香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

Lecture 10:
VHDL versus Verilog

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

Outline

- **VHDL** vs. **Verilog**
  - Background
  - Popularity and Syntax
  - Operators
  - Overall Structure
  - External I/O Declaration
  - Concurrent Statement
  - Sequential Statement
  - Wire vs. Reg
  - Structural Design
  - Design Constructions
  - Case Study: Flip-flop

VHDL
Very High Speed Integrated Circuit
Hardware Description Language

Verilog
HDL

What are VHDL and Verilog?

VHDL

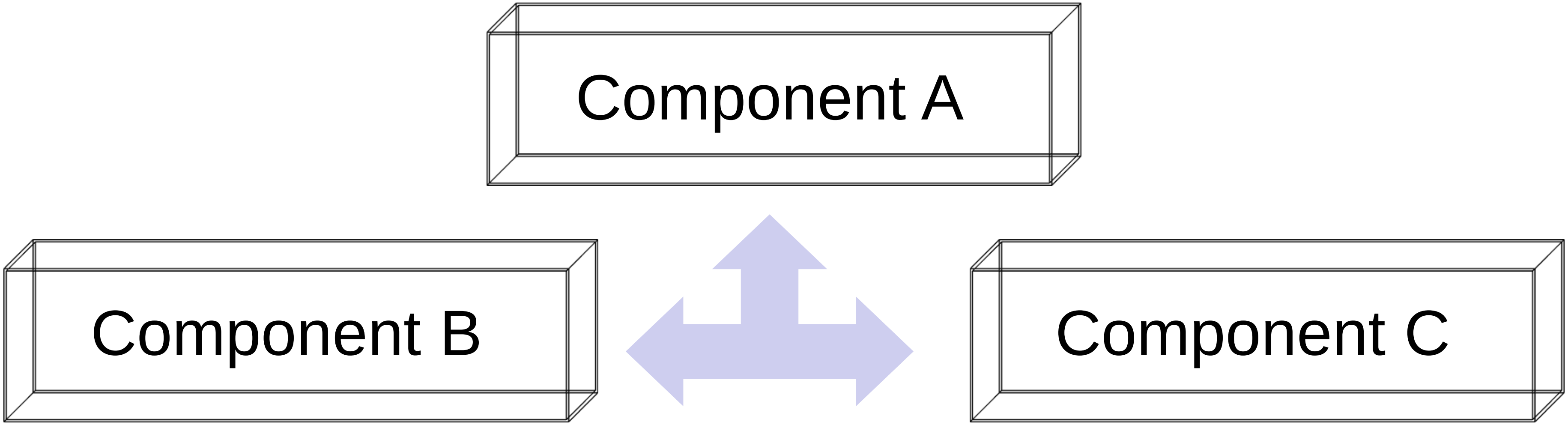
Very High Speed Integrated Circuit
Hardware Description Language

Verilog HDL

- They are both hardware description languages for modeling hardware.
- They are each a notation to describe the behavioral and structural aspects of an electronic digital circuit.

VHSIC Hardware Description Language

- VHDL uses top-down approach to partition a design into small building blocks (i.e., `components`).
  - **Entity**: Describe interface signals and basic building blocks.
  - **Architecture**: Describe behavior, each entity can have multiple Architectures.

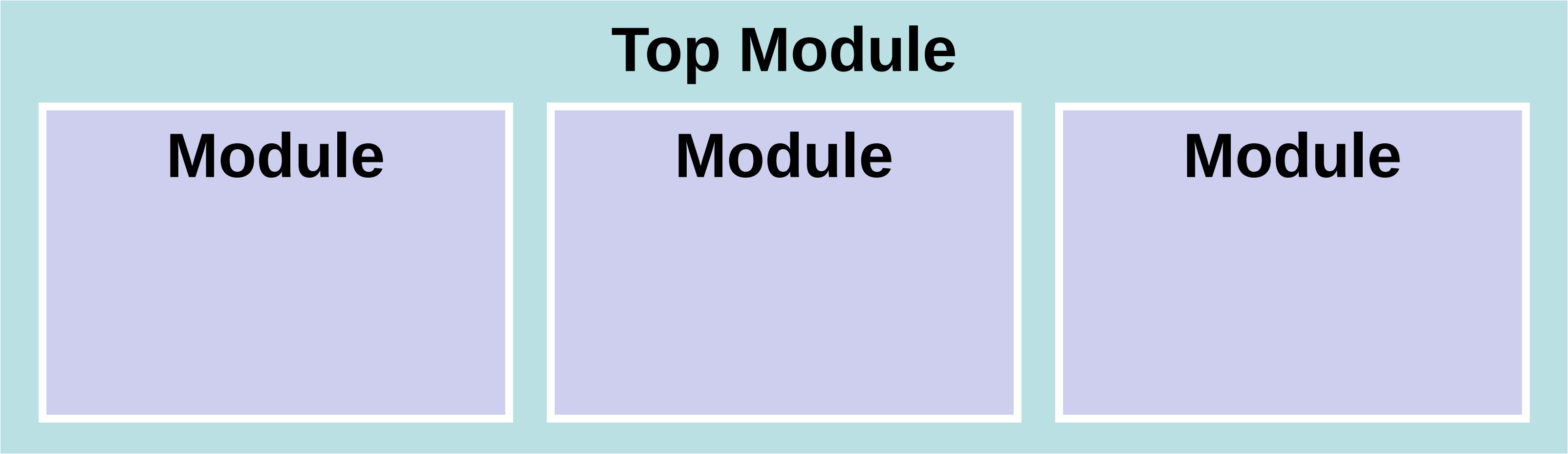


Connected by `port map` in architecture body

VHSIC stands for **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit.

Verilog

- In Verilog, the building block is called `module`.
  - There is only one module per file (**.v**) usually.
  - Modules connect through their ports (similarly as in VHDL).
  - A top level module invokes instances of other modules.



Connected by relating I/O and internal wires

Popularity and Syntax

VHDL
Very High Speed Integrated Circuit
Hardware Description Language

Verilog
HDL

Popularity	
VHDL is more popular with European companies.	**Verilog** is more popular with US companies.

Programming Style (Syntax)	
VHDL is similar to Ada programming language.	**Verilog** is similar to C/Pascal programming language.
VHDL is NOT case-sensitive.	**Verilog** is case-sensitive.
VHDL is more “verbose” than **Verilog**.	

Outline

- **VHDL** vs. **Verilog**
  - Background
  - Popularity and Syntax
  - Operators
  - Overall Structure
  - External I/O Declaration
  - Concurrent Statement
  - Sequential Statement
  - Wire vs. Reg
  - Structural Design
  - Design Constructions
  - Case Study: Flip-flop

VHDL

Very High Speed Integrated Circuit Hardware Description Language

Verilog HDL

Operators

	VHDL	Verilog		VHDL	Verilog
Add	+	+	Bitwise Negation	**not**	~
Subtract	-	-	Bitwise NAND	**nand**	~&
Multiplication	*	*	Bitwise NOR	**nor**	~|
Division	/	/	Bitwise XNOR	**xnor**	~^
Modulo	**mod**	%	Greater (or Equal)	>, >=	>, >=
Absolute	**abs**	N/A	Less (or Equal)	<, <=	<, <=
Exponentiation	**	**	Logical Equality	=	==
Concatenation	&	{ , }	Logical Inequality	/=	!=
Left Shift	**sll**	<<	Logical AND	**and**	&&
Right Shift	**srl**	>>	Logical OR	**or**	||
Bitwise AND	**and**	&	Logical Negation	**not**	!
Bitwise OR	**or**	|	Case Equality	N/A	===
Bitwise XOR	**xor**	^	Case Inequality	N/A	!==

Overall Structure

VHDL (.vhd)

```
-- Library Declaration
library IEEE;
...
-- Entity Declaration
entity ex is
...
end ex
-- Architecture Body
architecture arch of ex is
begin
...
end arch;
```

Verilog (.v)

```
// One Module
module ex ( ... );




...





endmodule
```

External I/O Declaration

VHDL

-- Entity Declaration

```
entity ex is
port(a, b: in std_logic;
c: in std_logiv_vector(3
down to 0),
y: out std_logic);
end ex
```

-- Architecture Body

```
architecture arch of mux is
begin
...
end ex;
```

Verilog

// One Module

```
module ex ( a, b, c, y );
input a, b;
input[3:0] c;
output y;
```

or

```
module ex (
input a, input b,
input[3:0] c, output y );
...

endmodule
```

Concurrent & Sequential Statements

VHDL

```
-- Entity Declaration
entity ex is
...
end ex
architecture arch of ex is
begin
-- concurrent statements
process ( sensitivity list )
begin
  -- sequential statements
end process;
end arch;
```

Verilog

```
// One Module
module ex ( ... );
-- concurrent statements
always @( sen. list or event )
begin
  -- sequential statements
end
endmodule
```

1) Concurrent Statement

VHDL: *inside architecture body, outside the* `process`

```
signal a, b: std_logic_vector(7 downto 0); -- array
signal c, d, e: std_logic;
a(3 downto 0) <= b(7 downto 4);
b(7 downto 4) <= "0000";
c <= d and e; -- bitwise AND
```

`LHS <= RHS;`

- **LHS** must be **signal**.
- The **LHS** will be updated whenever **RHS** changes.

Verilog: *outside the* `always@` *block*

```
wire [7:0] a, b; // array
wire c, d, e;
assign a[3:0] = b[7:4];
assign b[7:4] = 'b0000; // binary
assign c = d & e; // bitwise AND
```

`assign LHS = RHS;`

- **LHS** must be **wire**.
- The **LHS** will be updated whenever **RHS** changes.

Class Exercise 10.1

Student ID: ___________ Date:

Name: ________________ _______

- Translate the following VHDL program to Verilog:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity abc is
  port (a,b,c: in std_logic;
        y: out std_logic);
end abc;
architecture abc_arch of
abc is
signal x : std_logic;
begin
  x <= a nor b;
  y <= x and c;
end abc_arch;
```

2) Sequential Statement

VHDL

```
architecture arch of ex is
begin
process( sensitivity list )
variable a, b, c;
begin
   -- LHS could be signals
     (suggested) or variables


  -- variable assignment (:=)


  -- signal assignment (<=)
end;
end arch;
```

Verilog

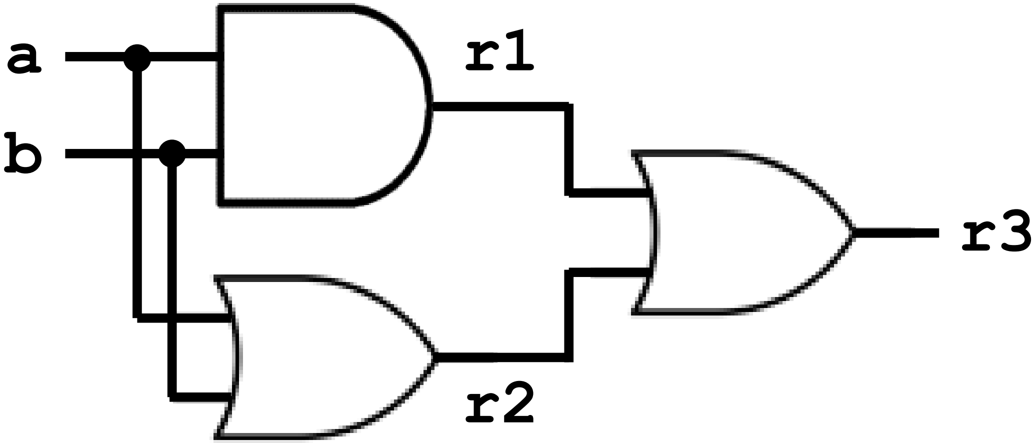
```
module ex (…);
reg a, b, c;
always @( sen. list or event )
begin

   // LHS must be reg (not wire)


   // blocking assignment (=)


   // non-blocking assignment
     (<=)
end
endmodule
```

2) Sequential Statement: Blocking

- **Blocking assignments (=)** in a sequential block (i.e., `always@`) are executed before the execution of the statements that follow it.
  - All blocking assignments are executed in a sequential way.
- **Usage**: Use blocking assignments in **`always@`** blocks to synthesize combinational logic (i.e., no clock!).

```
reg r1, r2, r3;
...
always @ (a, b)
begin
  r1 = a & b;
  r2 = a | b;
  r3 = r1 | r2;
end
```

sensitivity list `(a, b) OR (a or b)`



Class Exercise 10.2

Student ID: ___________ Date:

Name: _______________ _______

- Translate the following Verilog program to VHDL:

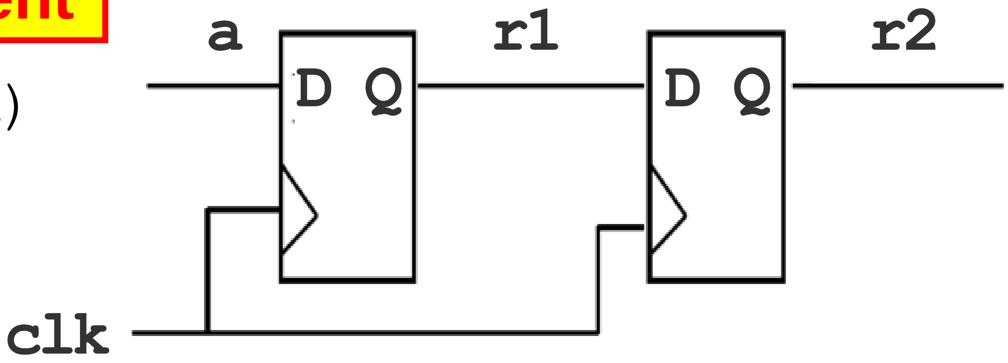
```
reg r1, r2, r3;
...
always @ (a, b)
begin
  r1 = a & b;
  r2 = a | b;
  r3 = r1 | r2;
end
```

2) Sequential Statement: Non-Blocking

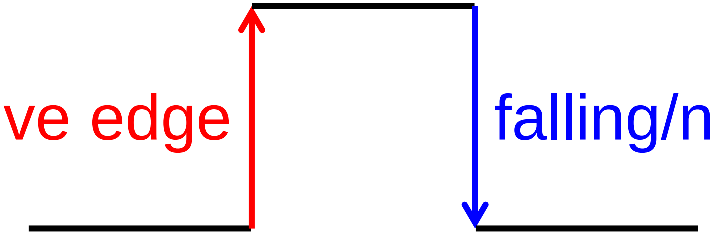
- **Non-blocking assignments (<=)** in a sequential block (i.e., `always@`) are executed within the same time step regardless of the order.
  - All non-blocking assignments will take effect at the next clock edge (concurrently, not sequentially!).
- **Usage**: Use non-blocking assignments in **`always@`** blocks to synthesize sequential logic (i.e., has clock!).

```
reg r1, r2;
...
always @ (posedge clk)
begin
→r1 <= a;
→r2 <= r1;
end
```

event



Edge Detection



rising/positive edge

falling/negative edge

VHDL

```
process (clk)
begin
   ...
   if rising_edge(CLK)
               or
   if falling_edge(CLK)
   ...
end
```

Verilog

```
always @ (posedge clk)
             or
always @ (negedge clk)
begin

   ...

end
```

Class Exercise 10.3

Student ID: ___________ Date:

Name: ________________ _______

- Translate the following Verilog program to VHDL:

```
reg r1, r2;
...
always @ (posedge clk)
begin
  r1 <= a;
  r2 <= r1;
end
```

“`wire`” vs. “`reg`” in Verilog

- **Wire**: Has *no* memory
  - It must be **physical wire** in the circuit.
  - It does *not* hold the value.
  - **Usage:** Cannot use “**`wire`**” in the LHS of assignments inside **`always@`** blocks!
- **Reg**: Has memory
  - It could be a **flip-flop** or a **physical wire**.
  - It holds the value until a new value is assigned.
  - **Usage:** Cannot use “**`reg`**” in the LHS of assignments outside **`always@`** blocks!

Outline

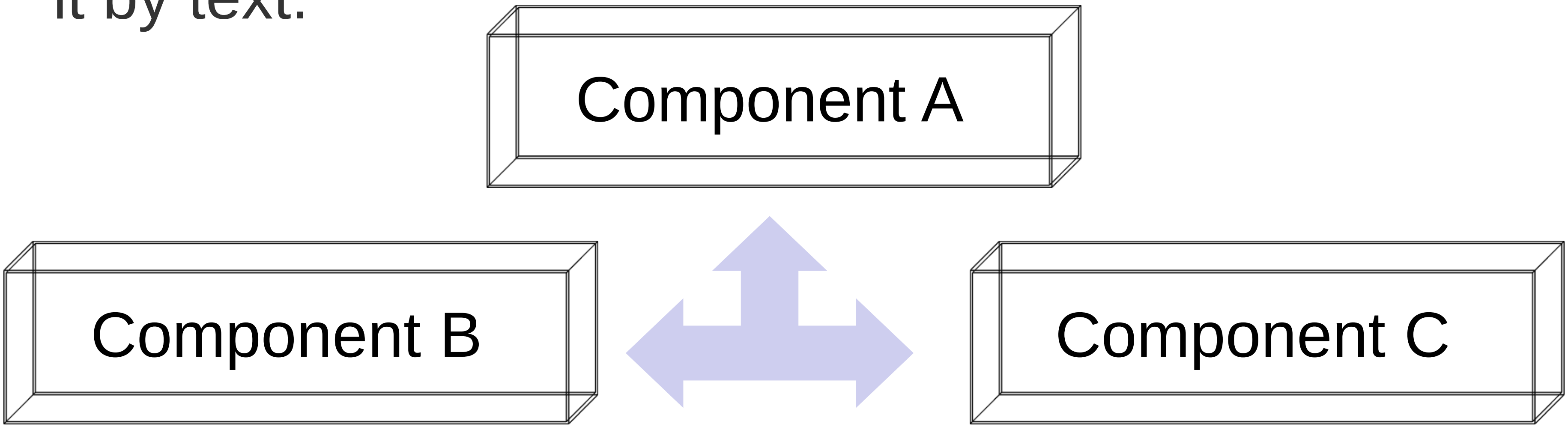
- **VHDL** vs. **Verilog**
  - Background
  - Popularity and Syntax
  - Operators
  - Overall Structure
  - External I/O Declaration
  - Concurrent Statement
  - Sequential Statement
  - Wire vs. Reg
  - Structural Design
  - Design Constructions
  - Case Study: Flip-flop

VHDL
Very High Speed Integrated Circuit
Hardware Description Language

Verilog
HDL

Structural Design in VHDL (1/2)

- Structural Design in VHDL: Like a circuit but describe it by text.



Connected by `port map` in architecture body

- **Design Steps**:

  Step 1: Create `entities`

  Step 2: Create `components` from `entities`

  Step 3: Use "`port map`" to relate the components

Structural Design in VHDL (2/2)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

Step 1

```
entity and2 is
port (a,b: in STD_LOGIC;
        c: out STD_LOGIC );
end and2;
architecture and2_arch of and2 is
begin
  c <= a and b;
end and2_arch;
```

```
---------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

Step 1

```
entity or2 is
port (a,b: in STD_LOGIC;
        c: out STD_LOGIC );
end or2;
architecture or2_arch of or2 is
begin
  c <= a or b;
end or2_arch;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------
entity test is
port ( in1: in STD_LOGIC; in2: in STD_LOGIC;
       in3: in STD_LOGIC;
      out1: out STD_LOGIC );
end test;
architecture test_arch of test is
```

Step 2

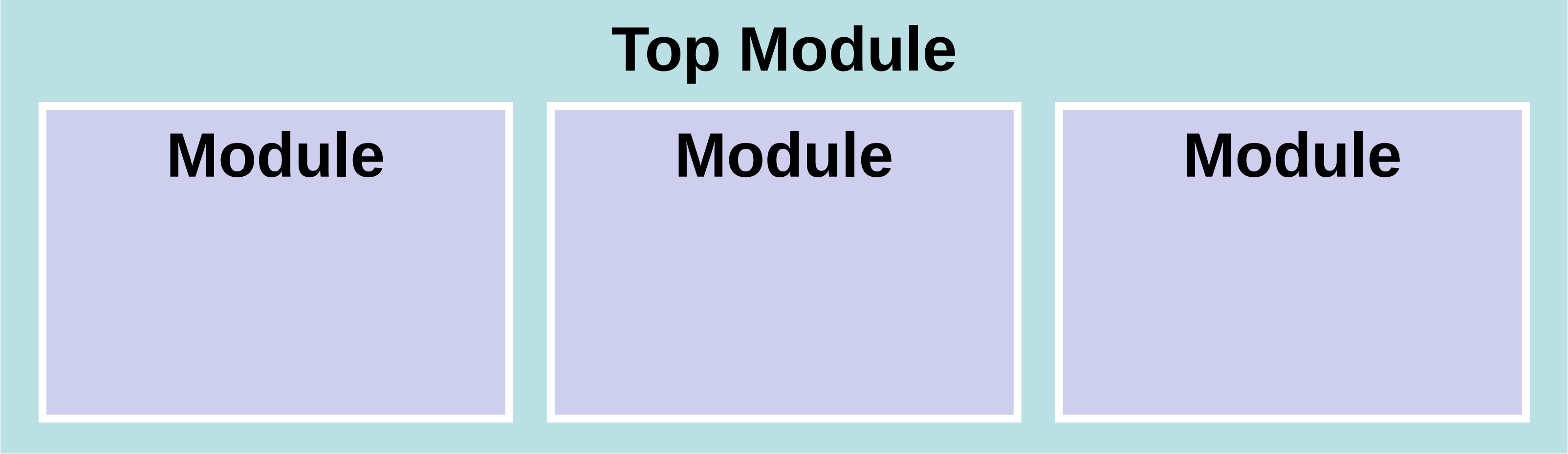
```
component and2 --create component
  port (a,b: in std_logic; c: out std_logic);
end component ;
component or2 --create component
  port (a,b: in std_logic; c: out std_logic);
end component ;
signal inter_sig: std_logic;
begin
```

Step 3

```
  label1: and2 port map (in1, in2, inter_sig);
  label2: or2 port map (inter_sig, in3, out1);
end test_arch;
```

Structural Design in Verilog (1/2)

- Structural Design in Verilog: One top module, several (sub) modules.



Connected by relating I/O and internal wires

- **Design Steps**:

  Step 1: Create (sub) `module(s)` (usually in separate **.v** files)

  Step 2: Define a `top-module` to interconnect `module(s)`

Structural Design in Verilog (2/2)

and2.v

Step 1

```
module and2(
   input a,
   input b,
   output c
   );
   assign c = a & b;
endmodule
```

or2.v

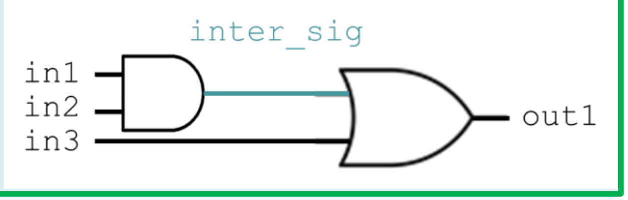
Step 1

```
module or2(
   input a,
   input b,
   output c
   );
   assign c = a | b;
endmodule
```

top_module.v

Step 2

```
module top_module(
   input in1, input in2, input in3,
   output out1 );
   wire inter_sig;
   and2 and2_ins(
      .a(in1),
      .b(in2),
      .c(inter_sig)
   );
   or2 or2_ins(
      .a(inter_sig),
      .b(in3),
      .c(out1)
   );
endmodule
```

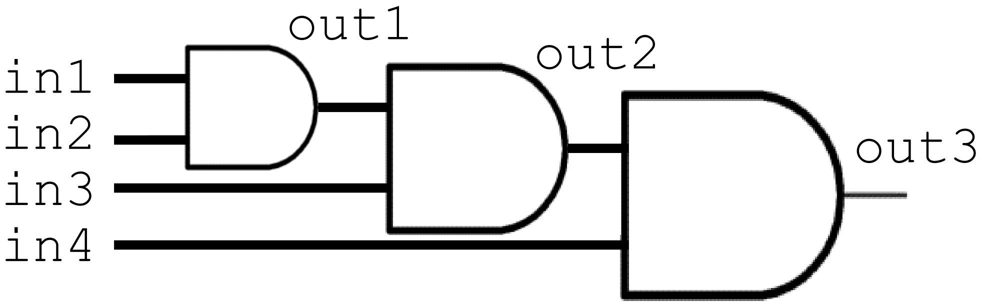


Class Exercise 10.4

Student ID: ____________ Date:

Name: __________________ ________

- Implement the following circuit in Verilog:



Outline

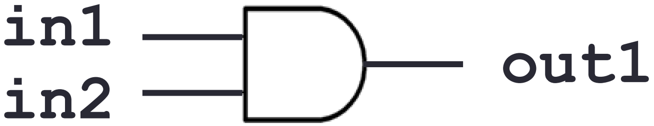
- **VHDL** vs. **Verilog**
  - Background
  - Popularity and Syntax
  - Operators
  - Overall Structure
  - External I/O Declaration
  - Concurrent Statement
  - Sequential Statement
  - Wire vs. Reg
  - Structural Design
  - Design Constructions
  - Case Study: Flip-flop

VHDL

Very High Speed Integrated Circuit
Hardware Description Language

Verilog
HDL

Design Constructions (1/4)



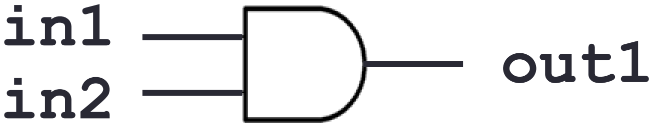
VHDL: **`when-else`** *(concurrent, outside **`process`**)*

```
architecture arch of ex is
begin
  out1 <= '1' when in1 = '1' and in2 = '1' else '0';
end arch ex_arch;
```

Verilog: **`assign ? :`** *(concurrent, outside **`always@`** block)*

```
module ex (...);
  assign out1 = (in1=='b1 && in2=='b1) ? 'b1 : 'b0;
  // 'b: binary; 'o: octal; 'd: decimal; 'h: hexadecimal
endmodule
```

Design Constructions (2/4)



VHDL: **`if-then-else`**

(sequential, inside ***`process`****)*

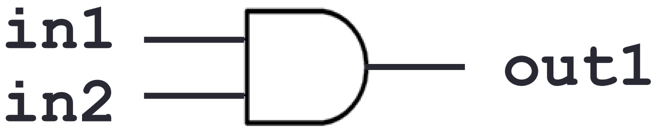
```
process(in1, in2)
begin
  if in1='1' and in2='1'
  then
    out1 <= '1';
  else
    out1 <= '0';
  end if;
end process;
```

Verilog: **`if-else`**

(sequential, inside ***`always@`****)*

```
always @(in1, in2)
begin
  if (in1=='b1 && in2=='b1)
  begin
        out1 = 'b1;
  end
  else
  begin
    out1 = 'b0;
  end
end
```

Design Constructions (3/4)



VHDL: **case-when**

(sequential, inside ***process****)*

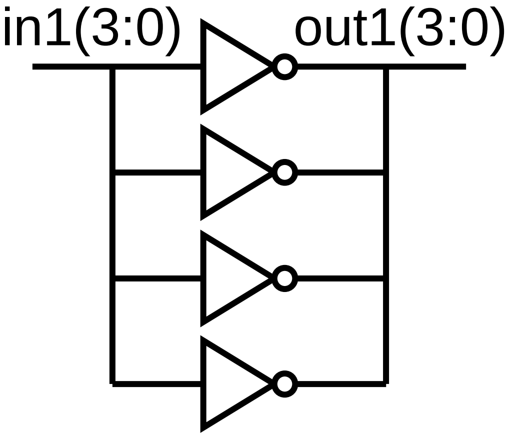
```
process(b)
begin
  case b is
    when "00"|"11" =>
      out1 <= '0';
      out2 <= '1';
    when others    =>
      out1 <= '1';
      out2 <= '0';
  end case;
end process;
```

Verilog: **case**

(sequential, inside ***always@****)*

```
always @(b)
begin
  case (b)
    'b00 || 'b11:
      out1 = 'b0;
      out2 = 'b1;
    default:
      out1 = 'b1;
      out2 = 'b0;
  endcase
end
```

Design Constructions (4/4)



VHDL: `for-in-to-loop`

*(sequential, inside **`process`**)*

```
process(in1)
begin
  for i in 0 to 3 loop
    out1(i) <= not in1(i);
  end loop;
end process;
```

Verilog: `for-loop`

*(sequential, inside **`always@`**)*

```
always @(in1)
begin
  for(idx=0; idx<4; idx+=1)
  begin
    out1[idx] = ~in1[idx];
  end
end
```

Outline

- **VHDL** vs. **Verilog**
  - Background
  - Popularity and Syntax
  - Operators
  - Overall Structure
  - External I/O Declaration
  - Concurrent Statement
  - Sequential Statement
  - Wire vs. Reg
  - Structural Design
  - Design Constructions
  - Case Study: Flip-flop

VHDL
Very High Speed Integrated Circuit
Hardware Description Language

Verilog
HDL

Posedge Flip-flop with Sync Reset (1/2)

VHDL

```
entity dff is
port(D,CLK,RESET:
     in std_logic;
     Q: out std_logic);
end dff;
architecture dff_arch of
dff is begin
  process(CLK) begin
    if rising_edge(CLK) then
      if (RESET = '1') then
        Q <= '0';
      else
        Q <= D;
      end if;
    end if;
  end process;
end dff_arch;
```

Verilog

```
module dff(
    input D,
    input CLK,
    input RESET,
    output reg Q);
always @(posedge CLK)
begin

  if (RESET) begin
    Q <= 1'b0;
  end
  else begin
    Q <= D;
  end

end
endmodule
```

Posedge Flip-flop with Sync Reset (2/2)

- **`Input`** must be **`wire`**.
- **`Output`** could be either **`wire`** or **`reg`**.
  - The default option is **`wire`**.
  - But you can specify an **`output`** as **`wire`** or **`reg`** depending on how you will assign it a value.

VHDL

```
dff is begin
  process(CLK) begin
    if rising_edge(CLK) then
      if (RESET = '1') then
        Q <= '0';
      else
        Q <= D;
      end if;
    end if;
  end process;
end dff_arch;
```

Verilog

```
module dff(
    input D,
    input CLK,
    input RESET,
    output reg Q);
always @(posedge CLK)
begin

  if (RESET) begin
    Q <= 1'b0;
  end
  else begin
    Q <= D;
  end

end
endmodule
```

Posedge Flip-flop with Async Reset (1/2)

VHDL

```
entity dff is
port(D,CLK,RESET:
     in std_logic;
     Q: out std_logic);
end dff;
architecture dff_arch of
dff is begin
  process(CLK,RESET) begin
    if (RESET = '1')
    then
      Q <= '0';
    elsif rising_edge(CLK)
    then
      Q <= D;
    end if;
end process;
end dff_arch;
```

Verilog

```
module dff(
    input D,
    input CLK,
    input RESET,
    output reg Q);
always @(posedge CLK or
         posedge RESET)
begin
  if (RESET) begin
    Q <= 1'b0;
  end
  else begin
    Q <= D;
  end

end
endmodule
```

Posedge Flip-flop with Async Reset (2/2)

VHDL

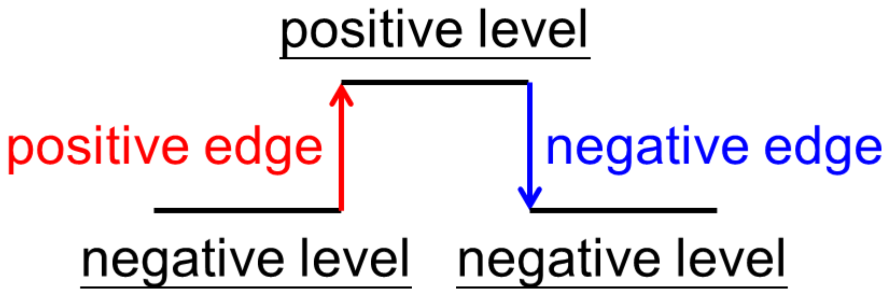
```
entity dff is
port(D,CLK,RESET:
      in std_logic;
      Q: out std_logic);
end dff;
```

Question: What if we do not specify “**`posedge`**” for the **`RESET`** signal?

[Synth 8-434] mixed level sensitive and edge triggered event controls are **not supported** for synthesis!

Verilog

```
module dff(
        input D,
        input CLK,
        input RESET,
        output reg Q);
always @(posedge CLK or
         ~~posedge~~ RESET)
begin
  if (RESET) begin
    Q <= 1'b0;
  end
  else begin
    Q <= D;
  end

end
endmodule
```



Summary

- **VHDL** vs. **Verilog**
  - Background
  - Popularity and Syntax
  - Operators
  - Overall Structure
  - External I/O Declaration
  - Concurrent Statement
  - Sequential Statement
  - Wire vs. Reg
  - Structural Design
  - Design Constructions
  - Case Study: Flip-flop

VHDL

Very High Speed Integrated Circuit
Hardware Description Language

Verilog
HDL