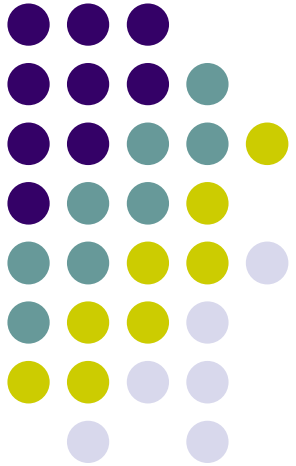
SEMICON Solutions

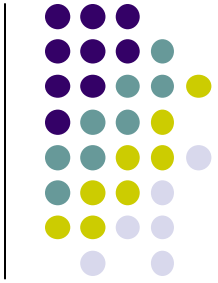
Verilog để synthesis



Trình bày: Đặng Tường Dương

15/07/2010

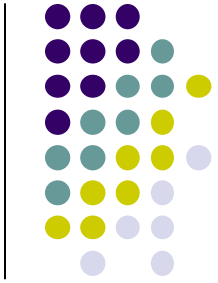
Hướng dẫn Coding?



Hướng dẫn về clocks and resets

- Coding để synthesis
- Phân vùng để synthesis
- RTL code hữu dụng
- Phụ đề
  - Thực hành coding cơ bản
  - Coding for portability

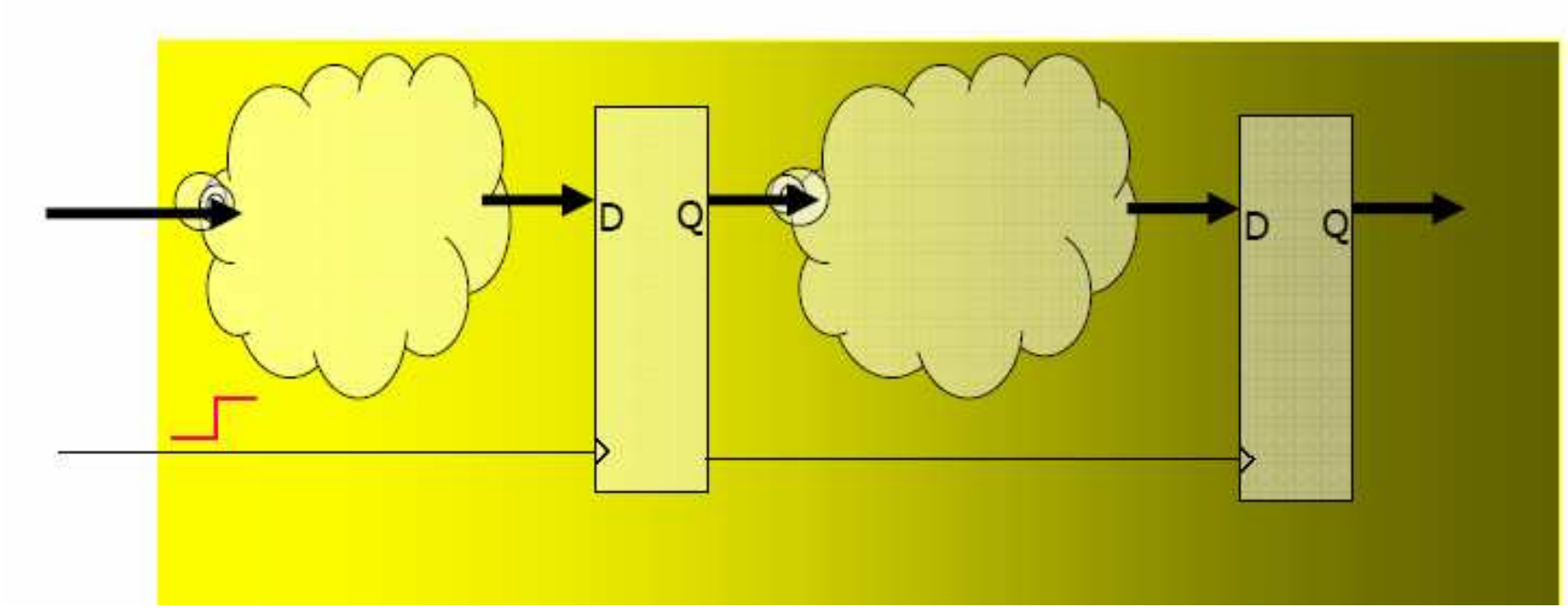
Hướng dẫn về clocks and resets



- Một cấu trúc clock đơn giản đó là dễ hiểu, phân tích và duy trì.
- Thêm vào đó, một cấu trúc clock đơn giản giúp xây dựng kết quả synthesis tốt hơn.
- Tránh dùng cạnh clock hỗn độn.
- Tránh dùng bộ đệm clock
- Tránh dùng gated clocks
- Tránh dùng clock được tạo ra từ bên trong.
- Gated clocks và những thiết kế công suất thấp
- Tránh dùng reset được tạo ra từ bên trong

Cấu trúc đường clock lý tưởng

- Cấu trúc đường clock mong đợi
  - Một clock chung
  - FF kích cạnh lên chỉ cho các thiết bị tuần tự



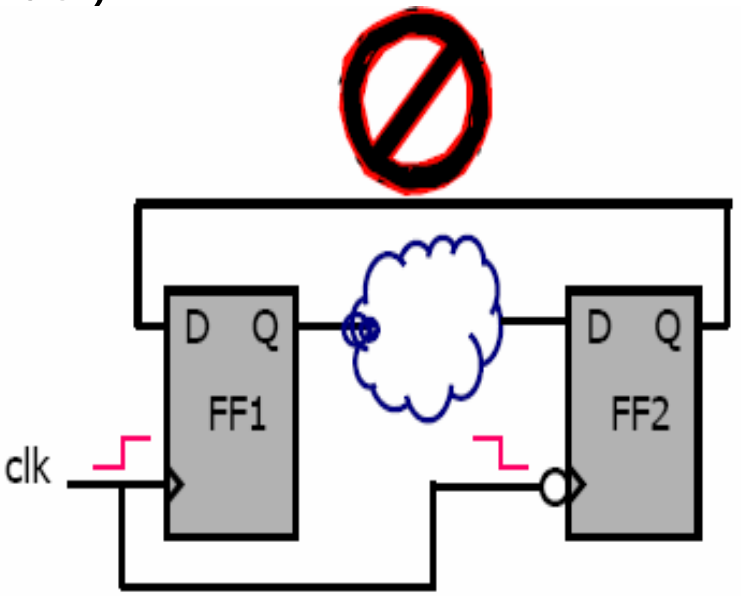
Tránh dùng cạnh clock hỗn độn

- Hướng dẫn
  - Tránh dùng cả cạnh lên và cạnh xuống kích các FFs
- Các cạnh hỗn hợp có thể dùng cho các mục đích timing cải tiến như DDR (tốc độ truyền dữ liệu gấp đôi)
- Sự quan tâm cần thiết.
  - Chú ý diễn biến trên từng chu

  kỳ nhiệm vụ trong hoạt động thiết kế
  - Cách kiểm tra cơ bản

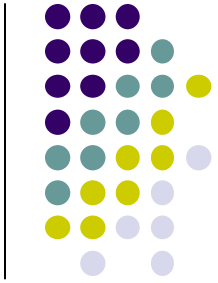
  (Scan-based testing)

  yêu cầu xử lí riêng rẽ cho FFs được kích cạnh dương và âm.



Bad example: mixed clock edges

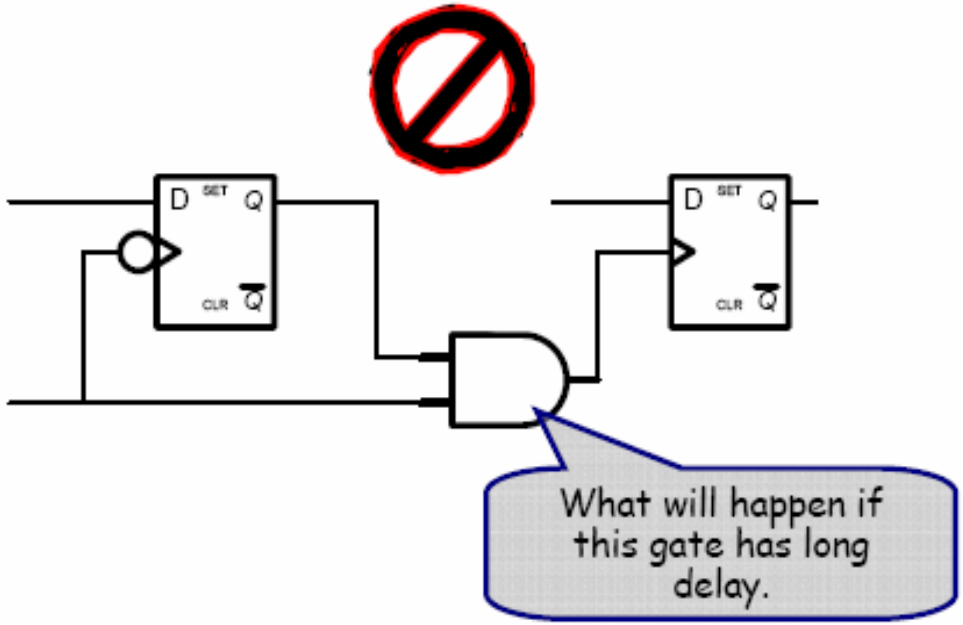
Tránh dùng bộ đệm clock



- Hướng dẫn
  - Tránh dùng các bộ đệm clock trong RTL code.
    - Bộ đệm clock được chèn sau synthesis như port của thiết kế vật lý.
    - Trong RTL code tổng hợp, đường clock được xem như là đường nối lý tưởng không có trễ (without delay).
    - Trong quá trình đặt và định tuyến (place and route), công cụ chèn cây clock sẽ chèn cấu trúc tốt nhất tiến gần đến lý tưởng, mạng phân phối clock cân bằng nhất có thể.

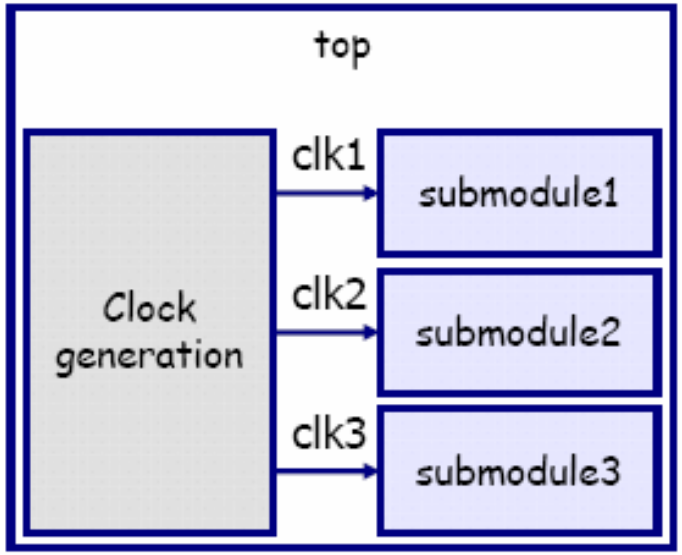
Tránh gated clock

- Hướng dẫn
  - Tránh coding gated clocks trong RTL, vì các mạch tạo cổng clock theo công nghệ và phụ thuộc timing
  - Timing không thích hợp có thể gây ra clock bị sai hoặc gai xung, set-up timg và hold time bị vi phạm.

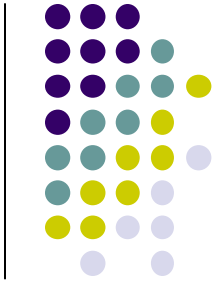


Gated clocks và những thiết kế công suất thấp

- Hướng dẫn
  - Giữ mạch tạo clock và reset (đối với 1 gated clock, 1 clock hoặc reset được tạo bên trong) khi 1 module riêng rẽ ở mức top
  - Phần khỏi thiết kế để mà tất cả các mạch logic trong 1 module đúng chỉ 1 clock và 1 reset
  - Cách ly mạch tạo clock và reset để sử dụng việc phân tích thời gian chuẩn và kỹ thuật chèn scan.



Tránh dùng resets tạo bên trong

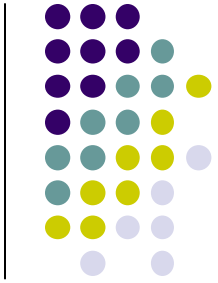


- Hướng dẫn
  - Tránh reset điều kiện, tạo bên trong nếu có thể.
  - Nói chung, tất cả các thanh ghi nên dùng chung một reset.
  - Nếu một reset điều kiện yêu cầu, tạo tín hiệu riêng cho tín hiệu reset, và cách li mạch logic reset điều kiện trong một module

```
module poor_style (clk, rst, a, b);
  input clk;
  input rst;
  input a, b;
  reg [7:0] my_reg;
  always @ (posedge clk or rst or a or b) begin
       if ((rst or a or b) == 1) my_reg = 8'b0;
       else begin
          ...
       end
  end
endmodule
```

```
module better_style (rst_out, rst_in, a, b);
  output rest_out;
  input rst_in;
  input a, b;
  wire rst_out = rst_in | a | b;
endmodule
module better_style (clk, rst);
  input clk, rst;
  reg [7:0] my_reg;
  always @ (posedge clk or rst) begin
       if (rst == 1) my_reg = 8'b0;
       else begin
          ...
       end
  end
endmodule
```

Coding để synthesis



- Để tạo code cải thiện thời gian biên dịch tốt nhất và những kết quả synthesis.
  - Kiểm tra
  - Đặc tính
  - Đơn giản hóa phân tích thời gian tĩnh
  - Hành vi mạch cổng giống như code RTL gốc
- Những thanh ghi
- Tránh chốt (latches)
- Tránh mạch hồi tiếp tổ hợp
- Xác định sensitivity list
- Gán Blocking and non-blocking
- Diễn tả case v.s. diễn tả if-then-else
- Coding máy trạng thái (state machines)

Các thanh ghi

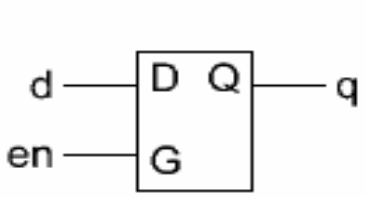
- Hướng dẫn
  - Các thanh ghi (flip-flops) là cơ chế Cho mạch tuần tự (như được so với chốt (latches).
  - Sử dụng tín hiệu reset của người thiết kế để tạo giá trị ban đầu cho thanh ghi.
    - Không dùng diễn tả “initial” để Tạo giá trị ban đầu cho tín hiệu vì nó không thể tổng hợp (synthesizable).
      - Những điều này có thể gây ra sự khác biệt mô phỏng trước và sau synthesis.

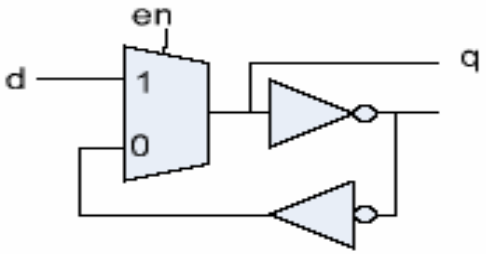
```
// process with synchronous reset
always @ (posedge clk)
  begin
    if (reset == 1'b1) begin
      ... something ...
    end else begin
      ... something else ...
    end
  end // INFER_REG_PROC
```

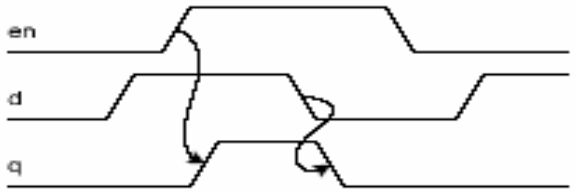
```
// process with asynchronous reset
always @ (posedge clk or negedge rst_n)
  begin
    if (reset == 1'b0) begin
      ... reset behavior
    end else begin
      ... normal behavior
    end
  end // INFER_REG_PROC
```

Tránh dùng chốt (1/2)

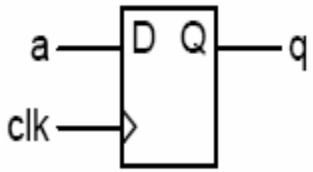
- Luật: tránh dùng chốt
  - Chốt
    - Khi CLK = 1, chốt sẽ truyền
      - D sẽ truyền đến Q như một bộ đệm
    - Khi CLK = 0, the latch is opaque
      - Q sẽ giữ giá trị cũ cũa nó độc lập với D
    - Truyền chốt hoặc chốt level-sensitive latch

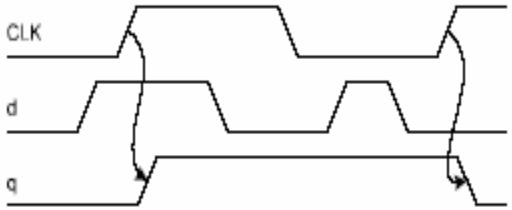






- Flip-flop
  - Khi CLK lên, D được chép đến Q
  - Trong tất cả thời gian, Q giữ giá trị của nó
  - FF kích cạnh lên, master slave flip-flop



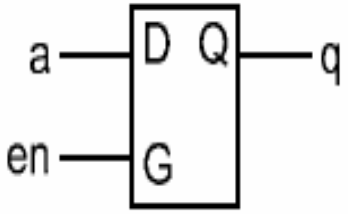


SEMICON

Tránh dùng chốt (2/2)

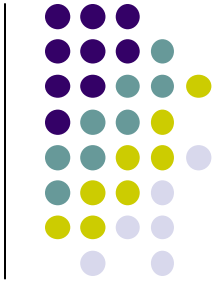
- Điều gì tạo nên chốt không dự định
  - Phép gán nhỡ
  - Điều kiện nhỡ
- Làm thế nào tránh chốt không dự định
  - Tránh chốt bằng cách gán hoàn toàn ngõ ra cho tất cả ngõ ra cho các điều kiện ngõ vào dùng else và default.

```
// process causes latch inference
always @ (a or en)   if (en==1'b1) q = a;
 // its semantics imply that q must hold
 // its old value when en is low.
```




```
// process without latch inference
always @ (a or en)   if (en==1'b1) q = a; else q = 1'b0;
```

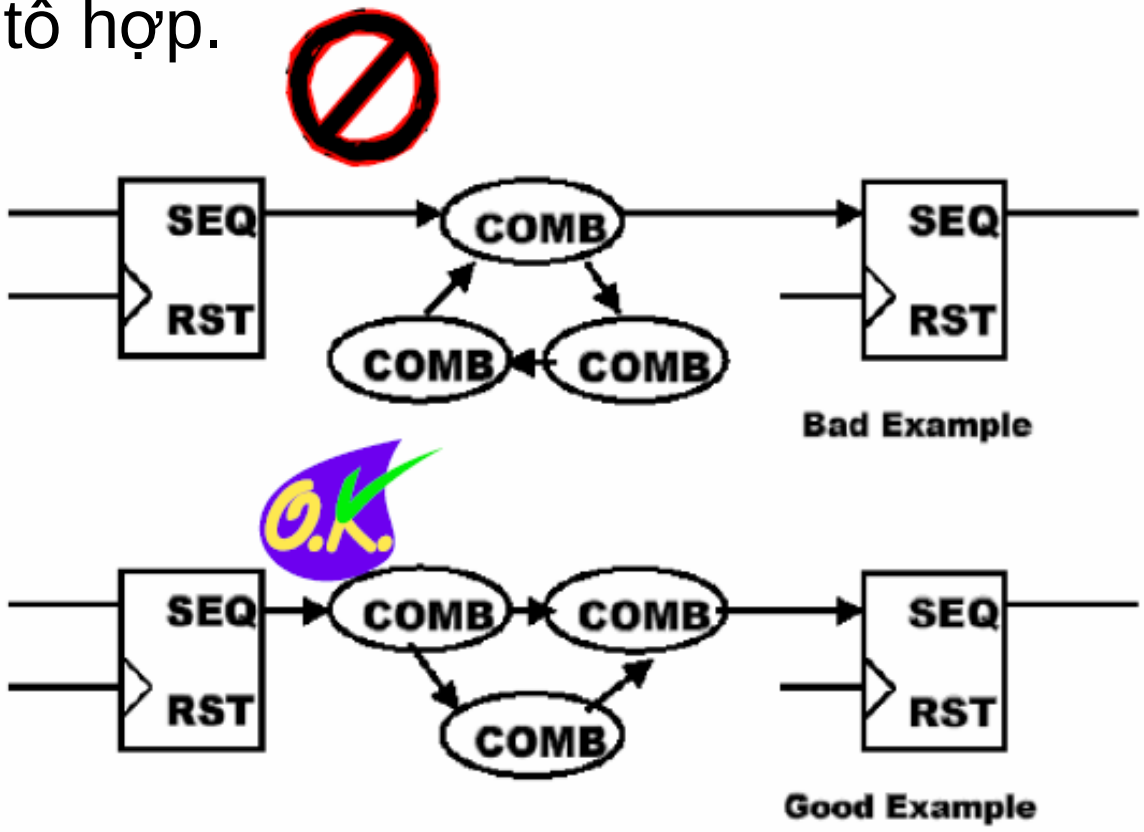
Tránh dùng chốt (2/2)



```
always @(a or b or en)
begin
    If (en) begin
        Out1 = a & b;
        Out2 = a | b;
    end
    else
       begin
           out1 = a ^ b;
         // out2 = a & !b;
       end
```

Tránh hồi tiếp tổ hợp (combinational feedback)

- Hướng dẫn
  - Tránh hồi tiếp tổ hợp.



Xác định sensitivity list

- Luật: đặt một list biến nhạy trong mỗi khối always.
  - Nếu không, hành vi thiết kế trước synthesis có thể khác netlist sau synthesis.
- Hướng dẫn
  - Đối với khối tổ hợp

(không có thanh ghi hoặc chốt), liệt kê biến nhạy phải gồm mọi tín hiệu được đọc.

  - Đối với khối tuần tự, liệt kê

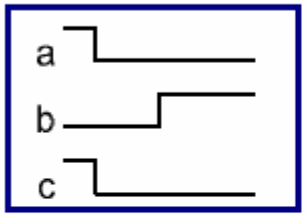
biến nhạy phải gồm tín hiệu clock được đọc.

  - Đảm bảo liệt kê biến nhạy

chỉ chứa nhưng tín hiệu cần thiết.

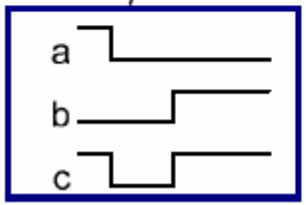
    - Thêm những tín hiệu không
    - cần thiết sẽ làm chậm đi việc mô phỏng.

```
// process with incomplete sensitivity list
always @ (a)   c = a or b;
```



Pre-synthesis simulation waveform

Synthesis netlist

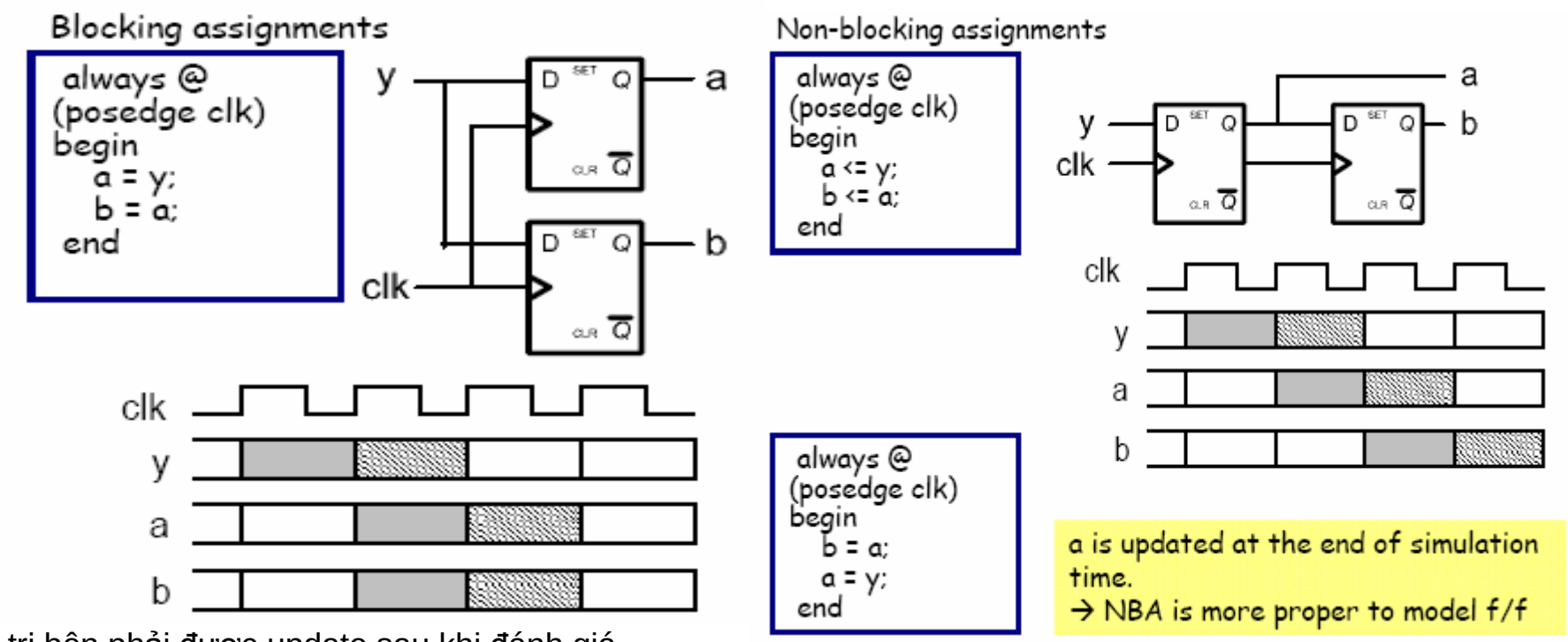


Post-synthesis simulation waveform

```
// process with complete sensitivity list
always @ (a or b)   c = a or b;
```

Gán blocking/ non-blocking

- Gán Blocking sẽ thực thi theo một thứ tự tuần tự
  - Trong khi gán non-blocking thực thi cùng lúc.
  - Luật: dùng gán **non-blocking (<=) trong khối always**



Giá trị bên phải được update sau khi đánh giá.

→ Gây ra điều kiện chạy đua

Điều kiện chạy đua (1/2)

- Chuyện gì xảy ra nếu ta lấy mẫu tín hiệu a trong diễn ta always khác?

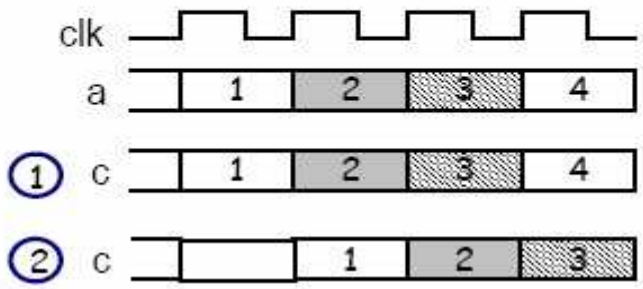
Blocking assignments

```
always @ (posedge clk)  begin
   a = y;
end

always @ (posedge clk) begin
   c = a
end
```

a is updated right after the evaluation.

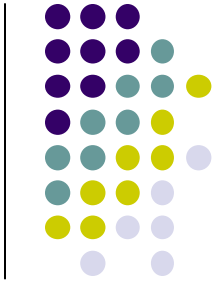
a is sampled right after the evaluation.

Q. What will happen to signal c? Choose one.



→Answer is "It depends on simulator implementation"
→ We call this condition "Race condition"
→ We should avoid this condition

Điều kiện chạy đua (2/2)



- Để tránh điều kiện chạy đua, dùng gán non-blocking

Non Blocking assignments

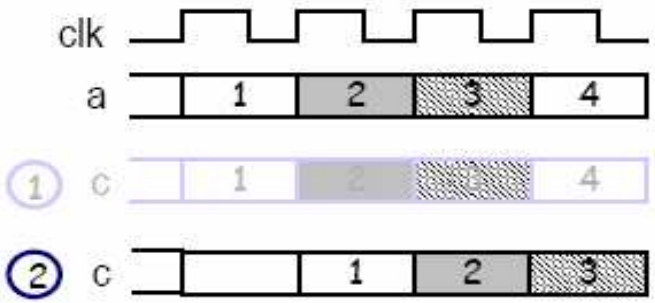
```
always @ (posedge clk)  begin
   a <= y;
end

always @ (posedge clk) begin
   c = a
end
```

a is updated at the end of simulation time.

a is sampled right after the evaluation.

Q. What will happen to signal c? Choose one.

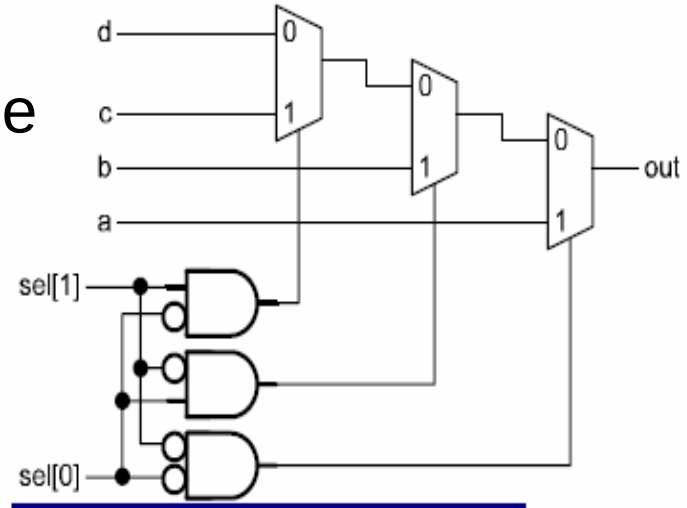
→ Answer is 2 regardless of simulator implementation



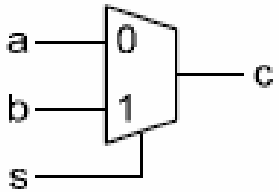
Diễn tả Case và If-then-else

- Một diễn tả case tạo ra một (bộ kênh) multiplexer một mức đơn
- Trong khi một diễn tả if-then-else tạo ra bộ kênh chọn tổ hợp, được mã hóa ưu tiên.
- Hướng dẫn
  - Bộ kênh là mạch chạy nhanh hơn.

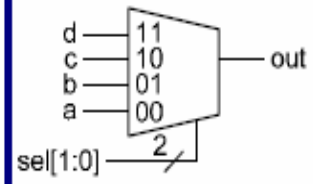
```
always @ (sel or a or b or c or d)
  if (sel == 2'b00) out = a;
  else if (sel == 2'b01) out = b;
  else if (sel == 2'b10) out = d;
  else out = d;
```




```
assign c = (s) ? b : a;
```



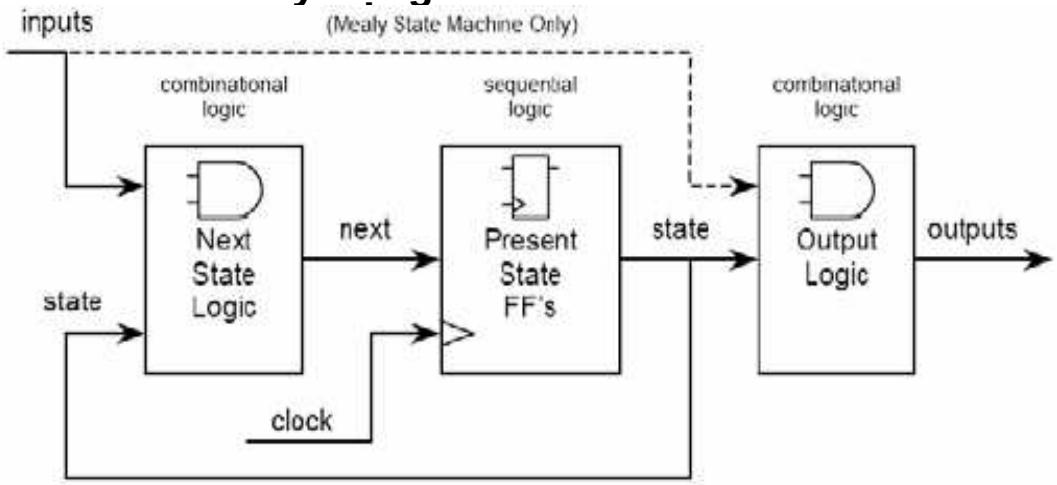

```
always @ (sel or a or b or c or d)
  case (sel)
   2'b00:   out = a;
   2'b01:   out = b;
   2'b10:   out = c;
   default: out = d;
  endcase
```



SEMICON

Coding máy trạng thái

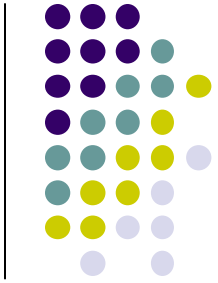
- **Hướng dẫn**
  - **Tách rời diễn tả máy trạng thái thành 2 quy trình**
    - **Mạch tổ hợp**
    - **Mạch tuần tự**
  - **Dùng diễn tả `define để định nghĩa vector trạng thái.**
  - **Giữ logic FSM và logic non-FSM trong những module tách rời.**
  - **Gán giá trị mặc định cho máy trạng thái.**




```
.....
 reg [1:0]  state, next_state; // FSM state
 //-----------------------------------------
 parameter IDLE    = 2'h0;
 parameter FOUND   = 2'h1;
 parameter MODULUS = 2'h2;
 parameter DONE    = 2'h3;
 //-----------------------------------------
always @ (state or input)
  case (state)
     IDLE:   begin
               ...
               next_state <= FOUND;
             end  end // IDLE
     FOUND:  begin
               ...
               next_state <= DONE;
             end // FOUND
     endcase
 always @ (posedge clk or negedge rst_b)
     if (rst_b == 1'b0) state <= IDLE;
     else state <= next_state;
endmodule
```

SEMICON

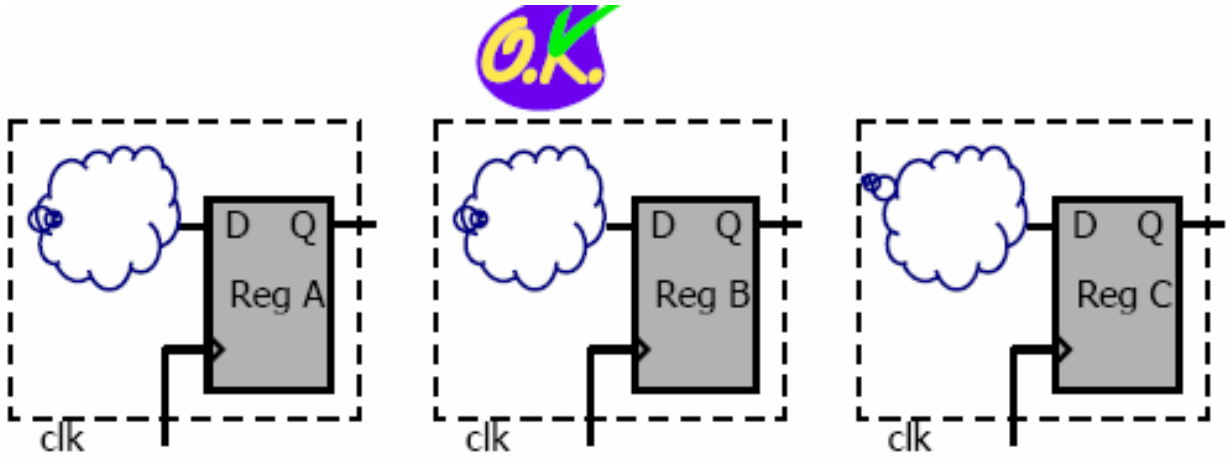
Phân chia thiết kế để synthesis



- Đặt thanh ghi trước tất cả ngõ ra
- Đinh vị khối logic tổ hợp bên trong một module đơn lẻ
- Tách rời các module có mục đích thiết kế khác nhau
- Tránh dùng Logic bất đồng bộ
- Toán tử số học: gộp thành khối

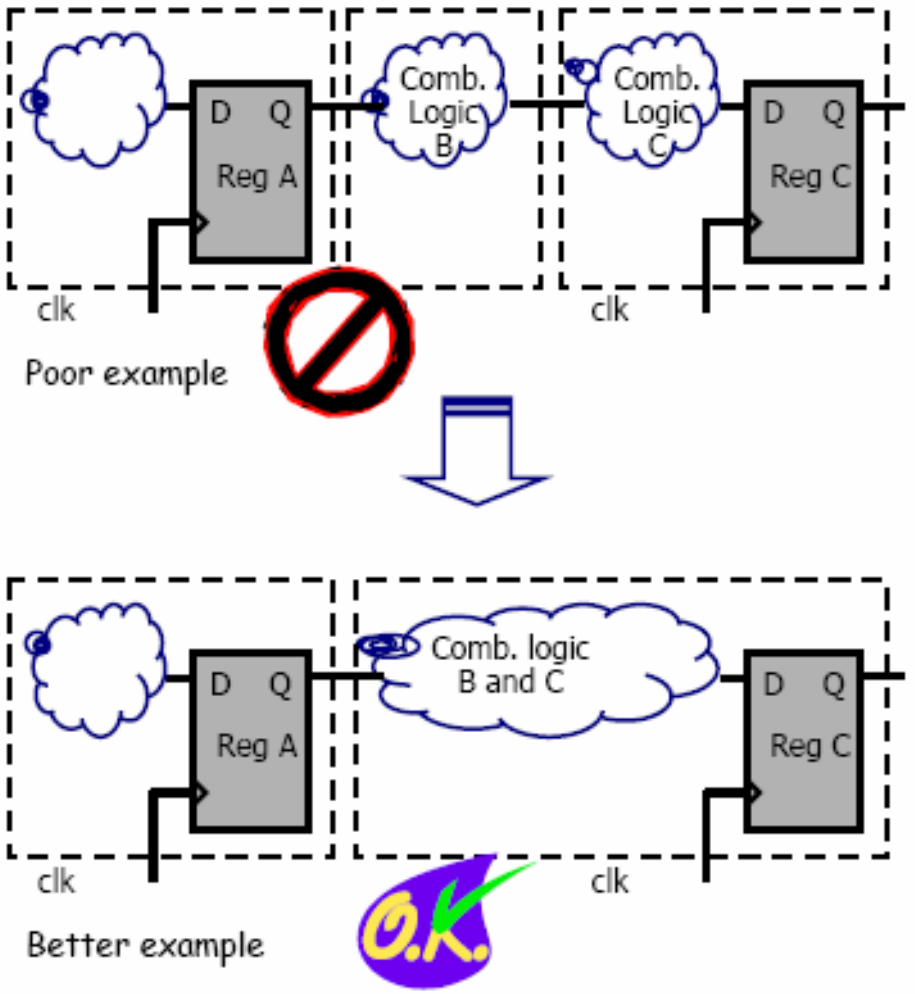
Đặt thanh ghi trước các ngõ ra

- Hướng dẫn
  - Đối với mỗi khối trong thiết kế theo hierarchy dung thanh ghi cho tất cả các tín hiệu ngõ ra.
    - Đạt thanh ghi cho các tín hiệu ngõ ra từ mỗi khối làm đơn giản hóa quy trình synthesis vì nó làm đường lái ngõ ra tốt và delay ngõ vào tốt hơn.



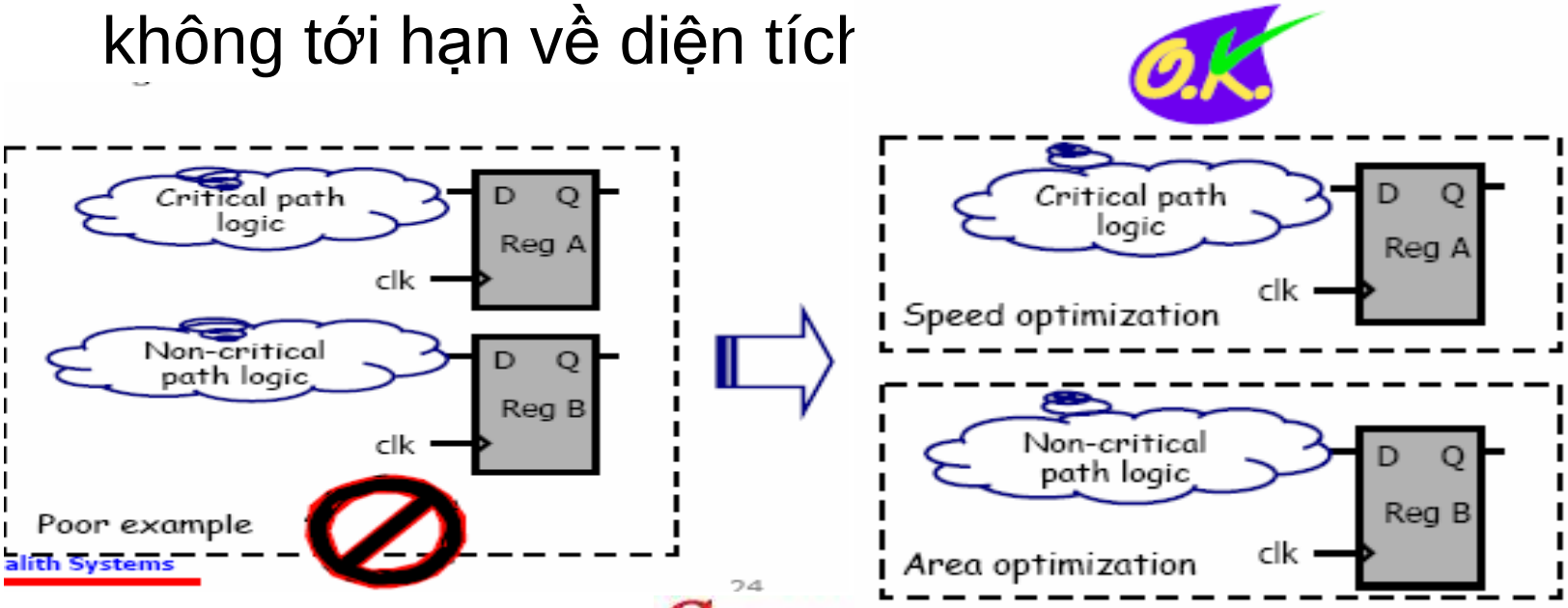
Đinh vị khối logic tổ hợp bên trong một module đơn lẻ

- Hướng dẫn
  - Giữ các mạch tổ hợp liên quan với nhau trong cùng một module.
    - Công cụ tổng hợp tự do hơn trong việc tối ưu thiết kế khi các khối tổ hợp có liên quan nhau được đặt trong cùng một module.
    - Giữ các mạch logic tổ hợp liên quan trong cùng một module se dễ dàng tính độ dự trữ timing và cho phép mô phỏng nhanh hơn.



Tách rời các module có mục đích thiết kế khác nhau

- Hướng dẫn
  - Giữ các đường logic tới hạn trong một module riêng từ các đường logic không tới hạn.
  - Công cụ Synthesis co thể tối ưu tốc độ các đường logic tới hạn cùng lúc tối ưu đường logic không tới hạn về diện tích

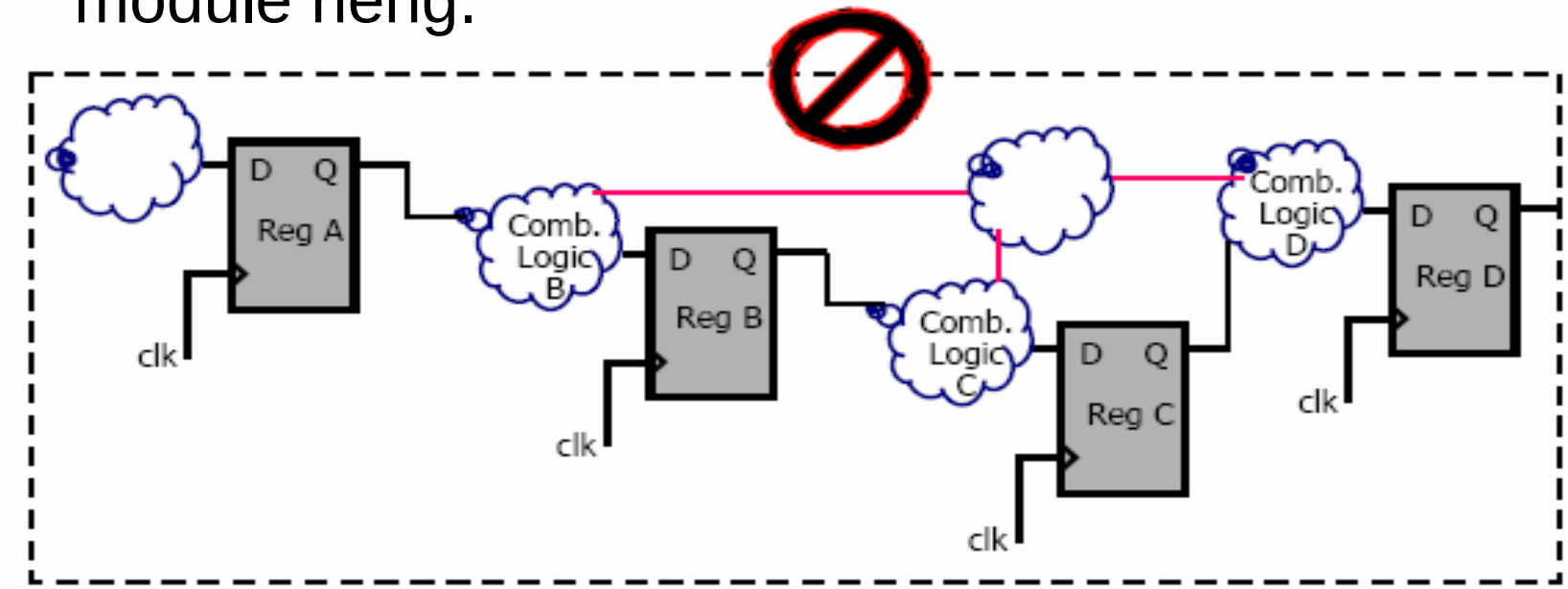


Tránh dùng Logic bất đồng bộ

- Hướng dẫn
  - Tránh dùng mạch bất đồng bộ.
  - Nếu logic bất đồng bộ yêu cầu, phân chia mạch bất đồng bộ trong một module riêng rẽ từ module đồng bộ.
  - Cách ly mạch bất đồng bộ trong một module rời tạo cho code kiểm tra dễ dàng hơn.
  - Mạch bất đồng bộ cần được xem xét cẩn thận để kiểm tra chức năng và timing.

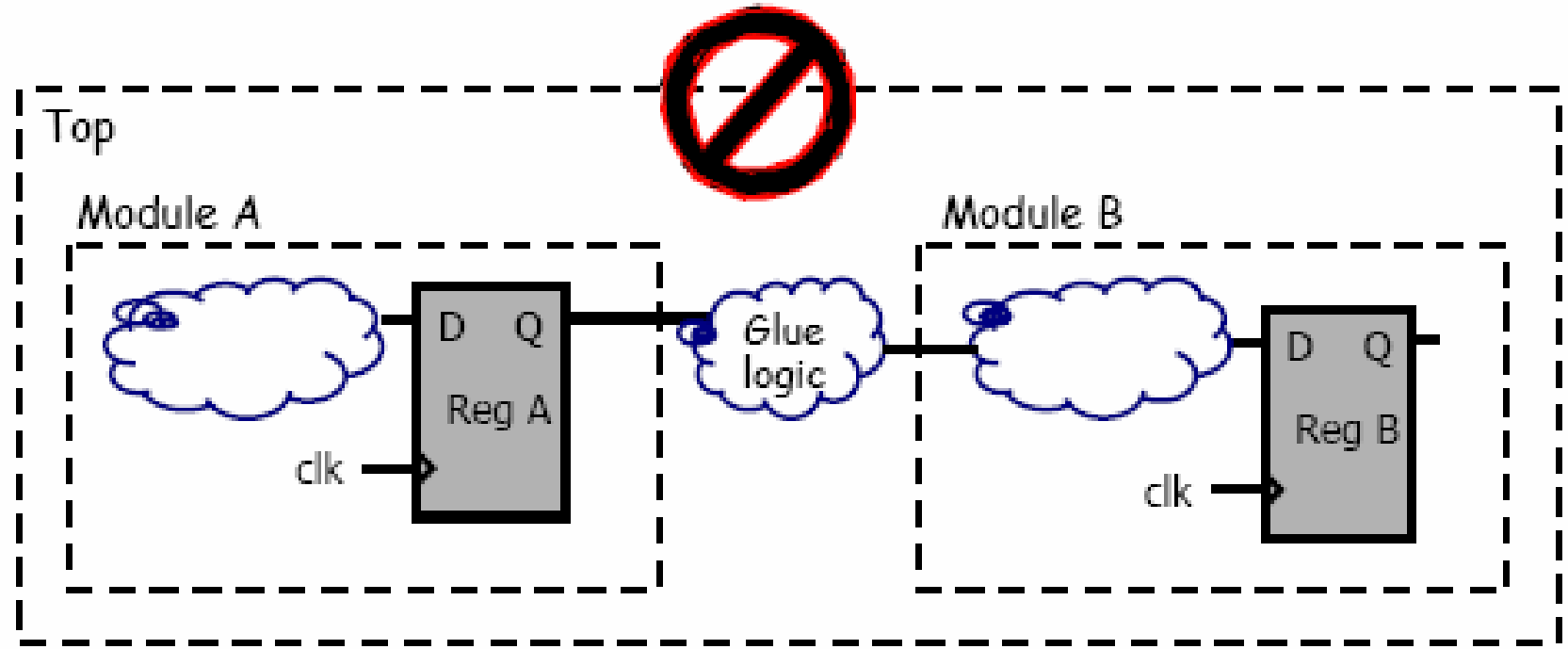
Tránh đường nhiều chu kỳ

- Tránh các đường nhiều chu kỳ
  - Trong đường nhiều chu kỳ, dữ liệu di ngang qua đường tử một thanh ghi đến thanh ghi khác trong nhiều chu kỳ clock.
  - Đường nhiều chu kỳ là khó khăn để phân tích đúng.
- Nếu cần, giữ poin-to-point ngoại lệ trong một module riêng.

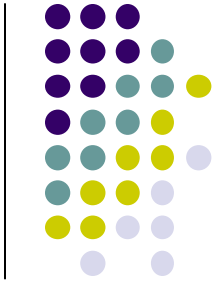


Tránh mạch glue logic ở module top

- Hướng dẫn
  - Đừng đặt logic gate-level ở module top của hierarchy thiết kế.
  - Cho phép hierarchy thiết kế chứa đựng gates chỉ ở mức lá của cây hierarchy.

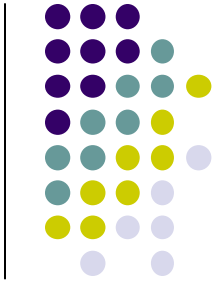


Code RTL hữu dụng

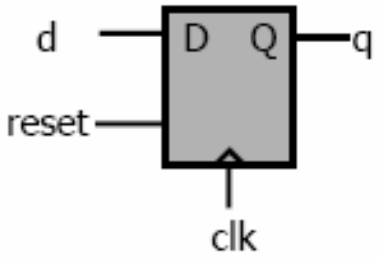


- Trong tóm tắt, code RTL mô tả tốt là luôn được yêu cầu.
- Chuẩn bị code RTL cho nhưngr thành phần cơ bản sau:
  - Mạch tuần tự
    - Flip flop
- Mạch tổ hợp
  - AND gate
  - MUX
  - Comparator
  - Encoder/Decoder
  - Tristate buffer

Code RTL hữu dụng



- Flip Flop description




```
//Flip Flop:
always @( posedge clk )
        q <= d ;
```

```
//Flip Flop with asynchronous reset:
always @( posedge clk or posedge reset)
if (reset)
        q <= 0 ;
else
        q <= d ;
```

```
//Flip Flop with asynchronous reset:
always @(posedge clk or negedge reset)
if (~resetb)
        q <= 0 ;
else
        q <= d ;
```

```
//Flip Flop with synchronous reset:
always @(posedge clk)
if (reset)
        q <= 0 ;
else
        q <= d ;
```

```
//Flip Flop with synchronous reset:
always @(posedge clk)
if (~resetb)
        q <= 0 ;
else
        q <= d ;
```

Code RTL hữu dụng

- 3 mô tả khác nhau của cổng AND

```
//COMBINATIONAL LOGIC :
reg out;
always @( a or b ) out <= a & b;
```

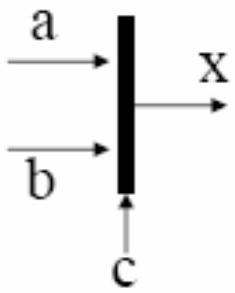
```
//COMBINATIONAL LOGIC :
wire out;
assign out = a & b;
```

```
//COMBINATIONAL LOGIC :
wire out = a & b;
```

Code RTL hữu dụng

- MUX

```
//MUX:
always @( a or b or c )
if ( c )      x <= a ;
else          x <= b ;
```



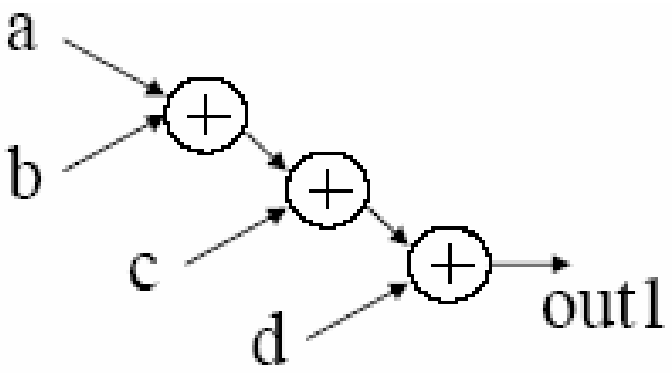
- Comparator

```
//COMPARATORS:

always @( a or b )
begin
            if ( a= =b ) y <= 1;
            else y <= 0;
end
```

Code RTL hữu dụng

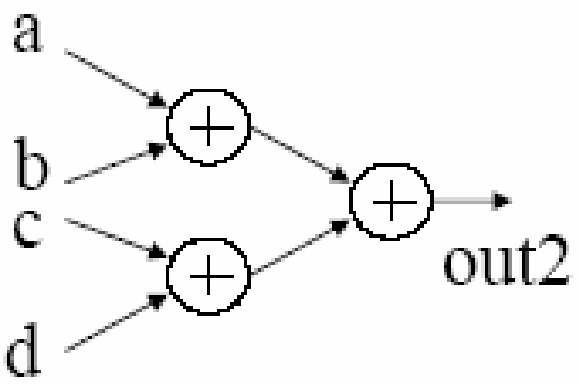
- Cấu trúc khối tổ hợp

```
//COMBINATIONAL BLOCK STRUCTURE:
always @( a or b or c or d )
begin
        out1 <= a + b + c + d ;
        out2 <= (a+b) + (c+d);
end
```

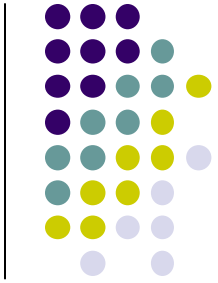


- Bộ giải mã

```
//DECODER:
input [1:0] in;
always @( in )
        case (in)
        2'b00 : out <= 4'b0001;
        2'b01 : out <= 4'b0010;
        2'b10 : out <= 4'b0100;
        2'b11 : out <= 4'b1000;
        endcase
```



Code RTL hữu dụng



```
//ENCODER:
always @( a )
begin
        if(a == 4'b0001) y <= 2'b00;
        else if(a == 4'b0010) y <= 2'b01;
        else if(a == 4'b0100) y <= 2'b10;
        else if(a == 4'b1000) y <= 2'b11;
        else y <= 2'bxx;
end
```

```
//ENCODER:
always @( a )
        case ( a )
        4'b0001: y <= 2'b00;
        4'b0010: y <= 2'b01;
        4'b0100: y <= 2'b10;
        4'b1000: y <= 2'b11;
        default : y <= 2'bxx;
        endcase
```

```
//PRIORITY ENCODER:
always @( a )
begin
        if(a[3]) y <= 2'b00;
        else if(a[2]) y <= 2'b01;
        else if(a[1]) y <= 2'b10;
        else if(a[0]) y <= 2'b11;
        else y <= 2'bxx;
end
```

```
//PRIORITY ENCODER:
always @( a )
        case ( a )
        4'b1xxx: y <= 2'b11;
        4'b01xx: y <= 2'b10;
        4'b001x: y <= 2'b01;
        4'b0001: y <= 2'b00;
        default : y <= 2'bxx;
        endcase
```

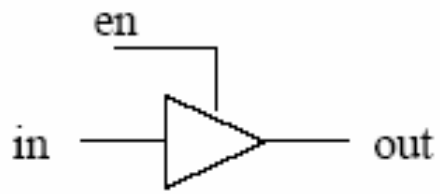
Code RTL hữu dụng

- Bộ đệm 3 trạng thái

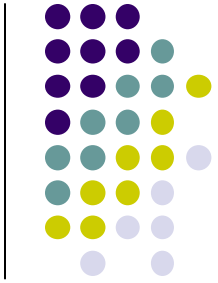
```
//Tri-state buffer:
always @( en or in) begin
    if ( en )           out <= in ;
    else                out <= 1'bz ;
end
```

```
//Tri-state buffer:
wire out;
assign out = en0 ? in0 : 1'bz ;
```

```
//Tri-state buffer
wire out = en0 ? in0 : 1'bz ;
```

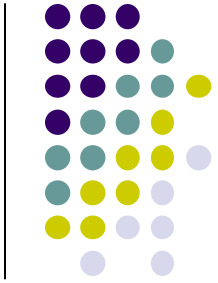


Thực hành coding cơ bản



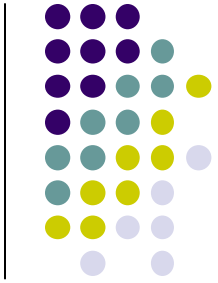
- Tập trung các mạch đơn giản và cấu trúc RTL cơ bản
- Cách đặt tên chung
- Thêm Header của các file
- Dùng comments
- Giữ lệnh trong những dòng riêng biệt
- Chiều dài line
- Lùi dòng
- Không dùng từ câm trong Verilog
- Thứ tự port
- Kết nối port
- Dùng function

Đặt tên chung



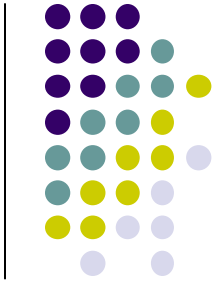
- Rule: Phát triển đặt tên cho thiết kế phải được lưu trong tài liệu và dùng cùng với thiết kế.
- Hướng dẫn
- Dùng các ký tự chữ thường cho toàn bộ các tên tín hiệu, tên biến và tên port.
- Dùng ký tự hoa cho tên các biến hằng và các loại được định ngĩa do người sử dụng đặt.
- Dùng tên có nghĩa tên của tín hiệu, port, function, và parameters.
- Dùng tên ngắn nhưng mô tả tên của parameters.
- Dùng ‘clk’ như tiền tố cho tất cả clock trong thiết kế.
- Dùng ‘_n’ (hoặc ‘_b’) như hậu tố cho các tín hiệu tích cực.
- Dùng ‘rst’ cho các tín hiệu reset. Nếu tích cực mức thấp, dùng ‘rst_n’.
- Dung một thứ tự bit cho tín hiệu bus nhiều bit.
- Dùng cùng tên hoặc tên tương tự cho tên của port và tín hiệu được kết nối.

Đặt tên chung



convention	Use
*_b	Active low signal (rst_b)
*_r	Output of a register (count_r)
*_a	Asynchronous signal (addr_strobe_a)
*_pn	Signal used in the nth phase (enable_p2)
*_nxt	Data before being registered into a register with the same name
*_z	Trisate internal signal
clk*	Use 'clk' as a prefix for clock signals
rst*	Use 'rst' as a prefix for reset signals

Thêm tiêu đề vào tập tin mã nguồn



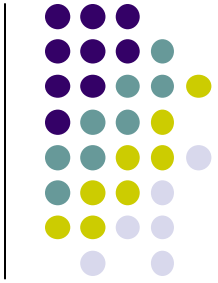
Nguyên tắc: Đặt vào đầu mỗi tập tin mã nguồn, bao gồm cả bản thảo

Cú Pháp:

- Tên tập tin.
- Tác giả.
- Mô tả các chức năng.
- Danh sách các tính năng chính.
- Ngày tập tin đã được tạo.
- Lịch sử sửa đổi bao gồm: ngày sửa đổi, tên tác giả và các chức năng bị thay đổi.

```
/*************************************************
 * Copyright (c) 2005 by Ando Ki.
 * All right reserved.
 *
 * http://www.dynalith.com
 *************************************************
 * File: euclide.v
 * Author: Ando Ki
 * Abstract: Euclide GCD (greatest common divisor)
 *           computes the greatest common divisor
 *           of two non-negative integers.
 * Date: 2005.09.30
 * Revision history:
 *      2005.10.01  Ando Ki   0.2  refinement
 *      2005.09.20  Ando Ki   0.1  original
 *************************************************
 */

module GCD ( clk, reset, a, b, valid, c, done);
  ...
  ...
endmodule
```

Dùng chú thích

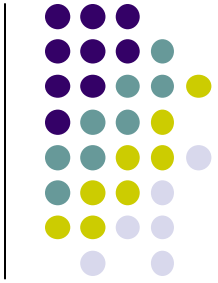


Nguyên tắc: sử dụng chú thích một cách thích hợp để giải thích các quy trình, chức năng, và khai báo các kiểu và kiểu phụ

Hướng dẫn:

- Sử dụng chú thích để giải thích các cổng, tín hiệu, và các biến, hoặc các nhóm tín hiệu, biến.
- Chú thích phải được tóm tắt, súc tích, và giải thích rõ ràng.
- Chú thích phải được đặt có logic, gần mã mà nó mô tả.

Viết các lệnh thành các dòng riêng

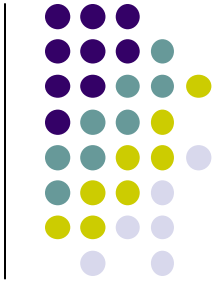


Quy tắc:

Sử dụng một dòng cách riêng biệt cho mỗi trạng thái.

- Cách sắp xếp này dễ đọc và dễ cải tiến.

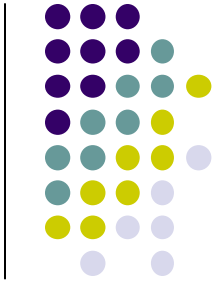
Chiều dòng mỗi dòng



Quy tắc:

Giữ cho chiều dài mỗi dòng khoảng 720 ký tự hoặc ít hơn.

Sự thụt dòng



Nguyên tắc:

• Thụt dòng giúp cho dễ đọc các đoạn mã hoặc các đoạn mã lồng vào nhau.

Hướng dẫn:

• Sử dụng 2 dấu cách:

Thụt dòng lớn để hạn chế chiều dài của dòng khi các mã có lồng vào nhau.

• Tránh dùng các tabs:

Sự khác biệt giữa người biên tập và người sử dụng làm cho vị trí của tab không thể đoán được.

Không được sử dụng các từ khóa

Quy tắc:

Không sử dụng các từ khóa của HDL (Verilog và VHDL) cho tên hoặc các thành phần khác trong RTL.

Thiết kế vĩ mô phải được dịch từ Verilog sang VHDL

Sắp xếp các cổng

Quy tắc:

• Khai báo các cổng theo trình tự logic, và giữ trình tự này trong suốt quá trình thiết kế.

Hướng dẫn:

• Khai báo cổng theo từng dòng và kèm theo chú giải.

• Khai báo cổng theo cách sau:

Sơ đồ cổng và sơ đồ chung

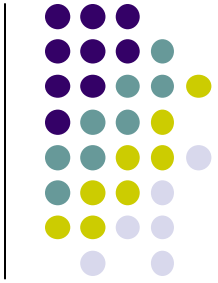
Hướng dẫn:

Luôn luôn sử dụng rõ ràng giữa sơ đồ cổng và sơ đồ chung, bằng cách sử dụng tên liên đới chứ ko phải tên vị trí.

Để lại 1 hàng trống giữa các lối vào, ra để dễ đọc.

```
//------------------------------------------------
 gcd Ugcd (
        .clk   (clk),
        .reset(reset),
        .A     (A),
        .B     (B),
        .valid  (valid),
        .C     (C),
        .done  (done)
 );
```

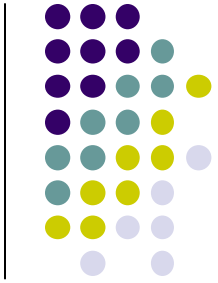
Sử dụng các chức năng



- Hướng dẫn:
  - Sử dụng các chức năng khi có thể thay vì phải lặp lại cùng 1 đoạn mã.

```
//-----------------------------------------------------
 function [`WIDTH-1:0] conv_addr;
    input  [`WIDTH-1:0] input_addr;
    input               offset;
    begin
        ...
        ...
    end
 endfunction
```

Mã hóa linh động



• Viết mã là 1 công nghệ riêng biệt. tương thích với các công cụ mô phỏng khác nhau, và dễ dàng dịch từ Verilog sang VHDL (hoặc từ VHDL sang Verilog).

• Không sử dụng các giá trị khó mã hóa.

• Thêm tập tin.

• Tránh nhúng các tập lệnh.

• Sử dụng thư viện độc lập.

Không dùng các giá trị khó mã hóa

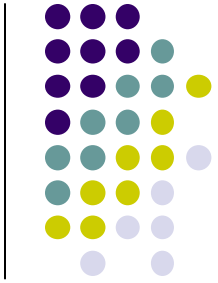
Không sử dụng các giá trị khó mã hóa.

Sử dụng hằng số thay vì các giá trị khó mã hóa.

```
//-----------------------------
 wire [7:0] my_in_bus;
 reg  [15:0] my_reg;
```

```
`define MY_BUS_SIZE 8
`define MY_REG_SIZE 16
parameter MY_BUS_SIZR = `MY_BUS_SIZE
parameter MY_REG_SIZE = `MY_REG_SIZE
//-----------------------------------------
  wire [MY_BUS_SIZE-1:0] my_in_bus;
  reg  [MY_REG_SIZE-1:0] my_reg;
```

Thêm tập tin

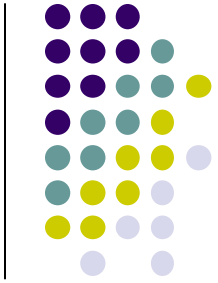


Hướng dẫn:

Sử dụng để định nghĩa trạng thái cho thiết kế bằng cách viết riêng biệt tập tin và tên tập tin.

Ex: DesignName_params.v.

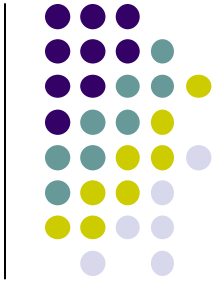
Tránh nhúng các tập lệnh



Hướng dẫn:

Tránh sử dụng các tập lệnh trong mã nguồn để mã linh động hơn.

Sử dụng các thư viện riêng.



Sử dụng các DesignWare Foundatioin để nâng cao tính độc lập trong công nghệ.

- Tránh chèn các cổng vào trong thiết kế:

Thiết kế mức cổng khó đọc, nên khó nâng cấp và tái sử dụng.

- Công nghệ để làm mức cổng cồng kềnh.

- Sử dung thư viện GTECH, nếu cổng phải sử dụng:

- GTECH chứa các cổng logic và các thành phần . Ex: AND, OR, NOT, Flipflop

- Thư viện DesignWare Foundation bao gồm các: Phép cộng, so sánh, tăng ,giảm. Sin, cos, chia, arithmetic và barrel shifters.

- Thư Viện DesignWare cải thiện timing, thiết kế nội, và kiến trúc.

- Thư viện DesignWare giúp thiết kế với hiệu suất cao mà còn gọn nhẹ.

Câu Hỏi & Trả Lời

icdesign1@semiconvn.com
icdesign2@semiconvn.com

Password: yovi2008