Introduction to VHDL

Homework 2

Chang Meng

Original Slides by Alessandro Tempia Calvino

Integrated Systems Laboratory (LSI)
EPFL

September 26, 2024

VHDL

VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

- A language to describe digital hardware systems

Uses:

- Modeling (documentation)
- Testing and validation (simulation)
- Design entry for automatic synthesis

VHDL

Goals:

- Make the design process more reliable, minimizing cost and time
- Avoid design errors
- Increase the design productivity

Other popular HDL languages:

- Verilog
- SystemC
- System Verilog

VHDL

VHDL..

- is case insensitive
- strongly typed
- comments −− till end of line

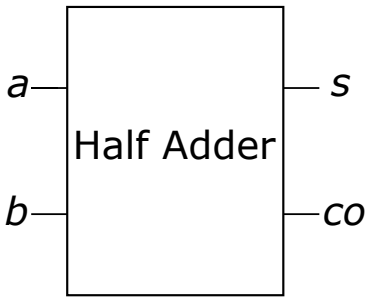
VHDL

VHDL..

- is case insensitive
- strongly typed
- comments −− till end of line

VHDL is not a programming language! It is a description language!
The keyword is **concurrency**!

Example: Half-Adder



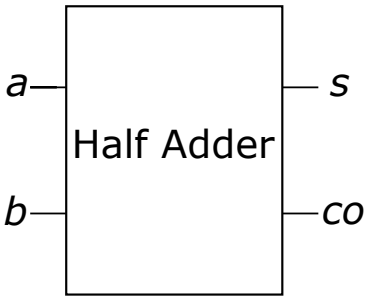
Signals:

- a, b: primary inputs
- s: sum
- co: carry out

Functions:

- $s = a \oplus b$
- $co = a \wedge b$

Example: Half-Adder




```
entity half_adder is
    port (
        a  : in std_logic;
        b  : in std_logic;
        s  : out std_logic;
        co : out std_logic
    );
end half_adder;
```

Example: Half-Adder

- entity declares a new type of component (e.g., "half_adder")
- port( ); lists inputs and outputs of the entity
  - "*a : in std_logic*" → *a* is an **input** port of type **std_logic**
  - "*s : out std_logic*" → *s* is an **output** port of type **std_logic**
  - ports are separated by semicolons

```
entity half_adder is
    port (
        a  : in std_logic;
        b  : in std_logic;
        s  : out std_logic;
        co : out std_logic
    );
end half_adder;
```

Library

Standard logic:

- IEEE 1164 standard
- package std_logic_1164: contains the enumerated types std_logic and std_ulogic

Library and standard package to always include:

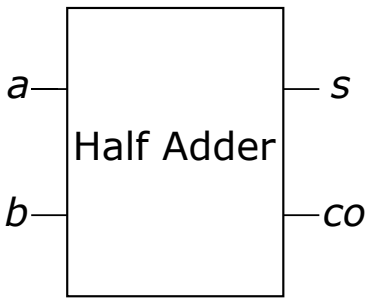
```
library ieee;
use ieee.std_logic_1164.all;
```

Library

Possible values of std_logic type

- 'U': uninitialized
- 'X': unknown
- '0': logic 0
- '1': logic 1
- 'Z': high impedance
- 'W': weak unknown, can't tell if 0 or 1
- 'L': weak 0
- 'H': weak 1
- '-': don't care

Example: Half-Adder architecture




```
entity half_adder is
    port (
        a  : in std_logic;
        b  : in std_logic;
        s  : out std_logic;
        co : out std_logic
    );
end half_adder;

architecture HA_df of half_adder is
begin
    s <= a xor b;
    co <= a and b;
end HA_df;
```

Example: Half-Adder architecture

The architecture defines the entity's functionality

- It describes the behavior or the inner implementation of an entity
- There may be several architectures for the same entity (for simulation, use configurations!)
- Each architecture must be bound to an entity

Signals are assigned with <=

```
entity half_adder is
    port (
        a  : in std_logic;
        b  : in std_logic;
        s  : out std_logic;
        co : out std_logic
    );
end half_adder;

architecture HA_df of half_adder is
begin
    s <= a xor b;
    co <= a and b;
end HA_df;
```

Architecture

```
architecture architecture_name of name_of_entity is
    -- Declarations
        -- components
        -- signals
        -- constants
        -- functions
        -- procedures
        -- types
begin
    -- concurrent statements
end architecture_name;
```

Architecture

An architecture can be described in different abstraction levels:

- Dataflow
- Behavioral (process)
- Structural (component instantiation)
- Mixed

Architecture

Dataflow

```
architecture HA_df of half_adder is
begin
    s <= a xor b;
    co <= a and b;
end HA_df;
```

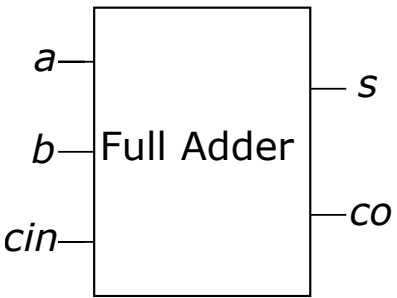
Every assignment is concurrent!

Behavioral

```
architecture HA_beh of half_adder is
begin
    process(a, b) -- sensitivity list
    begin
        s <= a xor b;
        co <= a and b;
    end process;
end HA_beh;
```

A process is one single concurrent operation, statements are sequential, concurrency among multiple interacting processes

Example: Full-Adder



Signals:

- a, b, cin: primary inputs
- s: sum
- co: carry out

Functions:

- $s = a \oplus b \oplus cin$
- $co = (a \wedge b) \vee (a \wedge cin) \vee (b \wedge cin)$

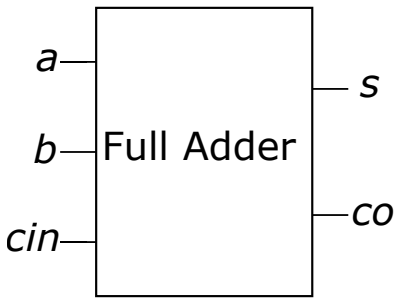
Example: Full-Adder

```
entity full_adder is
    port (
        a  : in std_logic;
        b  : in std_logic;
        ci : in std_logic;
        s  : out std_logic;
        co : out std_logic
    );
end full_adder;

architecture FA_df of full_adder is
begin
    s <= a xor b xor ci;
    co <= (a and b) or (a and ci) or (b and ci);
end FA_df;
```

Example: Full-Adder

It can be constructed starting from two half adders and additional logic

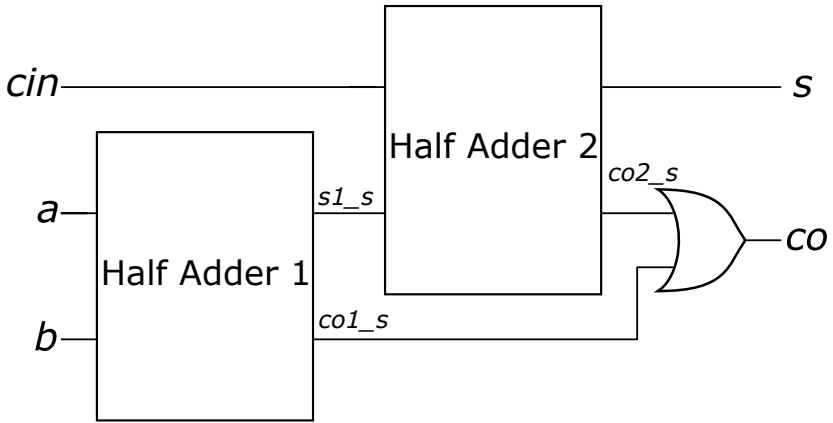


Functions:

- $s = a \oplus b \oplus cin$
- $co = ((a \oplus b) \wedge cin) \vee (a \wedge b)$

Example: Full-Adder

It can be constructed starting from two half adders and additional logic



Example: Full-Adder

Define a mixed architecture:

- **Component declaration**: match the corresponding entity declaration exactly
- Use of signals for the internal connections
- **Component instantiation**: occurrence of a component in a circuit

```
architecture FA_struct of full_adder is
    component half_adder is
        port (
            a  : in std_logic;
            b  : in std_logic;
            s  : out std_logic;
            co : out std_logic);
    end component;

    signal s1_s, co1_s, co2_s : std_logic;
begin
    HA_1: half_adder port map (a, b, s1_s, co1_s);
    HA_2: half_adder port map (ci, s1_s, s, co2_s);
    co <= co1_s or co2_s;
end FA_struct;
```

Example: Full-Adder

Component instantiation:

- **Label** for the name of the instance (e.g., "HA_1:")
- Name of the entity
- port map: maps component ports to signals (HA_1 is specified, HA_2 is positional)

```
HA_1: half_adder port map ( a => a,
                            b => b,
                            s => s1_s,
                            co => co1_s);

HA_2: half_adder port map (ci, s1_s, s, co2_s);
```

Example: 2-bit Ripple Carry Adder

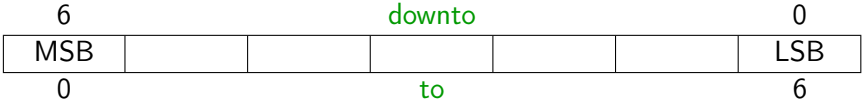
- Adds 2-bit numbers A and B
- 2-bit output sum
- 1-bit output carry

```
entity adder2 is
    port (
        a  : in std_logic_vector(1 downto 0);
        b  : in std_logic_vector(1 downto 0);
        s  : out std_logic_vector(1 downto 0);
        co : out std_logic
    );
end adder2;
```

Example: 2-bit Ripple Carry Adder

std_logic_vector:

- indexed collection of std_logic
- indices can have a range that is descending or ascending



- TIP: stick to `downto` for normal vectors of std_logic (`to` is usually used in matrices)

Example: 2-bit Ripple Carry Adder

```
entity adder2 is
    port (
        a  : in std_logic_vector(1 downto 0);
        b  : in std_logic_vector(1 downto 0);
        s  : out std_logic_vector(1 downto 0);
        co : out std_logic
    );
end adder2;

architecture adder_struct of adder2 is
    component half_adder is
        port (
            a  : in std_logic;
            b  : in std_logic;
            s  : out std_logic;
            co : out std_logic);
    end component;

    component full_adder is
        port (
            a  : in std_logic;
            b  : in std_logic;
            ci : in std_logic;
            s  : out std_logic;
            co : out std_logic
        );
    end component;

    signal co1_s: std_logic;
begin

    HA_1: half_adder port map (a(0), b(0), s(0), co1_s);
    FA_1: full_adder port map (a(1), b(1),
                               co1_s, s(1), co);
end adder_struct;
```

Simulation

Testbench

- Not synthesizable VHDL code (cannot be translated to a physical design)
- Used for simulation $\rightarrow$ generate waveforms
- Used to check if the behavior looks correct
- Provides input values, the simulator generates output values
- It can assert expected output values

Simulation

```vhdl
entity adder_tb is
end added_tb;

architecture tb of adder_tb is
    component adder2 is
        port (
            a  : in std_logic_vector(1 downto 0);
            b  : in std_logic_vector(1 downto 0);
            s  : out std_logic_vector(1 downto 0);
            co : out std_logic
        );
    end component;

    signal a_s, b_s, s_s : std_logic_vector(1 downto 0);
    signal co_s : std_logic;

begin
    DUT: adder2 port map (a_s, b_s, s_s, co_s);

    process
    begin
        -- test all the input combinations
        a_s <= "00"; b_s <= "00";
        wait for 10 ns;
        a_s <= "00"; b_s <= "01";
        wait for 10 ns;
        a_s <= "00"; b_s <= "10";
        wait for 10 ns; -- etc
    end process;
end tb;
```

Shifting

Multiplications or divisions by powers of 2 or shifts can be realised as follows:

```
signal s, s_shift : std_logic_vector(7 downto 0);

-- multiplication by 4
s_shift <= s(5 downto 0) & "00";

-- division by 8
s_shift <= "000" & s(7 downto 3);
```

References

Books

- Vahid, F., 2007, VHDL for digital design, John Wiley & Sons
- Rushton, A., 2011. VHDL for logic synthesis, John Wiley & Sons

Online books

- Free Range VHDL

Web

- `https://vhdlguide.readthedocs.io/en/latest/index.html`
- `https://www.doulos.com/knowhow/vhdl_designers_guide/`