RECRLab
Reconfigurable Computing Research Laboratory

DIGITAL LOGIC DESIGN

VHDL Coding for FPGAs
Unit 1

- ✓ DESIGN FLOW
- ✓ DATA TYPES
- ✓ LOGIC GATES IN VHDL
- ✓ TESTBENCH GENERATION
- ✓ XILINX: I/O ASSIGNMENT
- ✓ USE OF **std_logic_vector**

Daniel Llamocca

OAKLAND UNIVERSITY

✓ *DESIGN FLOW*

- **Design Entry**: The circuit is specified using a Hardware Description Language (e.g., VHDL, Verilog).
- **Functional Simulation**: Also called behavioral simulation. Here, we will only verify the logical operation of the circuit. Stimuli is provided to the logic circuit, so we can verify the outputs behave as we expect.
- **Physical Mapping**: The inputs/outputs of our digital circuit are mapped to specific pins of the FPGA.
- **Timing Simulation**: It simulates the circuit considering its timing behavior (delays between inputs and outputs)
- **Implementation**: A configuration file (‘bitstream’ file) is generated and then downloaded onto the FPGA configuration memory.

✓ *DESIGN FLOW (Vivado Software)*

- **Synthesis**: In this step, the VHDL code is examined for syntax errors and warnings. While your code should be free of syntax errors, watch out for warnings and critical warnings. After this, we do behavioral simulation.
- **Simulate Behavioral Model:** We need to write a VHDL file called 'testbench' where we specify the stimuli to the logic circuit.
- **Implement Design** (Translate + Map + Place & Route)
- **Generate Programming File**: Here, a configuration file or bitstream (.bit) is generated. This file will configure the FPGA so that the logic circuit is implemented on it.
- **Configure Target Device** (Programming): The bitstream file is written onto the FPGA configuration memory so that a digital circuit is materialized. At this stage, we can verify whether the hardware is actually working.

✓ LOGIC DATA TYPES

- **Type:** This is how data (digital signals or even abstract constructs like variables) is specified in VHDL. Though different standards are available, a common one is the IEEE *std_logic_1164,* that allows for these basic types:
    - std_logic, std_logic_vector, std_logic_2d
        - The std_logic type defines nine (9) possible states for a 1-bit signal:
            - ‘U’ : Uninitialized
            - ‘X’ : Forced Unknown
            - ‘0’ : Zero
            - ‘1’ : One
            - ‘Z’ : High impedance
            - ‘W’ : Weak unknown
            - ‘L’ : Weak Zero
            - ‘H’ : Weak One
            - ‘-’ : Don’t care
- Other data types:
    - array (group of signals, or group of groups)
    - integer, user-defined

✓ *LOGIC DATA TYPES:*

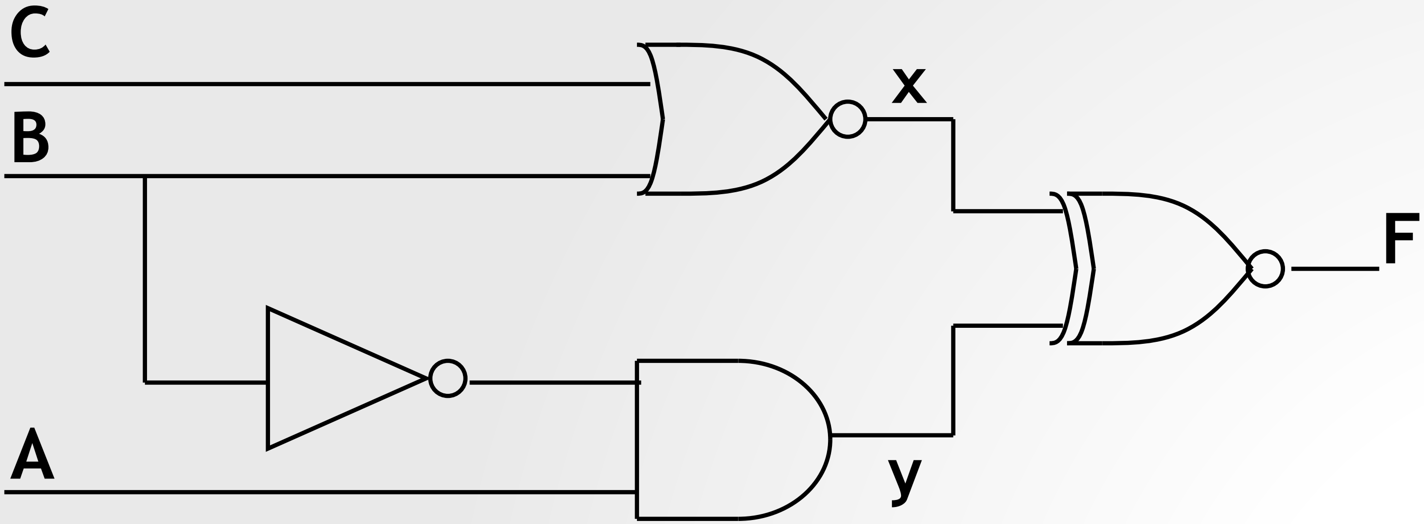
- A digital circuit includes internal signals and external signals (also called I/Os). In VHDL, I/O specification is called 'mode'.
- **Mode:** Physical characteristics of inputs/outputs of a logic circuit. The following modes are available in VHDL:
  - ✓ `IN` : Input port of a circuit
  - ✓ `OUT` : Output port of a circuit.

    VHDL syntax: in VHDL, it is not possible to feedback an output port to the input of the circuit.
  - ✓ `INOUT` : Bidirectional port (it can be an input or output at different times). It is very useful when implementing bidirectional buses.
  - ✓ `BUFFER` : Output port. In VHDL, if we define an output signal as `BUFFER`, the signal can be fed back as an input of the circuit. However, vendor support is inconsistent.

✓ *LOGIC GATES IN VHDL*

- **VHDL** allows for the specification of Boolean functions based on the following gates: **AND, OR, NOT, XOR, NAND, and NOR.**

- **EXAMPLE:** Write the VHDL code to implement the following circuit whose output is ‘F’:



- **Note:** In VHDL, A B C are inputs (`IN`), F is an output (`OUT`), and x and y are internal signals (`signal`).

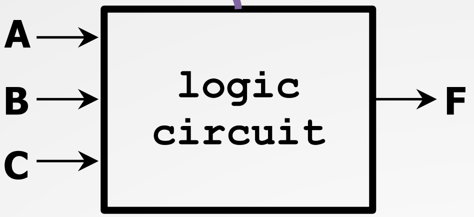
✓ LOGIC GATES IN VHDL

- **EXAMPLE:** VHDL code: *example.vhd*

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

```
entity example is
  port ( A, B, C: in std_logic;
             F: out std_logic);
end example;
```

I/Os are specified here



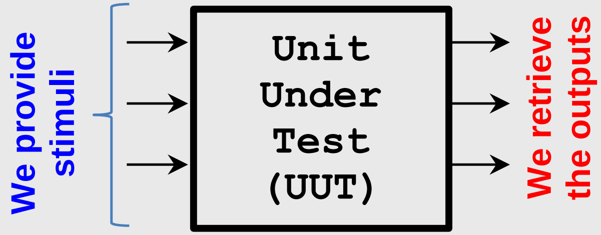
```
architecture struct of example is
    signal x,y: std_logic;
begin
    x <= C nor B;
    y <= A and not(B);
    F <= not(x xor y);
end struct;
```

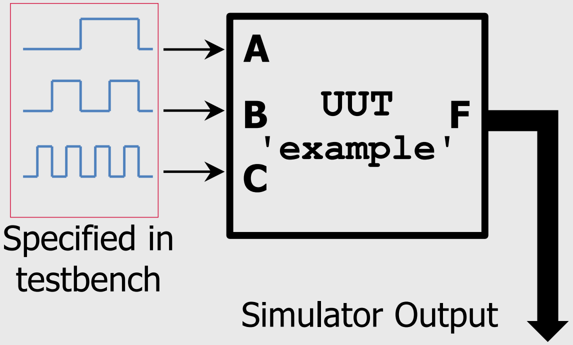
Internal Description of the logic circuit is specified here

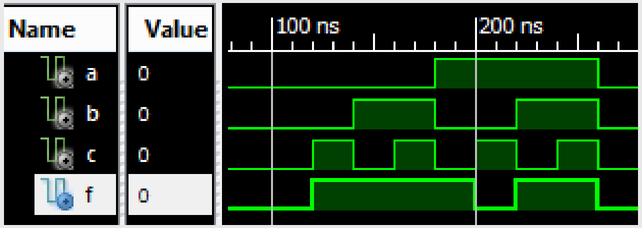
✓ TESTBENCH GENERATION

- EXAMPLE:

tb_example.vhd







Name	Value
a	0
b	0
c	0
f	0

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

entity tb_example is
end tb_example;

architecture behavior of tb_example is
  component example
     port ( A,B,C: in std_logic;
                F: out std_logic);
  end component;
  -- Inputs
  signal A: std_logic := '0'; -- default value
  signal B: std_logic := '0'; -- default value
  signal C: std_logic := '0'; -- default value
  -- Outputs
  signal f: std_logic;
begin
  uut: example port map (A=>A,B=>B,C=>C,F=>F);
  stim_proc: process -- Stimulus process
  begin
     wait for 100 ns -- reset state
     -- Stimuli:
     A <='0';B <='0';C <='0'; wait for 20 ns;
     A <='1';B <='0';C <='1'; wait for 20 ns;
     wait;
  end process;
end;
```

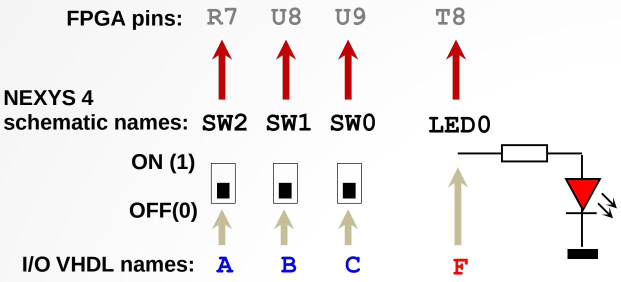
✓ *XILINX ISE: I/O ASSIGNMENT*

(obsolete)

UCF file: We need to map the I/Os of our logic circuit to physical FPGA pins. In a board (e.g., Nexys-4), these FPGA pins are connected to specific components: LEDs, switches, buttons, etc.

- **EXAMPLE:** The inputs A, B, C are assigned to switches. The output F is assigned to an LED (ON if F is '1'). The **Nexys-4 Artix-7 FPGA Board** is used.
- ISE 14.7: I/O standard must be specified for every pin
- UCF file: *example.ucf*

```
# Inputs
NET "A" LOC="U9" | IOSTANDARD="LVCMOS33"; #SW0
NET "B" LOC="U8" | IOSTANDARD="LVCMOS33"; #SW1
NET "C" LOC="R7" | IOSTANDARD="LVCMOS33"; #SW2

# Outputs
NET "F" LOC="T8" | IOSTANDARD="LVCMOS33"; #LED0
```



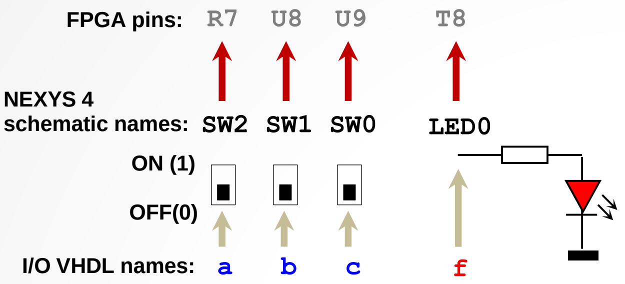
➢ **example.zip**: example.vhd, tb_example.vhd, example.ucf

✓ *VIVADO: I/O ASSIGNMENT*

XDC file: Here, we map the I/Os of our circuit to physical FPGA pins. In a board (e.g. Nexys-4), many FPGA pins are wired to specific components (LEDs, switches, buttons, etc.).

- **Example: Nexys-4 Artix-7 FPGA Board:**
  To connect the inputs a, b, c to SW2 SW1 SW0 and the output f to LED0, we assign a, b, c, f to the corresponding FPGA pins (this mapping information is provided by the board's manufacturer).
- Vivado: The I/O standard and pin name must be specified for every I/O port. Pin names are ***case-sensitive*** and must match the port names as specified in the VHDL entity.
  - *XDC file: example.xdc*

```
# Inputs
set_property PACKAGE_PIN U9 [get_ports {a}]
   set_property IOSTANDARD LVCMOS33 [get_ports {a}]
set_property PACKAGE_PIN U8 [get_ports {b}]
   set_property IOSTANDARD LVCMOS33 [get_ports {b}]
set_property PACKAGE_PIN R7 [get_ports {c}]
   set_property IOSTANDARD LVCMOS33 [get_ports {c}]
# Outputs
set_property PACKAGE_PIN T8 [get_ports {f}]
   set_property IOSTANDARD LVCMOS33 [get_ports {f}]
```

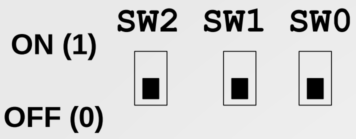


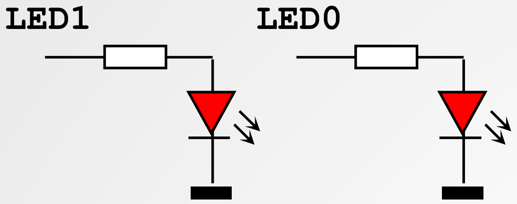
➢ **example.zip**: example.vhd, tb_example.vhd, example.xdc

✓ *EXAMPLE: Light Control*

- There are three available switches. We want LED1 ON when only one of the switches is in the ON position. And we want LED0 ON only when the three switches are in the ON position.

SW2	SW1	SW0	LED1	LED0
0	0	0	0	1
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1
1	1	1	0	0





$$\boxed{?}$$

$$LED1 = \overline{SW2}\,\overline{SW1}SW0 + \overline{SW2}SW1\overline{SW0} + SW2\overline{SW1}\,\overline{SW0}$$

$$\rightarrow LED1 = \overline{SW2}(SW1 \quad SW0) + SW2\overline{SW1}\,\overline{SW0}$$

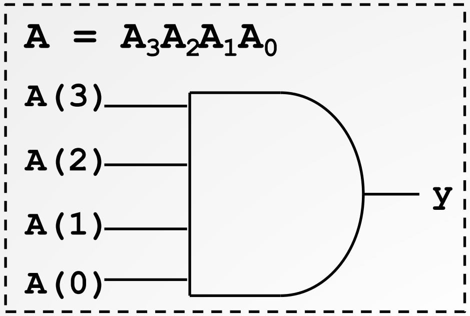
$$LED0 = \overline{SW2}\ +\ \overline{SW1}\ +\ \overline{SW0}$$

➢ **light_ctrl.zip:** light_ctrl.vhd, tb_light_ctrl.vhd, light_ctrl.ucf

✓ USE of std_logic_vector

- This type defines an array of bits.
- Here, we use the **std_logic_vector** type for an input signal.

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

entity test is
  port ( A: in std_logic_vector (3 downto 0);
         -- A: |A3|A2|A1|A0|
         y: out std_logic);
end test;

architecture struct of test is

begin
    -- The circuit represents an AND gate
    -- with 4 inputs: A(3), A(2), A(1), A(0)
    y <= A(3) and A(2) and A(1) and A(0);

end struct;
```



✓ USE of std_logic_vector

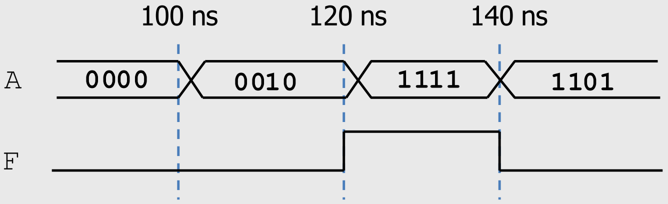
- Here, we use this type in a testbench for ‘test’.

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

entity tb_test is
end tb_test;

architecture behavior of tb_test is
  component test
     port ( A: in std_logic_vector(3 downto 0);
            F: out std_logic);
  end component;
  -- Inputs
  signal A: std_logic_vector(3 downto 0):= "0000";
  -- Outputs
  signal f: std_logic;
begin
  uut: test port map (A=>A,F=>F);
  stim_proc: process -- Stimulus process
  begin
     wait for 100 ns -- reset state
     -- Stimuli:
     A <= "0010"; wait for 20 ns;
     A <=   x"F"; wait for 20 ns;    -- A <="1111"
     A <= "11"&"01"; wait for 20 ns; -- A <="1101"
     wait;
  end process;
end;
```

* The values for a signal of type **std_logic_vector** can be specified in different ways (see this testbench).

* **A <=“0010”**: It is equivalent to:

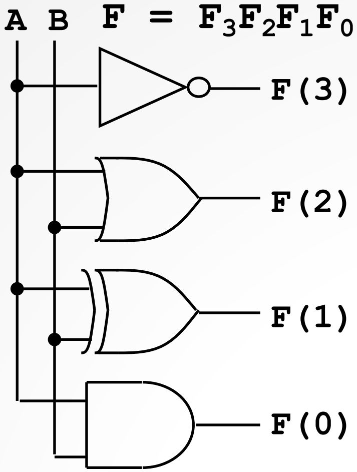
A(3)<=‘0’; A(2)<=‘0’;
A(1)<=‘1’; A(0)<=‘0’;



✓ USE of std_logic_vector

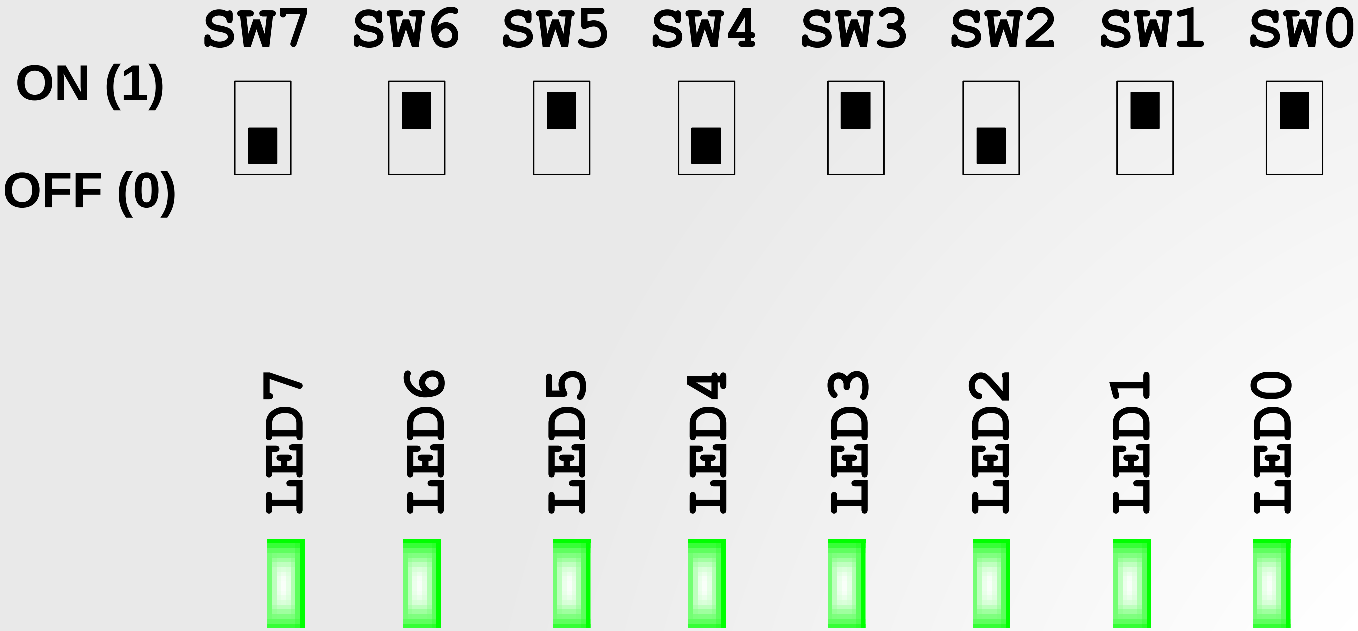
- In the example, we use the *std_logic_vector* type for an output signal.

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

entity tst is
  port ( A,B: in std_logic;
         F: out std_logic_vector (3 downto 0);
         -- F: |F3|F2|F1|F0
end tst;

architecture struct of tst is

begin
    F(0) <= A and B; F(1) <= A xor B;
    F(2) <= A or B; F(3) <= not(A);

end struct;
```



✓ *EXAMPLE: Security Combination*

- A lock is opened only when a certain combination of switches exist: Switches: `01101011`
- The lock will be represented by 8 LEDs. Open Lock ≡ All LEDS ON.



➢ **`sec_comb.zip`:** `sec_comb.vhd, tb_sec_comb.vhd, sec_comb.ucf`