# EE-421: Digital System Design

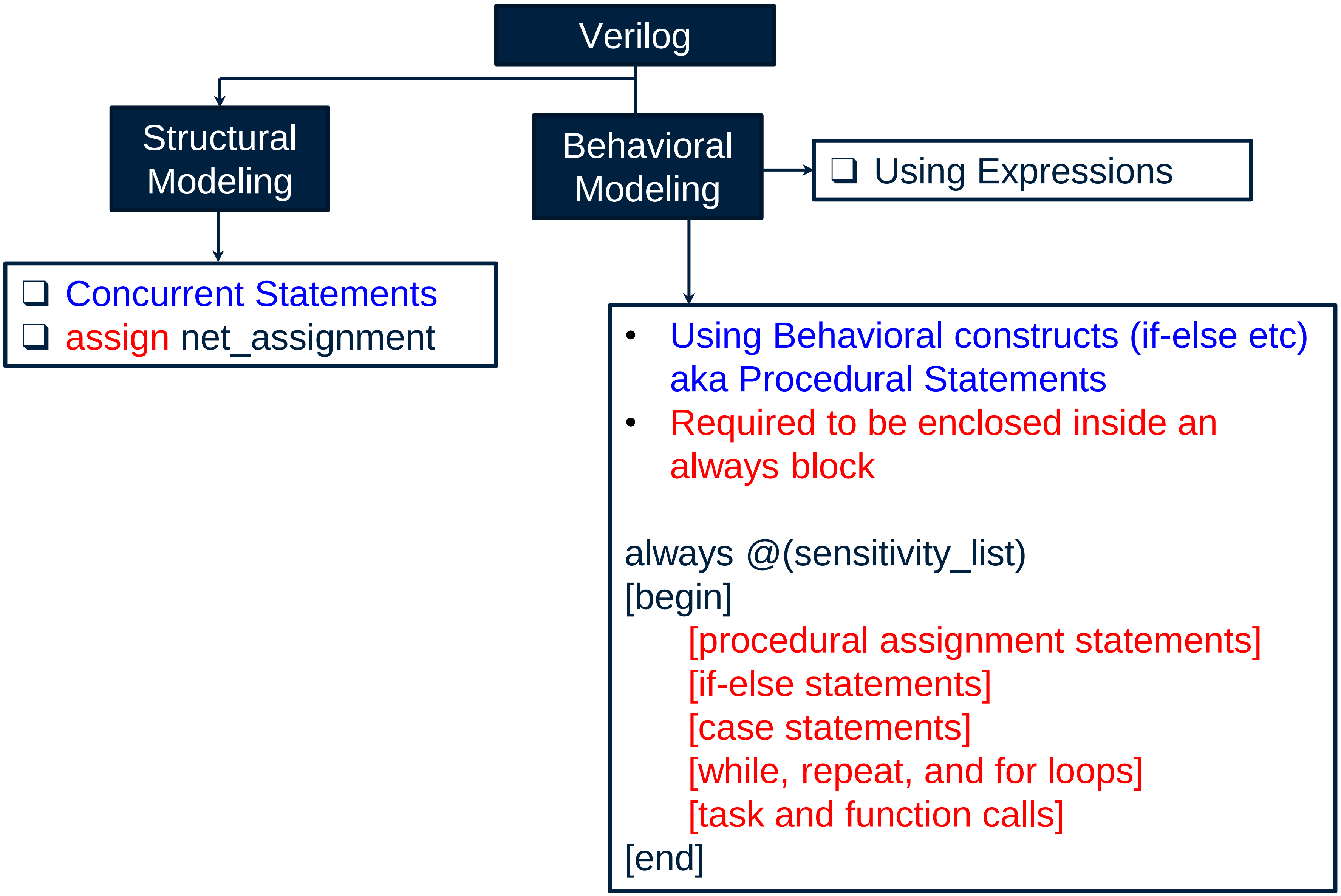
## Combinational Circuit Building Blocks Using CAD Tools - Adder

Instructor: Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

NUST

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE (SEECS)

# Recap: What have we seen so far . . .



- ❑ Concurrent Statements
- ❑ assign net_assignment

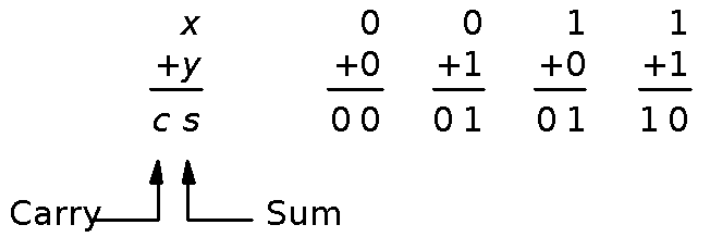
- ❑ Using Expressions

- Using Behavioral constructs (if-else etc) aka Procedural Statements
- Required to be enclosed inside an always block

```
always @(sensitivity_list)
[begin]
      [procedural assignment statements]
      [if-else statements]
      [case statements]
      [while, repeat, and for loops]
      [task and function calls]
[end]
```

## Design of Arithmetic Circuits – Full Adder

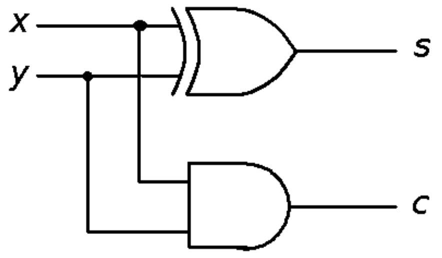
# Review: Half-Adder



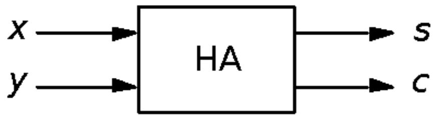
(a) The four possible cases

|  |  | Carry | Sum |
|---|---|---|---|
| $x$ | $y$ | $c$ | $s$ |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

(b) Truth table



(c) Circuit



(d) Graphical symbo

# Review: Full-Adder

| $c_i$ | $x_i$ | $y_i$ | $c_{i+1}$ | $s_i$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

(a) Truth table

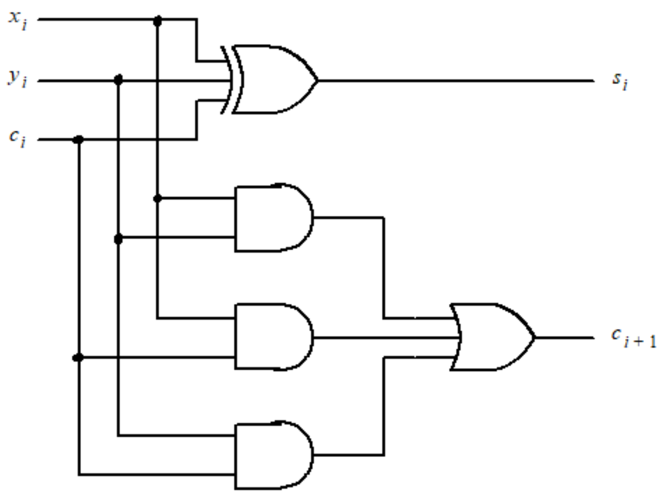
| $c_i$ \ $x_i y_i$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | 1 | | 1 |
| 1 | 1 | | 1 | |

$$s_i = x_i \oplus y_i \oplus c_i$$

| $c_i$ \ $x_i y_i$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | | 1 | |
| 1 | | 1 | 1 | 1 |

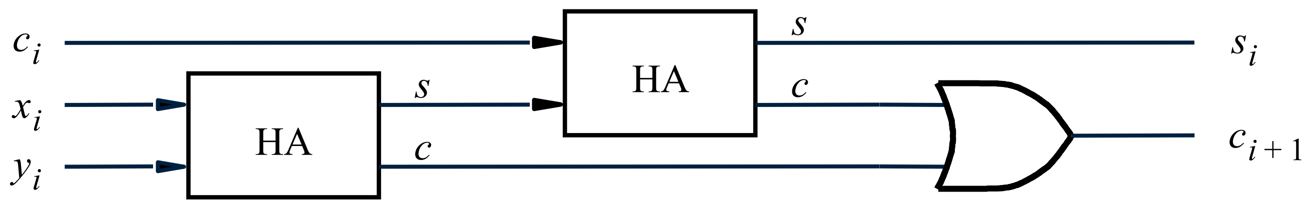
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

(b) Karnaugh maps

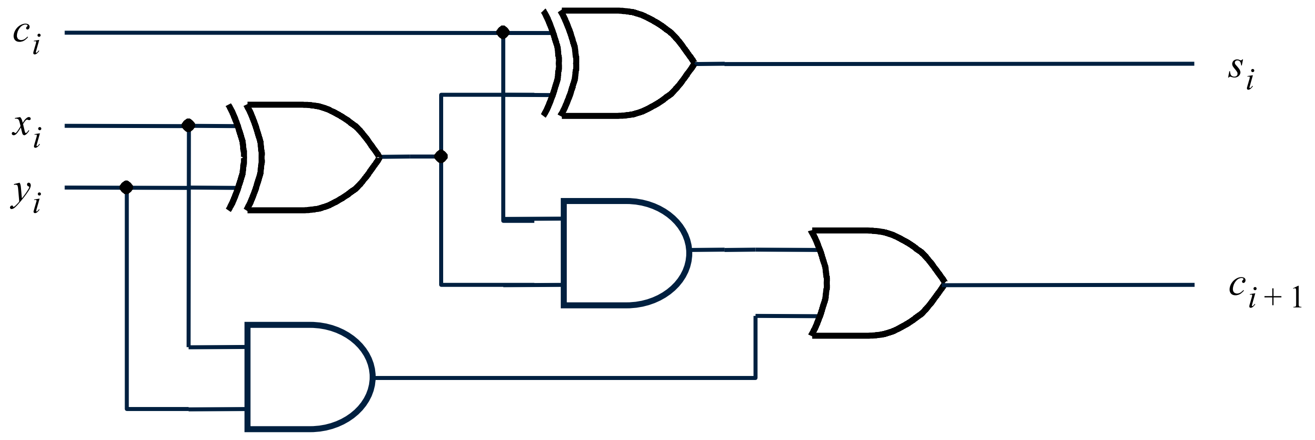


(c) Circuit

# Review: Full-Adder using Half-Adders



(a) Block diagram



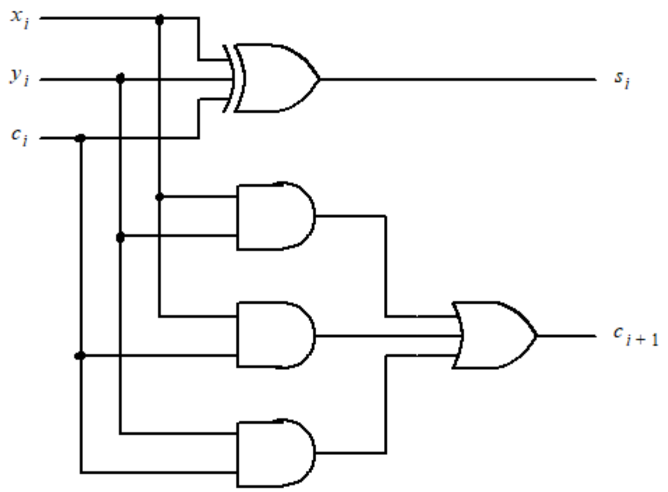
(b) Detailed diagram

# Full-Adder using Gate-Level Primitives

```
module fulladd (Cin, x, y, s, Cout);
    input Cin, x, y;
    output s, Cout;

     xor (s, x, y, Cin);
     and (z1, x, y);
     and (z2, x, Cin);
     and (z3, y, Cin);
     or (Cout, z1, z2, z3);

endmodule
```

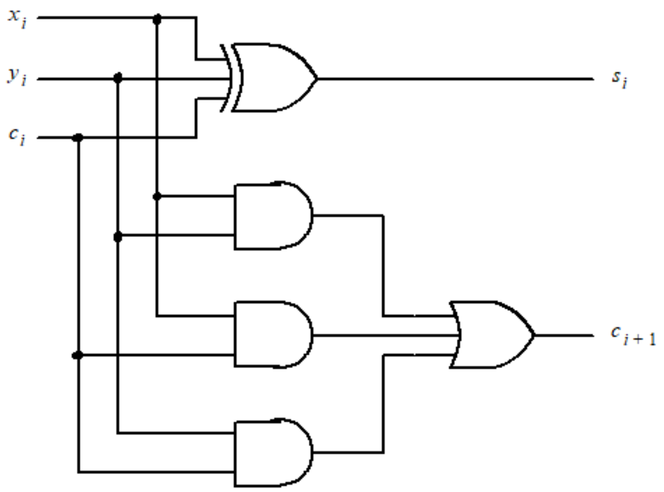


## Full-Adder using Continuous Assignment

```
module fulladd (Cin, x, y, s, Cout);
        input Cin, x, y;
        output s, Cout;

        assign s = x ^ y ^ Cin;
        assign Cout = (x & y) | (x & Cin) | (y & Cin);

endmodule
```



## Full-Adder at Behavioral

```
module fulladd (Cin, x, y, s, Cout);
    input Cin, x, y;
    output reg s, Cout;

    always @(x, y, Cin)
      {Cout, s} = x + y + Cin;

endmodule
```

## Design of Arithmetic Circuits – Multi-bit Adder

# Four-bit Adder – Using Hierarchical Design Approach

```
module adder4 (carryin, x3, x2, x1, x0, y3, y2, y1, y0, s3, s2, s1, s0, carryout);
    input carryin, x3, x2, x1, x0, y3, y2, y1, y0;
    output s3, s2, s1, s0, carryout;

    fulladd stage0 (carryin, x0, y0, s0, c1);
    fulladd stage1 (c1, x1, y1, s1, c2);
    fulladd stage2 (c2, x2, y2, s2, c3);
    fulladd stage3 (c3, x3, y3, s3, carryout);

endmodule

module fulladd (Cin, x, y, s, Cout);
    input Cin, x, y;
    output s, Cout;

    assign s = x ^ y ^ Cin,
    assign Cout = (x & y) | (x & Cin) | (y & Cin);

endmodule
```

## Code Improvement using Vectors

```
module adder4 (carryin, X, Y, S, carryout);
    input carryin;
    input [3:0] X, Y;
    output [3:0] S;
    output carryout;
    wire [3:1] C;

    fulladd stage0 (carryin, X[0], Y[0], S[0], C[1]);
    fulladd stage1 (C[1], X[1], Y[1], S[1], C[2]);
    fulladd stage2 (C[2], X[2], Y[2], S[2], C[3]);
    fulladd stage3 (C[3], X[3], Y[3], S[3], carryout);

endmodule
```

# Code Improvement using Parametrization

```
module addern (carryin, X, Y, S, carryout);
    parameter n=32;
    input carryin;
    input [n-1:0] X, Y;
    output reg [n-1:0] S;
    output reg carryout;
    reg [n:0] C;
    integer k;

    always @(X, Y, carryin)
    begin
        C[0] = carryin;
        for (k = 0; k < n; k = k+1)
        begin
            S[k] = X[k] ^ Y[k] ^ C[k];
            C[k+1] = (X[k] & Y[k]) | (X[k] & C[k]) | (Y[k] & C[k]);
        end
        carryout = C[n];
    end

endmodule
```

## Alternate Style using CASE

```
module fulladd (Cin, x, p_state, s, Cout);
  input Cin, x, p_state;
  output reg s, Cout;

  always @(Cin, x, p_state)
  begin
    case ( {Cin, x, p_state} )
      3'b000:  {Cout, s} = 'b00;
      3'b001:  {Cout, s} = 'b01;
      3'b010:  {Cout, s} = 'b01;
      3'b011:  {Cout, s} = 'b10;
      3'b100:  {Cout, s} = 'b01;
      3'b101:  {Cout, s} = 'b10;
      3'b110:  {Cout, s} = 'b10;
      3'b111:  {Cout, s} = 'b11;
    endcase
  end

endmodule
```

# Code Improvement using Generate

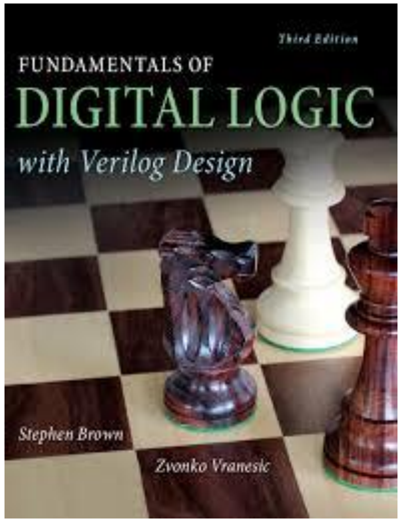
```
module addern (carryin, X, Y, S, carryout);
  parameter n = 32;
  input carryin;
  input [n-1:0] X, Y;
  output [n-1:0] S;
  output carryout;
  wire [n:0] C;

  genvar i;
  assign C[0] = carryin;
  assign carryout = C[n];
  generate
    for (i = 0; i <= n-1; i = i+1)
    begin:addbit
      fulladd stage (C[i], X[i], Y[i], S[i], C[i+1]);
    end
  endgenerate
endmodule
```

```
module fulladd (Cin, x, y, s, Cout);
      input Cin, x, y;
      output s, Cout;
      assign s = x ^ y ^ Cin,
             Cout = (x & y) | (x & Cin) | (y & Cin);
endmodule
```

# Recommended Reading

- Digital System Design with Verilog HDL, 3/e, b **S**tephen Brown and **Z**vonko Vranesic. [**S&Z**]
  - S&Z,
    - Chapter-3 for review
    - 3.5 for Verilog



## THANK YOU