

Nội dung chương 4

- Giới thiệu
- Khái niệm cơ bản
 - Cấu trúc cơ bản của mạch thiết kế
 - Khai báo
 - Tích hợp Module
- Mạch tổ hợp
- Xử lý tuần tự

Giới thiệu

- Verilog Hardware Description Language (Verilog-HDL) ra đời năm 1987
- Chuẩn hóa: IEEE 1364 năm 1995, cuối cùng IEEE 1364-2001
- Cho phép mô tả hardware không chỉ ở mức cổng (gate level), mức chuyển dịch thanh ghi (register-transfer level - RTL), mà còn cho phép mô tả theo thuật toán

Qui định

- Phạm vi sử dụng Verilog-HDL:
 - Thiết kế hardware
 - Mô phỏng
- Quy tắc ngôn ngữ:
 - Dùng: chữ số, chữ cái, dấu “_” (VD: x_0, y_1, ...)
 - Có phân biệt chữ in hoa và chữ thường
 - Với tên biến và tên Module: không dùng “_” ở đầu và cuối, hoặc 2 dấu “__” (VD: _x, y_, a__0, ...)

Từ khóa

always and assign attribute begin buf bufif0 bufif1 case casex casez cmos deassign default defparam disable edge else end endattribute endcase endfunction	endmodule endprimitive endspecify endtable endtask event for force forever fork function highz0 highz1 if ifnone initial inout input integer join medium module	large macromodule nand negedge nmos nor not notif0 notif1 or output parameter pmos posedge primitive pull0 pull1 pulldown pullup rcmos real realtime	reg release repeat rnmos rpmos rtran rtranif0 rtranif1 scalared signed small specify specparam strength strong0 strong1 supply0 supply1 table task time tran	tranif0 tranif1 tri tri0 tri1 triand trior trireg unsigned vectored wait wand weak0 weak1 while wire wor xnor xor
---	--	---	---	---

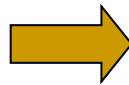
1. Những khái niệm cơ bản

- Cấu trúc cơ bản của mạch thiết kế
- Khai báo
- Biểu diễn giá trị số, Logic
- Phương pháp thiết kế
- Tích hợp Module
- Toán tử số học, Logic

Cấu trúc cơ bản của Verilog-HDL

- Cấu trúc cơ bản của mạch thiết kế là Module. Tất cả phần mô tả mạch thiết kế nằm trong khai báo Module và Endmodule

```
module ModuleName (PortList);  
  
    Port declaration  
    Net declaration  
    Register declaration  
    Parameter declaration  
  
    Mô tả mạch thiết kế  
  
endmodule
```



```
module HALFADD (A, B,  
                SUM, CARRY);  
  
    input A, B;  
    output SUM, CARRY;  
  
    assign SUM = A ^ B;  
    assign CARRY = A & B;  
  
endmodule
```

Khai báo

- Cổng giao tiếp ra bên ngoài Module

```
input clk, reset;  
input [7:0] bus1, bus2;  
inout [15:0] databus;
```

- Kết nối (Net) và thanh ghi (Register) bên trong Module

```
wire enable;  
wire [7:0] bus;
```

```
reg ff1, ff2;  
reg [3:0] cnt;  
reg [7:0] mem [0:255];
```

- Tham số

```
parameter STEP = 1000;  
parameter MEMSIZE = 1024;  
reg [7:0] mem [0: MEMSIZE - 1];
```

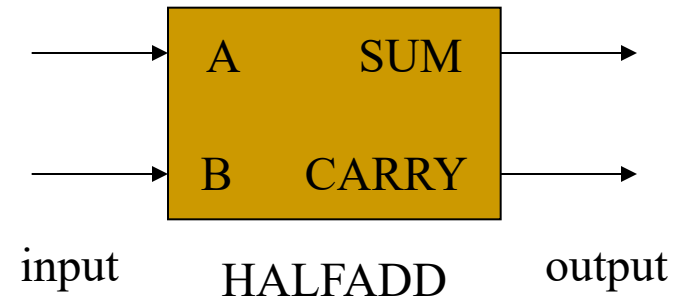
Mô tả mạch thiết kế

- Sau phần khai báo là phần mô tả cấu trúc và sự hoạt động mạch thiết kế

```
assign SUM = A ^ B;  
assign CARRY = A & B;
```

```
endmodule
```

Mô tả tín hiệu xử lý



Chú thích

- Sử dụng “//” cho mỗi dòng lệnh

- /*...*/ cho một khối

- Ví dụ:



```
module HALFADD (A, B,  
                SUM, CARRY);  
// Bộ cộng nửa  
  
input A, B;  
output SUM, CARRY;  
  
/* Đây là phần mô tả  
   mạch thiết kế */  
assign SUM = A ^ B;  
assign CARRY = A & B;  
  
endmodule
```

Biểu diễn giá trị logic

- Verilog HDL có 4 giá trị logic

Logic Value	Description
0	zero, low, or false
1	one, high, or true
z or Z	high impedance (tri-stated or floating)
x or X	unknown or uninitialized

Biểu diễn giá trị hệ cơ số

■ Cú pháp: *size'base value*

- *size* (optional): Độ lớn theo bit. Mặc định là số 32-bit
- *'base* (optional): Hệ cơ số. Mặc định là hệ thập phân
- *value*: Giá trị thiết lập

Base	Symbol	Legal Values
binary	b or B	0, 1, x, X, z, Z, ?, _
octal	o or O	0-7, x, X, z, Z, ?, _
decimal	d or D	0-9, _
hexadecimal	h or H	0-9, a-f, A-F, x, X, z, Z, ?, _

Biểu diễn giá trị hệ cơ số (cont.)

■ Ví dụ:

Examples	Size	Base	Binary Equivalent
4'd10	4 bits	decimal	1010
'o7	unsized	octal	0...00111 (32-bits)
1'b1	1 bit	binary	1
8'Hc5	8 bits	hex	11000101
6'hF0	6 bits	hex	110000 (truncated)
6'hF	6 bits	hex	001111 (zero filled)
6'hZ	6 bits	hex	ZZZZZZ (Z filled)

Khai báo tín hiệu

- Khai báo kiểu tín hiệu
 - Có 2 loại:
 - *Register* (**reg**): Latch và Flipflop
 - *Net* (**wire**): đường kết nối tín hiệu
 - Lệnh điều khiển:
 - *Register* signal: **always, function**
 - *Net* signal: **assign**

Lược bỏ khai báo

- Verilog-HDL cho phép lược bỏ khai báo
- VD1:
 - ❑ Lược bỏ khai báo Net
 - ❑ Tín hiệu khai báo cho Port mặc định là kiểu Net và có thể lược bỏ
- VD2:
 - ❑ Những kết nối 1 bit bên trong Module: có thể lược bỏ, Tuy nhiên, khó cho việc Debug

```
module DFF (CLK, D, Q);  
input CLK, D;  
output Q;  
Wire CLK, D; // Có thể lược bỏ  
reg Q;  
  
always @ (posedge CLK)  
begin  
    Q <= D;  
  
end  
endmodule
```

```
module RSFF(SB, RB, Q);  
input SB, RB;  
output Q;  
Wire tmp; // Có thể lược bỏ  
reg Q;  
  
nand na1 (Q, SB, tmp);  
nand na2 (tmp, Q, RB);  
  
endmodule
```

Tín hiệu nhiều bit

- Khai báo tín hiệu nhiều bit:
 - Kiểu qui định: [MSB: LSB]

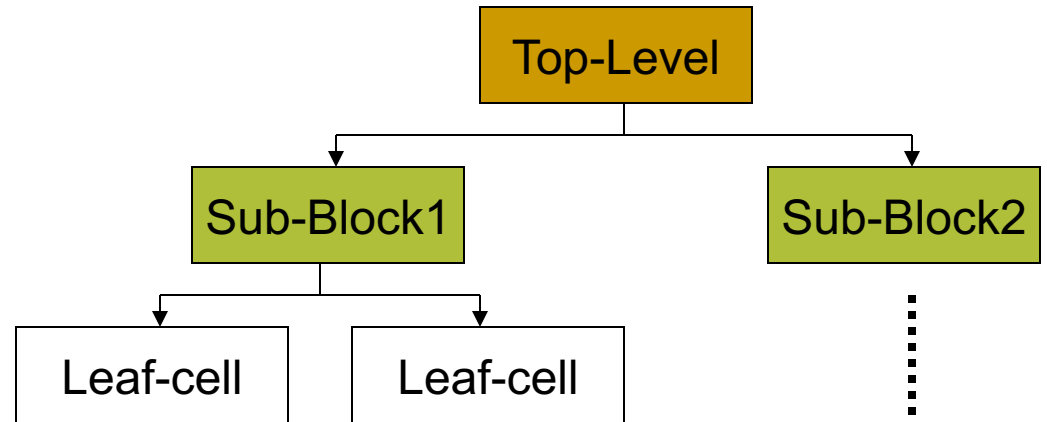
```
input [7:0] bus1, bus2;  
wire [15:0] databus;  
reg [15:8] addr_hi;  
reg [7:0] addr_lo;
```

- Tuyển chọn 1 phần tín hiệu nhiều bit:

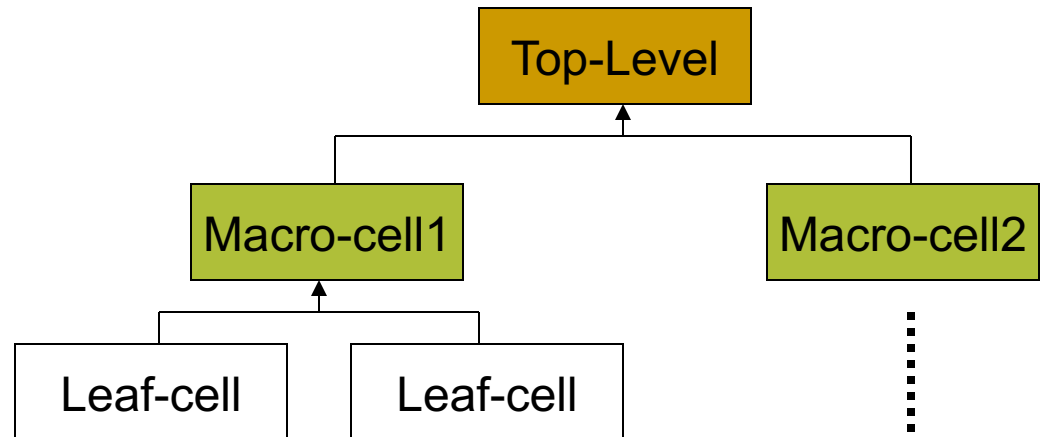
```
input [3:0] a, b; input [7:0] bus_a;  
wire [7:0] c; wire [3:0] d;  
  
assign c[7:4] = a;  
assign c[3:0] = b;  
assign d = bus_a[3:0];
```

Phương pháp thiết kế

■ Top-down design



■ Bottom-up design



Tích hợp Module (Module Instance)

- **Cú pháp:** `Module_name Instance_name (Port_List);`
 - `Module_name = Instance_name` là chấp nhận được
 - `Port_List`: có 2 cách mô tả

PP kết nối	Kết nối theo tên Port	Kết nối theo thứ tự Port
Cú pháp	<code>(.port_name(signal), .port_name(signal), ...)</code>	<code>(signal, signal, ...)</code>
Đặc điểm	Số cổng lớn, tình trạng kết nối vẫn tốt	Thứ tự sai, tình trạng kết nối sẽ sai

Example of Module Instance

```
module DFF4 (CLK, D, Q);
```

```
input CLK;
```

```
input [3:0] D;
```

```
output [3:0] Q;
```

```
// Kết nối theo tên Port
```

```
DFF DFF2 (.CLK(CLK), .D(D[2]), .Q(Q[2]));
```

```
DFF DFF3 (.D(D[3]), .CLK(CLK), .Q(Q[3]));
```

```
DFF DFF1 (.CLK(CLK), .D(D[1]), .Q(Q[1]));
```

```
DFF DFF0 (.CLK(CLK), .D(D[0]), .Q(Q[0]));
```

```
Kết nối theo thứ tự Port
```

```
// DFF DFF2 (CLK, D[2], Q[2]);
```

```
// DFF DFF3 (CLK, D[3], Q[3]);
```

```
// DFF DFF1 (CLK, D[1], Q[1]);
```

```
// DFF DFF0 (CLK, D[0], Q[0]);
```

```
endmodule
```

```
module DFF (CLK, D, Q);
```

```
input CLK, D;
```

```
output Q;
```

```
reg Q;
```

```
always @ (posedge CLK)
```

```
begin
```

```
Q <= D;
```

```
end
```

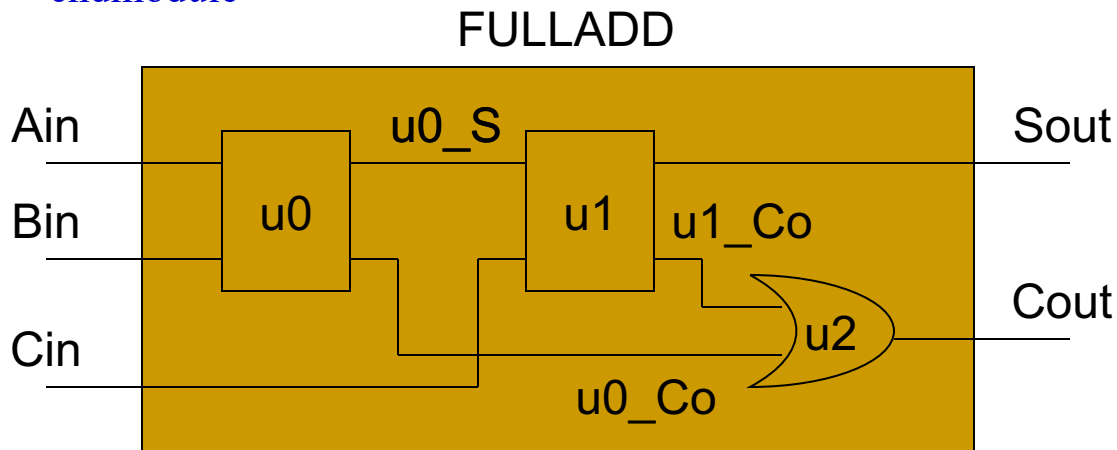
```
endmodule
```

Ví dụ thiết kế Full Adder

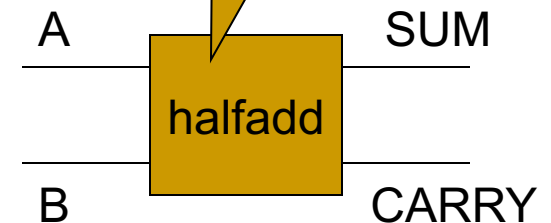
```
module fulladd (Ain, Bin, Cin, Sout, Cout);  
  input Ain, Bin, Cin;  
  output Sout, Cout;  
  wire u0_S, u0_Co, u1_Co;
```

```
  halfadd u0(.A(Ain), .B(Bin), .SUM(u0_S), .CARRY(u0_Co));  
  halfadd u1(.A(u0_S), .B(Cin), .SUM(Sout), .CARRY(u1_Co));  
  orunit u2(.A(u0_Co), .B(u1_Co), .Y(Cout));
```

```
endmodule
```



```
module HALFADD (A, B, SUM, CARRY);  
  input A, B;  
  output SUM, CARRY;  
  
  assign SUM = A ^ B;  
  assign CARRY = A & B;  
endmodule
```



Toán tử (operators)

- Toán tử trong Verilog-HDL tương tự toán tử trong C language

- Tuy nhiên:  `a = b = c = 1'b0;`

	Usage	Description
Arithmetic Operators		
+	$m + n$	Add n to m
-	$m - n$	Subtract n from m
-	$-m$	Negate m (2's complement)
*	$m * n$	Multiply m by n
/	m / n	Divide m by n
%	$m \% n$	Modulus of m / n

Toán tử (operators) cont.

Bitwise Operators		
\sim	$\sim m$	Invert each bit of m
$\&$	$m \& n$	AND each bit of m with each bit of n
$ $	$m n$	OR each bit of m with each bit of n
\wedge	$m \wedge n$	Exclusive OR each bit of m with n
$\sim \wedge$ $\wedge \sim$	$m \sim \wedge n$ $m \wedge \sim n$	Exclusive NOR each bit of m with n
Reduction Operators		
$\&$	$\& m$	AND all bits in m together (1-bit result)
$\sim \&$	$\sim \& m$	NAND all bits in m together (1-bit result)
$ $	$ m$	OR all bits in m together (1-bit result)
$\sim $	$\sim m$	NOR all bits in m together (1-bit result)
\wedge	$\wedge m$	Exclusive OR all bits in m (1-bit result)
$\sim \wedge$ $\wedge \sim$	$\sim \wedge m$ $\wedge \sim m$	Exclusive NOR all bits in m (1-bit result)

Toán tử (operators) cont.

Logical Operators		
!	!m	Is m not true? (1-bit True/False result)
&&	m && n	Are both m and n true? (1-bit True/False result)
	m n	Are either m or n true? (1-bit True/False result)
Equality Operators (compares logic values of 0 and 1		
==	m == n	Is m equal to n? (1-bit True/False result)
!=	m != n	Is m not equal to n? (1-bit True/False result)
Identity Operators (compares logic values of 0, 1, X and Z		
===	m === n	Is m identical to n? (1-bit True/False results)
!==	m !== n	Is m not identical to n? (1-bit True/False result)

Toán tử (operators) cont.

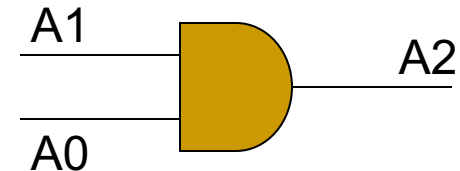
Relational Operators		
<	$m < n$	Is m less than n? (1-bit True/False result)
>	$m > n$	Is m greater than n? (1-bit True/False result)
<=	$m \leq n$	Is m less than or equal to n? (True/False result)
>=	$m \geq n$	Is m greater than or equal to n? (True/False result)
Logical Shift Operators		
<<	$m \ll n$	Shift m left n-times
>>	$m \gg n$	Shift m right n-times
Miscellaneous Operators		
? :	$sel?m:n$	If sel is true, select m: else select n
{ }	$\{m,n\}$	Concatenate m to n, creating larger vector
{ } { }	$\{n\{m\}\}$	Replicate m n-times

Phép tính Bit

\sim	$\&$	$ $	\wedge	$\sim\wedge$
(NOT)	(AND)	(OR)	(XOR)	(Ex-NOR)

- Ví dụ:

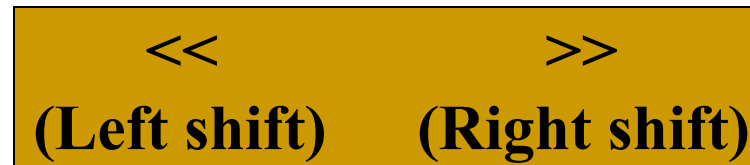
```
wire A2, A1, A0, B2, B1, B0;  
assign A2 = A1 & A0;
```



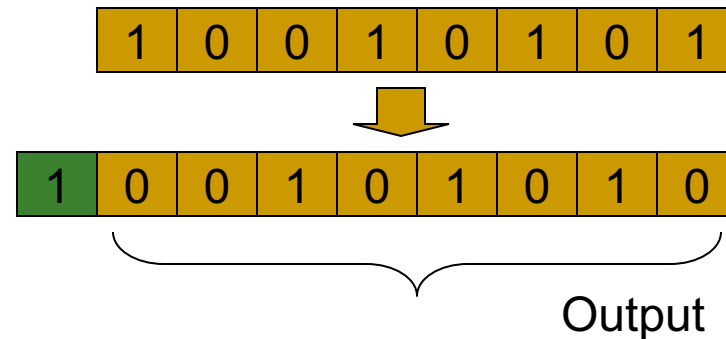
- Phép tính trong () được ưu tiên:
 - ```
assign B2 = ~((A1 & A0) | (~B1 ^ B0));
```



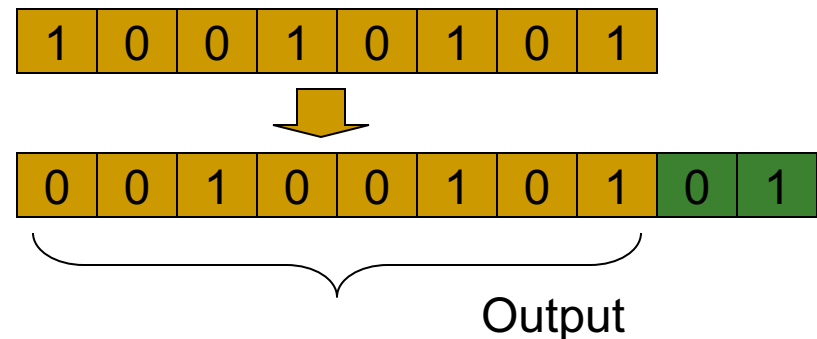
# Phép dịch Bit



- `wire [7:0] a, b;`  
`assign a = b << 1;`



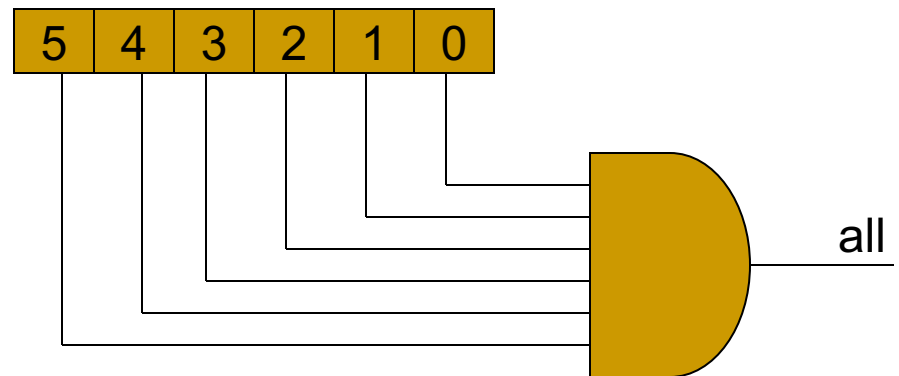
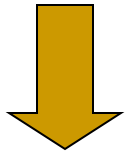
- `wire [7:0] a, b;`  
`assign a = b >> 2;`



# Phép tính rút gọn

|       |          |      |         |          |              |
|-------|----------|------|---------|----------|--------------|
| $\&$  | $\sim\&$ | $ $  | $\sim $ | $\wedge$ | $\sim\wedge$ |
| (NOT) | (AND)    | (OR) | (NOR)   | (XOR)    | (Ex-NOR)     |

- `reg [5:0] cnt;`  
`assign all = & cnt;`  
`assign parity = ^ cnt;`



```
assign all = cnt[5] & cnt[4] & cnt[3] & cnt[2] & cnt[1] & cnt[0];
assign parity = cnt[5] ^ cnt[4] ^ cnt[3] ^ cnt[2] ^ cnt[1] ^ cnt[0];
```

# Phép tính liên kết

{ }

## ■ Phép liên kết ở vế phải:

- Các bit được sắp xếp theo trật tự MSB  $\rightarrow$  LSB

```
wire [15:0] databus;
wire [7:0] addr_hi; wire [7:0] addr_lo;
assign databus = {addr_hi, addr_lo};
```

## ■ Phép liên kết ở vế trái:

- Nếu vế trái  $>$  vế phải thì các bit MSB của vế phải được lập về 0
- Vế trái  $<$  vế phải thì các bit MSB của vế phải bị cắt

```
input [3:0] a, b, sum;
wire carry;
assign {carry, sum} = a + b;
```

## ■ Phép liên kết lặp

```
{4{2'b10}}
= {{2'b10}, {2'b10}, {2'b10}, {2'b10}}
= 8'b10101010
```

# Phép tính so sánh

==    !=    ===    !==    <    <=    >    >=

- Các phép so sánh cho kết quả True, False
- === và !== là phép so sánh đồng nhất, sử dụng cho các tín hiệu có trạng thái x, z
- Riêng <= còn được sử dụng cho phép gán

Đây là phép gán

```
If (a == b)
 c <= 1'b1;
Else
 c <= 1'b0;
```

# Phép tính Logic

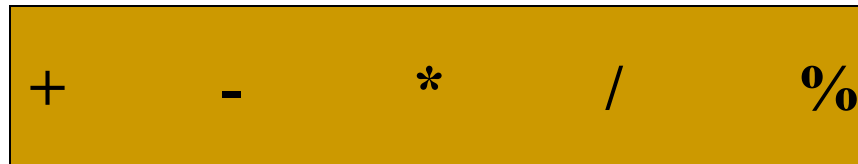
!    &&    ||

- Các phép logic cho kết quả True, False (phủ định, AND, OR)

Thường sử dụng với **If**

```
If ((a == b) && (a1 == b1))
 c <= 1'b1;
Else
 c <= 1'b0;
```

# Phép tính số học



- Riêng các phép “/” và “%” không thể tổ hợp (synthesize), chỉ có thể dùng đến mức mô phỏng

# Phép tính lựa chọn

?

:

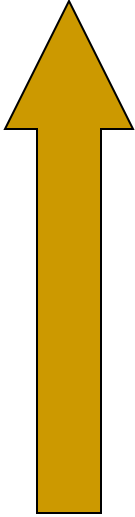
- Sử dụng với **assign** giống như sử dụng lệnh **if else**

```
wire [7:0] a, b, c;
assign a = (sel == 1'b1)? b: c;
```

TRUE

FALSE

# Trật tự ưu tiên

| Operator Precedence                                                                                                                                        |                                                                                                                                         |
|------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <p>! ~<br/>* / %<br/>+ -<br/>&lt;&lt; &gt;&gt;<br/>&lt; &lt;= &gt; &gt;=<br/>== != === !==<br/>&amp; ~&amp; ^ ~^<br/>  ~ <br/>&amp;&amp;<br/>  <br/>?:</p> | <p>highest precedence</p>  <p>lowest precedence</p> |



# Bài tập

1. Thiết kế bộ cộng nửa HALFADD 1-bit (tr.18)
2. Thiết kế bộ cộng đủ FULLADD 1-bit (tr.18)
3. Thiết kế bộ cộng đủ FULLADD8 8-bit từ bộ cộng đủ 1-bit
4. Thiết kế bộ cộng đủ FULLADD8 8-bit dùng toán tử “+”

## 2. Mạch tổ hợp

- Phát biểu assign
- Phát biểu always
- Phát biểu if, case, casex
- Phát biểu function

# Mạch tổ hợp (Combinational Circuit)

- Phát biểu **assign** được sử dụng để mô tả mạch tổ hợp

```
assign na = ~(in1 & in2); // NAND 2 đầu vào
assign out = (sel == 1'b1)? in1: in0; // Selector
assign sum = a + b; // Bộ cộng
```

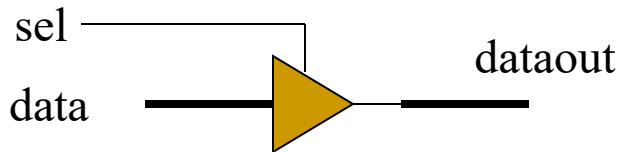
# Phát biểu *assign*

- Bộ chọn (Selector):

```
input [1:0] in0, in1; input sel;
output [1:0] out;

assign out = (sel == 1'b1)? in1: in0;
```

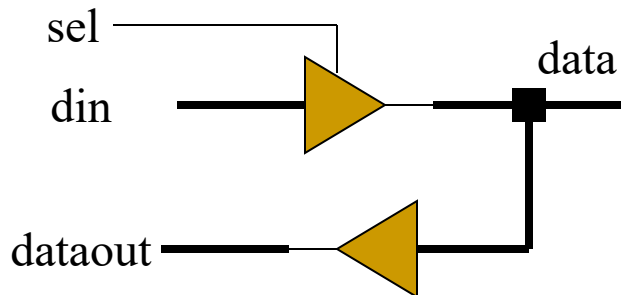
- Bộ đệm 3 trạng thái (three-state buffer):



```
input [7:0] data;
output [7:0] dataout; wire sel;

assign dataout = (sel)? data: 8'hz;
```

- Bộ đệm vào ra (inout buffer):



```
input [1:0] din;
inout [7:0] data;
wire sel; reg dataout;

assign data = (sel)? din: 8'hz;
always @(posedge CLK)
 dataout <= data;
```

# Phát biểu *always* (1)

- Các khối *always* được xử lý song song
- Các phát biểu nằm trong *always* được xử lý tuần tự

```
module Architecture(...);
input ...;
output ...; Xử lý đồng thời

 // concurrent statements
 always @(...)
 begin
 // sequential statements
 end

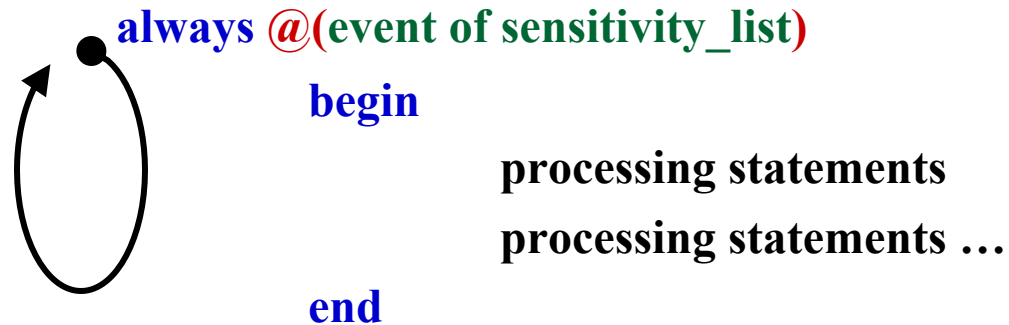
 Xử lý đồng thời

 // concurrent statements
 always @(...)
 begin
 // sequential statements
 end

endmodule
```

# Phát biểu *always* (2)

- **Cú pháp:**



- **Event:** sự kiện xảy ra
- **Sensitivity\_list:** danh sách tín hiệu nhạy cảm
- Các phát biểu được xử lý tuần tự từ trên xuống, sau phát biểu cuối cùng quá trình xử lý sẽ quay lại phát biểu đầu tiên
- Trường hợp có 1 phát biểu: có thể lược bỏ **begin end**

# Phát biểu *always* (3)

- Event có 2 dạng:
  - Thay đổi theo mức (level)
    - G thay đổi → phát biểu trong *always* được thực hiện
    - Có thể tập hợp nhiều tín hiệu nhạy cảm như: G, D, ...
    - Sử dụng cho mạch tổ hợp, latch
  - Thay đổi theo sườn (edge)
    - Theo sườn lên
    - Theo sườn xuống
    - Sử dụng cho Flip-Flop

```
// concurrent statements
always @(G)
begin
// sequential statements
end
```

```
// concurrent statements
always @(posedge CLK)
begin
// sequential statements
end
```

```
// concurrent statements
always @(negedge CLK)
begin
// sequential statements
end
```

# Phát biểu *always* (4)

- Sử dụng *always* để thiết kế mạch tổ hợp
- Có thể kết hợp sử dụng với *if*, *case*, *casex*

```
module SEL4TO1 (in, sel, out);
input [3:0] in; input [1:0] sel;
output out; reg out;

always @(sel or in)
begin
 if (sel == 2'h0)
 out = in[0];
 else if (sel == 2'h1)
 out = in[1];
 else if (sel == 2'h2)
 out = in[2];
 else
 out = in[3];
end
endmodule
```



# Phát biểu *if*

## ■ Cú pháp:

```
□ if (condition1) begin
 processing statements;
end else if (condition2) begin
 processing statements;
end else begin
 processing statements;
end
```

- Điều kiện đầu tiên có độ ưu tiên cao nhất
- Trường hợp chỉ có 1 phát biểu **begin end** có thể lược bỏ

```
if (sel == 2'h0)
 out = in[0];
else if (sel == 2'h1)
 out = in[1];
else if (sel == 2'h2)
 out = in[2];
else
 out = in[3];
```

# Phát biểu *case*, *casex*

## ■ Cú pháp:

### □ *case* (signal)

const: processing statements;

const: processing statements;

const: processing statements;

*default*: processing statements;

*endcase*

- Với các trạng thái tín hiệu X, Z → sử dụng *casex*

```
case (sel)
 0: out = in[0];
 1: out = in[1];
 2: out = in[2];
 3: out = in[3];
 default: out = 0;
endcase
```

```
casex (enc)
 8'b1xxxxxxx: out = 3'h7;
 8'b01xxxxxx: out = 3'h5;
 8'b001xxxxx: out = 3'h3;
 8'b00001xxx: out = 3'h2;
 8'b000001xx: out = 3'h1;
 default: out = 3'hx;
endcase
```

# Phát biểu *function*

- Cú pháp:

Có thể nhiều đầu vào

```
function [...] func_name;
input port_name;
begin
 processing statements;
end
```

- Có thể kết hợp sử dụng với  
if, case, casex

- Có thể gọi nhiều lần

```
module DEC2TO4 (in, out);
input [1:0] in;
output [3:0] out;
```

```
function [3:0] dec;
input [1:0] in;
begin
 case (in)
 0: dec = 4'h1;
 1: dec = 4'h2;
 2: dec = 4'h4;
 3: dec = 4'h8;
```

```
 endcase
endfunction
```

```
 assign out = dec(in);
endmodule
```

---

## 3. Xử lý tuần tự

- Sử dụng *always*
- Reset đồng bộ và không đồng bộ
- Bài tập

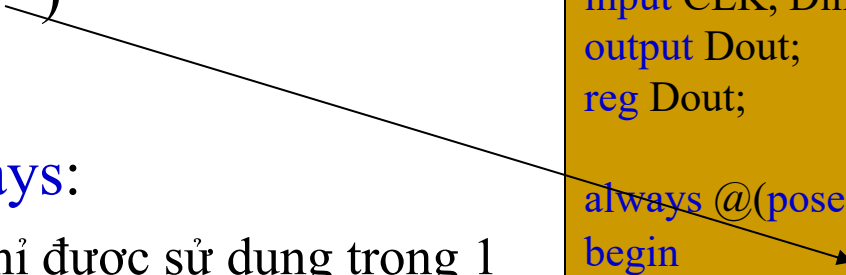
# Xử lý tuần tự

- Thiết kế cơ bản với **always**: xử lý tuần tự (dùng “<=”)
- Qui định cho **always**:
  - 1 tín hiệu đầu ra chỉ được sử dụng trong 1 **always**
  - 1 tín hiệu không dùng cho nhiều dạng sự kiện:
    - **always** @(posedge CLK or negedge CLK)
  - Trong 1 **always** không dùng nhiều clock
    - **always** @(posedge CLK1 or posedge CLK2)

```
module D_FF(CLK, Din, Dout);
input CLK, Din;
output Dout;
reg Dout;

always @(posedge CLK)
begin
 Dout <= Din;
end

endmodule
```

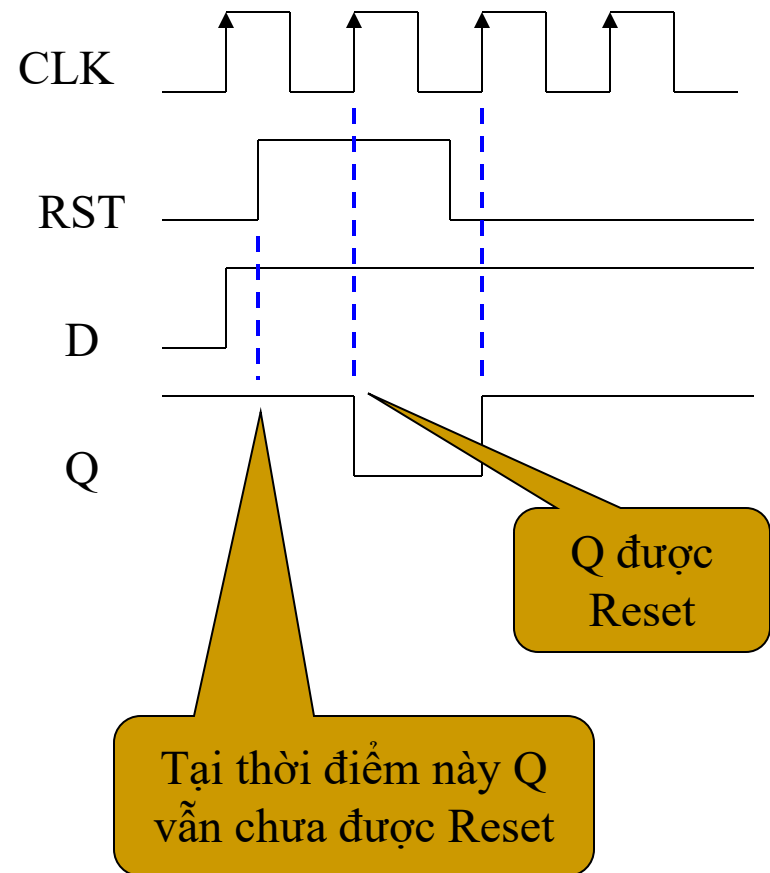


# DFF với Reset đồng bộ

```
module SYNC_FF(CLK, RST, D, Q);
input CLK, RST, D;
output Q;
reg Q;

always @(posedge CLK)
begin
 if (RST == 1)
 Q <= 0;
 else
 Q <= D;
 end
end

endmodule
```

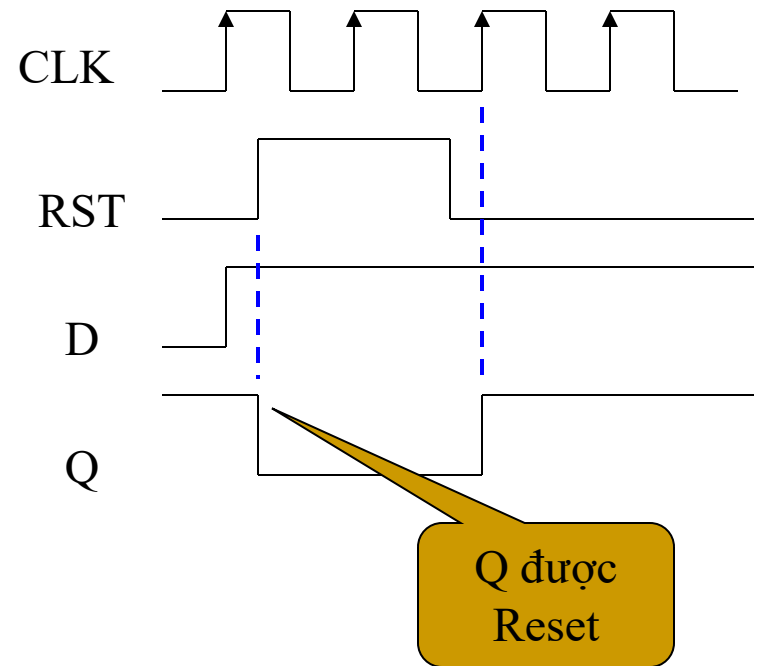


# DFF với Reset không đồng bộ

```
module SYNC_FF(CLK, RST, D, Q);
input CLK, RST, D;
output Q;
reg Q;

always @(posedge CLK or posedge RST)
begin
 if (RST == 1)
 Q <= 0;
 else
 Q <= D;
 end
end

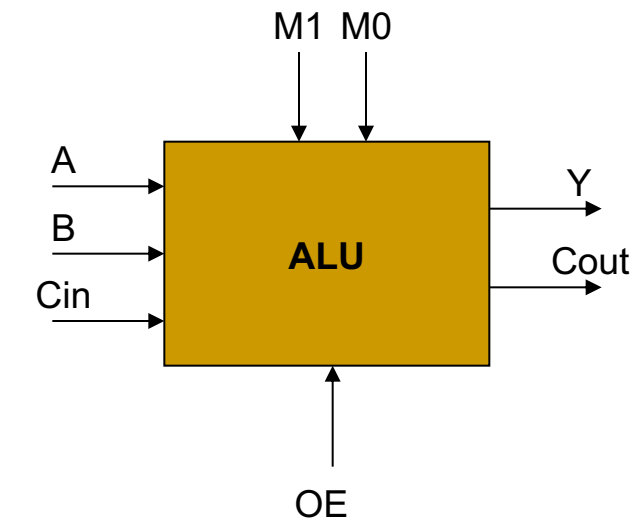
endmodule
```



# Bài tập lớn

## 1. Thiết kế bộ xử lý logic số học (Arithmetic Logic Unit - ALU)

| T/h         | Hướng | Độ lớn | Ý nghĩa           |
|-------------|-------|--------|-------------------|
| <b>A</b>    | Vào   | 8-bit  | Dữ liệu vào       |
| <b>B</b>    | Vào   | 8-bit  | Dữ liệu vào       |
| <b>Cin</b>  | Vào   | 1-bit  | Carry in          |
| <b>M0</b>   | Vào   | 1-bit  | Điều khiển mode   |
| <b>M1</b>   | Vào   | 1-bit  | Điều khiển mode   |
| <b>OE</b>   | Vào   | 1-bit  | Điều khiển đầu ra |
| <b>Y</b>    | Ra    | 8-bit  | Giá trị tính toán |
| <b>Cout</b> | Ra    | 1-bit  | Carry out         |



| Mode       | M1 | M0 | Ra                     |
|------------|----|----|------------------------|
| <b>ADD</b> | 0  | 0  | $Y = A + B$            |
| <b>SUB</b> | 0  | 1  | $Y = A - B$            |
| <b>AND</b> | 1  | 0  | $Y = A \text{ and } B$ |
| <b>OR</b>  | 1  | 1  | $Y = A \text{ or } B$  |

| T/h       | Giá trị | Ý nghĩa điều khiển      |
|-----------|---------|-------------------------|
| <b>OE</b> | 1       | Cho phép đầu ra Y       |
| <b>OE</b> | 0       | $Y = Z$ (trở kháng cao) |