Design with Verilog

Chap 6 – Introduction to HDL (d)

Credit to: MD Rizal Othman
Faculty of Electrical & Electronics Engineering
Universiti Malaysia Pahang

Ext: 6036

VERILOG HDL

- Basic Unit – A module
- Module
  - Describes the functionality of the design
  - States the input and output ports
- Example: A Computer
  - Functionality: Perform user defined computations
  - I/O Ports: Keyboard, Mouse, Monitor, Printer

Module

- General definition

```
module module_name ( port_list );
    port declarations;
    …
    variable declaration;
    …
    description of behavior
endmodule
```

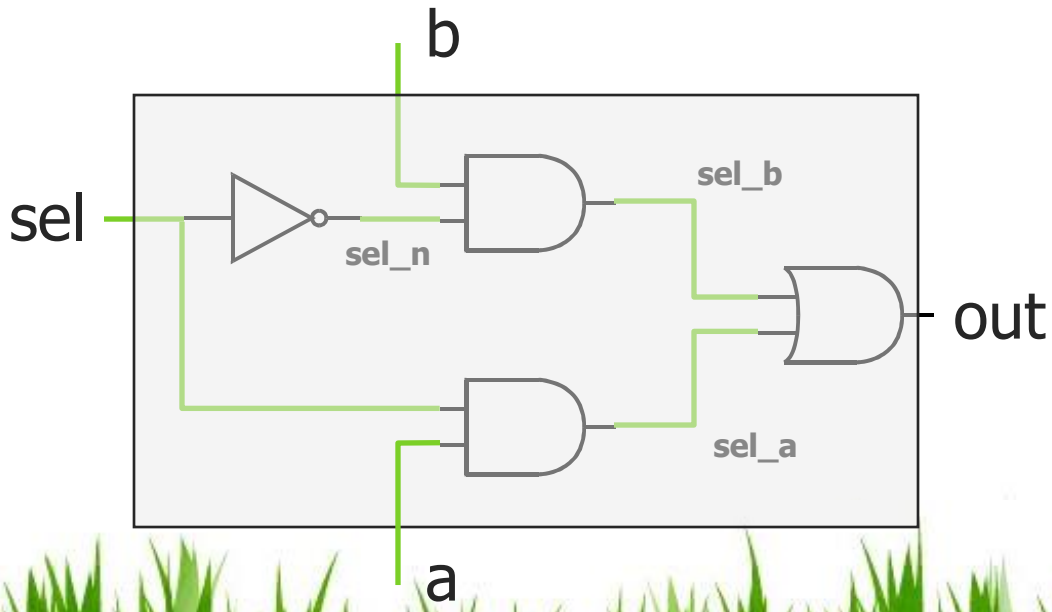
Example

```
module HalfAdder (A, B, Sum Carry);
input A, B;
output Sum, Carry;
assign Sum = A ^ B;
//^ denotes XOR
assign Carry = A & B;
// & denotes AND
endmodule
```

Dataflow

- **Dataflow**: Specify output signals in terms of input signals
- Example:

```
assign out = (sel & a) | (~sel & b);
```



Dataflow Modeling

- Uses continuous assignment statement
  - Used to describe combinational logic
  - Output of the circuit evaluated whenever an input changes, i.e continuously assigned
  - Can be used ONLY for nets, NOT for register variables

  - Format: assign [delay] net = expression;
  - Example: assign sum = a ^ b;

- **Delay**: Time duration between assignment from RHS to LHS
- All continuous assignment statements execute concurrently
- Order of the statement does not impact the design

Dataflow Modeling (cont.)

- Delay can be introduced
  - Example: assign #2 sum = a ^ b;
  - "#2" indicates 2 time-units
- Associate time-unit with physical time
  - `timescale time-unit/time-precision
  - Example: `timescale 1ns/100 ps
- Timescale

  `timescale 1ns/100ps
  - 1 Time unit = 1 ns
  - Time precision is 100ps (0.1 ns)
  - 10.512ns is interpreted as 10.5ns

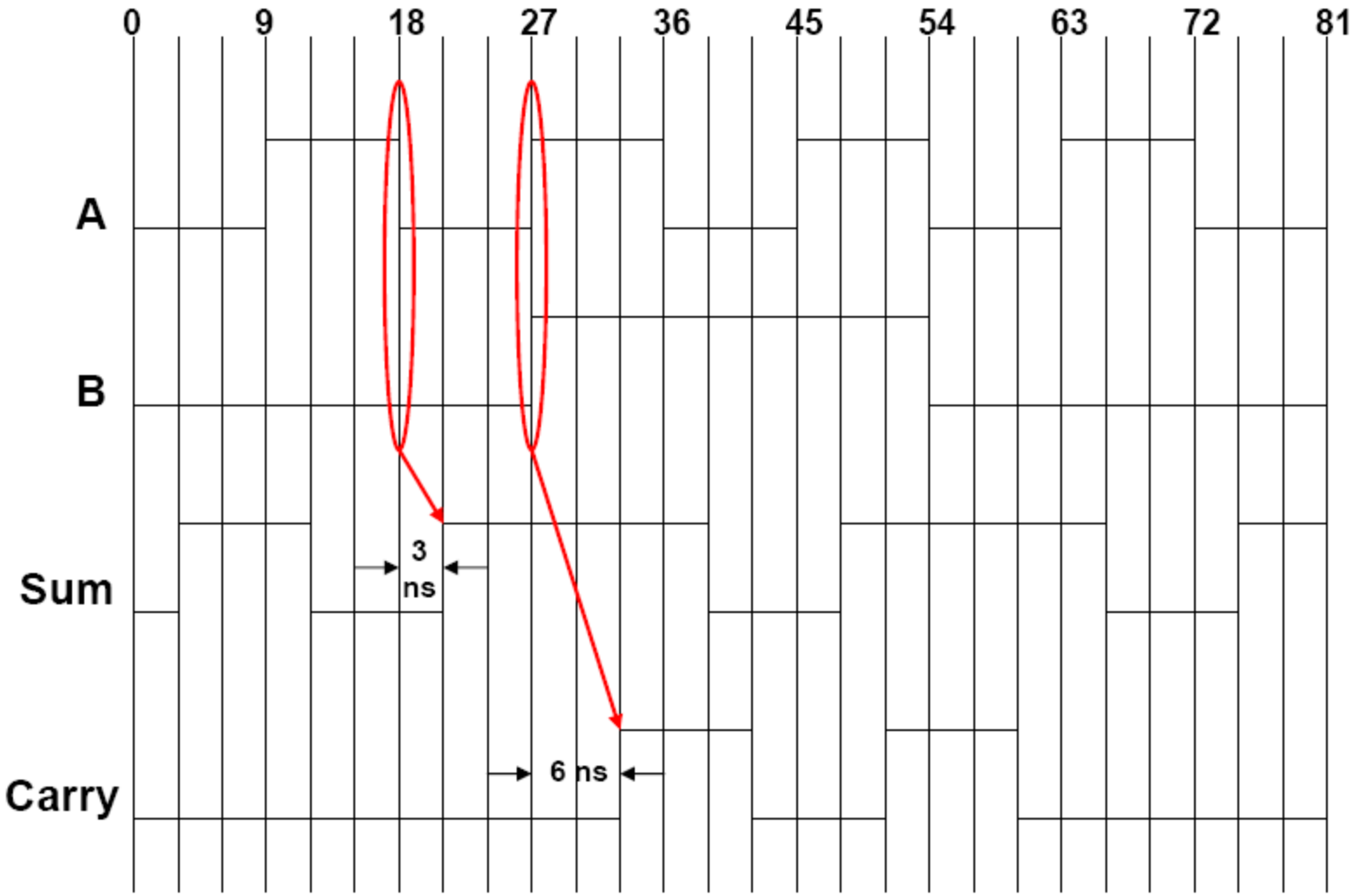
Dataflow Modeling (cont.)

- Restrictions on Data Types:
  - Can use only ***wire*** data type
  - Cannot use ***reg*** data type

- Example:

```
`timescale 1ns/100ps
module HalfAdder (A, B, Sum, Carry);
  input A, B;
  output Sum, Carry;
  assign #3 Sum = A ^ B;
  assign #6 Carry = A & B;
endmodule
```

Dataflow Modeling (cont.)



1. Implementation of a 2x4 decoder.

```
module decoder_2x4 (out, in0, in1);

output out[0:3];
input in0, in1;

// Data flow modeling uses logic operators.
assign out[0:3] = { ~in0 & ~in1, in0 & ~in1,~in0 & in1, in0 & in1 };
endmodule
```

2. Implementation of a Full adder.

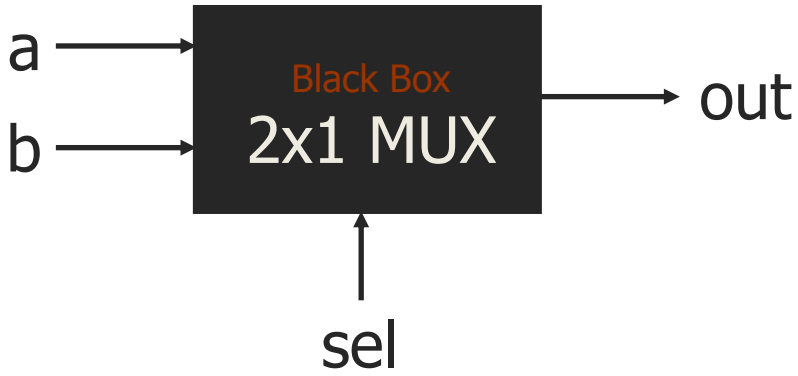
```
module full_adder (sum, c_out, in0, in1, c_in);

output sum, c_out;
input in0, in1, c_in;

assign { c_out, sum } = in0 + in1 + c_in;

endmodule
```

Behavioral Modeling

- **Behavioral**: Algorithmically specify the behavior of the design

- Example:

```
if (select == 0) begin
        out = b;
end
else if (select == 1) begin
        out = a;
end
```



2. Behavioral Modeling

- Behavioral modeling is used to describe complex circuits.
- It is primarily used to model sequential circuits, but can also be used to model pure combinatorial circuits.
- The behavior of the design is described using procedural constructs: initial Statements and always Statements

Behavioral Modeling (cont.)

- always statement : Sequential Block
  - Sequential Block: All statements within the block are executed sequentially
- When is it executed?
  - Occurrence of an event in the sensitivity list
  - Event: Change in the logical value
- Statements with a Sequential Block: Procedural Assignments
  - can drive only **reg data type**.
  - Which means left-side data (output) type cannot be nets.
  - Can occur only within an initial or an always statement.
  - It can model both combinational and sequential logic.

Behavioral Modeling (cont.)

- **Example:**

```
module mux_2x1(a, b, sel, out);
   input a, a, sel;
   output reg out;
   always @(a or b or sel)
   begin
    if (sel == 1)
        out = a;
    else out = b;
   end
endmodule
```

Behavioral Modeling (cont.)

- Delay in Procedural Assignments
  - Inter-Statement Delay
  - Intra-Statement Delay
- Inter-Assignment Delay
  - Example:

    Sum = A ^ B;

    #2 Carry = A & B;
  - Delayed execution
- Intra-Assignment Delay
  - Example:

    Sum = A ^ B;

    Carry = #2 A & B;
  - Delayed assignment

Procedural Statement

- Two Procedural Statement
  - initial Statement
  - always Statement
- initial Statement : Executes only once
- always Statement : Executes in a loop
- Example:

```
...
initial begin
  Sum = 0;
  Carry = 0;
end
...
```

```
...
always @(A or B) begin
  Sum = A ^ B;
  Carry = A & B;
end
...
```

Event Control

- Event Control
  - Edge Triggered Event Control
  - Level Triggered Event Control
- Edge Triggered Event Control

  @ (posedge CLK) //Positive Edge of CLK

  Curr_State = Next_state;

@ negedge	@ posedge
1 → x	0 → x
1 → z	0 → z
1 → 0	0 → 1
x → 0	x → 1
z → 0	z → 1

- Level Triggered Event Control

  @ (A or B) //change in values of A or B

  Out = A & B;

```
module cont_proc (in1, in2, out1_cont, out2_cont, out1_proc, out2_proc);
input in1, in2;
output out1_cont, out2_cont, out1_proc, out2_proc;

wire in1, in2 ,out1_cont, out2_cont;
reg out1_proc, out2_proc;
//continuous assignment
assign #2 out1_cont = in1 | in2;
assign #1 out2_cont = in1 | in2;

always@(in1, in2) begin
//procedural assignment
 #2 out1_proc = in1 | in2;
 #1 out2_proc = in1 | in2;
end
endmodule
```

Loop Statements

- Loop Statements
  - Repeat
  - While
  - For
- Repeat Loop
  - Example:

```
repeat (Count)
  sum = sum + 5;
```

  - If condition is a x or z it is treated as 0

Loop Statements (cont.)

- While Loop
  - Example:

```
while (Count < 10) begin
  sum = sum + 5;
  Count = Count +1;
end
```

  - If condition is a x or z it is treated as 0

- For Loop
  - Example:

```
for (Count = 0; Count < 10; Count = Count + 1) begin
  sum = sum + 5;
end
```

Conditional Statements

- if Statement
- Format:

```
if (condition)
   procedural_statement
else if (condition)
   procedural_statement
else
   procedural_statement
```

- Example:

```
if (Clk)
   Q = 0;
else
   Q = D;
```

Conditional Statements (cont.)

- Case Statement
- Example 1:

```
case (X)
  2'b00: Y = A + B;
  2'b01: Y = A – B;
  2'b10: Y = A / B;
endcase
```

Conditional Statements (cont.)

- Variants of case Statements:
  - casex and casez
- casez – z is considered as a don't care
- casex – both x and z are considered as don't cares
- Example:

```
casez (X)
  2'b1z: A = B + C;
  2'b11: A = B / C;
endcase
```

1. Implementation of a full adder.

```
module full_adder (sum, c_out, in0, in1, c_in);

output sum, c_out;
reg sum, c_out

input in0, in1, c_in;

always @(in0, in1, c_in)
{c_out, sum} = in0 + in1 + c_in;

endmodule
```

2. Implementation of a 8-bit binary counter.

```
module ( count, reset, clk );

output [7:0] count;
reg [7:0] count;

input reset, clk;

// consider reset as active low signal

always @( posedge clk, negedge reset)
begin
if(reset == 1'b0)
count <= 8'h00;
else
count <= count + 8'h01;
end

endmodule
```

- ***Implementation of a Full adder using Dataflow modeling***

```
module full_adder (sum, c_out,
in0, in1, c_in);

output sum, c_out;
input in0, in1, c_in;

assign { c_out, sum } = in0 + in1 +
c_in;

endmodule
```

- ***Implementation of a full adder using Behavioral modeling***

```
module full_adder (sum, c_out,
in0, in1, c_in);

output sum, c_out;
reg sum, c_out

input in0, in1, c_in;

always @(*)
{c_out, sum} = in0 + in1 + c_in;

endmodule
```

Homework & Discussion

3. Implementation of a 4x1 multiplexer using Dataflow and Behavioral modeling.

Module…

….

….

….

endmodule

3. Structural Modeling

1. Data flow:
   - To design combinational ONLY
   - Use ***assign*** statement
2. Behavior
   - To design combinational and sequential circuits
   - Uses arithmetic expressions, procedural assignments (***initial*** and ***always***), or other Verilog control flow structures

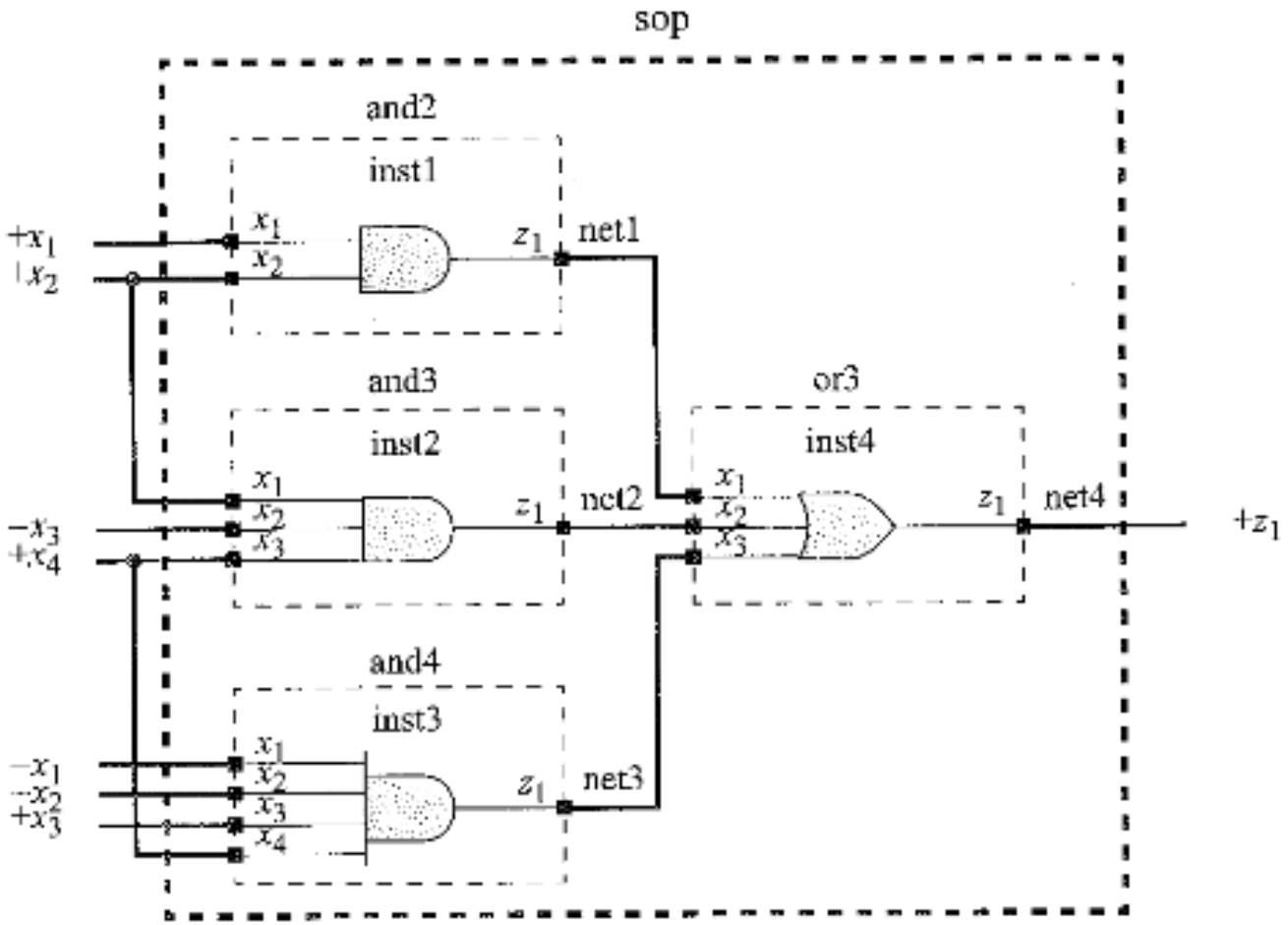
3. Structural Modeling

- A top-down design by combining lower-level logic modules into a larger structural module.
- Each lower-level modules (sub-modules) must have previously compiled and tested for correct functional operation
- Structural modeling can be described using build-in gate primitives or module instances in any combination
- Interconnections between instances are specified using nets

3. Structural Modeling (Example…)

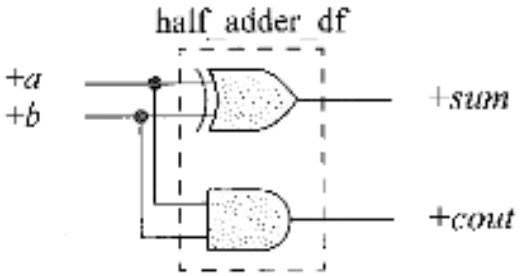
Design the circuit based on expression:

z1 = x1x2 + x2x3'x4 + x1'x2'x3x4



3. Structural Modeling (Example…)

Design a full adder circuit using structural modeling

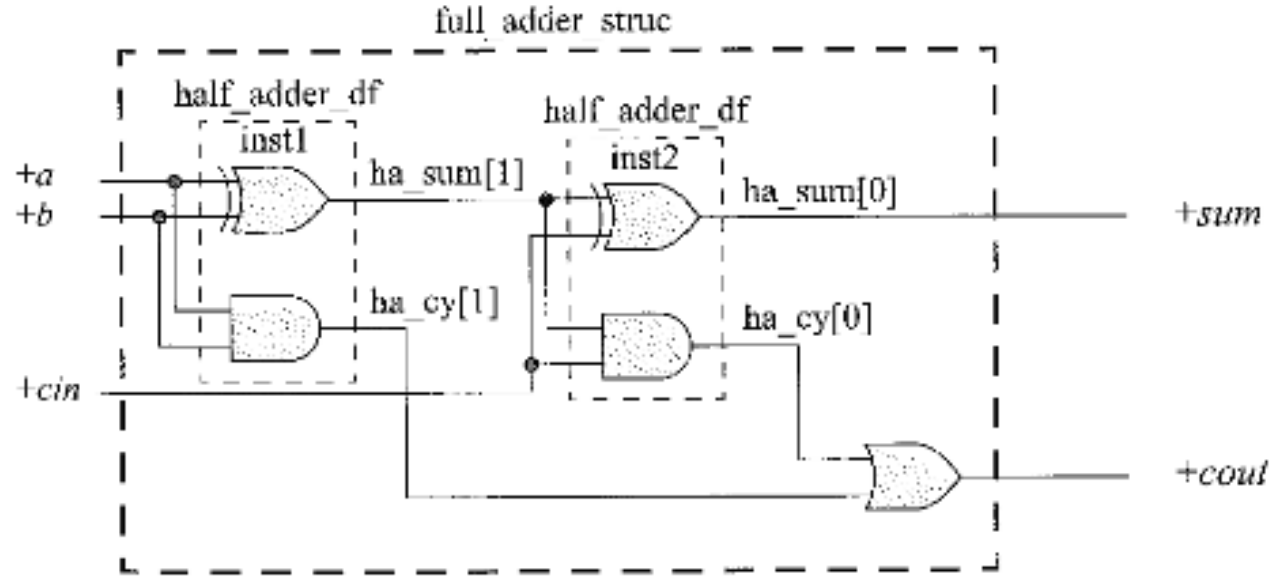


$$sum = a \ominus b \ominus cin$$

$$cout = cin(a \oplus b) + ab$$

$$sum = a \oplus b$$

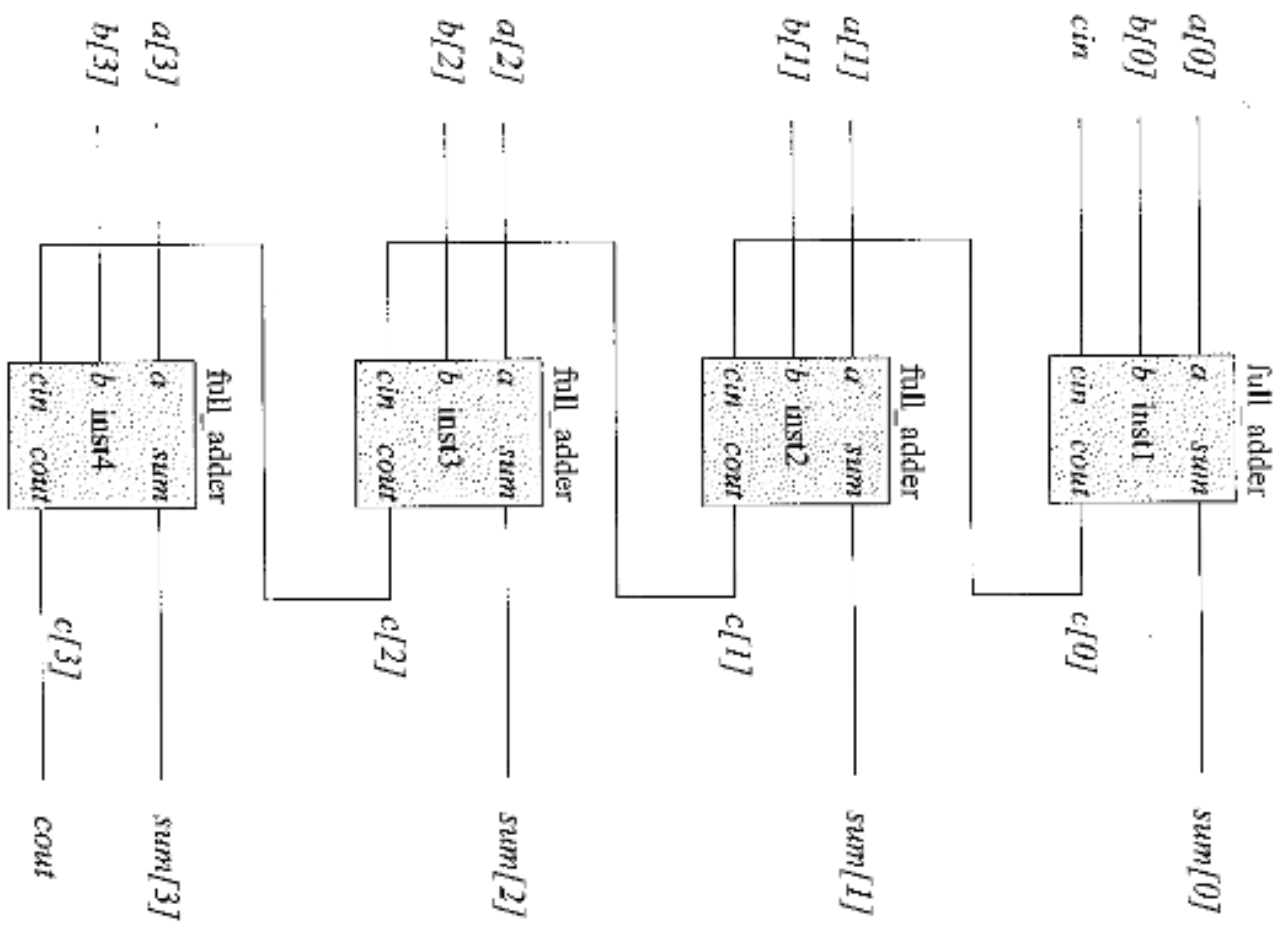
$$cout = ab$$



Group discussion

3. Structural Modeling (homework?…)

Design a four-bit ripple adder circuit using structural modeling

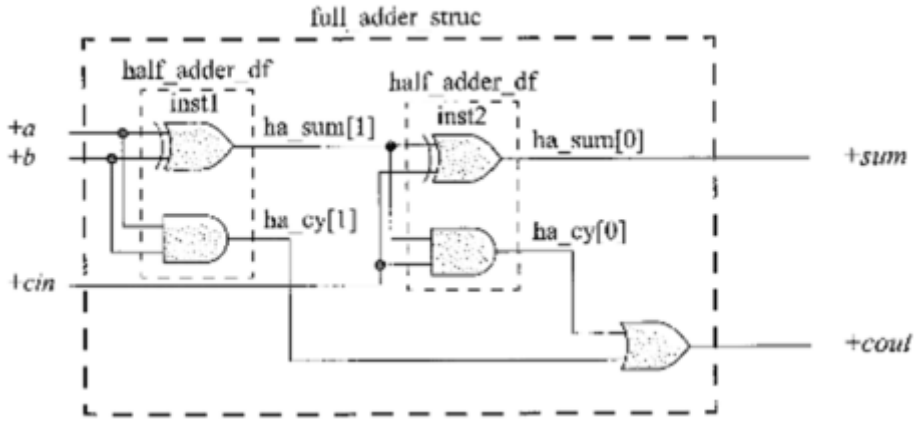


Group discussion

4. Mixed-design Modeling

- Incorporates different modeling styles in the same module
- Includes gate, module instantiations, continuous assignment and behavioral constructs
- For example, a full adder can be designed using built-in primitives, dataflow and behavioral modeling

4. Mixed-design Modeling




```
//mixed-designed full adder
Module full_adder_mix
(a,b,cin,sum,cout);
// list inputs and outputs
Input a, b, cin;
Output sum, cout;
// define reg and wire
Reg cout;
Wire a, b, cin;
Wire sum;
Wire net1;
```

```
//built-in primitive
Xor (net1, a,b);

//behavioral
Always @ (a or b or cin)
Begin
    cout=cin & (a^b)|(a&b);
End

//dataflow
Assign sum=net1^cin;
endmodule
```

Exercise & Discussion

- Finite state machine (FSM)