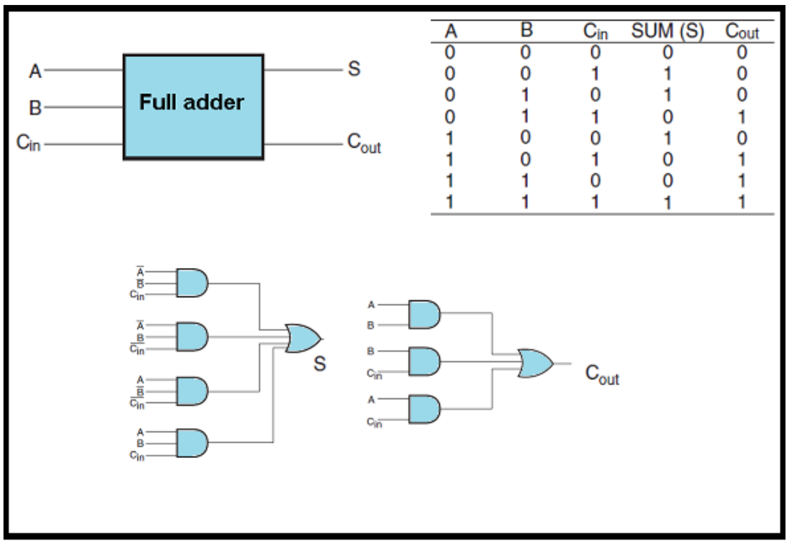
DAY-3
FULL ADDER

EXPLANATION:

Full Adder is the adder that adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM. The C-OUT is also known as the majority 1's detector, whose output goes high when more than one input is high. A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to another. we use a full adder because when a carry-in bit is available, another 1-bit adder must be used since a 1-bit half-adder does not take a carry-in bit. A 1-bit full adder adds three operands and generates 2-bit results.



A	B	C_{in}	SUM (S)	C_{out}
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

Logical expression for Sum,

Sum, $S = A \oplus B \oplus C_{in} = A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in}$

Logical expression for Carry, Carry $= AB + BC_{in} + AC_{in}$

APLLICATIONS:

- Full adders are used in ALUs (arithmetic logic units) of CPUs of computers.
- Full adders are used in calculators.
- Full adders also help in carrying out multiplication of binary numbers.
- Full adders are also used to realize critic digital circuits like multiplexers.
- Full adders are used to generate memory addresses.
- Full adders are also used in generation of program counterpoints.
- Full adders are also used in GPU (Graphical Processing Unit).

RTL CODE:

DATA FLOW METHOD:

```
module fulladder(s,c2,a,b,c);

  output s,c2;

  input a,b,c;

  assign {c2,s}=a+b+c; // by using concatenation operator

endmodule
```

STRUCTURAL METHOD:

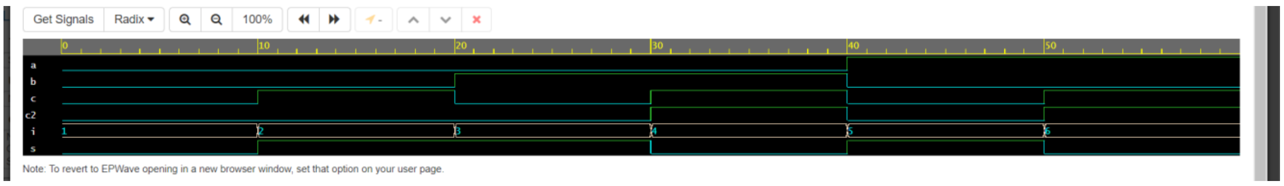
```
module fulladder(s,c2,a,b,c);

  output s,c2;

  input a,b,c;

  wire w1,w2,w3,w4 ;

  xor  x1 (w1,a,b);

   xor x2(s,w1,c);

   and a1(w2,a,b);
and a2(w3,b,c);

and a3(w4,c,a);

or    c1(c2,w2,w3,w4);

endmodule
```

BEHAVIORAL METHOD:

```
module fulladder(s,c2,a,b,c);

  output s,c2;

  input a,b,c;

  always@(*)

   begin

     s=a^b^c;

     c2=a&b|b&c|c&a;

    end

endmodule
```

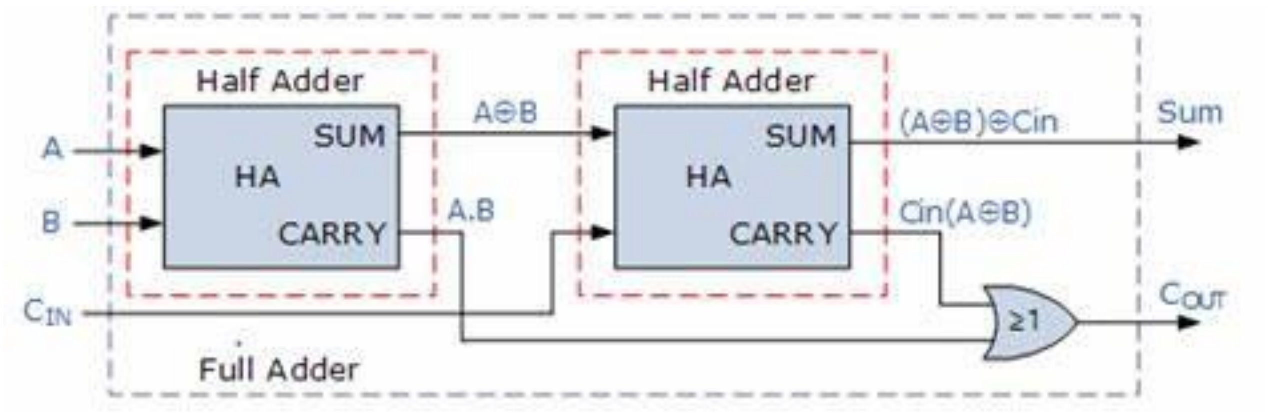
TEST BENCH:

```
module testbench;

  reg a,b,c;

  wire s,c2;

  integer i;

  fulladder a1(s,c2,a,b,c);
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
  initial
    begin
      a=0;b=0;c=0;
    end
  initial
    begin
      for(i=1;i<8;i=i+1)
        begin
          #10 {a,b,c}=i;
        end
    end
  initial
    begin
      #60 $finish();
    end
endmodule
```



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

FULLADDER USING TWO HALF ADDERS



From the logic diagram of the full adder using half adders, it is clear that we require two XOR gates, two AND gates and one OR gate for the implementation of a full adder circuit using half-adders.

However, the implementation of full adder using half adder has a major disadvantage that is the increased propagation delay. That means, the input bits must propagate through several gates in succession that increases the total propagation delay of the full adder circuit.

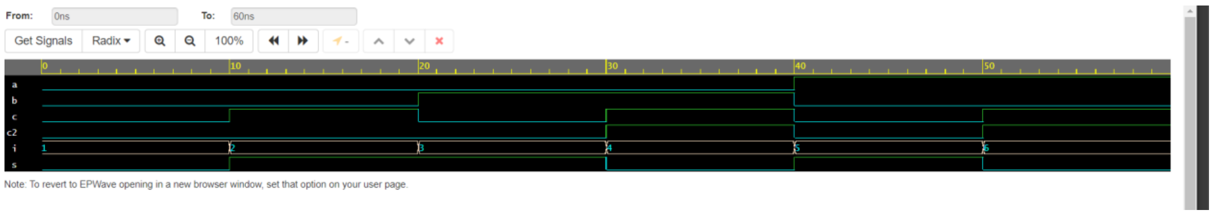
RTL CODE:

```
module fulladder(s,c2,a,b,c);
  output s,c2;
  input a,b,c;
  wire w1,w2,w3;
  assign w1=a^b;
  assign w2=a&b;
  assign s=w1^c;
  assign w3=w1&c;
  assign c2=w2|w3;
endmodule
```

TEST BENCH

```
module testbench;
  reg a,b,c;
  wire s,c2;
  integer i;
  fulladder a1(s,c2,a,b,c);
  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1);
    end
  initial
    begin
```

```
      a=0;b=0;c=0;
    end
  initial
    begin
      for(i=1;i<8;i=i+1)
        begin
          #10 {a,b,c}=i;
        end
    end
  initial
    begin
      #60 $finish();
    end
endmodule
```



From: 0ns To: 60ns

Note: To revert to EPWave opening in a new browser window, set that option on your user page.