Digital System Design

Lecture 3
VHDL Basics

Reading Assignment:

- Brown, “Fundamentals of Digital Logic with VHDL”, pp. 60 - 65, pp. 228 – 232
- Brown, “Appendix A: VHDL Reference”, pp. 779 - 791

Learning Objective:

- Provide a brief introduction of VHDL features

VHDL Introduction

- VHDL is a hardware description language (HDL)
  - We use VHDL to write a program
  - A VHDL program describes hardware
    - Just like a schematic describes hardware
  - An HDL program is used to develop a chip
    - Design
    - Synthesis
    - Simulation of chip

- Why an HDL program, why not schematics ?
  - Real life circuits are too complex to be designed (described) by schematics
  - There would be too many and complex schematics

VHDL Introduction

- VHDL was developed in the 1980s under Department of Defense (DoD) sponsorship
- VHDL stands for ?

  Very High Speed Integrated Circuit
  Hardware
  Description
  Language

- Today VHDL is widely used across the industry, around the world
  - Established as IEEE standard IEEE 1076 in 1987
  - Extended in IEEE standard IEEE 1164 in 1993
  - In 1996, IEEE 1076.3 became a VHDL synthesis standard

VHDL Introduction

- Another common HDL language : **Verilog HDL**
- VHDL v.s. Verilog
  - VHDL is more like C++; Verilog is more like C
  - VHDL is ***STRONGLY*** typed; Verilog is ***WEAKLY*** typed
  - VHDL is not case sensitive; while Verilog is case sensitive
    - In VHDL, "A" or "a" does not matter
- Knowing one HDL language helps one learn another HDL language faster

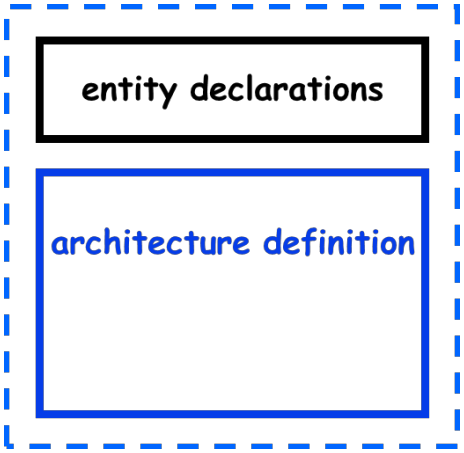
VHDL v.s. Verilog (video)

Important Terminology

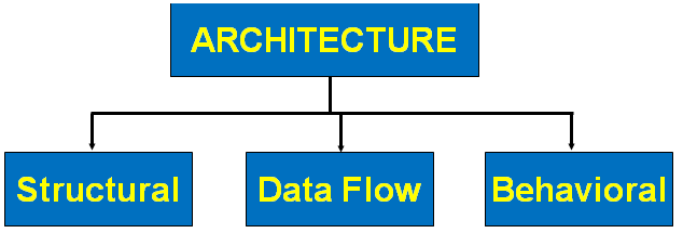
- *Simulation* is the prediction of the behavior of a design
  - VHDL provides many features suitable for the simulation of digital circuit designs
  - *Functional simulation* approximates the behavior of a hardware design by assuming that all outputs change at the same time
  - *Timing simulation* predicts the exact behavior of a hardware design
- *Synthesis* is the translation of a design into a netlist file that describes the structure of a hardware design
  - VHDL was not designed for the purpose of synthesis
  - Not all VHDL statements are synthesizable

VHDL Basics

- A VHDL program is a collection of **modules**
  - A module consists of an **entity** and an **architecture**
- Entity: shows inputs and outputs



- Architecture : internal description : ***implementation***
  - It can be written in one of three different detail levels
    - Structural (very detailed)
    - Dataflow (less detailed)
    - Behavioral (least detailed)



Entity

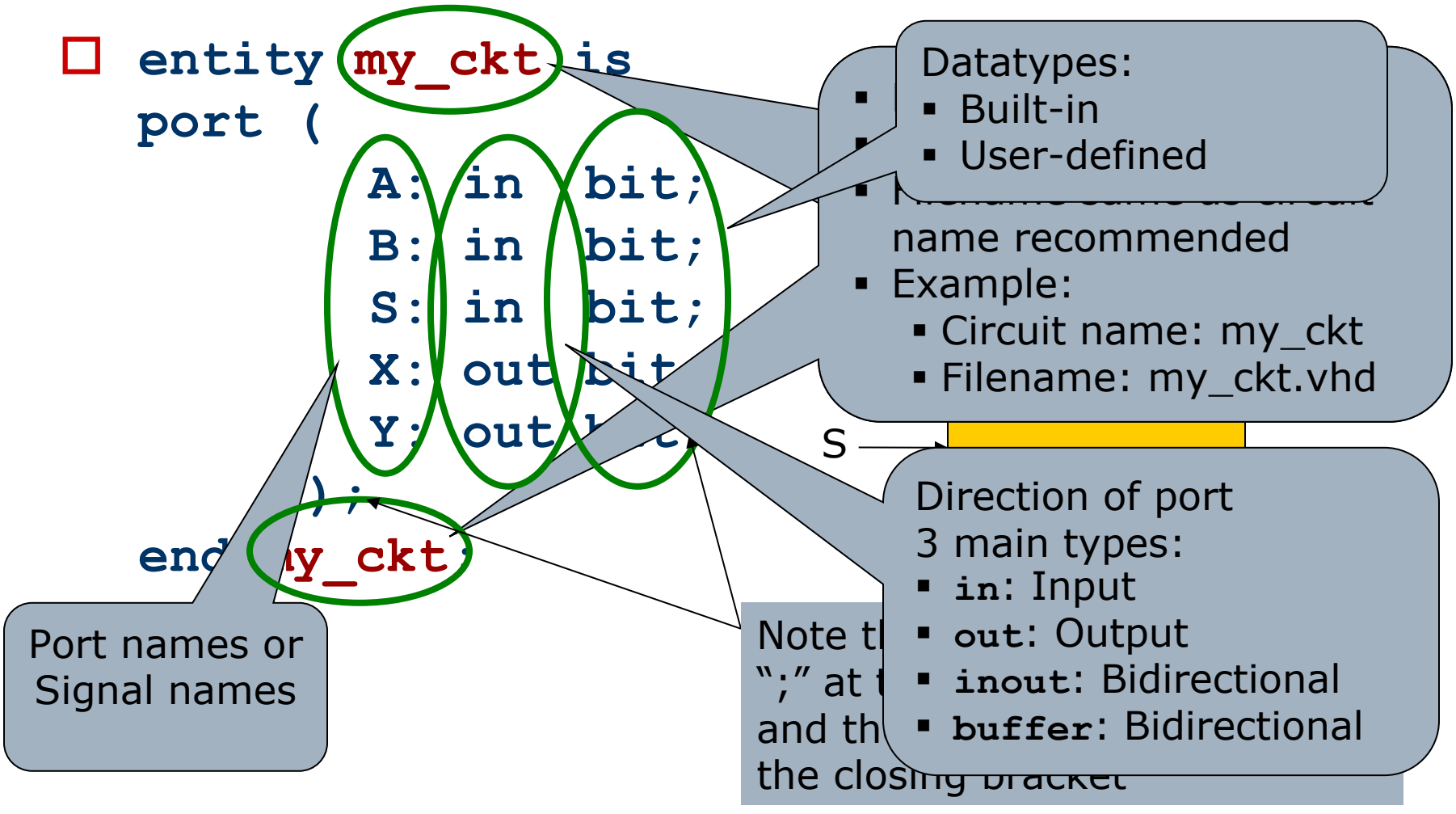
- The entity declares ports
  - Inputs and outputs
    - Digital circuit input *signals*
    - Digital circuit output *signals*

- *Syntax :*

```
entity entity-name is
        port (signal-names : mode data-type ;
              signal-names : mode data-type ;
                    ....
              signal-names : mode data-type)
end entity-name ;
```

NOTE:
In the PORT declaration, the semi-colon is used as a separator.

Entity



NOTE:
A buffer is an output that can be "read" by the architecture of the entity.

Built-in Datatypes

- Scalar (single valued) signal types:
  - **bit**
  - **boolean**
  - **Integer (32-bit, signed)**
  - **Real (32-bit, float)**
  - Examples:
    - `A: in bit;`
    - `G: out boolean;`
    - `K: out integer range -2**4 to 2**4-1;`
- Aggregate (collection) signal types:
  - **bit_vector**: array of bits representing binary numbers
  - **signed**: array of bits representing signed binary numbers
  - Examples:
    - `D: in bit_vector(0 to 7);`
    - `E: in bit_vector(7 downto 0);`
    - `M: in signed (4 downto 0);`
      `--signed 5 bit_vector binary number`

- Built-in data types work well for simulation but not so well for synthesis
- Built-in data types are suitable for use inside an architecture but should not be used for external pins

Architecture

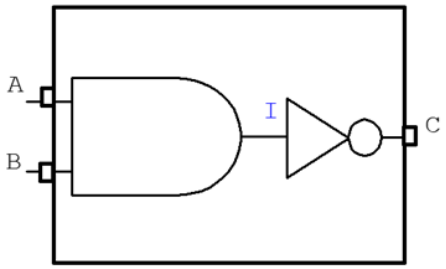
- The architecture describes the internal operations

```
architecture architecture-name of entity-name is
        type declarations
        signal declarations
        constant declarations
        function definitions
        component declarations
begin
        concurrent statement
        ....
        concurrent statement
end architecture-name ;
```

- Multiple architectures can be created for a particular entity with each architecture optimized with respect to a design goal:
  - Performance
  - Area
  - Power Consumption
  - Ease of Simulation

Architectures

```
ARCHITECTURE structure OF NAND2 IS
Signal I:BIT;
BEGIN
  Cell1:AND_2 port map(I1=>A, I2=>B, O1=>I);
  Cell2:INVERTER port map(I1=>I, O1=>C);
END structure;
```



A structural description is just like the schematic

```
architecture Dataflow of NAND2 is
begin
   C<=A nand B;
end Dataflow;
```

A dataflow description is just like the logic equation

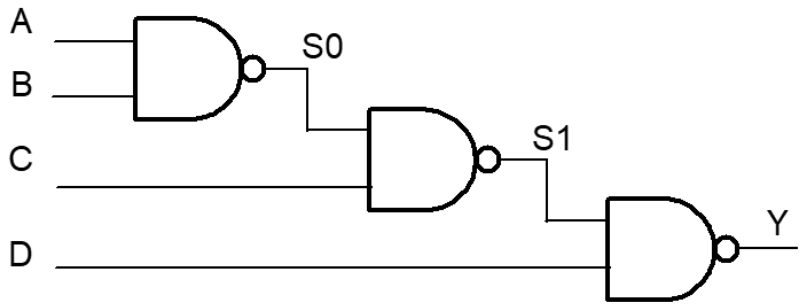
```
architecture behavior of NAND2 is
begin
process (A,B)
begin
  if (A='1') and (B='1') then
     C<='0';
  else C<='1';
  end if ;
end process;
end behavior;
```

A behavior description is just like the software programming

Signal

- SIGNAL data objects represent the logic signals, or wires in a circuit.
- Can be declared
  - in the declarative section of an architecture
  - in the declarative section of a package

```
....
signal S0,S1 : std_logic;
begin
   S0 <= A nand B;
   S1 <= S0 nand C;
   Y <= S1 nand D;
end a;
```

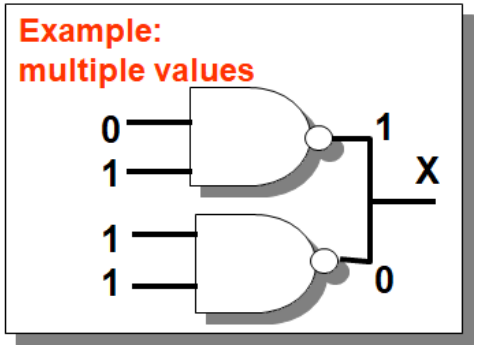


➢ But physical wires cannot be modelled accurately using only ‘0’ and ‘1’. Additional values are needed to represent the state of a wire.

Multi-Valued Logic Representation

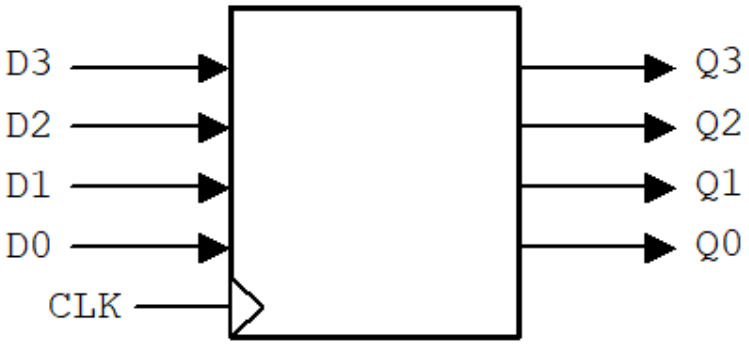
- IEEE 1164 package provides 9 logic-value representations.

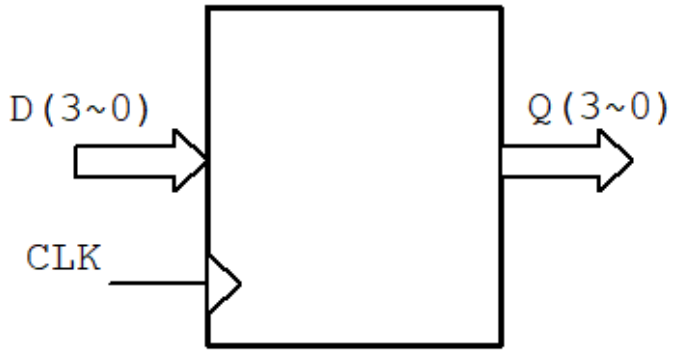
Uninitialized	'U'	Weak 1	'H'
Don't Care	'-'	Weak 0	'L'
Forcing 1	'1'	Weak Unknown	'W'
Forcing 0	'0'	High Impedance	'Z'
Forcing Unknown	'X'		



- Signals can be driven with multiple values.
- Signals can be resolved when conflicting values have been driven

Aggregate Signals






```
Signal D3,D2,D1,D0：Std_Logic;
Signal Q3,Q2,Q1,Q0：Std_Logic;
……
   Q3<=D3;
   Q2<=D2;
   Q1<=D1;
   Q0<=D0;
```

```
Signal D：Std_Logic_Vector(3 downto 0);
Signal Q：Std_Logic_Vector(3 downto 0);
……
     Q<=D;
```

Libraries and packages

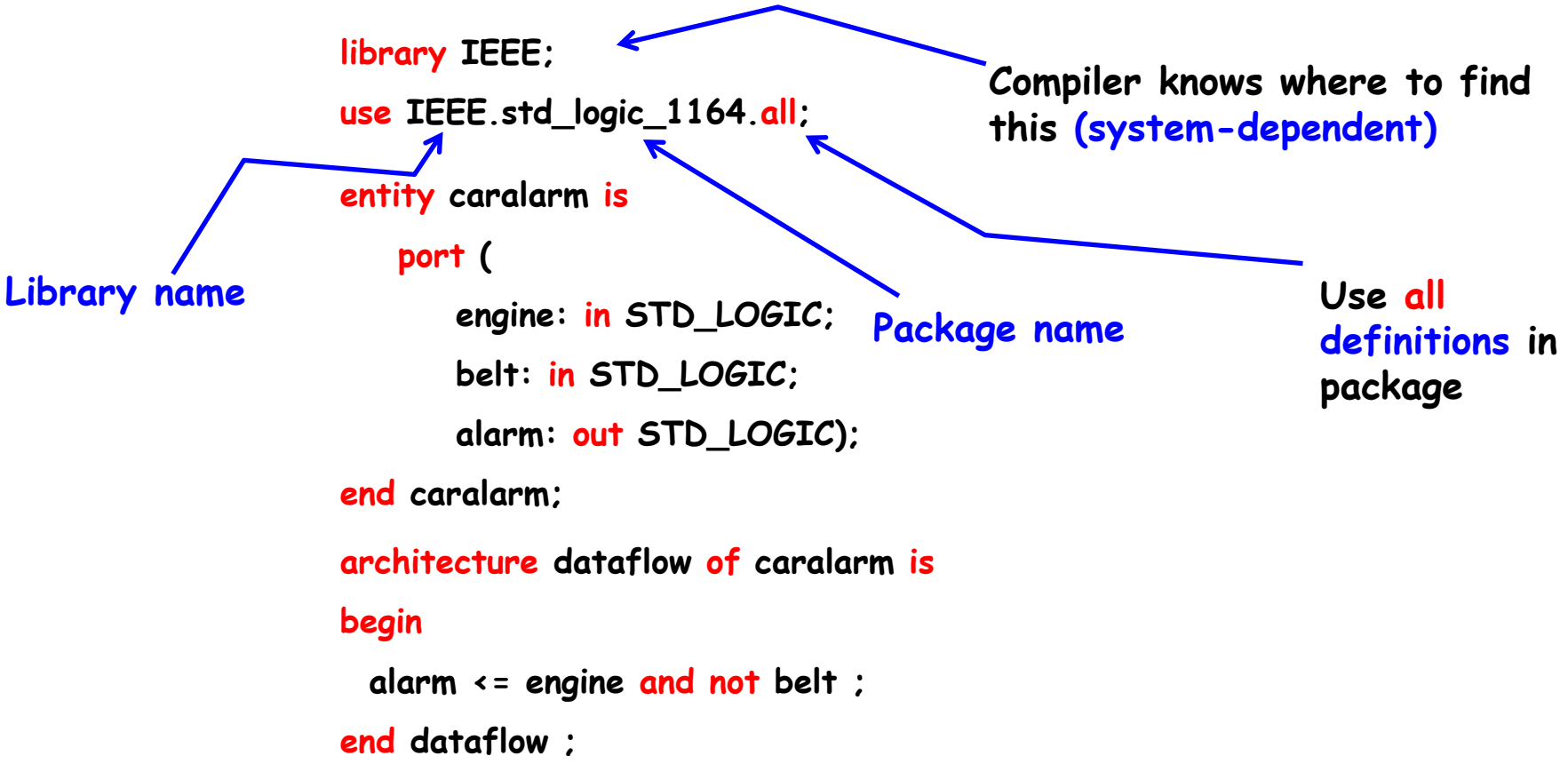
- Libraries provide a set of packages, components, and functions that simplify the task of designing hardware
- Packages provide a collection of related data types and subprograms
- The following is an example of the use of the `ieee` library and its `std_logic_1164` package:

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;
```

A VHDL Example

- Commonly used IEEE-1164 types:
  - STD_LOGIC (one bit)
  - STD_LOGIC_VECTOR(*range*) (multi-bit vector)



VHDL Predefined Operators

- Boolean operations
  - and AND
  - or OR
  - nand NAND
  - nor NOR
  - xor Exclusive OR
  - xnor Exclusive NOR
  - not Complement
- Logic operations
  - = equal
  - /= not equal
  - < less than
  - <= less than or equal
  - > greater than
  - >= greater than or equal
- numerical operations
  - + addition
  - - subtraction
  - * multiplication
  - / division
  - **mod** modulo division
  - **rem** modulo remainder
  - abs absolute value
  - ** exponentiation

Assignment Statements

```
SIGNAL a, b, c              : std_logic;
SIGNAL avec, bvec, cvec     : std_logic_vector(7 DOWNTO 0);

-- Concurrent Signal Assignment Statements
-- NOTE: Both a and avec are produced concurrently
a         <= b AND c;
avec      <= bvec OR cvec;

-- Alternatively, signals may be assigned constants
a         <= '0';
b         <= '1';
c         <= 'Z';
avec      <= "00111010";              -- Assigns 0x3A to avec
bvec      <= X"3A";                   -- Assigns 0x3A to bvec
cvec      <= X"3" & X"A";             -- Assigns 0x3A to cvec
```

Synthesis v.s. Simulation

- All synthesizable designs can be simulated
- Not all simulation designs can be synthesized

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY simple_buffer IS
      PORT (   din      : IN     std_logic;
               dout     : OUT    std_logic );
END simple_buffer;

ARCHITECTURE behavioural1 OF simple_buffer IS
BEGIN
      dout <= din AFTER 10 ns;
END behavioural1;
```

- The input din is assigned to dout after 10 ns
  - Can this represent a real-world system? YES
  - Can this be implemented in a device? PERHAPS
  - Can this be implemented in all devices? NO
- This architecture can be simulated but not synthesized