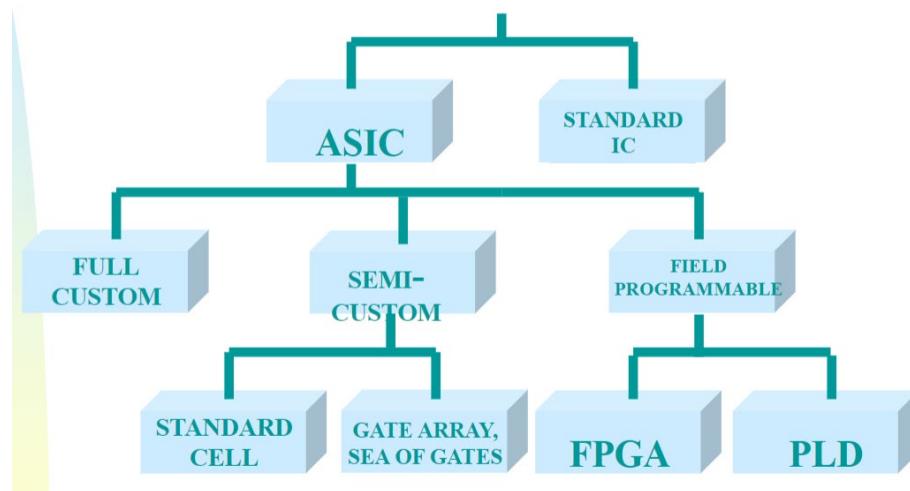


## Integrated circuit design

### Lecture 1 : Design of complex digital systems

# IC Implementation Technologies



- Sử dụng FPGA/full custom against microprocessor/dsp hardware is beneficial in speed. Dùng micro/dsp thì ta có thể viết code để nó làm nhiều mục đích khác nhau, nhưng vì có code nên tốc độ sẽ chậm hơn sử dụng FPGA/full custom.
- Nếu không quá khắt khe về speed thì sử dụng micro/dsp sẽ tiện hơn.

#### Design requirement: system representation (views)

- Behavioural view:
    - o Describe functionalities and i/o behaviour
    - o Treat the system as black box, no need to know what is inside
  - Structural view:
    - o Describe the internal implementation (components, connections..)
    - o Viewed as block diagram
  - Physical view :
    - o Add more details on structural view: component size, component locations...
- Focus on behavioural and structural view

#### VHDL for synthesis

- The level of design that is most suitable for synthesis is register transfer level. Your code must target that level

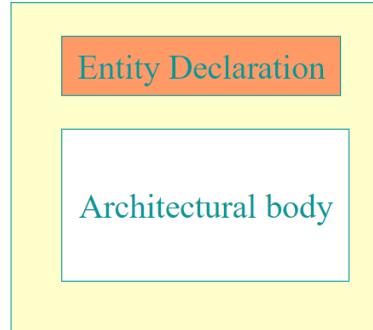
## Lecture 2 : VHDL

#### VHDL design objects:

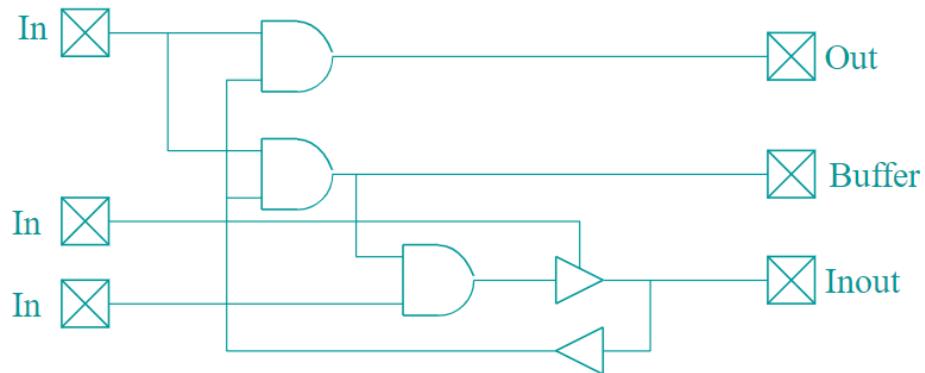
##### 1. Entity:

- Bao gồm 2 phần là entity declaration và architectural bodies (1 hoặc nhiều cái)

# Design Entity



- Entity declaration dùng để declare các ports, direction of ports, and its type of data
- Directions of port:
  - o In: input, **có thể assign giá trị của input**
  - o Out:output, **không thể assign giá trị của output** => dùng signal
  - o Inout (**chỉ dùng để design data bus**): bi-directional port. Nó có thể xuất hiện dc ở 2 phía trong phép gán (vì vừa là input vừa là output). Chỉ dùng cho data bus vì dùng nó sẽ nhân đôi chi phí, vd 1 signal 8bit, dùng inout sẽ tạo ra 2 cái buffer = 16 bit
  - o Buffer (**not widely used**): dùng cho internal feedback



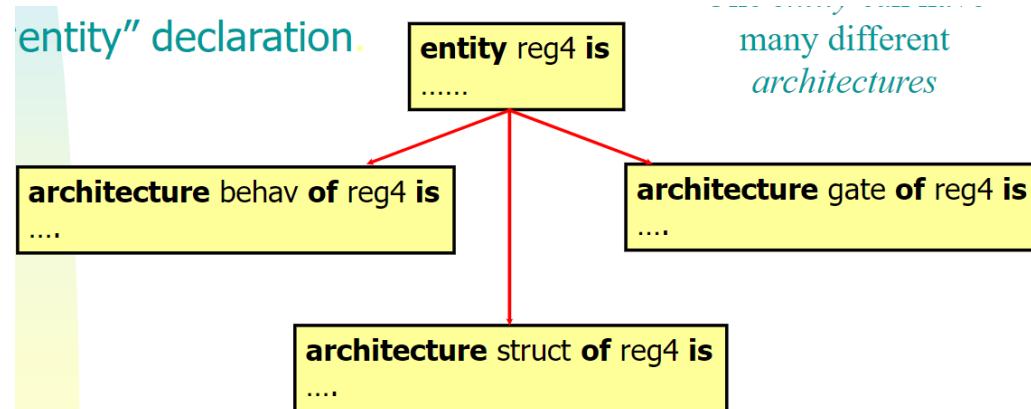
```
entity reg4 is
    port (d0,d1,d2,en,clk: in std_logic;
          q0,q1,q2: out std_logic);
end entity reg4;
architecture behave of reg4 is
begin
    [concurrent statements]
end architecture behave;
```

## 2. Concurrent statements:

- **Những statement này được trigger chỉ khi 1 trong những input của nó bị thay đổi**  
VD:  $x \leq c$  and  $(a \text{ or } b)$ ;  
Thì chỉ khi nào a,b hoặc c thay đổi giá trị thì statement trên mới được trigger

## 3. Architecture:

- Có thể có **nhiều architecture** cho 1 entity



- Giữ **begin** và **end** là **concurrent statements**. Nếu bỏ sequential statements (vd **nếu if**) vào thì sẽ bị syntax **error**
- **Sequential** ko thể được dùng một mình ở trong architecture, nó chỉ đc **dùng bên trong process**. Và nó chỉ được **update giá trị ở cuối process**

#### 4. Process:

- **Process** itself đc coi như là 1 **concurrent statements**, nhưng những lệnh **bên trong** process là **sequential** statements
- Phải bỏ hết những **input** vào trong **sensitivity list** của process, vì khi 1 thành phần trong sensitivity list bị đổi giá trị, process đc triggered.
- **Không** đc **bỏ dù** vào sensitivity list vì nó sẽ kích hoạt process mà ko làm đc gì
- 2 processes có thể communicate với nhau bằng cách để **output** của 1 process làm **sensitivity list** của **process còn lại**
- Những cái sequential statements trong process **chỉ đc update ở cuối process**.

#### 5. Package:

- Một tập hợp những định nghĩa có sẵn để dễ code hơn (vd data type)
- Package bao gồm header file và body file

#### 6. Configuration:

- Dùng để chọn 1 architecture body của 1 cái entity nào đó

#### 7. Library:

- Bao gồm entities, architectures, package header, configuration

#### Data types:

- Integer: 0,1,-4, ...
  - o For non-decimal: base#number#
 

VD: 2#001110# (mình muốn dùng 001110 là số binary), 8#76771# (mình muốn dùng 76771 là số oct)
- Real: 0.01, 0.3...
- Character: 'z', 'A',...
- String: "clockA"...
- Boolean
- Natural : từ 0 -> n
- Positive: từ 1 -> n
  - ⇒ **Note:** nên dùng natural và positive khi phù hợp, đừng lạm dụng integer.  
VD: for counter, dùng natural chứ đừng dùng integer.

#### Precision and range:

- Nên tập **xác định range** của input chứ đừng dùng maximum, vì như vậy mình sẽ **tiết kiệm được bộ nhớ**  
e.g. variable x : integer range 0 to 255;
- Tại vì mình sẽ ko muốn dùng 1 cái variable với 32-bit trong khi chỉ cần 8-bit là đủ

#### Enumerated data type:

- allow us to define any data type
- **syntax:**  
Type four\_value is ('0', '1', 'Z', 'X')  
Type macro\_op is (load, store, add, sub, move)
- **Default value** của kiểu giá trị mới đó sẽ là giá trị **thành phần đầu tiên**
- **Case-sensitive:** mặc dù VHDL là case-insensitive language nhưng kiểu enumerated là case-sensitive. Phải dùng đúng như những gì khai báo

#### STD\_logic:

Là một kiểu dữ liệu dành cho unsigned.Bao gồm **9 giá trị**:

- 'U': not initialized
  - o Khi một biến kiểu std\_logic được gọi thì giá trị **default** của nó là 'U'
  - o **Useful để phát hiện error** vì sau khi code chạy xong, nếu 1 biến nào đó **vẫn** có giá trị là 'U' thì ta biết biến đó ko dc update trong quá trình chạy code
- 'X': Unknown, conflict
  - o X xuất hiện khi
  - o => khi xuất hiện X, chắc chắn là lỗi
  - o Chỉ dùng cho mục đích phát hiện lỗi, vì vậy ko được dùng X để gán cho biến
- '0': low (0)
- '1': high (1)
- 'Z': high impedance
  - o Xuất hiện trong trường hợp **tri-state** buffer. Nếu en='1' thì output được gán giá trị của input, nhưng nếu **en='0'** thì nó giống như **open circuit**, high impedance => giá trị của output là 'Z'
  - o Ta **có thể gán** 'Z' cho output, nhưng **ko so sánh được** (e.g. if q ='Z' then... -> can't)
- 'W': weak unknown
- 'L': weak low.
  - o Models a pull-down: thường cho thêm cái resistor nối đất vào để kéo nó xuống '0'
- 'H': weak high
  - o Models a pull-up: thường cho thêm cái resistor nối nguồn vào để kéo lên '1'
- '-': don't care
  - o **Có thể gán** don't care cho output, nhưng **ko so sánh được**

#### Signals:

- Used for interconnection of wires between components
- Connect concurrent statements to form models
- Syntax:  
**signal A: std\_logic\_vector(3 downto 0) := "1010";**
- **Initial values** bị **ignored** bởi synthesis tool
- **Delay value** cũng bị **ignored** bởi synthesis tool
- **Signal ở bên trong process ko được update ngay lập tức, mà dc update sau khi end process**

### Variables:

- for local storage of data, **bắt buộc** phải dùng **bên trong process**, ra **bên ngoài** sẽ **ko access** được variable.
- **variable thay đổi ngay lập tức chứ ko như signal**
- syntax:  
**Variable v\_name: type := initial\_value**

### Constant:

- syntax:  
**constant name: type := initial\_value**

### Array:

- syntax:  
**TYPE bus8 IS ARRAY (7 DOWNTO 0) OF std\_logic;**  
**SIGNAL X : bus8**
- có thể dùng array để tạo ra mảng 2 chiều: thay std\_logic = std\_logic\_vector
- **concatenation:**
  - o dùng dấu **&**
  - o **2 thành phần** phải có **cùng data type** mới concatenate được
    - e.g.  
signal z : std\_logic\_vector(2 downto 0);  
signal a, b, c: std\_logic;  
z <= a & b & c suy ra z(2) = a, z(1) = b...
- **Slice of an array:**
  - o Is a sub-array
  - o Chiều của slice phải **cùng chiều** với **parent array** (to/downto)
- **Aggregates:**
  - o Chỉ định trực tiếp giá trị cho từng vị trí trong array
  - o e.g. Signal z : std\_logic\_vector (3 downto 0)  
z(3) <= a\_bit; hoặc  
z <= (3 => a\_bit, 1 => c\_bit, others => b\_bit)

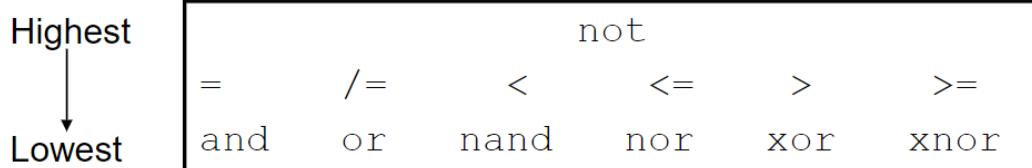
### VHDL operators:

- **logical** : and, nand, or, nor, xor, xnor, not
- **Shift operator:**
  - o Sll n: shift left logical
    - e.g. z <= A\_bit sll 2;
  - o Srl n: shift right logical
  - o Sla n: shift left arithmetic
  - o Sra n: shift right arithmetic
  - o Rol n: rotate left logical
  - o Ror n: rotate right logical
- **Type conversion (cast):**

data type of a	to data type	conversion function / type casting
unsigned, signed	std_logic_vector	std_logic_vector(a)
unsigned, std_logic_vector	unsigned	unsigned(a)
unsigned, signed	std_logic_vector	std_logic_vector(a)
unsigned, signed	integer	to_integer(a)
natural	unsigned	to_unsigned(a, size)
integer	signed	to_signed(a, size)

- **So sánh, relational:**

- o = : equal to
- o /= : not equal to
- o <,>,<=,>=



## LECTURE 3: BEHAVIOURAL MODELING, SYNTHESIS VIEW

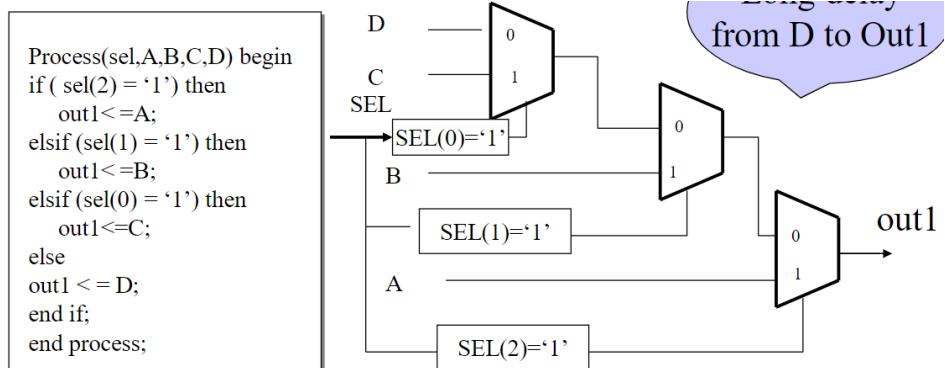
### Process:

- Nếu là process với sensitivity list thì ko đc có wait, và ngược lại
- Process with sensitivity list
  - o Được kích hoạt khi 1 thành phần trong list thay đổi
- Process with wait statement:
  - o Nó sẽ chạy cho đến khi gặp wait statement, và sau đó dừng lại
    - Wait on a,b -> chừng nào a,b thay đổi thì tiếp tục
    - Wait until (condition) -> chừng nào thỏa điều kiện thì tiếp tục
      - **Cẩn thận:** tất cả những câu lệnh ở sau dòng wait này đều sẽ được thực hiện khi điều kiện thỏa mãn. Nghĩa là nó có thể tạo ra register ko mong muốn
    - Wait for time-expression -> chừng nào đủ time thì tiếp tục
      - Chỉ dùng cho simulation, bị bỏ qua trong synthesis
    - Wait -> wait vĩnh viễn

### Sequential statements: if statement

- **Latches:**
  - o Là khi thỏa điều kiện thì giá trị output đc cập nhật, còn ko thì giá trị output bị giữ nguyên
  - o Khi dùng if cần thận hãy đảm bảo rằng **mọi trường hợp đều đã được gán**, nếu ko sẽ tạo ra latch
  - o **Để tránh** được latch thì nên có **default value** trước khi dùng if
- **If elsif with priorities:**

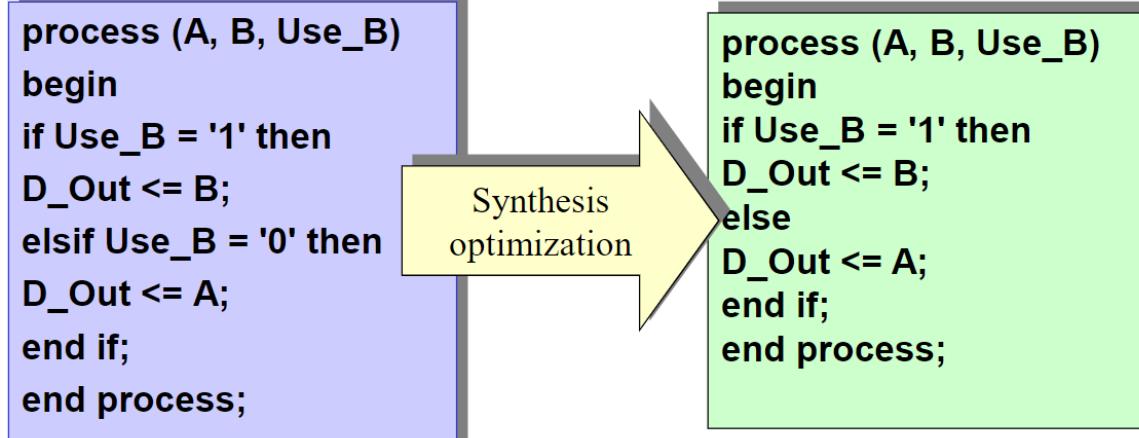
- Khi dùng if elsif thì cái có **highest priority** sẽ là cái **multiplexer cuối cùng**



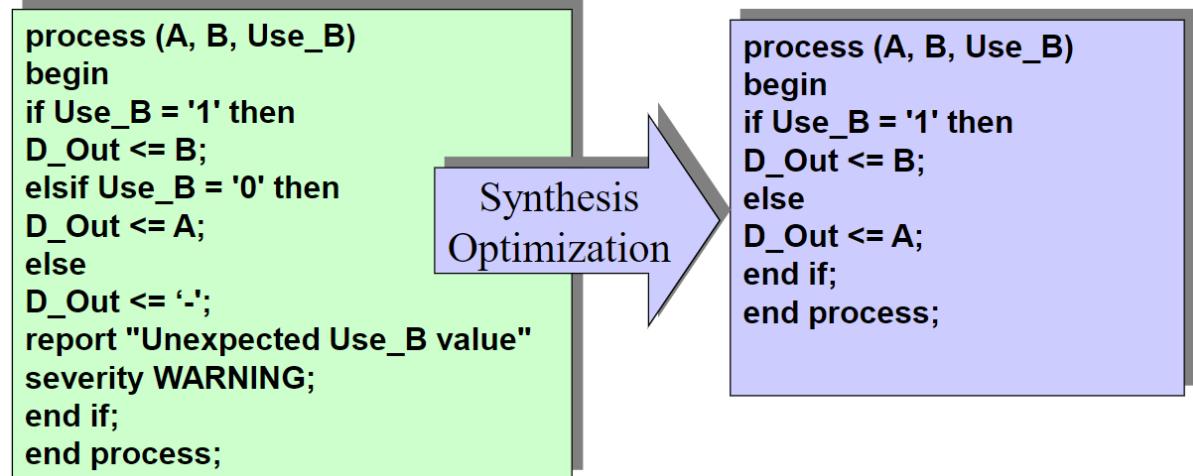
Ta thấy if sel(2) = 1 là if đầu tiên, thì nó sẽ là mux cuối cùng (gần output nhất)

#### Code optimisation:

- Đối với **synthesis tool**, nó **chỉ quan tâm** đến **bit value**, nghĩa là nếu biến đó có kiểu là std\_logic thì nó chỉ coi biến đó có 2 giá trị là '0' và '1'.



- Bên trái **mặc dù thiếu else** nhưng vẫn **ko tạo ra latch** vì synthesis tool chỉ quan tâm đến bit value, nghĩa là chỉ quan tâm đến 0 và 1, nên nó **optimise** đoạn code đó, bỏ đi elsif thay bằng else



#### Case statements:

- Syntax:

```

case expression is
    when value_1 =>
        sequential statements
    when value_2 =>
        sequential statements
    when value_x to value_y =>
        sequential statements
    when value_x | value_y =>
        sequential statements
    when others =>
        sequential statements;
end case;

```

- **Để tránh latches:**
  - o Phải **luôn dùng others** để đảm bảo **ko tạo ra latches**
  - o Khi có 1 **signal** xuất hiện, phải **gán giá trị** cho nó **trong tất cả** các **branches** của case statement

```

process (Data_In)
begin
case Data_In is -- Signal Data_In was declared as follows:
-- signal Data_In is INTEGER range 0 to 15;
when 0 => Out_1 <= '1';-- What about Out_2, Out_3, Out_4 ?
when 1 | 3 => Out_2 <= '1';-- What about Out_1, Out_3, Out_4 ?
when 4 to 7 | 2 => Out_3 <= '1';-- What about Out_1, Out_2, Out_4 ?
when others => Out_4 <= '1';-- What about Out_1, Out_2, Out_3 ?
end case;
end process;

```

Để khắc phục latch ở trường hợp trên thì thêm default value vào trước case

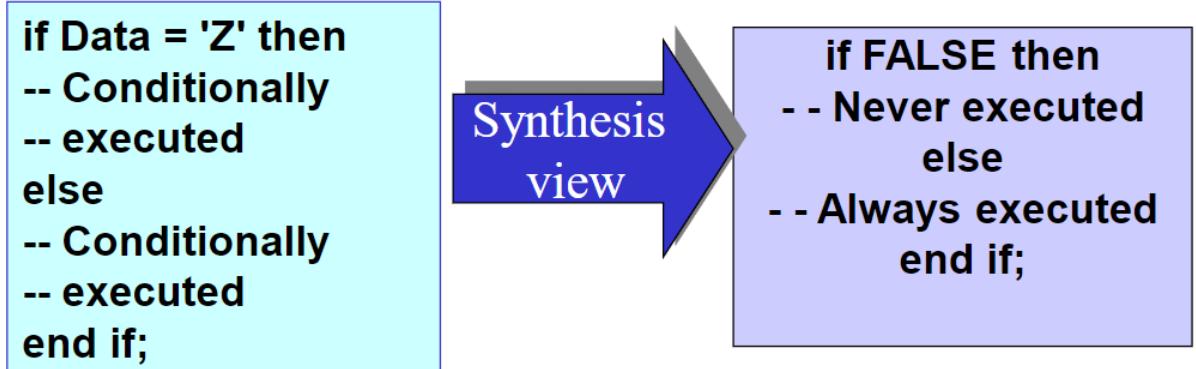
```

process (Data_In)
begin
Out_1 <= '0';
Out_2 <= '0';
Out_3 <= '0';
Out_4 <= '0';
case Data_In is -- Signal Data_In was declared as
follows:
-- signal Data_In is INTEGER range 0 to 15;
when 0 => Out_1 <= '1';
when 1 | 3 => Out_2 <= '1';
when 4 to 7 | 2 => Out_3 <= '1';
when others => Out_4 <= '1';
end case;
end process;

```

### Z value in synthesis:

- Không được gán Z cho signal -> error
- Khi so sánh 1 signal với 'Z', mặc dù synthesis cho phép điều này, nhưng nó sẽ coi như là always false



### Don't care value in synthesis:

- Comparison with don't care always return false
- Thay vì dùng don't care để compare hãy dùng aggregate:

**Wrong:**

```
case req is
when "1- -" => Y<= "10";
when "01 -" => Y<= "10";
when "100" => Y<= "10";
When others => Y<= "00";
end case;
```

**Correct:**

```
y <= "10" when req(3)='1' else
      "01" when req(3 downto 2)="01" else
      "00" when req(3 downto 1)="001" else
      "00";
```

### Combinational modelling rules:

- Tất cả input phải được ghi vào sensitivity list của process
- Không được thêm gì ngoài input vào sensitivity list
- Code ở trong signal ko được sensitive to rising/falling edge  
e.g. wait for sel='1' -> sẽ tạo ra rising edge, tạo ra register ko mong muốn
- Cẩn thận phòng tránh hết latches

### Clock:

- Wait until clk = '1' (rising edge) = wait until (clk'event and clk='1')
 Wait until clk = '0' (falling edge) = wait until (clk'event and clk='0')
 Wait until (rising\_edge(clk))
 Wait until (falling\_edge(clk))

- ⇒ **Ghi nhớ:** sau wait until thì mọi assignments đều là registers
- Vì vậy **tốt hơn** hết là **dùng sensitivity list và if statement** để control được cái nào là register cái nào không
  - **Asynchronous:** reset has **higher priority** than clock
 

```
async_rst : process (clk, rst_n) -- use exactly this sensitivity list
begin
  if rst_n = '0' then
    -- Assign bootup values for DFFs in this branch. Use constant values only!
    -- Do not read signals or input ports! Comb. logic in reset signal is very bad.
  elsif clk'event and clk = '1' then
    -- Assign the values of normal operation in this branch.
    -- No other conditions to elsif.
    -- Do not use a logical condition with clk'event and clk = '1' ( ie.
    (clk'event and clk = '1' and En = '1')
    -- Use nested if-statements instead.
  end if;
  -- No else of elsif branches here
  -- No signal assignments here
end process async_rst;
```

**Không** được thêm **else** vào statement

Những signal được ghi **bên trong elsif và end if là registers**
  - **Synchronous:** chỉ có **đuynhất clk** trong **sensitivity list**

```
sync_rst : process (clk) -- use exactly this sensitivity list
begin
  if clk'event and clk = '1' then
    if sync_rst_n = '0' then
      -- Assign bootup values for DFFs in this branch.
      -- No other conditions here. Assign constant reset values only.
    else
      -- Assign the values of normal operation in this branch.
    end if;
    -- No elsif branches here.
  end if;
  -- No else of elsif branches here
  -- No signal assignments here
end process sync_rst;
```

**Không** được thêm dư **elsif**

### Non-synthesisable VHDL:

- **Dual-edge triggered register/counter:** là những flip flop mà có thể đc triggered trên cả rising edge và falling edge
  - o Ko thể implement dual-edge trong 1 process được

### Loop statement:

- Những lệnh ở bên trong loop statement đều là những lệnh sequential
- **For loop** thì **synthesis** được, còn while loop với no scheme loop thì ko synthesis được

```
for N in 3 downto 1 loop  
    shift_reg(N) <= shift_reg(N-1);  
end loop;
```

- Có những attribute: **'high', 'low', 'range'**  
VD: signal x: std\_logic\_vector(7 downto 0)  
=> x'high = 7, x'low = 0, x'range = 7 downto 0  
=> for i in x'low to x'high loop = for i in 0 to 7 loop = for i in x'range loop
- **Avoid unnecessary repeated computation:**

```
for K in 1 to 7 loop  
    if K > (A - 1) then  
        S(K) <= '1';  
    else  
        S(K) <= '0';  
    end if;  
end loop;
```

- o Nếu làm như trên thì ta sẽ có 7 subtracter và 7 comparator
- o Vì vậy nên tránh tính toán ở trong loop, mà hãy làm trước khi vào loop

```
Temp := A - 1;  
for K in 1 to 7 loop  
    if K > Temp then  
        S(K) <= '1';  
    else  
        S(K) <= '0';  
    end if;  
end loop;
```

- o Làm như vậy thì chỉ còn 1 subtracter và 7 comparator

### Other sequential circuit:

- **Null:** a sequential statement that **doesn't cause any action**, doesn't make any assignment
- **Next:** chỉ được dùng bên trong loop, dùng để **skip current iteration** khi thỏa điều kiện. Will **generate a latch**. Dùng **default value** để **prevent it**.
- **Exit:** chỉ được dùng trong loop, dùng để thoát khỏi loop khi thỏa điều kiện

### Variable in clocked processes:

- Nếu như variable in clocked process mà được **read trước khi write** thì giá trị của nó sẽ được lưu vào **flip flop (register)**. Còn nếu write trước khi read thì không tạo ra flip flop (register) process (CLK)

```
variable Q : std_logic;
begin
  if CLK'event and CLK='1' then
    PULSE <= D and not(Q);
    Q := D;
    -- PULSE and Q act as registers
  end if;
end process;
```
- Trường hợp trên: Q là variable trong clocked process và được read trước khi write => Q:=D sẽ là 1 flip flop (register)

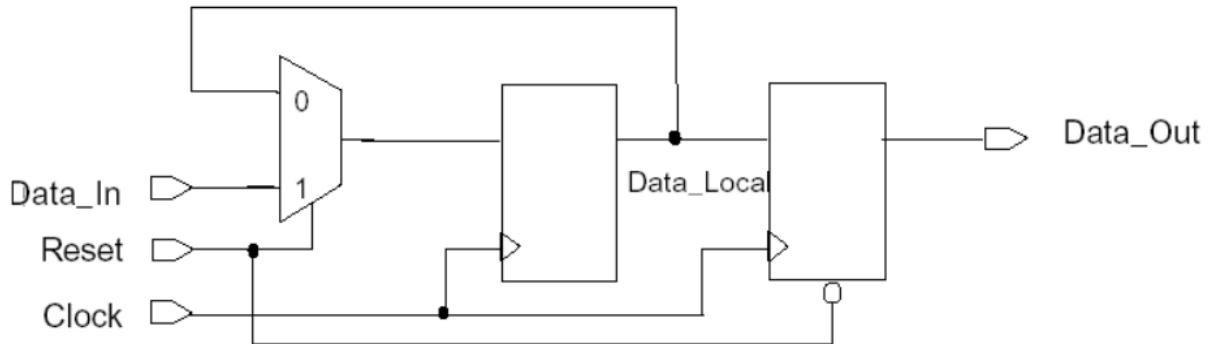
```
Process(clk)
  Variable a,b,c: std_logic;
begin
  if rising_edge(clk) then
    a := input;
    b := a;
    c := b;
    output <= c;
  end if;
end process;
```
- Còn trường hợp này a,b,c được write trước khi read => sẽ ko tạo ra flip flop

### Unexpected reset signal dependency:

```

process (Clock, Reset)
begin
if Reset = '0' then
Data_Out <= '0';
elsif Clock'EVENT and Clock = '1' then
Data_Local <= Data_In;
Data_Out <= Data_Local;
end if;
end process;

```

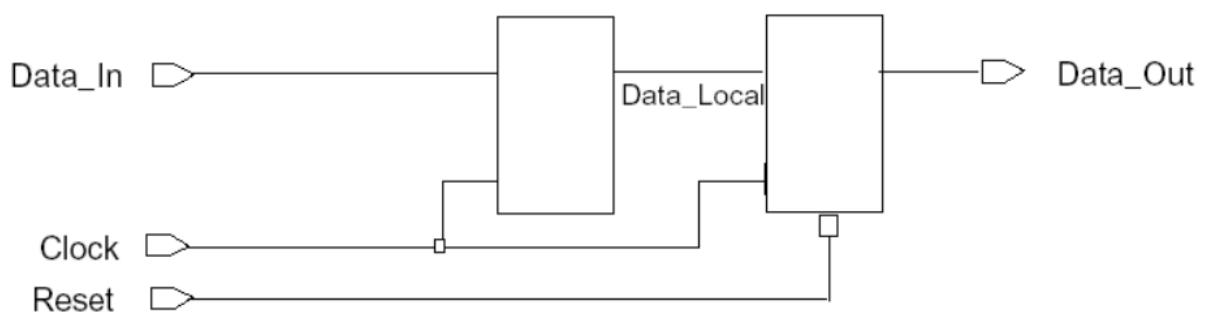


- Khi reset bằng 0 thì chỉ có Data\_out đc reset, còn Data\_Local thì vẫn giữ nguyên => kết quả sẽ bị sai
- Resolve:

```

process (Clock, Reset)
begin
if Reset = '0' then
Data_Out <= '0';
elsif Clock'EVENT and Clock = '1' then
Data_Out <= Data_Local;
end if;
end process;
process
begin
wait until Clock = '1';
Data_Local <= Data_In;
end process;

```

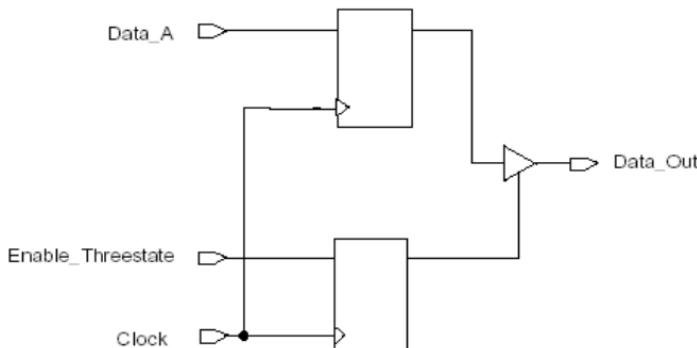


### Registering of synchronous tri-state enabler

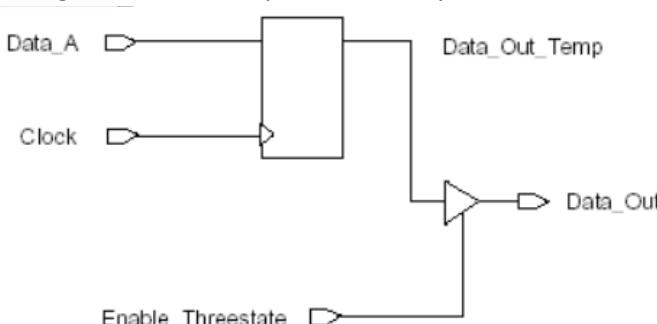
- Cần thận khi muốn dùng tri-state on rising edge of clock vì nó có thể làm cho 'en' của tri-state buffer bị latched (bị bỏ vô register)

```
process (Clock)
begin
if Clock'EVENT and Clock = '1' then
if Enable_Threestate = '0' then
Data_Out <= 'Z';
else
Data_Out <= Data_A;
end if;
end if;
end process;
```

- Khi viết như trên thì 'en' của tri-state phải chờ clock edge => 'en' bị bỏ vô register.



- Trong khi cái ta muốn phải là như này



- Vì vậy phải cho điều kiện của 'en' là highest priority

```
process (Clock, Enable_Threestate)
begin
if Enable_Threestate = '0' then
Data_Out <= 'Z';
elsif Clock'EVENT and Clock = '1' then
Data_Out <= Data_A;
end if;
end process;
```

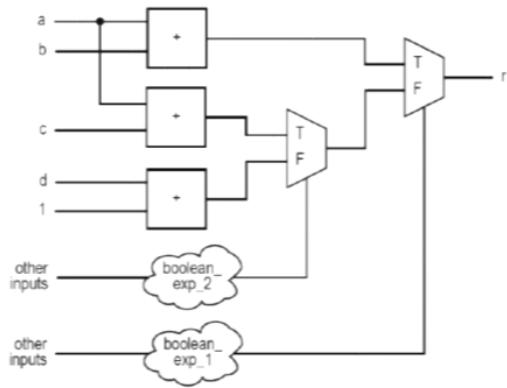
### Gated clock:

- Là cho clk and en hoặc clk or en => gated clock
- Avoid gated clock

### Resource sharing (important):

- Không bao giờ có thể share resource giữa các **processes**
- Trade-offs: time, nếu chọn share resource thì thời gian chạy sẽ tốn lâu hơn
- How to write the code that can share:
  - Tránh implement adder trong conditional statement:

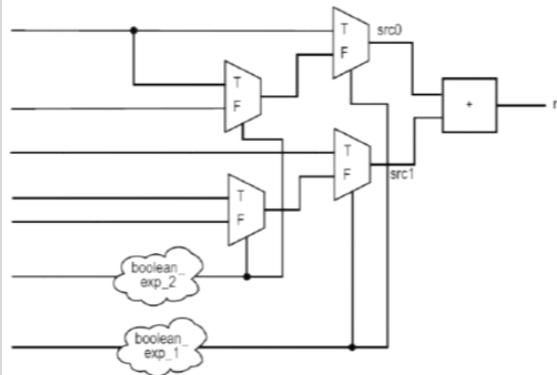
```
process(a,b,c,d,...)
begin
  if boolean_exp_1 then
    r <= a+b;
  elsif boolean_exp_2 then
    r <= a+c;
  else
    r <= d+1;
  end if;
end process;
```



2 adders, 1 incrementer, 2 mux

Mà chỉ implement adder ở bên ngoài conditional statement:

```
process(a,b,c,d,...)
begin
  if boolean_exp_1 then
    src0 <= a;
    src1 <= b;
  elsif boolean_exp_2 then
    src0 <= a;
    src1 <= c;
  else
    src0 <= d;
    src1 <= "00000001";
  end if;
end process;
r <= src0 + src1;
```



1 adder, 4 mux

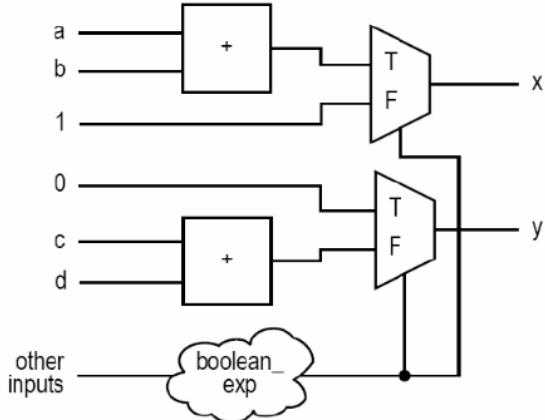
- Ví dụ 2:

```

process(a,b,c,d,...)
begin
  if boolean_exp then
    x <= a + b;
    y <= (others=>'0');
  else
    x <= (others=>'1');
    y <= c + d;
  end if;
end process;

```

2 adders, 2 mux

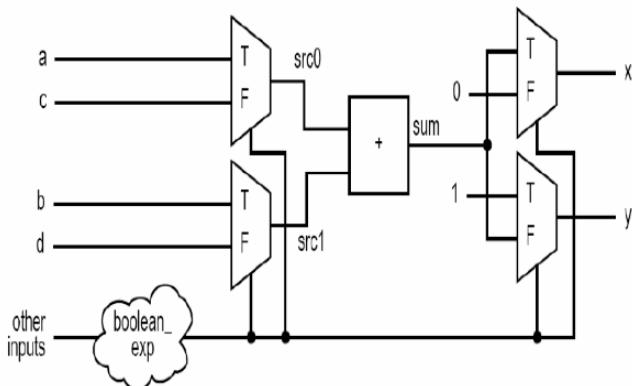


```

begin
  if boolean_exp then
    src0 <= a;
    src1 <= b;
    x <= sum;
    y <= (others=>'0');
  else
    src0 <= c;
    src1 <= d;
    x <= (others=>'1');
    y <= sum;
  end if;
end process;
sum <= src0 + src1;

```

1 adder, 4 mux



- **Common sub expression:**

run  $\leq r1 + r2;$

stop  $\leq r3 - (r1 + r2);$

- o R1+r2 là common sub-expression. 2 adders sẽ được tạo ra trong khi nó có kết quả giống hệt nhau

- o Dùng **temp** cho common sub-expression

**tmp := (r1 + r2); ...**

run  $\leq \text{tmp}; ...;$

- o stop  $\leq r3 - \text{tmp};$

- **Dùng case statement rất tốt cho sharing**

## Concurrent statements: data flow modelling

- Concurrent statements ko đc sử dụng bên trong process
- **Concurrent statements triggered when a signal in their sensitivity list changes**
- Sự khác nhau giữa concurrent và sequential:
  - o Sequential statements thì đc execute hết, và execute theo thứ tự
  - o Concurrent statements thì ko đc execute hết, chỉ những statement nào có sensitivity list bị đổi thì mới execute

### Conditional signal assignment:

- Là **concurrent version** của if statement
- Chỉ có **duy nhất 1 output**
- 2 cái dưới đây là tương đương nhau:

$Z \leq A$  when ( $X > 5$ ) else  
 $B$  when ( $X < 5$ ) else  
 $C;$

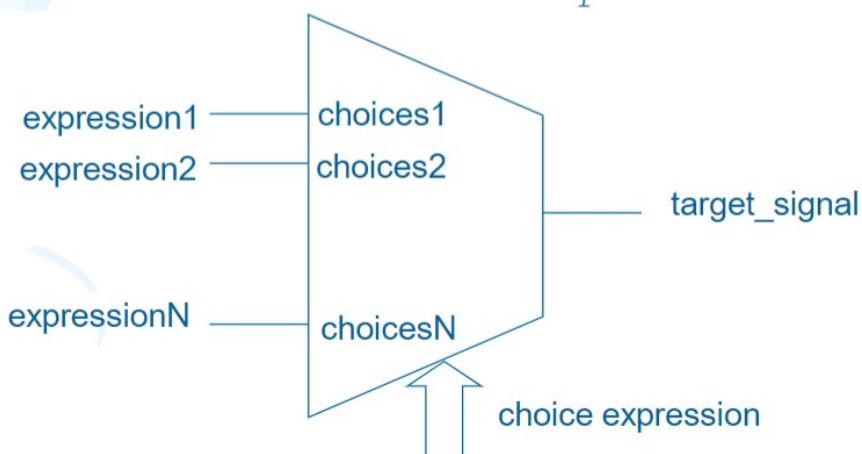
```
if ( X > 5) then  
    Z <= A;  
elsif ( X < 5) then  
    Z <= B;  
else  
    Z <= C;
```

- **Bắt buộc** phải có else => không bao giờ xảy ra latch

### Selected signal assignment statement:

- Dùng để implement a multiplexer

```
with choice_expression select  
    target_signal <= expression1 when choices1,  
                                expression2 when choices2,  
                                . . .  
                                expressionN when choicesN;
```



- đây là **concurrent version** của case statement
- chỉ có duy nhất 1 output

with X select

Z <= B when 0 to 4;  
C when 5;  
A when others;

```
case X is
when 0 to 4 =>
Z <= B ;
when 5 =>
Z <= C ;
when others =>
Z <= A ;
end case;
```

### Examples using data flow design style:

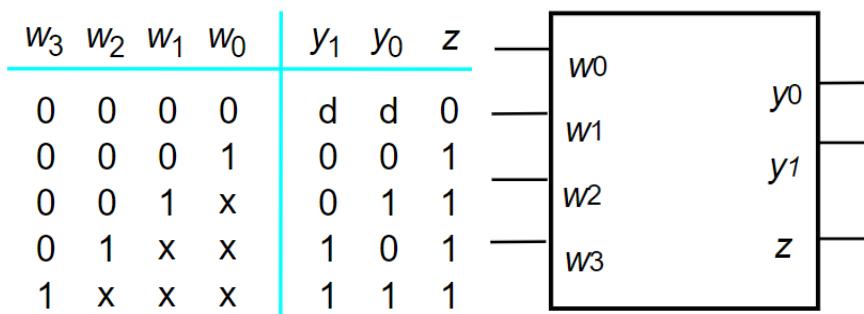
- 2 to 1 multiplexer:

F <= w0 when s='0' else w1;

- Tri-state buffer:

output <= input when (ena='0') else  
others => 'Z';

- Priority encoder:



Y <= "11" when w3='1' else  
"10" when w2='1' else  
"01" when w1='1' else  
"00" when w0='1' else  
"--"

Z <= '0' when w="0000" else '1'

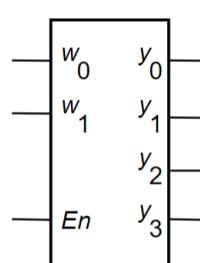
- 4 to 1 multiplexer:

WITH s SELECT

f <= w0 when "00",  
w1 when "01",  
w2 when "10"  
w3 when OTHERS

- 2 to 4 decoder:

$en$	$w_1$	$w_0$	$y_0$	$y_1$	$y_2$	$y_3$
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0



```

enw = en & w;
WITH enw SELECT
    y <=    "1000" when "100"
            "0100" when "101"
            "0010" when "110"
            "0001" when "111"
            "0000" when others

```

#### Arithmetic operators in VHDL:

- Để sử dụng dc arithmetic operations cho std\_logic\_vector thì cần phải include đầy đủ library cho nó:

**LIBRARY ieee;**

**USE ieee.std\_logic\_1164.all;**

**USE ieee.std\_logic\_unsigned.all;**

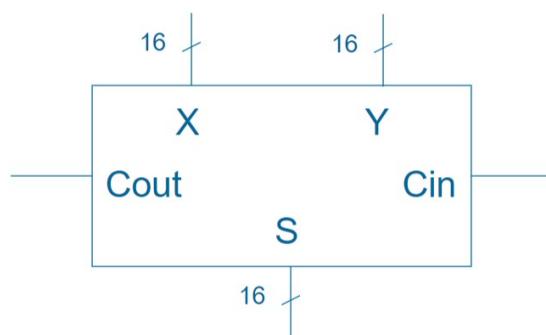
**or**

**USE ieee.std\_logic\_signed.all;**

- Sau khi include thư viện thì ta có thể dùng +, -, \*
- **Phép cộng:**

- o Khi gọi a+b thì nó sẽ tự động tạo ra 1 temp để lưu giá trị a+b. **Size** của **temp** sẽ là **max(size của a, size của b)**
- o Vì vậy ví dụ nếu cộng 2 vector **16 bits** a và b, thì temp phải có **ít nhất 17 bits** để có gì còn lưu dc carry.
- o Ta sẽ thêm vào a (hoặc b) 1 bit ở đầu, lúc đó size a = 17, size b = 16 => size temp = 17
- o Đối với **unsigned** thì luôn có thể thêm số **0** vào ở đầu, vì nó ko thay đổi giá trị. Còn đối với **signed** thì nếu đó là số **âm**, ta **thêm 1** vào đầu, nếu là số **dương** thì **thêm 0** vào đầu.

**Ví dụ 16-bit unsigned adder:**



```

ENTITY adder16 IS
PORT ( Cin          : IN      STD_LOGIC ;
        X, Y         : IN      STD_LOGIC_VECTOR(15 DOWNTO 0) ;
        S            : OUT     STD_LOGIC_VECTOR(15 DOWNTO 0) ;
        Cout         : OUT     STD_LOGIC );
END adder16 ;

```

Để lưu lại tổng 2 vector 16-bit thì cần 1 signal SUM 17 bits

```

ARCHITECTURE Behavior OF adder16 IS
    SIGNAL Sum : STD_LOGIC_VECTOR(16 DOWNTO 0) ;
BEGIN
    Sum <= ('0' & X) + Y + Cin ;
    S <= Sum(15 DOWNTO 0) ;
    Cout <= Sum(16) ;
END Behavior ;

```

Vì là **unsigned** nên **thêm số 0** vào ở đầu X (hoặc Y). Lúc này **temp** sẽ tự động có size là **17 bit**

Sau khi có dc SUM (17 bits) thì **kết quả** sẽ là **16 bits của SUM. Carry out** sẽ là **bit 17** của SUM

#### Ví dụ 16-bit signed adder:

- Cũng giống trên nhưng thay thư viện unsigned thành signed
- Trước khi gán sum thì check :

```

If x(16)='1' then Sum <= ('1' & X) + Y + Cin;
Else thì giống ví dụ trên

```

- **Phép so sánh (bằng, bé, lớn):**

- o Khác với phép cộng, phép so sánh có **code giống nhau** cho trường hợp **signed** và **unsigned**. Tại vì khi gọi library tương ứng, thì nó đã xử lý việc so sánh sao cho đúng type đó rồi

#### Ví dụ 4-bit number comparator

```

ENTITY compare IS
    PORT ( A, B           : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           AeqB, AgtB, AltB : OUT      STD_LOGIC ) ;
END compare ;

```

ARCHITECTURE Behavior OF compare IS  
BEGIN

```

        AeqB <= '1' WHEN A = B ELSE '0' ;
        AgtB <= '1' WHEN A > B ELSE '0' ;
        AltB <= '1' WHEN A < B ELSE '0' ;

```

END Behavior ;

Signed hay unsigned đều code như thế này

## LECTURE 4: FINITE STATE MACHINE CONTROLLER

### Programmable vs non-programmable controller :

- **Programmable:**
  - o Có program counter trả vào next instruction
  - o Instruction được lưu trong RAM hoặc ROM
  - o Vd: microprocessor
- **Non-programmable:**
  - o Chỉ thực hiện được 1 công dụng

### Finite state machine:

- Là dụng cụ để model the behaviour of a sequential system
- FSM có nhiều states, input sẽ kết hợp với current state để tạo ra next state

### State memory:

- FSM dùng **flip flop** để lưu trữ các states.
- Nếu FSM có  **$2^n$  states** thì nó cần dùng **n flip flops**.

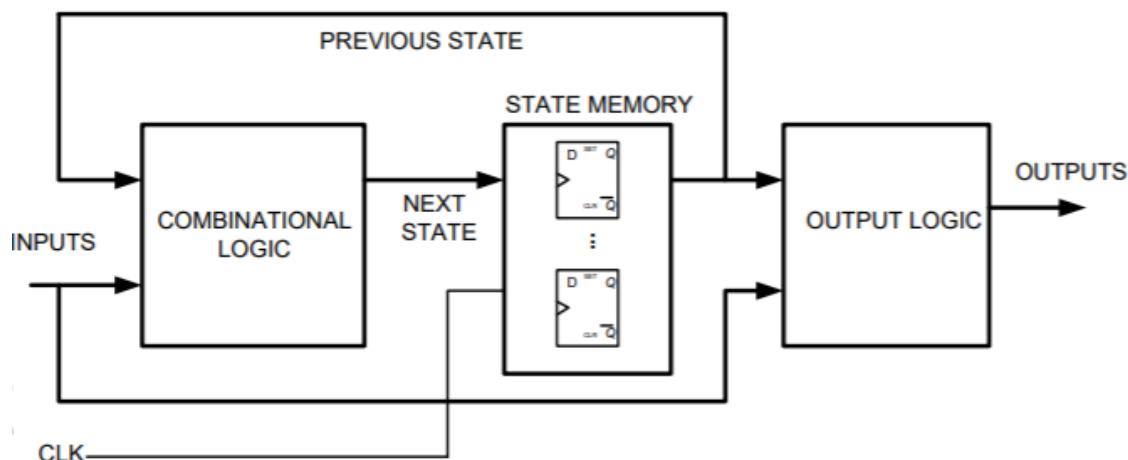
### Mealy machine:

- Output depends on current **state + input**.
- Cần **ít states** hơn moore, đồng nghĩa với cần **ít flip flop**.
- Output được cập nhật mỗi khi **input** bị **thay đổi**

### Moore machine:

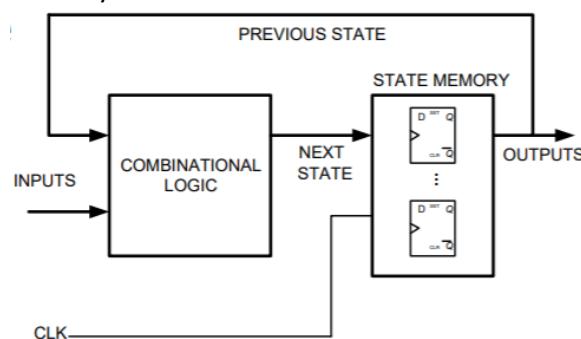
- Output depends on **only current state**.
- Cần **nhiều states** hơn mealy, đồng nghĩa cần **nhiều flip flop**
- Output chỉ dc cập nhật nếu **state** gần nhất bị **thay đổi**

### Coding solution:



#### - 1 process solution:

- o Sẽ ko có output logic, và chỉ duy nhất 1 process để xử lý combinational logic + state memory



#### - 2 processes solution:

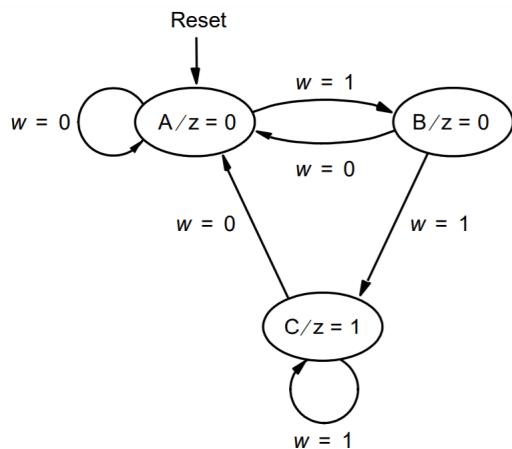
- o 1 process sẽ để xử lý combinational logic và state memory
- o 1 process sẽ để xử lý output logic

- **3 processes solution:**
  - o 1 process để xử lý combinational logic
  - o 1 process để xử lý state memory
  - o 1 process để xử lý output logic

**Synthesis tool:**

- Synthesis tool can understand FSM if:
  - o State transitions are described in a process sensitive to clk and asynchronous reset only
  - o Outputs are described as concurrent statements outside the process

**Example FSM code:**



**1. 1 process solution**

- **Current state thay đổi trong lúc clock edge luôn**
- **Chỉ có duy nhất 1 biến cho state đó là current state**

entity main is

```

port (
  clk, reset, w: in std_logic;
  z : out std_logic);
  
```

end main;

architecture Behavioral of main is

type states is (A,B,C);

signal crt\_state: states;

begin

process(clk,reset)

begin

if (reset = '1') then

crt\_state <= A;

```

elsif (clk'event and clk = '1') then
    case crt_state is
        when A =>
            if (w = '1') then
                crt_state <= B;
            else
                crt_state <= A;
            end if;
        when B =>
            IF w = '0' THEN
                crt_state <= A ;
            ELSE
                crt_state <= C ;
            END IF ;
        WHEN C =>
            IF w = '0' THEN
                crt_state <= A ;
            ELSE
                crt_state <= C ;
            END IF ;
        when others => y <= A;
    end case;
end if;
end process;
z <= '1' when crt_state = C else '0';
end Behavioral;

```

## 2. 2 processes solution

- Gồm có **current state** và **next state**
- Process 1 để xử lý next state: **next state chỉ thay đổi khi input hoặc current state thay đổi**
- Process 2 để xử lý **update current state, trên clock edge**

```
entity twoprocesses is
    port ( clk, reset, w: in std_logic;
           z : out std_logic);
end twoprocesses;
```

```
architecture Behavioral of twoprocesses is
```

```
type states is (A,B,C);
```

```
signal crt_st, next_st: states;
```

```
begin
```

```
process(w, crt_st)
```

```
begin
```

```
    case crt_st is
```

```
        when A =>
```

```
            if (w = '1') then
```

```
                next_st <= B;
```

```
            else
```

```
                next_st <= A;
```

```
            end if;
```

```
        when B =>
```

```
            if (w = '0') then
```

```
                next_st <= A;
```

```
            else
```

```
                next_st <= C;
```

```
            end if;
```

```
        when C =>
```

```
            if (w = '0') then
```

```
                next_st <= A;
```

```
            else
```

```
                next_st <= C;
```

```
            end if;
```

```
    end case;
```

```

end process;

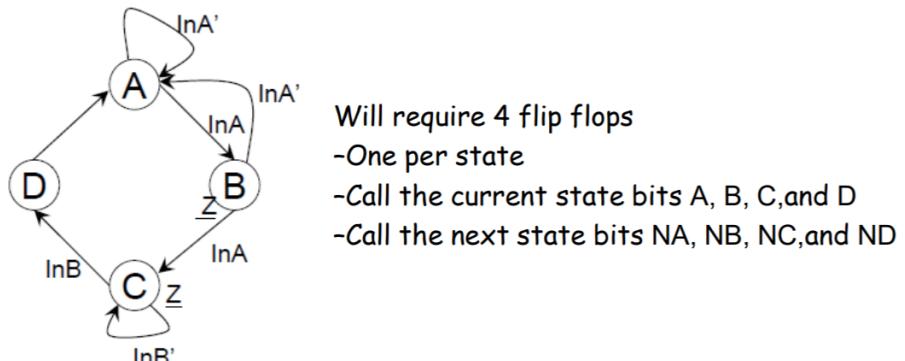
process (clk, reset)
begin
    if reset = '1' then
        crt_st <= A;
    elsif clk'event and clk = '1' then
        crt_st <= next_st;
    end if;
end process;

z <= '1' when crt_st = C else '0';
end Behavioral;

```

### State encoding – VHDL

- Encode VHDL ra circuit
- **Binary** encoding : synthesis by **default** sẽ đổi vị trí của states ra binary number
  - o Sử dụng **N** flip flop để lưu trữ  $2^N$  states  
e.g. type states is (A,B,C) thì A = 00, B = 01, C = 10  
Chỉ dùng 2 flip flop vì chỉ có 3 states
- **one-hot** encoding (1 FF/state method):
  - o sử dụng **N** flip flop cho **N** states  
e.g. type states is (A,B,C) thì A = 000, B = 010, C = 100
  - o states thường được biểu diễn bởi **logical equation**



$$\begin{aligned}
 NA &= A \cdot InA' + B \cdot InA' + D \\
 NB &= A \cdot InA \\
 NC &= B \cdot InA + C \cdot InB' \\
 ND &= C \cdot InB \\
 Z &= B + C
 \end{aligned}$$

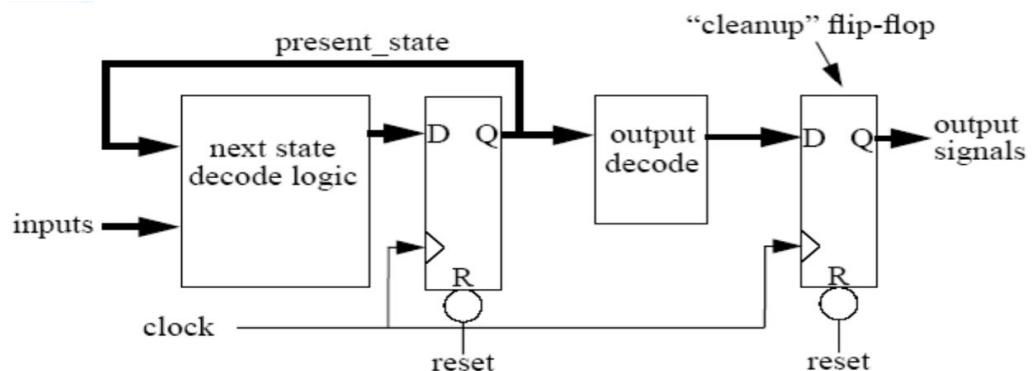
- **gray encoding:**
  - o best for **minimizing glitches** in output
  - o glitch được tạo ra vì khi output của flip flop propagate combinational circuit để tạo ra được output của circuit thì nó sẽ có thời gian delay khác nhau.
  - o E.g. q1 và q0, giả sử q1 delay ít hơn, nó sẽ trigger output trước, sau đó q0 mới đến, và lại trigger output => glitch

#### **RAM and ROM also use this model:**

- Input sẽ đi qua decoder để xác định state (đối với RAM/ROM thì gọi là địa chỉ). Tại địa chỉ này có dữ liệu đc lưu trong đó => có thể read hoặc write vào địa chỉ đó

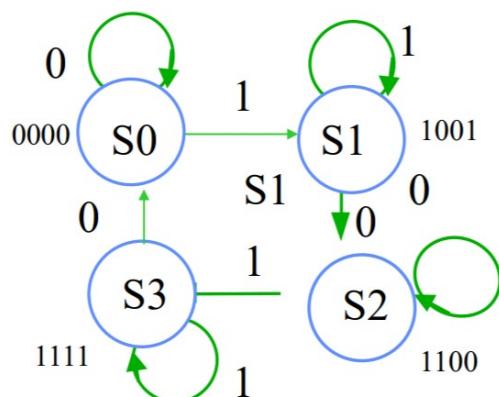
#### **Glitches:**

- Để tạo ra output clean, without gliches ta có 2 cách:
  - o Dùng gray encoding
  - o Dùng register output: đảm bảo rằng **mọi output** đều **come directly from a flip flop** output. Tại vì flip flop phải chờ clock edge nên ko thể tạo ra glitches đc



#### **Coding để không bị glitch (glitchless coding)**

Bỏ output vào trong case chứ ko bỏ ở bên ngoài nữa. VD:



- **Moore machine:** output chỉ thay đổi dựa vào state entity moore\_glitchless is
 

```

        port ( clk, reset, input: in std_logic;
               output: out std_logic_vector(3 downto 0));
      end moore_glitchless;
```

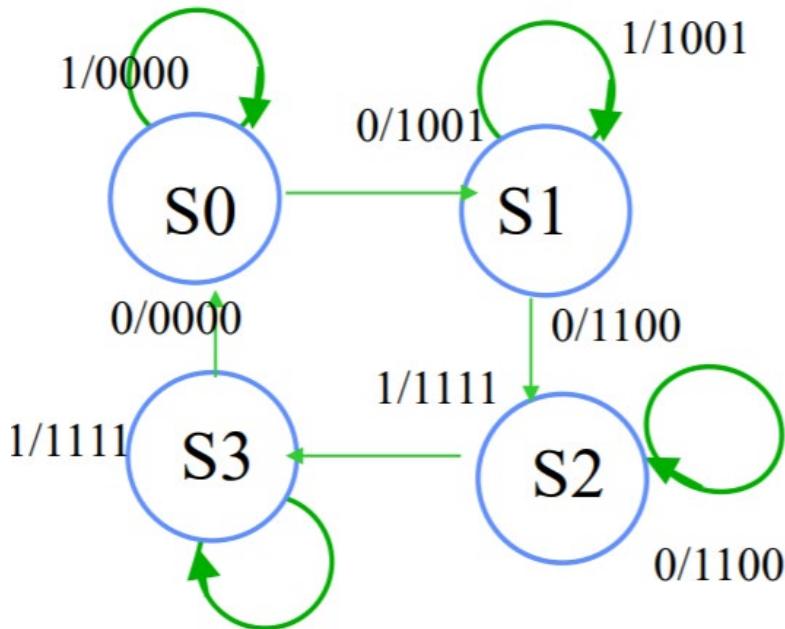
architecture Behavioral of moore\_glitchless is

```

type states is (s0,s1,s2,s3);
signal st: states;
begin
    process(clk,reset)
    begin
        if (reset = '1') then
            st <= s0;
            output <= "0000";
        elsif (clk'event and clk = '1') then
            case st is
                when s0 =>
                    if input = '0' then
                        st <= s0;
                    else
                        st <= s1;
                    end if;
                    output <= "0000";
                when s1 =>
                    if input = '0' then
                        st <= s2;
                    else
                        st <= s1;
                    end if;
                    output <= "1001";
                ....
            end case;
        end if;
    end process;
end Behavioral;

```

- **Mealy machine:** output bị thay đổi dựa vào cả state và input



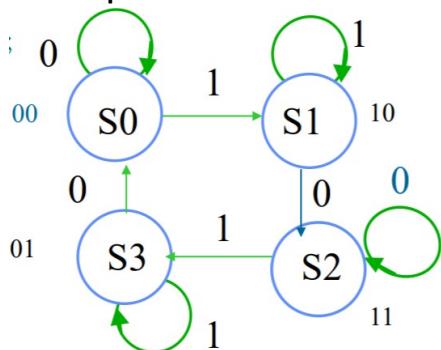
```

entity moore_glitchless is
    port ( clk, reset, input: in std_logic;
           output: out std_logic_vector(3 downto 0));
end moore_glitchless;

architecture Behavioral of moore_glitchless is
type states is (s0,s1,s2,s3);
signal st: states;
begin
begin
process(clk,reset)
begin
if (reset = '1') then
    st <= s0;
    output <= "0000";
elsif (clk'event and clk = '1') then
    case st is
        when s0 =>
            if input = '0' then
                st <= s1;
                output <= "1001";
            else
                st <= s0;
                output <= "0000";
            end if;
        when s1 =>
            if input = '0' then
                st <= s2;
                output <= "1100";
            else
                st <= s1;
                output <= "1001";
            end if;
        .....
    end case;
end if;
end process;
end Behavioral;

```

- **Gán output cho tên của state luôn:**



```

entity moore_glitchless is
    port ( clk, reset, input: in std_logic;
           output: out std_logic_vector(3 downto 0));
end moore_glitchless;

architecture Behavioral of moore_glitchless is
type states is array ( 1 downto 0) of std_logic;
Constant S0 : states := "00";
Constant S1 : states := "10";
Constant S2 : states := "11";
Constant S3 : states := "01";
signal st: states;
begin
    process(clk,reset)
    begin
        if (reset = '1') then
            st <= s0;
        elsif (clk'event and clk = '1') then
            case st is
                when s0 =>
                    if input = '0' then
                        st <= s0;
                    else
                        st <= s1;
                    end if;
                when s1 =>
                    if input = '0' then
                        st <= s2;
                    else
                        st <= s1;
                    end if;
                .....
            end case;
        end if;
    end process;
    output <= std_logic_vector(st);
end Behavioral;

```

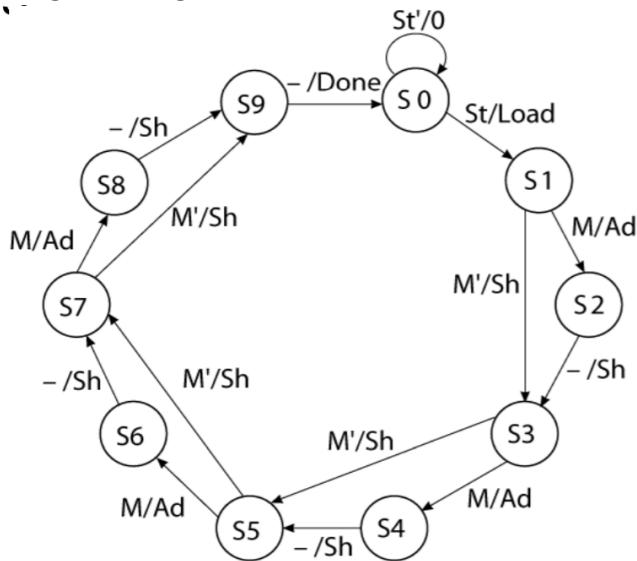
### FSM/ASM:

- State diagram: chỉ thích hợp với những mạch đơn giản, ít input/output
- Algorithmic state machine (ASM):
  - o Là 1 FSM nhưng thích hợp với những mạch phức tạp, với số lượng lớn input/output
  - o Có nhiều lợi ích hơn đối với việc sử dụng state diagram
  - o ASM giống như 1 cái flow diagram
- ASM blocks:
  - o One state box:
  - o One or more optional decision boxes

- One or more conditional output boxes
- ASM chart rules:
  - Transition của chart chỉ xảy ra trên **clock**
  - Transition của chart xảy ra ở **giữa các blocks**
  - Chỉ duy nhất **1 exit path** cho 1 input combination
  - Mỗi **closed loop** trong ASM phải chứa **1 state box**, bởi vì transition từ state này qua state khác xảy ra trên clock.

### VD: Multiplier design

- Dùng state diagram :



entity multiplier is

```

port ( clk, start: in std_logic;
      mer, mcand : in unsigned(3 downto 0);
      output : out unsigned(3 downto 0);
      done: out std_logic);
  
```

end multiplier;

architecture Behavioral of multiplier is

signal acc : unsigned(8 downto 0);

signal state : integer range 0 to 9;

alias M : std\_logic is acc(0);

begin

process (reset, clk)

begin

if (reset = '1') then

```

done <= '0';
output <= (others => '0');

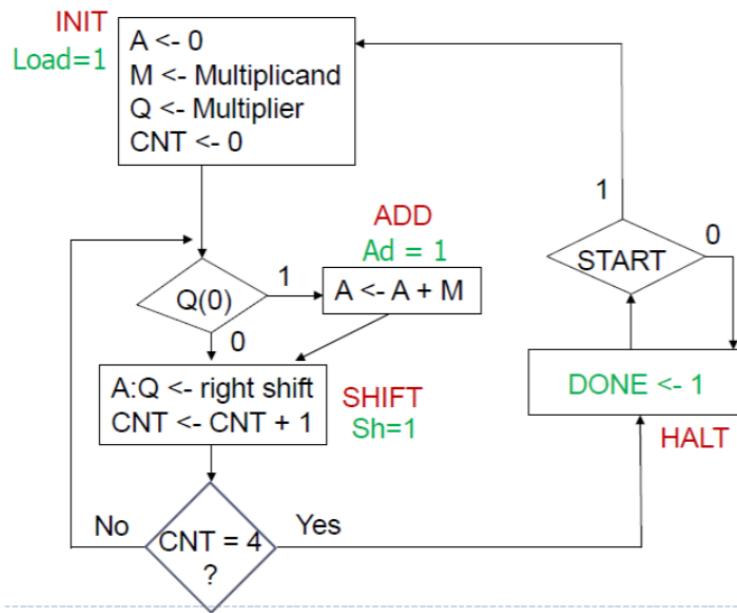
elsif (clk'event and clk = '1') then
    case state is
        when 0 =>
            if start = '1' then
                acc(8 downto 4) <= "0000";
                acc(3 downto 0) <= mer;
                state <= 1;
            end if;
        when 1|3|5|7 =>
            if M = '1' then
                acc(8 downto 4) <= ('0'&acc(7 downto 4)) +
                ('0'&mcand);
                state <= state+1;
            else
                acc(8 downto 0) <= '0' & acc(8 downto 1);
                state <= state +2;
            end if;
        when 2|4|6|8 =>
            acc(8 downto 0) <= '0' & acc(8 downto 1);
            state <= state +1;
        when 9 =>
            state <= 0;
    end case;
end if;
end process;

done <= '1' when state <= 9 else '0';

end Behavioral;

```

- Dùng ASM (better):



## LECTURE 5: HIERARCHICAL DESIGN

**Benefits of hierarchical design:**

- Xử lý được 1M gates
- Complexity management:
  - o Có thể quản lý **từng phần riêng biệt** trong 1 project lớn mà ko gây ảnh hưởng cho nhau
- Design reuse:
  - o Sử dụng modules có sẵn hoặc bên thứ 3

**Generics:**

- Là việc **pass a parameter vào trong entity hoặc component**
- Khai báo ở khu vực entity declaration, sau đó có thể được sử dụng như constant trong port declaration và architecture body

ENTITY regn IS

```

    GENERIC ( N : INTEGER := 16 ) ;
    PORT ( D : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
           Resetn, Clock : IN STD_LOGIC ;
           Q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;

```

END regn ;

**Component:**

- Declaration:
  - o Declare ở phần entity declaration
  - o Nếu **tên** của **component** và interface cần phải **giống** với **tên** của an **existing design** -> thì design đó mới được sử dụng
  - o Syntax:

```

component component_name
  generic
    generic_declaration;
    generic_declaration;
    . . .
  );
  port
    port_declaration;
    port_declaration;
    . . .
);
end component;

```

Ví dụ:

```

entity dec_counter is
  port(
    clk, reset: in std_logic;
    en: in std_logic;
    q: out std_logic_vector(3 downto 0);
    pulse: out std_logic
  );
end dec_counter;

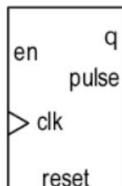
```

### Component declaration for dec\_counter

```

component dec_counter
  port(
    clk, reset: in std_logic;
    en: in std_logic;
    q: out std_logic_vector(3 downto 0);
    pulse: out std_logic
  );
end component;

```



```

architecture up_arch of dec_counter is
  signal r_reg: unsigned(3 downto 0);
  signal r_next: unsigned(3 downto 0);
  constant TEN: integer:= 10;
begin
  -- register
  process(clk,reset)
  begin
    if (reset='1') then
      r_reg <= (others=>'0');
    elsif (clk'event and clk='1') then
      r_reg <= r_next;
    end if;
  end process;
  -- next-state logic
  process(en,r_reg)
  begin
    r_next <= r_reg;
    if (en='1') then
      if r_reg=(TEN-1) then
        r_next <= (others=>'0');
      else
        r_next <= r_reg + 1;
      end if;
    end if;
  end process;
  -- output logic
  q <= std_logic_vector(r_reg);
  pulse <= '1' when r_reg=(TEN-1) else
    '0';
end up_arch;

```

Thì lúc này, ta có thể gọi dec\_counter as a component trong 1 architecture khác :

```

library ieee;
use ieee.std_logic_1164.all;
entity hundred_counter is
  port(
    clk, reset: in std_logic;
    en: in std_logic;
    q_ten, q_one: out std_logic_vector(3 downto 0);
    p100: out std_logic
  );
end hundred_counter;

architecture vhdl_87_arch of hundred_counter is
  component dec_counter
    port(
      clk, reset: in std_logic;
      en: in std_logic;
      q: out std_logic_vector(3 downto 0);
      pulse: out std_logic
    );
  end component;
  signal p_one, p_ten: std_logic;

begin
  one_digit: dec_counter
    port map (clk=>clk, reset=>reset, en=>en,
              pulse=>p_one, q=>q_one);
  ten_digit: dec_counter
    port map (clk=>clk, reset=>reset, en=>p_one,
              pulse=>p_ten, q=>q_ten);
  p100 <= p_one and p_ten;
end vhdl_87_arch;

```

- **Port map** có thể map như trên hoặc dùng **position association**:
  - o Cần thận vì khi đổi thứ tự của entity dec\_counter thì kết quả sẽ bị sai

```

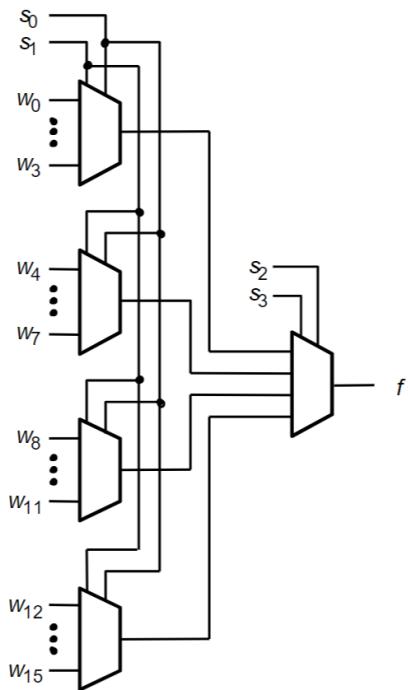
one_digit: dec_counter
  port map (clk, reset, en, q_one, p_one);
ten_digit: dec_counter
  port map (clk, reset, p_one, q_ten, p_ten);

```

### Generate scheme for component instantiations:

- Generate statements: bao gồm generation scheme và 1 tập hợp các concurrent statements
  - o For i in range generate  
end generate;
  - o If i in range generate  
end generate;
  - o ở **trong generate** là những **concurrent statements**

### Ví dụ sử dụng generate scheme:



Ví dụ có mạch như bên cạnh, thì:

- tạo 1 cái entity để implement 4 to 1 mux
- sau đó dùng component để sử dụng cái 4-1 mux đó 5 lần  
(Chú ý: 4 cái output của 4 cái mux đầu tiên ko thể port map trực tiếp với input của cái mux thứ 5 => phải dùng signal)

ENTITY mux4to1 IS

```
PORT ( w0, w1, w2, w3 : IN STD_LOGIC ;
       s : IN STD_LOGIC_VECTOR(1 DOWNTO 0) ;
       f : OUT STD_LOGIC );
```

END mux4to1 ;

ARCHITECTURE Dataflow OF mux4to1 IS

BEGIN

```
WITH s SELECT
      f <= w0 WHEN "00",
      w1 WHEN "01",
      w2 WHEN "10",
      w3 WHEN OTHERS ;
```

END Dataflow ;

ENTITY Example1 IS

```
PORT ( w : IN STD_LOGIC_VECTOR(0 TO 15) ;
       s : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
       f : OUT STD_LOGIC ) ;
```

END Example1 ;

## ARCHITECTURE Structure OF Example1 IS

```
COMPONENT mux4to1
    PORT (    w0, w1, w2, w3      : IN      STD_LOGIC ;
              s                  : IN      STD_LOGIC_VECTOR(1 DOWNTO
0) ;
              f                  : OUT     STD_LOGIC ) ;
END COMPONENT ;

SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;

BEGIN
    Mux1: mux4to1 PORT MAP ( w(0),  w(1),  w(2),  w(3),
                           s(1 DOWNTO 0), m(0) );
    Mux2: mux4to1 PORT MAP ( w(4),  w(5),  w(6),  w(7),
                           s(1 DOWNTO 0), m(1) );
    Mux3: mux4to1 PORT MAP ( w(8),  w(9),  w(10), w(11),
                           s(1 DOWNTO 0), m(2) );
    Mux4: mux4to1 PORT MAP ( w(12), w(13), w(14), w(15),
                           s(1 DOWNTO 0), m(3) );
    Mux5: mux4to1 PORT MAP ( m(0),  m(1),  m(2),  m(3),
                           s(3 DOWNTO 2), f ) ;
END Structure ;
```

- Thay vì vậy, ta có thể **dùng generate**

## ARCHITECTURE Structure OF Example1 IS

```
COMPONENT mux4to1
    PORT (    w0, w1, w2, w3      : IN      STD_LOGIC ;
              s                  : IN      STD_LOGIC_VECTOR(1 DOWNTO
0) ;
              f                  : OUT     STD_LOGIC ) ;
END COMPONENT ;

SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;

BEGIN
    G1: FOR i IN 0 TO 3 GENERATE
        Muxes: mux4to1 PORT MAP (
            w(4*i), w(4*i+1), w(4*i+2), w(4*i+3), s(1 DOWNTO 0), m(i) ) ;
        END GENERATE ;
        Mux5: mux4to1 PORT MAP ( m(0), m(1), m(2), m(3), s(3 DOWNTO 2), f ) ;
    END Structure ;
```

## LECTURE: CONFIGURATION

### Định nghĩa:

- Specifies how architectures are bound with entities

- Cho phép thay đổi components at sim time
- Bind a component with an entity and an architecture
- Tóm lại là nó dùng để xác định 1 entity sẽ sử dụng architecture nào

**Types of configuration:**

- **Configuration specification (inside architecture body):**

```

architecture vhdl_87_config_arch of hundred_counter is
component dec_counter
port(
    clk, reset: in std_logic;
    en: in std_logic;
    q: out std_logic_vector(3 downto 0);
    pulse: out std_logic
);
end component;
for one_digit: dec_counter
use entity work.dec_counter(down_arch);
for ten_digit: dec_counter
use entity work.dec_counter(down_arch);
signal p_one, p_ten: std_logic;
begin

```

- For.... use....

- **Configuration declaration (an independent design unit):**

- Thay vì viết bên trong architecture như trên thì ta có 1 file độc lập riêng biệt để làm việc đó

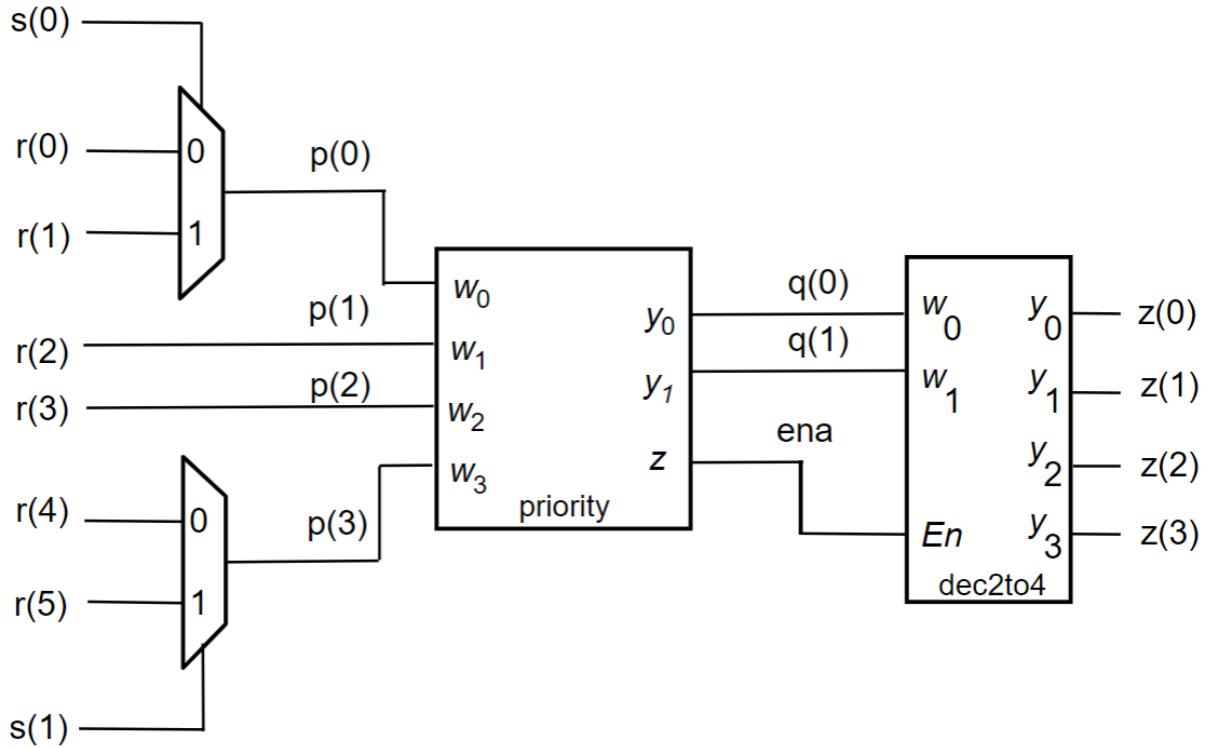
- Syntax:

```

configuration conf_name of entity_name is
  for archiecture_name
    for instance_label: component_name
      use entity lib_name.bound_entity_name(bound_arch_name);
    end for;
    for instance_label: component_name
      use entity lib_name.bound_entity_name(bound_arch_name);
    end for;
    .
  end for;
end;

```

**Ví dụ sử dụng configuration:**



```

ENTITY priority_resolver IS
  PORT (r      : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
        s      : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        z      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END priority_resolver;

```

ARCHITECTURE structural OF priority\_resolver IS

```

SIGNAL p : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL q : STD_LOGIC_VECTOR (1 DOWNTO 0);
SIGNAL ena : STD_LOGIC ;

```

**COMPONENT mux2to1**

```
PORT (w0, w1, s      : IN   STD_LOGIC ;
      f            : OUT  STD_LOGIC ) ;
```

**END COMPONENT ;**

**COMPONENT priority**

```
PORT (w      : IN   STD_LOGIC_VECTOR(3 DOWNTO 0) ;
      y      : OUT  STD_LOGIC_VECTOR(1 DOWNTO 0) ;
      z      : OUT  STD_LOGIC ) ;
```

**END COMPONENT ;**

**COMPONENT dec2to4**

```
PORT (w      : IN   STD_LOGIC_VECTOR(1 DOWNTO 0) ;
      En     : IN   STD_LOGIC ;
      y      : OUT  STD_LOGIC_VECTOR(0 TO 3) ) ;
```

**END COMPONENT ;**

**BEGIN**

```
u1: mux2to1 PORT MAP (w0 => r(0) ,
                      w1 => r(1),
                      s => s(0),
                      f => p(0));
```

```
p(1) <= r(2);
p(2) <= r(3);
```

```
u2: mux2to1 PORT MAP (w0 => r(4) ,
                      w1 => r(5),
                      s => s(1),
                      f => p(3));
```

```
u3: priority PORT MAP (w => p,
                      y => q,
                      z => ena);
```

```
u4: dec2to4 PORT MAP (w => q,
                      En => ena,
                      y => z);
```

**END structural;**

```
CONFIGURATION SimpleCfg OF priority_resolver IS
```

```
    FOR structural
```

```
        FOR ALL: mux2to1
```

```
            USE ENTITY work.mux2to1(dataflow);
```

```
        END FOR;
```

```
        FOR u3: priority
```

```
            USE ENTITY work.priority(dataflow);
```

```
        END FOR;
```

```
        FOR u4: dec2to4
```

```
            USE ENTITY work.dec2to4(dataflow);
```

```
        END FOR;
```

```
    END FOR;
```

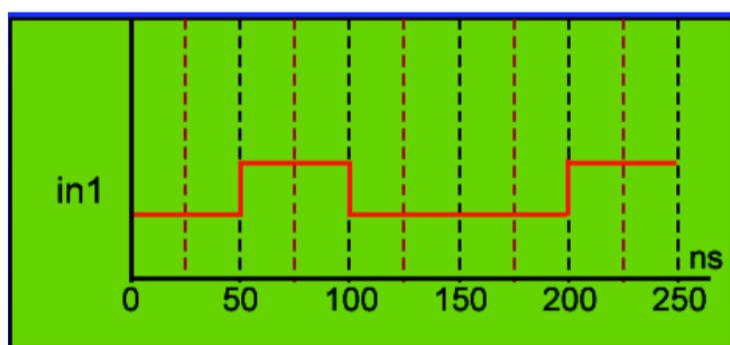
```
END SimpleCfg;
```

## TEST BENCHES

- Có 2 thứ để ta test đó là signal values (coi kết quả có đúng ko) and timing (coi có đạt được thời gian yêu cầu không)
- Có 3 kiểu test:
  - o Ktra signal values and timing manually
  - o Ktra signal values automatically, ktra timing manually
  - o Ktra signal values and timing automatically

Generate input for testing:

in1 <= '0', '1' after 50 ns, '0' after 100 ns, '1' after 200;



Generate all possible values for 2 inputs:

- Dùng for loop

```

SIGNAL test_ab : STD_LOGIC_VECTOR(1 downto 0);
SIGNAL test_sel : STD_LOGIC_VECTOR(1 downto 0);

BEGIN
    .....
    double_loop: PROCESS
    BEGIN
        test_ab <="00";
        test_sel <="00";
        for I in 0 to 3 loop
            for J in 0 to 3 loop
                wait for 10 ns;
                test_ab <= test_ab + 1;
            end loop;
            test_sel <= test_sel + 1;
        end loop;
    END PROCESS;
    .....
END behavioral;

```

**Generate periodical signals (clock, etc..)**

```

CONSTANT clk1_period : TIME := 20 ns;
CONSTANT clk2_period : TIME := 200 ns;
SIGNAL clk1 : STD_LOGIC;
SIGNAL clk2 : STD_LOGIC := '0';

BEGIN
    .....
    clk1_generator: PROCESS
        clk1 <= '0';
        WAIT FOR clk1_period/2;
        clk1 <= '1';
        WAIT FOR clk1_period/2;
    END PROCESS;

    clk2 <= not clk2 after clk2_period/2;
    .....
END behavioral;

```

### How to test automatically:

- Khi mà output quá phức tạp để test manually thì ta nên test automatic.
- Để test automatic thì ta dùng ASSERT (assert can be both concurrent statement and sequential statement)
- Assert dùng để viết message ra screen khi có lỗi xảy ra.
- Severity của assert:
  - o Note
  - o Warning: alerts users that condition is not expected, but not fatal
  - o Error: condition sẽ làm cho model work incorrectly
  - o Failure: alerts users that condition is catastrophic

```
library ieee;          -- load the ieee 1164 library
use ieee.std_logic_1164.all;    -- make the package "visible"
use work.nandgate;      -- use NAND gate model from 'work'

entity testnand is      -- top level entity of the test bench
end testnand;           -- has no ports

architecture stimulus of testnand is
component nand           -- first declare lower level entity
port (A,B: in std_logic;
      Y: out std_logic);
end component;

signal A,B: std_logic;   -- next, declare some local signals to
signal Y: std_logic;     -- assign values and observe

begin
  -- create an instance of the comparator circuit.
  NAND1: nand port map(A => A, B =>B, Y => Y);
```

} entity decl.  
for test bench

} Unit under  
test: includes  
local signals  
to apply stim  
and measure  
response

} Component  
instantiation

```

process
  constant PERIOD: time := 40ns;
begin
  A <= '1';
  B <= '1';
  wait for PERIOD;
  assert (Y='0')
    report 'Test failed' severity ERROR;
  A <= '1';
  B <= '0';
  wait for PERIOD;
  assert (Y='1')
    report 'Test failed' severity ERROR;
  A <= '0';
  B <= '1';
  wait for PERIOD;
  assert (Y='1')
    report 'Test failed' severity ERROR;
  A <= '0';
  B <= '0';
  wait for PERIOD;
  assert (Y='1')
    report 'Test failed' severity ERROR;
  wait;
end process;
end stimulus;

```

process  
describes the inputs to the circuit over time

Wait: provides specific delay value

**assert condition\_expression**  
**report text\_string**  
**severity severity\_level;**

*if condition\_expression fails then report string displayed and simulator takes action depending on severity.*

#### Pre-defined signal attributes:

- A'event: trả về gtri Boolean khi an event occurred on signal A
- A'active: Boolean, true nếu xảy ra a transaction
- A'last\_event: trả về thời gian tính từ last event cho tới lúc gọi lệnh này.
- A'last\_active: trả về thời gian tính từ last transaction cho tới lúc gọi lệnh này (transaction is any update to a signal, an update may or may not cause an event)
- A'last\_value: trả về giá trị của signal trước khi xảy ra event

#### Signal attributes that return signals:

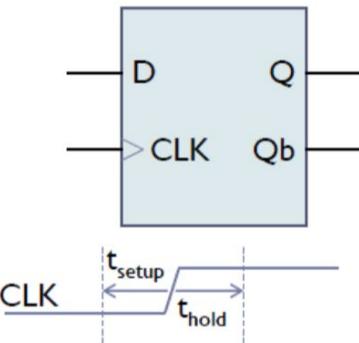
- A'delayed(T): trả về signal có type giống A nhưng bị delayed T thời gian.
- A'stable(T): Boolean, true khi signal A ko xảy ra bất kì event nào trong khoảng thời gian T.
  - o Not A'stable = A'event
- A'quiet(T): Boolean, true khi signal A ko xảy ra bất kì transaction nào trong khoảng thgian T.

#### Check hold and setup time:

```

check: process
begin
    wait until (clk'event and CLK = '1');
    assert (D'stable(setup_time))
        report "Setup time violation"
        severity ERROR;
    wait for hold_time;
    assert (D'stable(hold_time))
        report "Hold time violation"
        severity ERROR;
end process check;

```



D should be "stable" for  $t_{\text{setup}}$  prior to the clock edge and remain stable until  $t_{\text{hold}}$  following the clock edge.

## LAB2: RAM

- SRAM: static ram, là 1 array của các flip flops. It can be very fast and integrated with logic.
  - o Sử dụng 6 transistors để store 1 bit
- DRAM: dynamic ram, sử dụng a single capacitor được điều khiển bởi 1 công tắc để store a bit. Need to refresh periodically
  - o Sử dụng 1 transistor và 1 capacitor để store 1 bit

Các bước thực hiện:

- Initial -> enable? -> load 16bit, read mode, write mode... -> chia nửa ra để vào row address -> enable RAS -> nửa còn lại load vào column address -> enable CAS -> thực hiện read/write

Yêu cầu:

- Phải làm 2 designs (hỏi tutor xem làm như nào)
- Try at least 3 different state assignment (là gray decoding, binary decoding....) để compare được speed, area...
- Hỏi tutor xem 2 cái state assignments khác nhau có được coi là 2 different designs ko

Coi lại lecture nếu ko chắc chắn điều gì

## LECTURE: STATIC TIMING ANALYSIS AND PIPELINING

**Dynamic timing analysis:**

- Kiểm tra full behaviour của circuit bằng set of input stimulus vectors – viết test bench là dynamic timing analysis

**Static timing analysis:**

- Ko kiểm tra functionalities của circuit như testbench mà chỉ kiểm tra **timing performance** of circuit by checking all possible paths for timing violations
- STA sẽ **break design down into a set of timing paths**, tính toán **signal propagation delay** trên mỗi path, và kiểm tra **violation of timing constraints** ở trong design và ở input/output interface.