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

✓ *DESIGN FLOW*

- **Design Entry**: We specify the logic circuit 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 logic 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

✓ *DESIGN FLOW (ISE Software)*

- **Synthesis**: It makes sure that the VHDL file is syntax free. If ok, the logical circuit is ready for behavioral simulation.
- **Simulate Behavioral Model:** It requires the creation of a VHDL file in which we specify the stimuli to the logic circuit. This file is called 'testbench'.
- **Implement Design** (Translate + Map + Place & Route):
- **Generate Programming File**: Here, a configuration file (bitstream) is generated. This file will configure the FPGA so that the logic circuit is implemented on it.
- **Configure Target Device** (iMPACT software): Here, the configuration file (.bit file) previously created is downloaded onto the FPGA. At this stage, we can verify whether the actual hardware is actually working.

✓ LOGIC DATA TYPES

- **Type:** There are many ways to define data types in VHDL. A very common IEEE standard is *std_logic_1164*. The following types are available:
  - std_logic, std_logic_vector, std_logic_2d
    - The 'std_logic' type define nine (9) possible states:
      - `'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:
  - integer
  - array
  - User-defined

✓ *DATA TYPES:*

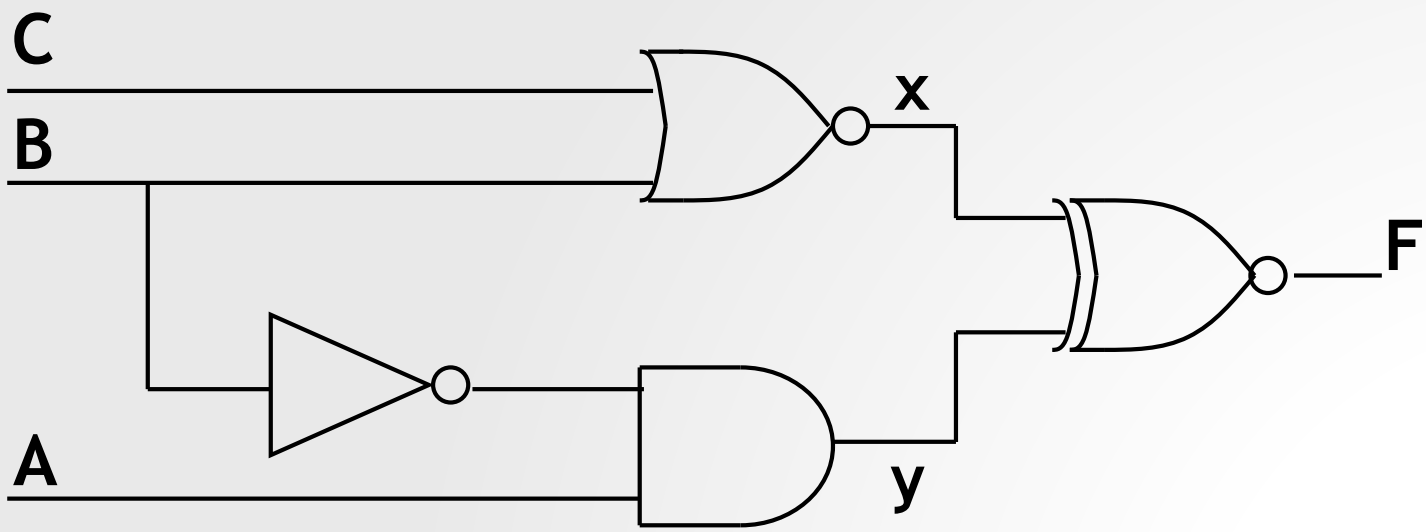
- **Mode:**
- Physical characteristics of inputs/outputs of a logic circuit. The following modes are available in VHDL:

  - IN : Input port of a circuit
  - OUT : Output port of a circuit
  - INOUT : Bidirectional port: It can be an input and an output at different times Very useful when working with bidirectional buses.
  - BUFFER : Output port of a circuit. It has the property that this output can be fed back to the circuit.

✓ *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 generate the output 'F' of the following circuit:

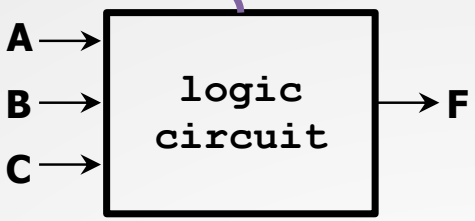


✓ LOGIC GATES IN VHDL

- **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;

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;
```

I/Os are specified here

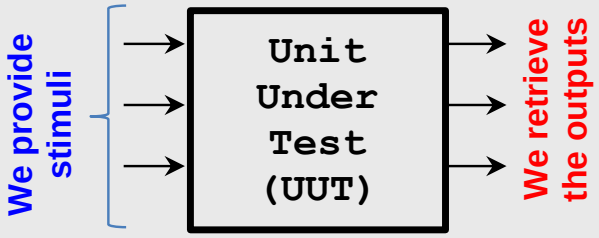


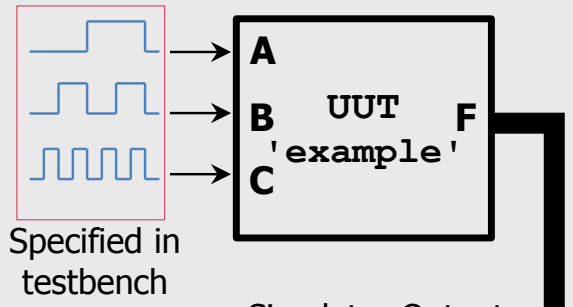
Internal Description of the logic circuit is specified here

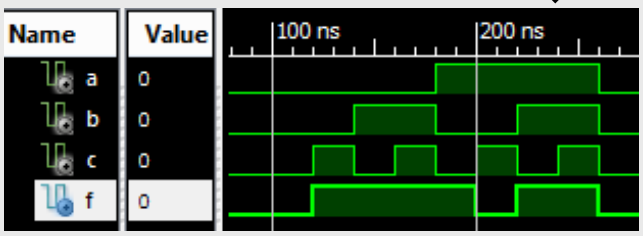
✓ TESTBENCH GENERATION

- **EXAMPLE:**

  tb_example.vhd







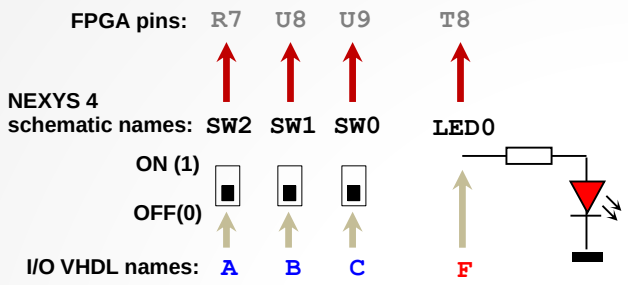

```
library ieee;
use ieee.std_logic_1164.all;

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';
  signal B: std_logic := '0';
  signal C: std_logic := '0';
  -- 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

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
```



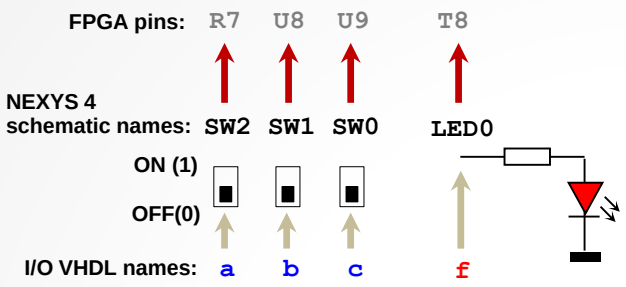
➢ **example.zip**: example.vhd, tb_example.vhd, example.ucf

✓ *VIVADO: I/O ASSIGNMENT*

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

- **Example: Nexys-4 Artix-7 FPGA Board:**
  The inputs a, b, c are assigned to switches. The output f is assigned to an LED (ON if F is '1').
- Vivado: The I/O standard and the pin must be specified for every pin. The pin names are ***case-sensitive***.
- *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}]
```

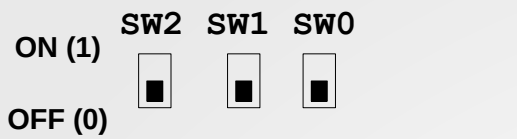


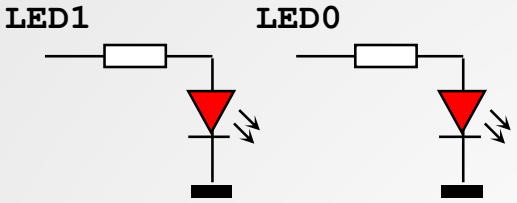
➢ **example.zip**: example.vhd, tb_example.vhd, example.xdc

✓ *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





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

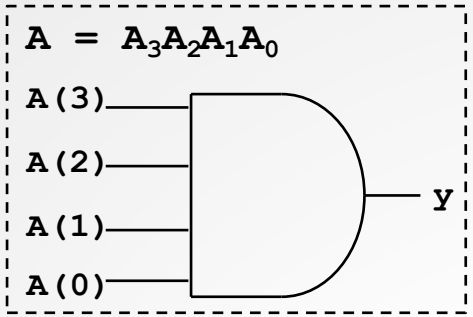
$$LED0 = \overline{SW2}\ +\ \overline{SW1}\ +\ \overline{SW0}$$

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

✓ std_logic_vector

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

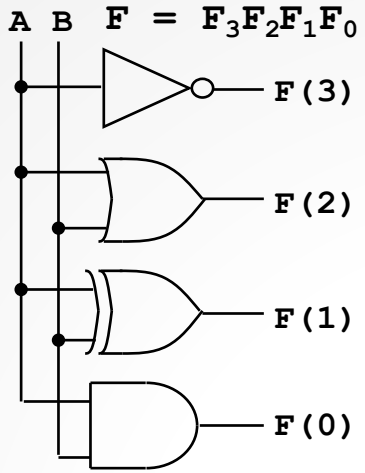
```
library ieee;
use ieee.std_logic_1164.all;

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;
```



✓ std_logic_vector

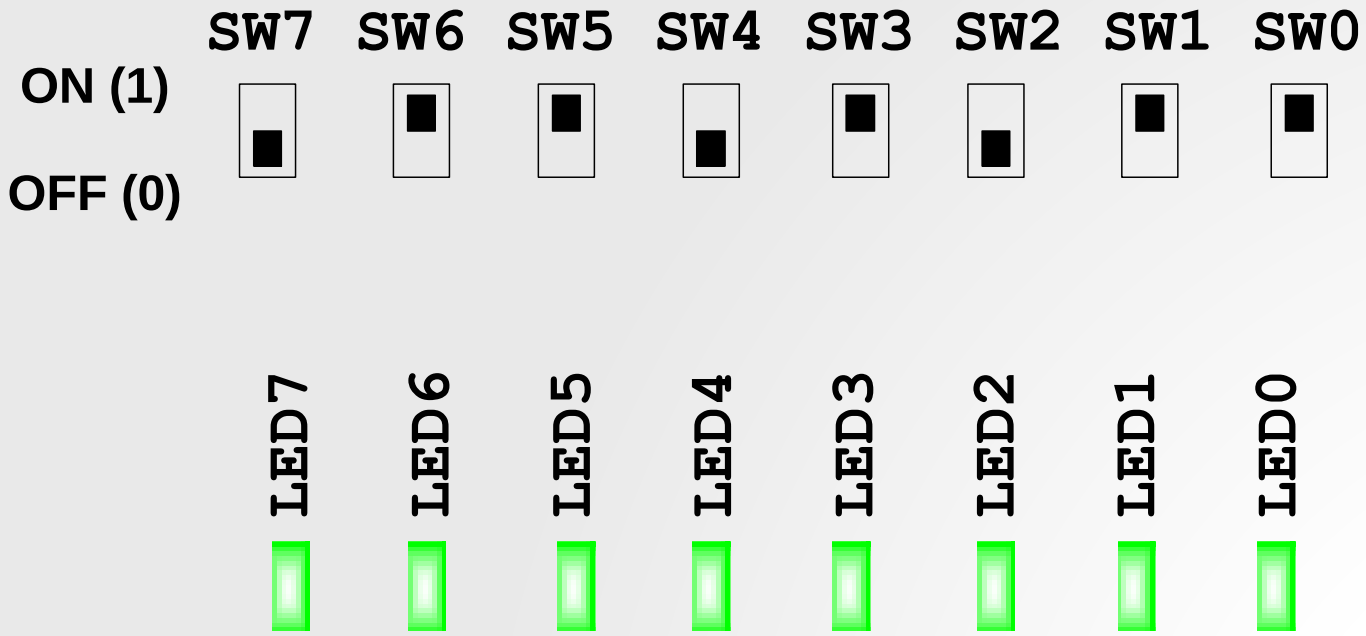
- In the example, we use the *std_logic_vector* type for an output signal.

```
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`