



UNITED STATES  
AIR FORCE  
ACADEMY



UNITED STATES  
AIR FORCE  
ACADEMY

# ECE 383 - Embedded Computer Systems II Lecture 2 - Digital System, Hierarchical Design, and testbench

→ sandbox\_tb.vhdl



# HW#1 VHDL Code

```
entity hw1 is
    port(    I3, I2, I1, I0    :    in std_logic;
           01, 00            :    out std_logic);
end hw1;

architecture structure of hw1 is

begin
    00 <= I3 or (not I2 and I1);
    01 <= I3 or I2;
end structure;
```

Is this Behavioral or Structural code?



UNITED STATES  
AIR FORCE  
ACADEMY

# Lesson Outline

---

- 1. Overview of HDLs**
- 2. Basic VHDL concepts by example**
- 3. Testbenches**



UNITED STATES  
AIR FORCE  
ACADEMY

---

# Overview of HDLs

# Programming Language

---

Python?

- Can we use C or Java as an HDL?

need

•  
•



# HDL vs Traditional PL

---


- **Traditional PL**
  - Modeled after a sequential process
  - Operations performed in a sequential order
  - Help human's thinking process to develop an algorithm step-by-step
  - Resemble the operation of a basic computer model
- **HDL**
  - Characteristics of digital hardware
    - Connections of parts
    - Concurrent operations
    - Concept of propagation delay and timing
  - Characteristics cannot be captured by traditional PLs
  - Require new languages: HDL



UNITED STATES  
AIR FORCE  
ACADEMY

# Modern Use of HDLs

---

- Formal Documentation
- Input to a simulator
- Input to a synthesizer Handwritten red lines pointing from the text 'Input to a synthesizer' to the words 'FPGA' and 'ASIC'.

RTL

# Characteristics of an HDL

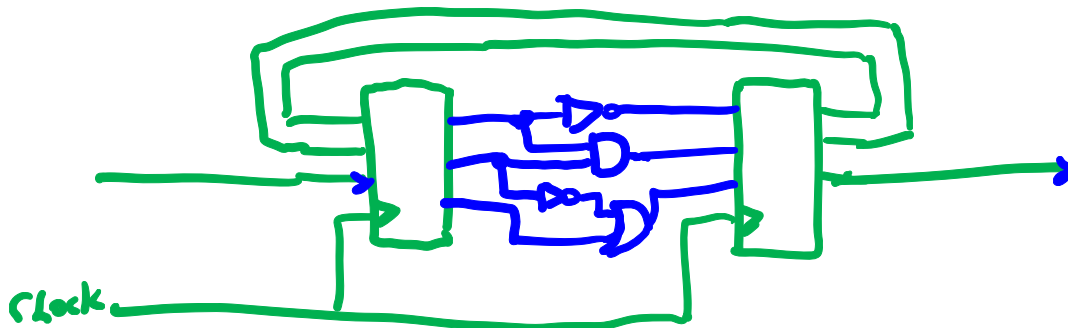
- Capture characteristics of a digital circuit:

Part

- ~~Entity~~ - basic building block (e.g. 7400 chips) **BBB**
- Connectivity - Connection of parts (e.g. wires)
- Concurrency - parallel operations
- Timing - schedule / order of multiple operations

- Must be able to describe a circuit in

- Gate level and RT level
- Structural view and behavioral view (*not* physical)







# Industry-Standard HDLs

- VHDL *← Dr York*
  - DoD initiative in 1980s
  - Transferred to IEEE to standardize
  - First released in 1987
  - Similar to Ada
  - Heavily used in FPGA industry
  - New versions: 1993, 2001, 2008
- Verilog *← Dr Bank*
  - Developed by industry
  - Released in early 1980s
  - Similar to C
  - Heavily used in ASIC industry
  - New versions: 1995, 2001, 2005
  - SystemVerilog is a superset of Verilog 2005



UNITED STATES  
AIR FORCE  
ACADEMY

---

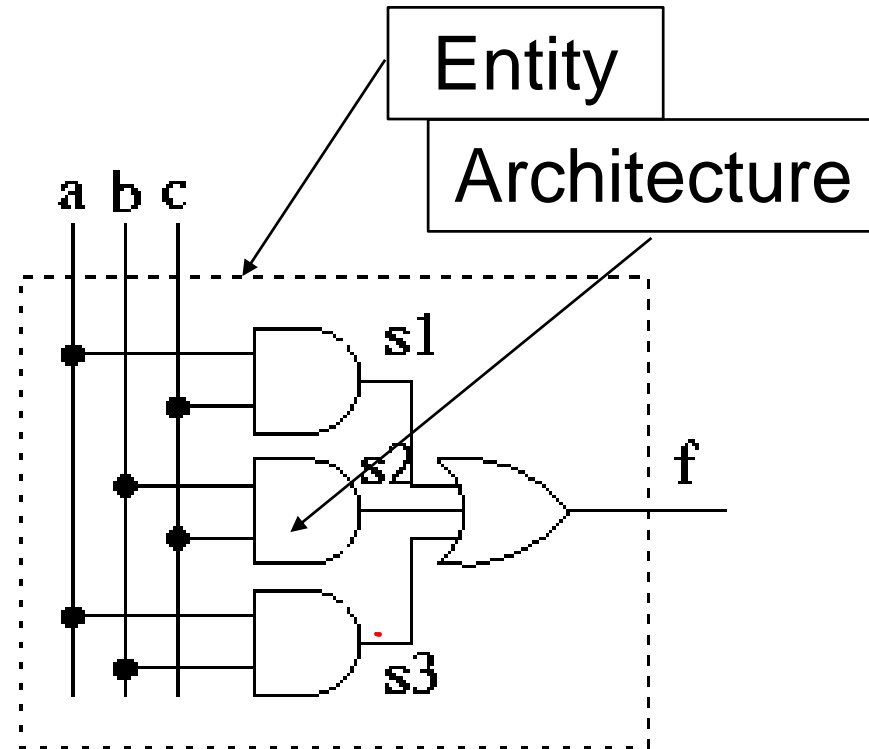
# Basic VHDL Concepts By Example

# Structural Description

## ■ Structural Description from Lesson 1

### Truth Table

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



- **Entity declaration**
  - i/o ports (“outline” of the circuit)
- **Architecture body**
  - Signal declaration – “wires”
  - Each concurrent statement
  - Can be thought as a circuit part
  - Contains timing information
  - Arch body can be thought as a “collection of parts”
- **What’s the difference between this and a C program?**

# Structural Description

---

- In structural view, a circuit is constructed by smaller parts.
- Structural description specifies the types of parts and connections.
- Essentially a textual description of a schematic
- Done by using “component” in VHDL
  - *First declared (make known)*
  - *Then instantiated (used)*



# Structural Description – Component Declaration

```
library IEEE; -- These lines are similar to a #include in C
use IEEE.std_logic_1164.all;
library unisim; -- Use these libraries if you are using primitive components
use unisim.vcomponents.all;
```

← AND2  
OR3

entity majority is

```
    port(
        a, b, c: in std_logic;
        f:      out std_logic);
end majority;
```

architecture structure of majority is

component AND2

```
    port ( i0, i1 : in std_logic;
           o      : out std_logic);
end component;
```

component OR3

```
    port ( i0, i1, i2 : in std_logic;
           o          : out std_logic);
end component;
```

```
signal s1, s2, s3: std_logic; -- wires which begin and end in the component
```

① declare

# Structural Description – Component Instantiation

Shorthand notation?

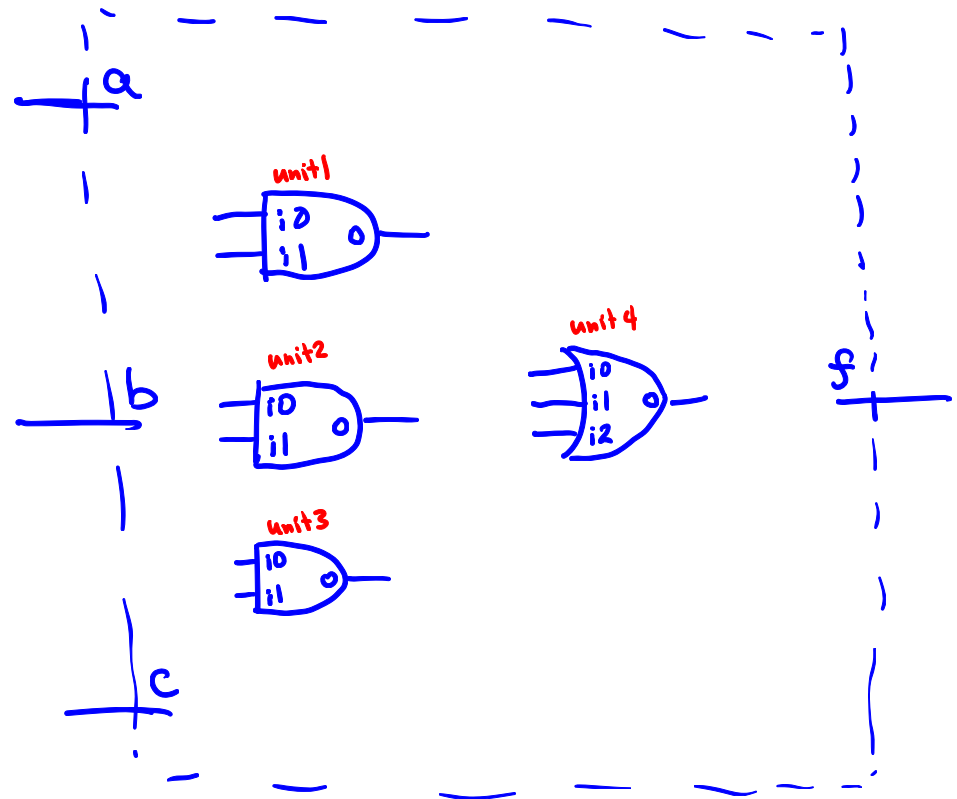
begin

```
unit1:    AND2
port map ( -- s1 <= a and b;
           i0 => a,
           i1 => b,
           o => s1);

unit2:    AND2
port map ( -- s2 <= b and c;
           i0 => b,
           i1 => c,
           o => s2);

unit3:    AND2
port map ( -- s3 <= a and c;
           i0 => a,
           i1 => c,
           o => s3);

unit4:    OR3
port map ( -- f <= s1 or s2 or s3;
           i0 => s1,
           i1 => s2,
           i2 => s3,
           o => f);
```



end structure;

# Lesson 1 example code

Is this Behavioral or Structural code?

```
entity majority is
    port( a, b, c: in std_logic;
          f: out std_logic);
end majority;

?
architecture structure of majority is
    signal s1, s2, s3: std_logic; -- wires which begin and end in the component

begin
    s1 <= a and b; -- These statements are called
    s2 <= b and c; -- concurrent signal assignments. CSA
    s3 <= a and c; -- They all happen at the same time
    f <= s1 or s2 or s3; -- unlike a regular programming lang.
end structure;
```





3<sup>rd</sup> way?

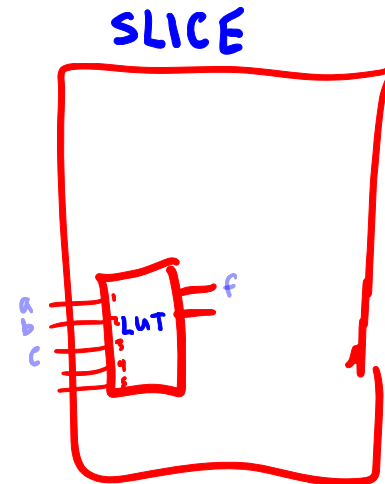
- A behavioral description of a component describes what the circuit does rather than how it is done.

```

library IEEE;
use IEEE.std_logic_1164.all;
entity majority is
    port(
        a, b, c:    in std_logic;
        f:          out std_logic);
end majority;
architecture Behavioral of majority is
begin
    f <=
        '0' when a='0' and b='0' and c='0' else
        '0' when a='0' and b='0' and c='1' else
        '0' when a='0' and b='1' and c='0' else
        '1' when a='0' and b='1' and c='1' else
        '0' when a='1' and b='0' and c='0' else
        '1' when a='1' and b='0' and c='1' else
        '1' when a='1' and b='1' and c='0' else
        '1';
    -- essentially an enumeration of a truth table
end Behavioral;

```

-- These lines are similar to a #include in C





4<sup>th</sup> way

### ■ Concatenation operator helps make code more readable

```
library IEEE;           -- These lines are similar to a #include in C
use IEEE.std_logic_1164.all;
entity majority is
    port(
        a, b, c:    in std_logic;
        f:          out std_logic);
end majority;
architecture Behavioral of majority2 is
    signal temp: std_logic_vector(2 downto 0);
begin
    temp <= a & b & c;
    f <=
        '0' when temp = "000" else
        '0' when temp = "001" else
        '0' when temp = "010" else
        '1' when temp = "011" else
        '0' when temp = "100" else
        '1' when temp = "101" else
        '1' when temp = "110" else
        '1';
end Behavioral;
```

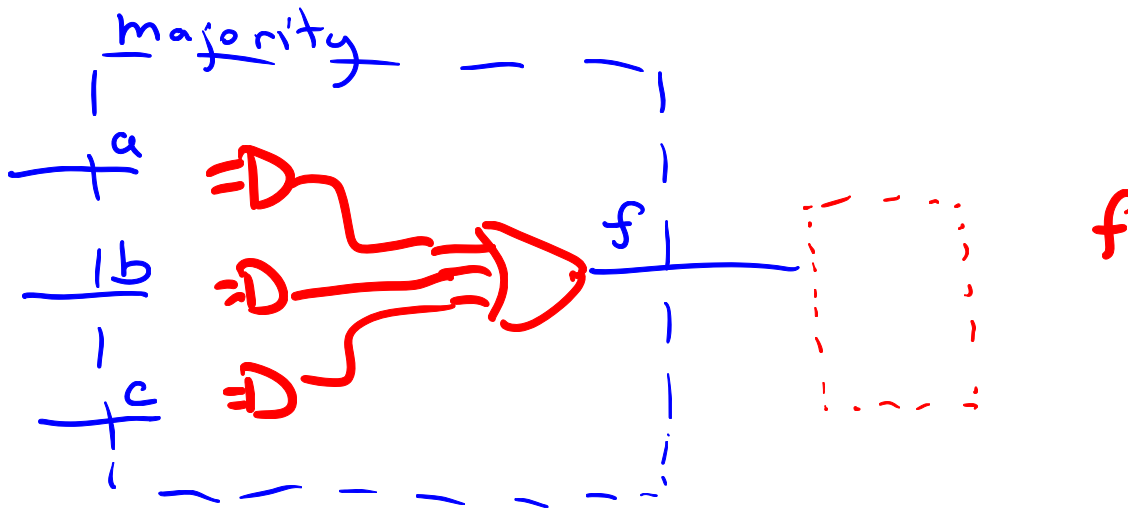
Concatenation Operator

Double quotes for std\_logic\_vectors

Did we need k-map simplification?

# Combinational vs Sequential

- What is the difference between combinational and sequential? ←





- In VHDL they are called literals (not a constant)

hexDigit : std\_logic\_vector (3 downto 0);

...hexDigit = x"D" else

...hexDigit = "0101" else

...hexDigit = d"12" else

HW#2?

C code?

VHDL

"=" – used to compare

"<=" – used to assign



UNITED STATES  
AIR FORCE  
ACADEMY

---

# Testbenches



# Testbench – Component Declaration and Instantiation

```
ENTITY majority_tb IS
END majority_tb;
```

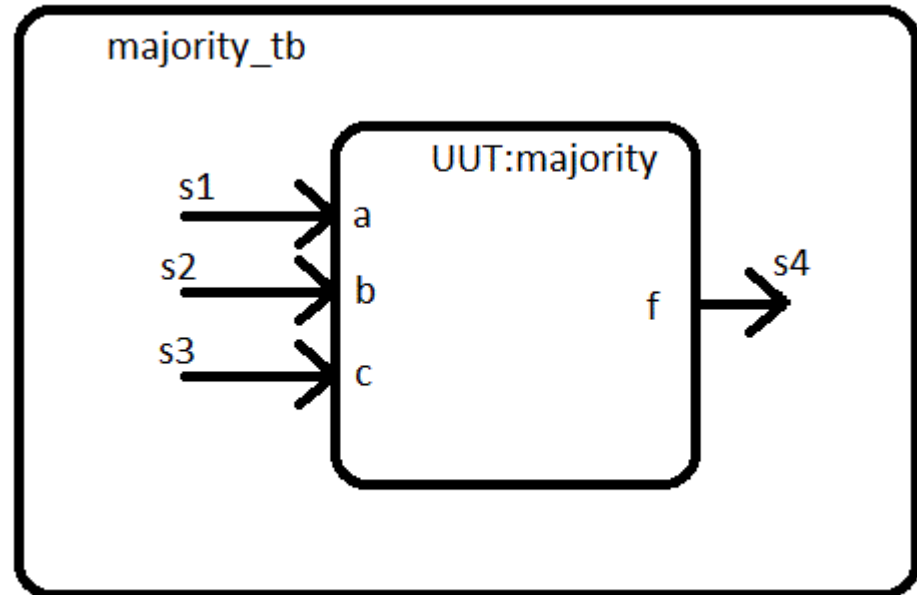
← Nothing in here? ? uut: majority PORT MAP (s1,s2,s3,s4):

```
ARCHITECTURE behavior OF majority_tb IS
```

```
COMPONENT majority
PORT( a : IN std_logic;
      b : IN std_logic;
      c : IN std_logic;
      f : OUT std_logic);
END COMPONENT;
signal s1, s2, s3, s4: std_logic;
begin
  uut: majority PORT MAP (
    a => s1,
    b => s2,
    c => s3,
    f => s4);
end
```

declare

instantiate





# Testbench – Component Declaration and Instantiation

## ■ Test Vector Setup:

1. CONSTANT TEST\_ELEMENTS:integer:=8;
2. SUBTYPE INPUT is std\_logic\_vector(2 downto 0);
3. TYPE TEST\_INPUT\_VECTOR is array (1 to TEST\_ELEMENTS) of INPUT;
4. SIGNAL TEST\_INPUT: TEST\_INPUT\_VECTOR := ("000", "001", "010", "011", "100", "101", "110", "111");

SEE TEST\_OUTPUT

## ■ Loop to apply the 8 test input vectors to majority circuit

1. for i in 1 to TEST\_ELEMENTS loop
2. testVector <= test\_input(i);
3. wait for 1 us;
4. assert f = test\_output(i)
5. report "Error in majority circuit for input " & integer'image(i)
6. severity failure; ? warning?
7. end loop;

# Simulation Experimentation

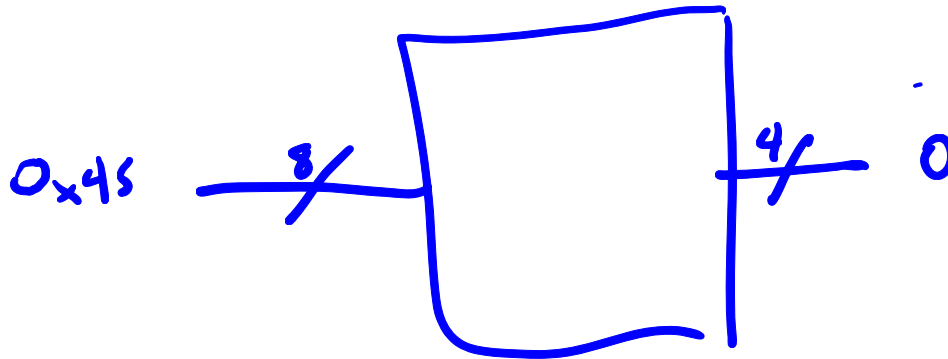
---

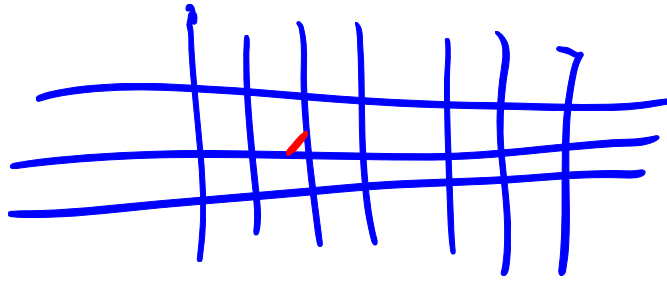
- How to add and remove waveforms to the waveform view.
- How to change the radix of a vector waveform
- How to change the colors of the waveforms.
- How to transcend the design hierarchy.
- How to observe signal values in design hierarchy.
- How to observe signals values in the VHDL code.
- How to save a load a simulation waveform wcfg file.





- Scan codes?
- Code, testbench, simulation plot (snip)





2. In the following problem, you will build vhdl function called **Scancode decoder** which processes keyboard scancodes. When you press a key on a keyboard, the keyboard sends an 8-bit code to the computer called a PS2 scancode. Each key has its own scancode listed below. The relationship between the keys and their scancode is not based on ASCII nor any other discernible pattern.

Keyboard Key	Scancode (in hex)
0	0x45
1	0x16
2	0x1E
3	0x26
4	0x25
5	0x2E
6	0x36
7	0x3D
8	0x3E
9	0x46



Build a function which converts an 8-bit scancode for the digits 0-9 into a 4-bit hexadecimal values.

<b>Nomenclature:</b>	<b>Scancode decoder</b>
<b>Data Input:</b>	D = std_logic_vector(7 downto 0);
<b>Data Output:</b>	H = std_logic_vector(3 downto );
<b>Control:</b>	none
<b>Status:</b>	none
<b>Behavior:</b>	Converts the scancode d, representing a the key of a decimal digit, into its 4-bit value. For example, if D = 25_16, the scancode for the character "4", then the converter should output H = 0100_2. Assume that the inputs are always legal hexadecimal scancodes.

Use the when statement syntax to describe the output in terms of the input.

see slide