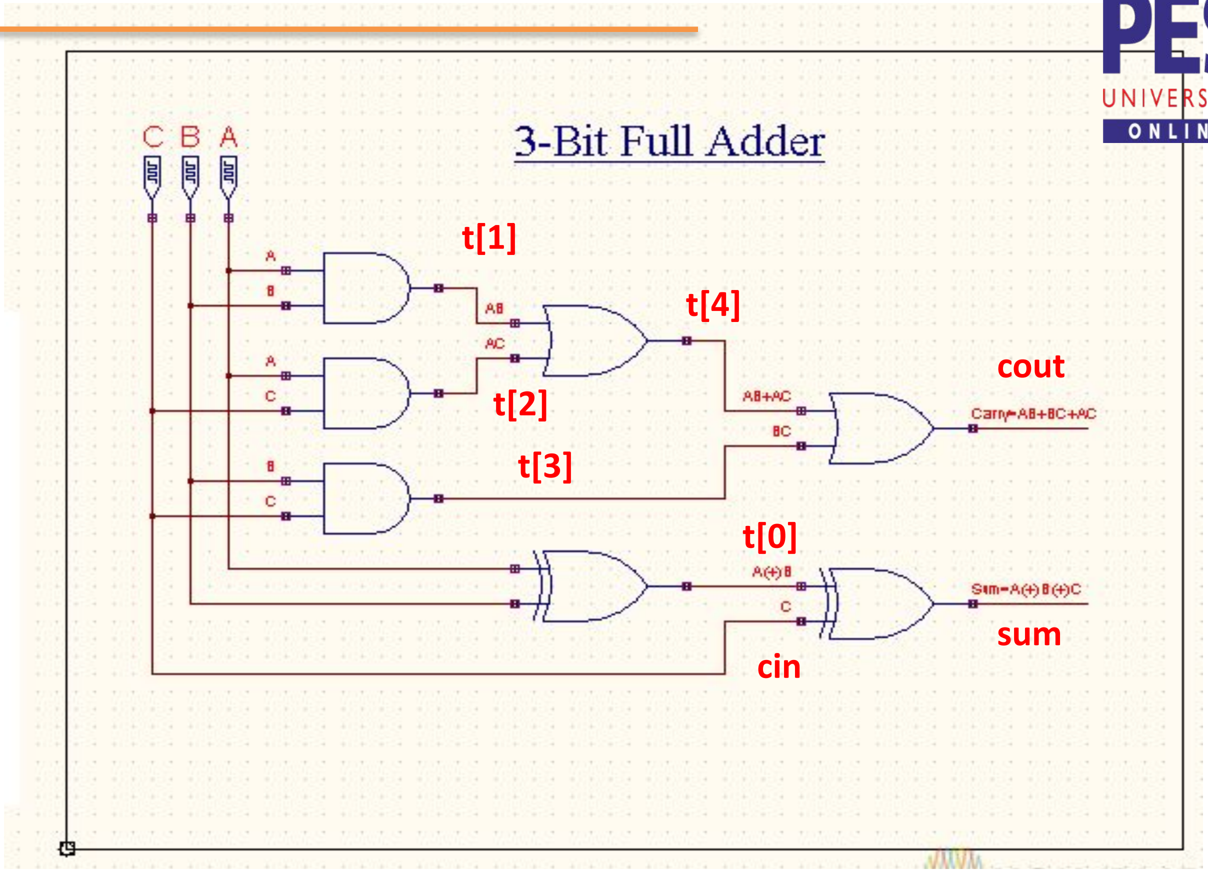
Step3: gtkwave halfadder_test.vcd

Full Adder



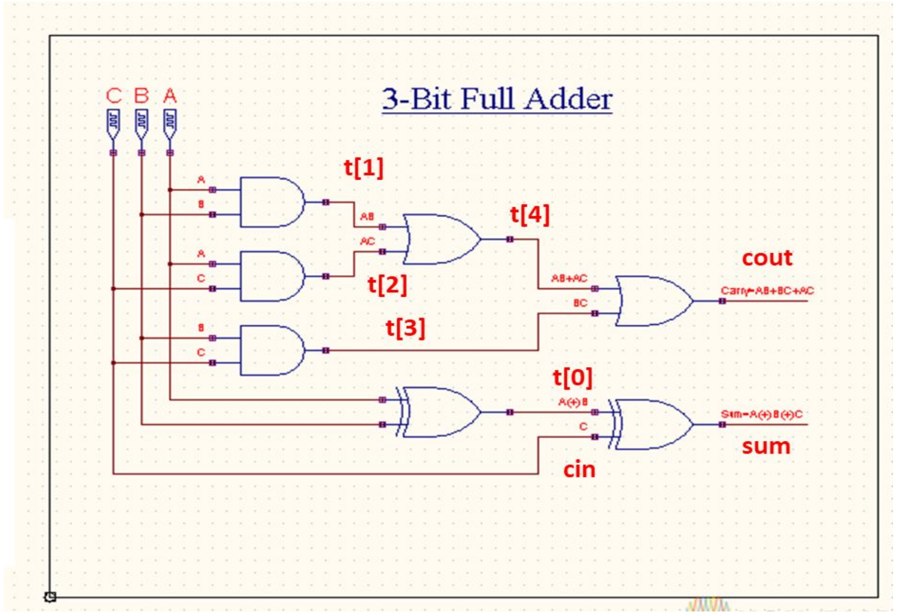
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Truth Table



Full Adder Verilog File fulladd.v

```
module fulladd(input wire a, b, cin, output wire sum, cout);
wire [4:0] t;
xor2 x0(----------);
xor2 x1(---------);
and2 a0(----------);
and2 a1(-----------);
and2 a2(-----------);
or2 o0(------------);
or2 o1(----------t);
endmodule
```



Full Adder Test Bench fulladd_tb.v

```
module fulladd_tb;
reg aa,bb,cc;
wire ss,cy;
fulladd add1(.a(aa), .b(bb), .cin(cc), .sum(ss), .cout(cy));
initial
begin
$dumpfile("fulladd_test.vcd");
$dumpvars(0, fulladd_tb);
end
initial
 begin
$monitor($time, "a=%b, b=%b, c=%b,sum=%b,carry=%b",aa,bb,cc,ss,cy);
```

Full Adder Test Bench(Cont......)

```
aa = 1'b0;bb = 1'b0;cc=1'b0;
#5  aa = 1'b0;bb = 1'b0;cc=1'b1;
#5  aa = 1'b0;bb = 1'b1;cc=1'b0;
#5  aa = 1'b0;bb = 1'b1;cc=1'b1;
#5  aa = 1'b1;bb = 1'b0;cc=1'b0;
#5  aa = 1'b1;bb = 1'b0;cc=1'b1;
#5  aa = 1'b1;bb = 1'b1;cc=1'b0;
#5  aa = 1'b0;bb = 1'b1;cc=1'b1;
end
endmodule
```

Full Adder Simulation (using basic gates)

Students have to complete the fulladd.v file

Execution Steps

Step1) iverilog -o test2 basic.v fulladder.v fulladder_tb.v

If the compilation went OK, you won't see any output. What this does is create a file called testfa that we can feed to the simulator.

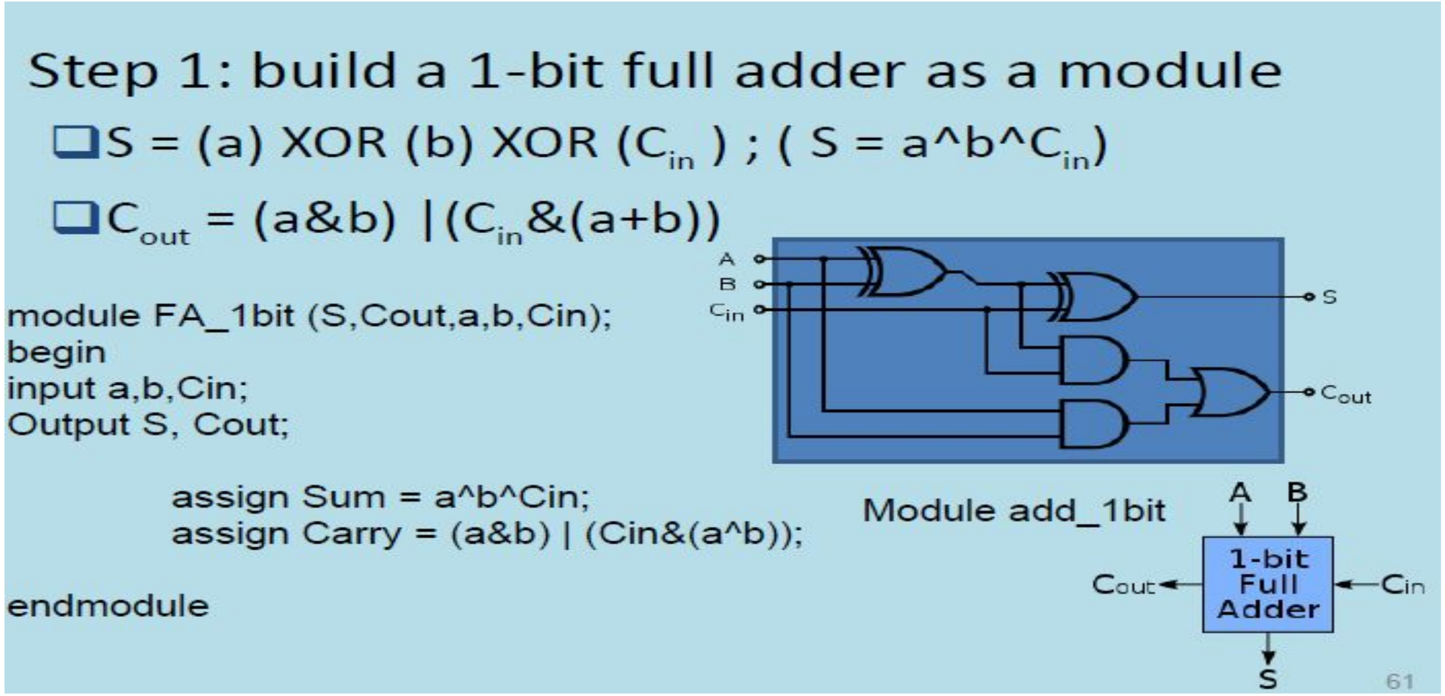
Step2) vvp test2

You can observe output on the console

Step3) gtkwave fulladder_test.vcd

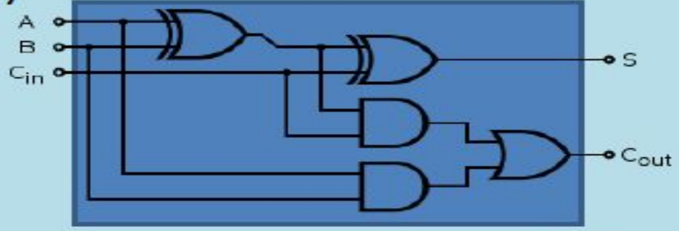
Output waveform will be observed.

4-bit Ripple Carry Adder



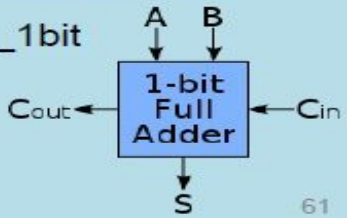
Step 1: build a 1-bit full adder as a module

- $S = (a) \text{ XOR } (b) \text{ XOR } (C_{in})$; ($S = a\hat{}b\hat{}C_{in}$)
- $C_{out} = (a\&b) \mid (C_{in}\&(a+b))$



```
module FA_1bit (S,Cout,a,b,Cin);
begin
input a,b,Cin;
Output S, Cout;

        assign Sum = a^b^Cin;
        assign Carry = (a&b) | (Cin&(a^b));

endmodule
```

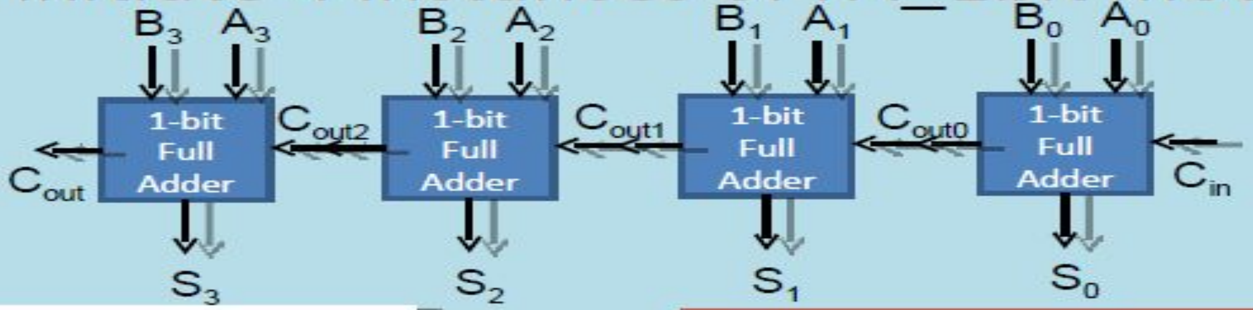
Module add_1bit



61

4-bit Ripple Carry Adder

- Step 2: initiate 4 instances of FA_1bit module



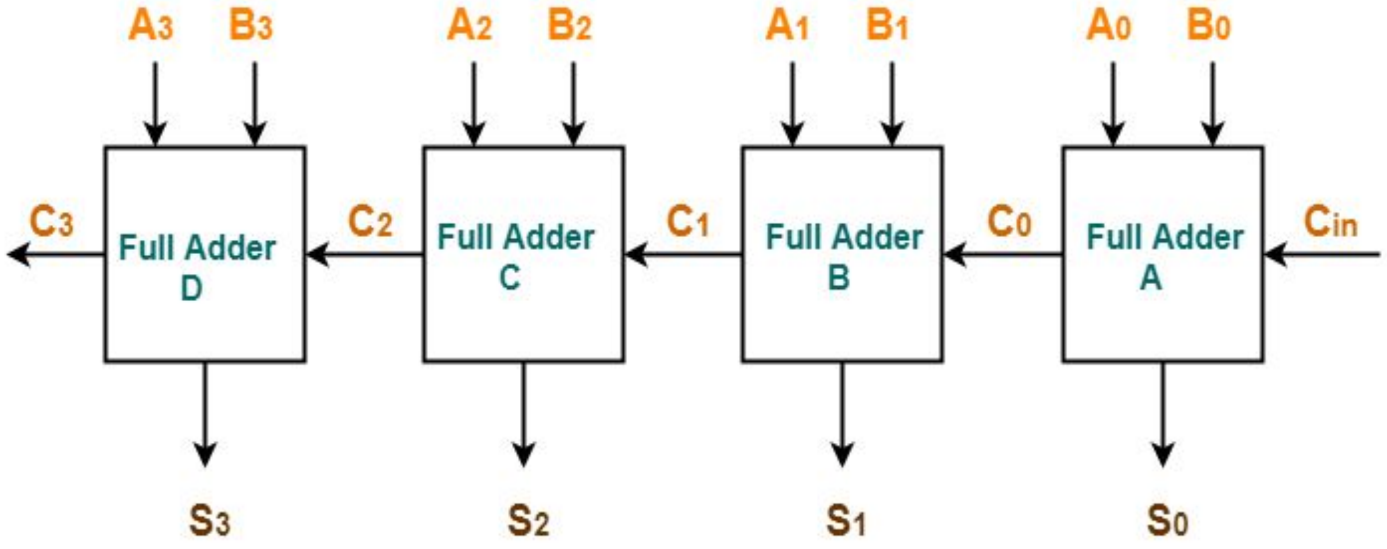
```
module FA_4bits (S,Cout,A,B,Cin);
begin
            input [3:0]   A, B;
            input                 Cin;
            output [3:0] S;
            output                Cout
            wire          Cout0, Cout1, Cout2

FA_1bit     FA1(S[0], Cout0,A[0],B[0],Cin);
FA_1bit     FA1(S[1], Cout1,A[1],B[1],Cout0);
FA_1bit     FA1(S[2], Cout2,A[2],B[2],Cout1);
FA_1bit     FA1(S[3], Cout,A[3],B[3],Cout2);
end
endmodule;
```

The inputs and the output are 4-bits wide

we need wires to propagate the carry from one stage to the next

you may name the instances with any name, but you have to maintain the order of the inputs and outputs

4-bit Ripple Carry Adder Block diagram



4-bit Ripple Carry Adder

4-bit Ripple Carry Adder

```
// Module 4-bit ripple carry adder
module fulladdR(input wire [3:0] a, b, input wire cin, output wire [3:0]
sum, output wire cout);
 // Instantiate full adder modules here
wire [2:0] c;
fulladd u0 (----------------------);
fulladd u1 (----------------------);
fulladd u2 (---------------------);
fulladd u3 (---------------------);
endmodule
```

4-bit Ripple Carry Adder Testbench rca_tb.v

```
`timescale 1 ns / 100 ps
`define TESTVECS 10
module tb;
reg clk, reset;
 reg [3:0] i0, i1;
 reg cin;
wire [3:0] o;
wire cout;
 reg [8:0] test_vecs [0:(`TESTVECS-1)];
integer i;
initial
begin
$dumpfile("rca_test.vcd");
$dumpvars(0,tb);
 end
```

Continue......

```
initial
begin
 reset = 1'b1; #12.5 reset = 1'b0; end  initial clk = 1'b0; always #5 clk =~ clk;
  initial begin
 test_vecs[0] = 9'b000000000;
 test_vecs[1] = 9'b000000001;
test_vecs[2] = 9'b000100010;
  test_vecs[3] = 9'b000100011;
test_vecs[4] = 9'b001000100;
  test_vecs[5] = 9'b001000101;
  test_vecs[6] = 9'b101010110;
   test_vecs[7] = 9'b101010111;
test_vecs[8] = 9'b111011110;
test_vecs[9] = 9'b111011111;
  end
```

Continue......

```
initial {i0, i1, cin, i} = 0;
 fulladdR u0 (i0, i1, cin, o, cout);
initial begin    #6
for(i=0;i<`TESTVECS;i=i+1)
   begin #10 {i0, i1, cin}=test_vecs[i];
end    #100 $finish;  end
always@(i0 or i1 or cin)
$monitor("At time = %t, i0=%b, i1=%b,cin=%b,Sum = %b,Carry %b",
$time,i0,i1,cin,o,cout);
endmodule
```

4-bit Ripple Carry Adder Truth Table

	a3	a2	a1	a0	b3	b2	b1	b0	cin		
	I0[3]	I0[2]	I0[1]	I0[0]	I1[3]	I1[2]	I1[1]	I1[0]	cin	Sum[3:0]	cout
TESTVECTOR[0]	0	0	0	0	0	0	0	0	0	0+0+0=	
TESTVECTOR[1]	0	0	0	0	0	0	0	0	1	0+0+1=	
TESTVECTOR[2]	0	0	0	1	0	0	0	1	0	1+1+0=	
TESTVECTOR[3]	0	0	0	1	0	0	0	1	1	1+1+1=	
TESTVECTOR[4]	0	0	1	0	0	0	1	1	0	2+3+0=	
TESTVECTOR[5]	0	0	1	0	0	0	1	1	1	2+3+1=	
TESTVECTOR[6]	1	0	1	0	1	0	1	1	0	A+B+0=	
TESTVECTOR[7]	1	0	1	0	1	0	1	1	1	A+B+1=	
TESTVECTOR[8]	1	1	1	0	1	1	1	1	0	E+F+0=	
TESTVECTOR[9]	1	1	1	0	1	1	1	1	1	E+F+1 =	

Ripple carry adder

Step1) iverilog -o testrca basicfa.v rca.v rca_tb.v
If the compilation went OK, you won't see any output.

Step2) vvp testrca
You can observe output on the console

Step3) gtkwave rca_test.vcd
Output waveform will be observed