# Chapter 14
# Design of Combinational Circuits using Verilog HDL

- Verilog Codes
  - Structural Verilog Code
  - Behavioral Verilog Code
    - Procedural Model
    - Continuous assign statement

# Structural Verilog Code of S output of FA

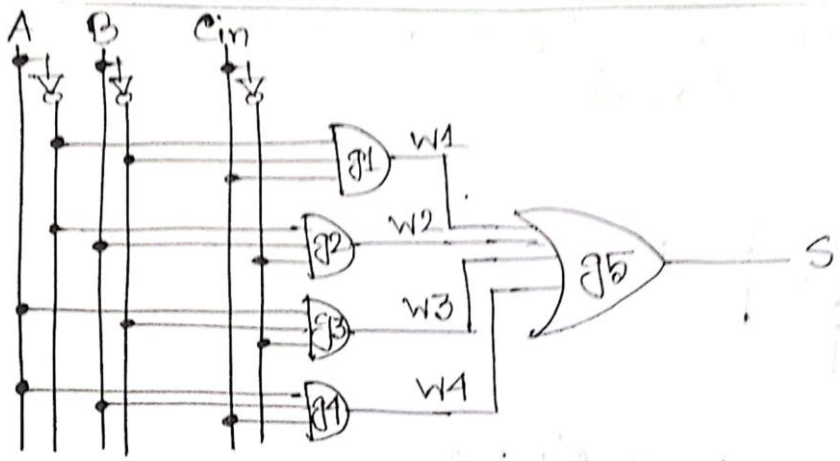



Fig: S output of FA circuit

Structural Verilog Code:

```
module sumofFAStructural (input A, B, Cin, output S);
wire w1, w2, w3, w4;
and g1 (w1, ~A, ~B, Cin),
    g2 (w2, ~A, B, ~Cin),
    g3 (w3, A, ~B, ~Cin),
    g4 (w4, A, B, Cin);
or g5 (S, w1, w2, w3, w4);
endmodule
```

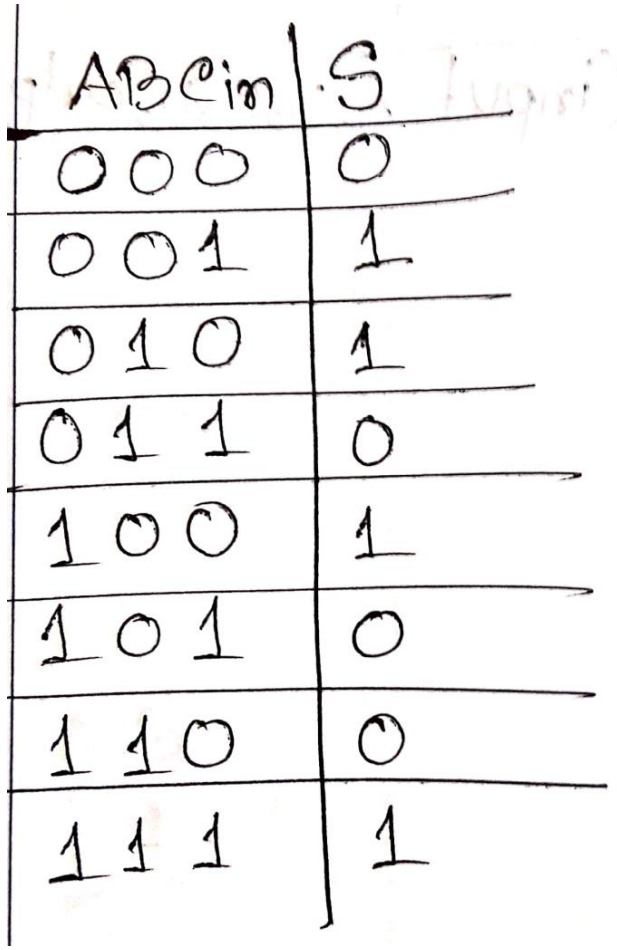
# Behavioral Verilog Code of S output of FA

**Procedural Model:**

```
module sumofFABehavioral(input A, B, Cin, output reg S);
always@(A, B, Cin) begin
S=0;
if (~A & ~B & Cin) S=1;
if (~A & B & ~Cin) S=1;
if (A & ~B & ~Cin) S=1;
if (A & B & Cin) S=1;
end
endmodule
```

| ABCin | S |
|---|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 1 |
| 011 | 0 |
| 100 | 1 |
| 101 | 0 |
| 110 | 0 |
| 111 | 1 |

```
module sumofFABehavioral(input A, B, Cin, output reg S);
always@(A, B, Cin) begin
S=1;
if (~A&~B&~Cin) S=0;
if (~A&B&Cin) S=0;
if (A&~B&Cin) S=0;
if (A&B&~Cin) S=0;
end
endmodule
```



| ABCin | S |
|---|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 1 |
| 011 | 0 |
| 100 | 1 |
| 101 | 0 |
| 110 | 0 |
| 111 | 1 |

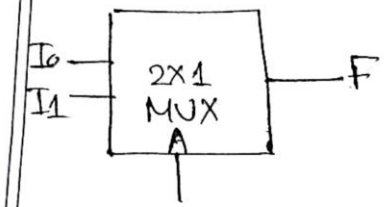
**Continuous assign statement:**

```
module sumofFAAssign(input A, B, Cin, output S);
assign S=(~A & ~B & Cin)|(~A & B & ~Cin)|(A & ~B & ~Cin)|(A & B & Cin);
endmodule
```

**Continuous assign statement:**

```
module sumofFAAssign(input A, B, Cin, output S);
assign S=(A|B|Cin)&(A|~B|~Cin)&(~A|B|~Cin)&(~A|~B|Cin);
endmodule
```

## Procedural description of a 2X1 MUX using if statement




| A | F |
|---|---|
| 0 | $I_0$ |
| 1 | $I_1$ |

$F = I_0A' + I_1A$

| AI0I1 | F |
|---|---|
| 0 0 0 | 0 |
| 0 0 1 | 0 |
| 0 1 0 | 1 |
| 0 1 1 | 1 |
| 1 0 0 | 0 |
| 1 0 1 | 1 |
| 1 1 0 | 0 |
| 1 1 1 | 1 |

**Procedural Model:**

```
module twoByOneMux(input A, I0, I1, output reg F);
always@(A, I0, I1)
if (A==0)
    F=I0;
else F=I1;
endmodule
```

**Continuous assign statement:**

```
module twoByOneMuxAssign(input A, I0, I1, output F);
assign F=(~A&I0)|(A&I1);
endmodule
```

# Procedural Description of sum of FA using case statement

```
module sumofFACase(input A, B, Cin, output reg S);
always@(A, B, Cin)
    case({A, B, Cin})
        3'b000: S=0;
        3'b001: S=1;
        3'b010: S=1;
        3'b011: S=0;
        3'b100: S=1;
        3'b101: S=0;
        3'b110: S=0;
        3'b111: S=1;
    endcase
endmodule
```

# Procedural Description of sum of FA using case statement with Default

```
module sumofFACaseWithDefault(input A, B, Cin, output reg S);
always@(A, B, Cin)
    case({A, B, Cin})
        3'b000: S=0;
        3'b011: S=0;
        3'b101: S=0;
        3'b110: S=0;
        default: S=1;
    endcase
endmodule
```