EEM412 Course
Introduction to IC Design – II

Instructor
Dr. İsmail Enis Ungan

VHDL

- Very High Speed Integrated Circuit Hardware Description Language
  - Product of the VHSIC program funded by DOD in the 1970s and 1980s.
  - The language was intended to be a standard means to document complex circuits so that a designer would understand an other designer's work easily.
  - Also, to be used as a modeling language that could be processed by software for simulation purposes.
  - Became a standard, IEEE 1076, in 1987. Updated and additional VHDL standard, IEEE 1164, was adopted. In 1996, IEEE 1076.3 became a VHDL synthesis standard.
  - Today, VHDL is an industry standard for description, modeling, and synthesis of digital circuits and systems. Only a subset of VHDL can be used to write codes which can be synthesized.

VHDL

- VHDL provides the capabilities of;
  - Power and Flexibility
    - Multiple levels of design description for controlling design implementation.
    - Supports design libraries and creation of reusable components.
    - Provides design hierarchies to create modular designs.
    - A single language for both design and simulation.
  - Device-Independent Design
    - Permits to create a design prior to device determination for implementation.
    - Permits multiple styles (behavioral, structural) of design description.
    - A design description may be implemented in many device architectures.
    - Allows the designer to concentrate on the design, instead of a device architecture for optimization of performance and utilization.

VHDL

- … VHDL provides the capabilities of;
  - Portability
    - A design description can be taken from one simulator/synthesizer/platform to an other.
    - VHDL design descriptions can be used in multiple projects.
  - Benchmarking
    - Device-independent design and portability allows to benchmark a design using different device architectures, and different synthesis tools. So that, one can evaluate the results and chooses the device that best fits the design requirements, or chooses the synthesis tool that gives the best quality.
  - Quick Time-to-Market and Low Cost
    - VHDL and programmable logic pair facilitate a speedy design process.
    - Designs are described quickly in VHDL and PLD eliminates NRE costs and facilitates quick design iterations by VHDL synthesis.

VHDL

- … VHDL provides the capabilities of;
  - ASIC Migration
    - ASIC vendor takes the design description in VHDL.
- Shortcomings
  - The logic implementations created by synthesis tools may be inefficient.
  - The quality of synthesis varies from tool to tool.

VHDL

- A digital system is modeled and designed in VHDL by the top-down design approach.
- A VHDL design consists of separate design units each of which is compiled and saved in library. The basic two design source units are
  - Entity
    - Describes the design's interface signals and represents the most basic building block in a design.
  - Architecture
    - Describes the design's behavior.

VHDL

- Entity
  - Defines a component name, inputs/outputs connections and related declarations.

```
entity abc4 is
   port ( a, b : in bit;
                 c : out bit );
end abc4;
```

- Architecture
  - Describes the behavior, interconnections, and components of a design entity. Includes concurrent statements and structure.

```
architecture arch2 of abc4 is
begin
   c <= (a xor b) and a;
end arch2;
```

VHDL

- Entity Port Modes
  - **In**: Data flows only into the entity.
  - **Out**: Data flows only from its source to the output port of the entity.
  - **Buffer**: Similar to mode “out” port, except that it allows internal feedback.
  - **Inout**: Used for bi-directional signals, which allows data to flow into or out of entity. It also allows internal feedback.

VHDL

- Example: 4-bit Equality Comparator

```
-- eqcomp4 is a four bit equality comparator
library ieee;
use ieee.std_logic_1164.all;
entity eqcomp4 is
        port ( a, b : in std_logic_vector(3 downto 0);
                equals : out std_logic ); -- equals is active “high”
end eqcomp4;
architecture dataflow of eqcomp4 is
begin
        equals <= '1' when (a=b) else '0';
end architecture dataflow;
```

VHDL

- Same Example (architecture part):

```
...
architecture behavioral of eqcomp4 is
begin
process (a,b)
begin
      if  a = b   then equals <= '1';
                  else equals <= '0';
      end if;
end process;
end architecture;
```

VHDL

- Same Example (architecture part):

```
...
architecture behavioral of eqcomp4 is
begin
label: process (a,b)
begin
        equals <= '0';
        if  a = b    then equals <= '1';
        end if;
end process label;
end architecture;
```

VHDL

- Same Example (architecture part):

```
...
architecture boolean_dataflow of eqcomp4 is
begin
        equals <=       not(a(0) xor b(0))
                    and not(a(1) xor b(1))
                    and not(a(2) xor b(2))
                    and not(a(3) xor b(3));
end architecture boolean_dataflow;
```

VHDL

- Same Example (including components):

...

component xnor2 **port** (a,b : **in** std_logic; z : **out** std_logic); **end component**;

component and4 **port** (a,b,c,d : **in** std_logic; z : **out** std_logic); **end component**;

architecture structural **of** eqcomp4 **is**

signal x : std_logic_vector(3 **downto** 0);

begin

u0: xnor2 **port map** (a(0), b(0), x(0));

u1: xnor2 **port map** (a(1), b(1), x(1));

u2: xnor2 **port map** (a(2), b(2), x(2));

u3: xnor2 **port map** (a(3), b(3), x(3));

u4: and4 **port map** (x(0), x(1), x(2), x(3), equals);

end architecture boolean_dataflow;

VHDL

- VHDL architectures are generally categorized in styles of;
  - Behavioral:
    - State machines
    - Sequential descriptions
    - Test benches
    - High level specifications
  - Data flow: implies a structure and behavior
    - Arithmetic operations
    - Concurrent signal assignment
    - Register transfers
  - Structural: defines interconnections of components
    - Physical information
    - Hierarchy of gates
    - Boolean equations

VHDL

- Behavioral style architecture

```
architecture my_arch of andgate is
begin
   process (a,b)
   begin
            if a = '1' and b = '1' then c <= '1' after 1ns;
                                   else c <= '0' after 1ns;
            end if;
   end process;
end my_arch;
```

VHDL

- Dataflow style architecture

```
architecture my_dataflow of xor2 is
begin
   c <= a xor b after m;
-- m is a parameter of delay defined in the entity of xor2
-- as generic ( m : time := 1.0ns);
end my_dataflow;
```

VHDL

- Structural style architecture

```
entity comparator is
   port ( a, b : in bit;
            c : out bit );
end comparator;
architecture my_structure of comparator is
signal i : bit; -- defines internal signal named as i.
component xor2 port ( x, y : in bit; z : out bit ); end component;
component inv port ( x : in bit; z : out bit ); end component;
begin
   c1 : xor2 port map ( a, b, i ); -- instance name is c1.
   c2 : inv port map ( i, c );
end my_structure;
```

VHDL

- Data Objects
  - They hold values of specified types. They belong to four classes; Constants, Signals, Variables, Files
  - Constant
    - Holds a value that can not be changed within a design description.
    - **constant** width : integer := 8;
  - Signals
    - Represent wires and also state of memory elements.
    - The symbol " <= " is used to indicate signal assignment.
    - **signal** count : **bit_vector** (3 **downto** 0);
  - Variables
    - Used only in processes and sub-programs.
    - Used for computational purposes during synthesis.
    - The symbol " := " is used for variable assignment.
    - **variable** result : **std_logic** := '0';

VHDL

- Alias
  - Not a data object.
  - An alternate identifier for an existing object.
  - **alias** top_addr : **std_logic_vector** (3 **downto** 0) **is** addr (31 **downto** 28);
- Data Types
  - Enumeration type is a list of values that an object of that type may have.
    - **type** states **is** (idle, preamble, data, jam, error);
      signal current_state : states;
    - **type** boolean **is** (FALSE, TRUE); -- already defined by IEEE 1076.
    - **type** bit **is** ('0', '1'); -- already defined by IEEE 1076.

VHDL

- … Data Types
  - … Enumeration type

```
type std_ulogic is ( 'U', -- uninitialized
                     'X', -- forcing unknown
                     '0', -- forcing 0
                     '1', -- forcing 1
                     'Z', -- high impedance
                     'W', -- weak unknown
                     'L', -- weak 0
                     'H', -- weak 1
                     '-' ); -- don't care
```

-- this type is already defined in IEEE 1164.

- To use these already defined types, place the following before an entity declaration;

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

VHDL

- … Data Types
  - Integer type is supported in the range $-(2^{31} - 1)$ to $(2^{31} - 1)$.

    variable a : **integer range** -255 **to** 255;
  - Array type

    type word **is array** (15 downto 0) **of** bit;

    signal my_bus : word;

    type table_8x4 **is array** (0 to 2, 0 to 3) **of bit**;

    constant ex_or : table_8x4 := (“0000”, “0011”, 1001”);

VHDL

- … Data Types
  - Record type has multiple elements of different types.

```
type iocell is record
    buffer_inp : bit_vector ( 7 downto 0 );
    enable : bit;
    buffer_out : bit _vector ( 7 downto 0);
end record;
signal  busa, busb, busc : iocell;
signal vec : bit_vector( 7 downto 0 );
busa.buffer_inp <= vec;
busb.enable <= '1';
busc <= busb;
```

VHDL

- Sequential Statements
  - They are found in a process, function, or procedure.
  - The collection of statements that make up a process constitutes a concurrent statement.
  - Electronic systems are concurrent and so are processes. If a design has multiple processes then these processes are concurrent with respect to one another and to other concurrent statements within the architecture.
  - Inside a process, signal assignment is sequential from a simulation standpoint. The order in which signal assignments are listed does affect how logic is synthesized.

VHDL

- … Sequential Statements
  - example

```
signal  step : std_logic;
signal addr : std_logic_vector( 7 downto 0 );

a_name : process (addr)
   begin
            step <= '0';
            if addr > X"0F" then step <= '1';
            end if;
   end process;
```

- … Sequential Statements
  - example

```
a_name : process (addr)
begin
if addr > X"0F" then step <= '1';
                else step <= '0';
end if;
end process;
```

  - example

```
a_name : process (addr)
begin
if addr > X"0F" then step <= '1';
end if;
-- else step retains its value.
-- This is referred to as implicit
   memory.
end process;
```

VHDL

- … Sequential Statements
  - example

```
process (addr)
begin
    case addr is
      when “001” => decode <= X”11”;
      when “111” => decode <= X”42”;
      when “101” => decode <= X”88”;
      when others => decode <= X”00”;
    end case;
end process;
```

- Synchronous Logic
  - example: D-FF

```
library ieee,
use ieee.std_logic_1164.all
entity dff is port (d, clk : in std_logic,
                    q : out std_logic);
end dff;
architecture example of dff is
begin
    process(clk)
    begin
      if clk’event and clk =‘1’ then q <= d;
      -- implicit memory for q.
      end if;
    end process;
end example;
```

VHDL

- Three-State Buffers and Bi-directional Signals
  - example:

```
entity ali is port
(count : buffer std_logic_vector(7 downto 0));
end ali;
-- one of the processes in the architecture is
my_buffer : process(oe,count)
    begin
      if oe='0' then count <= (others => 'Z');
      else count <= counter;
-- counter is the output of a  counter somewhere
-- in the architecture.
      end if;
    end process;
```

- … Three-State Buffers and Bi-directional Signals
  - example:

```
entity my_port is port
(count_io : inout std_logic_vector(7 downto 0));
end my_port;
counting : process (clk) begin
if clk'event and clk='0' then
    if load='1' then count <= count_io;
    elsif enable='1' then count <= count + 1;
    end if;
end process;
outputting : process (oe, count) begin
    if oe='0' then count_io <= (others => 'Z');
    else count_io <= count;
    end if;
end process;
```

VHDL

- Loops

  They are used to implement repetitive operations; consist of either “for” loops or “while” loops.

  – example: Resetting a FIFO using “while” loop.

```
reg_array : process(rst, clk)
variable j : integer := 0;
begin
    if rst='1' then
      while j < 7 loop
         fifo(j) <= (others =>'0');
         j := j + 1;
      end loop;
    else . . . . . .
```

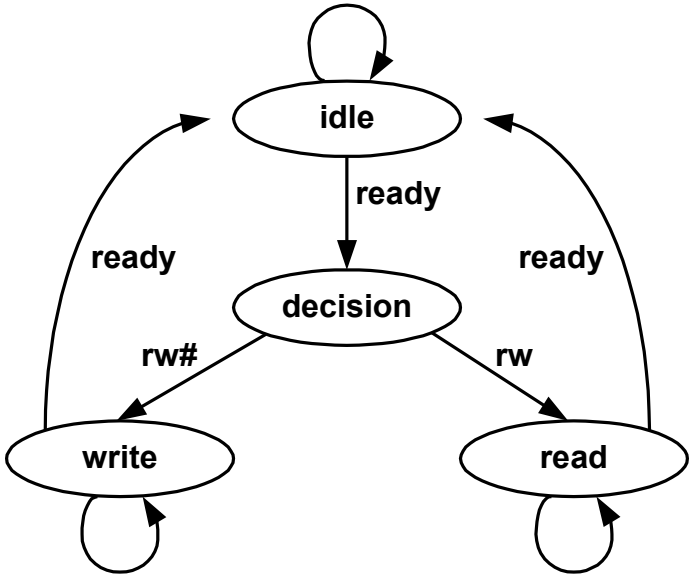
- … Loops

  – example: Resetting a FIFO using “for” loop.

```
reg_array : process(rst, clk)
begin
    if rst='1' then
      for j in 7 downto 0 loop
         if j = 4 then next;-- don't reset fifo(4)
         else fifo(j) <= (others =>'0');
         end if;
      end loop;
    else . . . . . .
```

VHDL

- State-Machine Design
  - Example



state	outputs	
	oe	we
idle	0	0
decision	0	0
write	0	1
read	1	0

- … State-Machine Design

```
architecture state_machine of example is
type statetype is (idle,decision,read,write);
signal present_state, next_state : state_type;
begin
my_machine : process(present_state,rw,ready)
begin
    case present_state is
    when idle =>
        oe<='0';
        we<='0';
        if ready='1' then next_state<= decision;
                    else next_state<= idle;
        end if;
```

VHDL

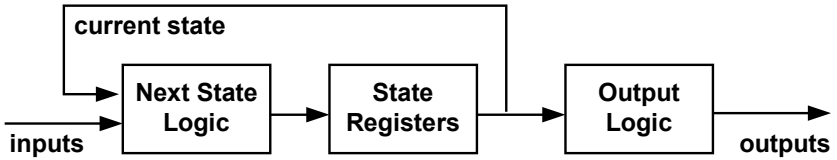
- … State-Machine Design

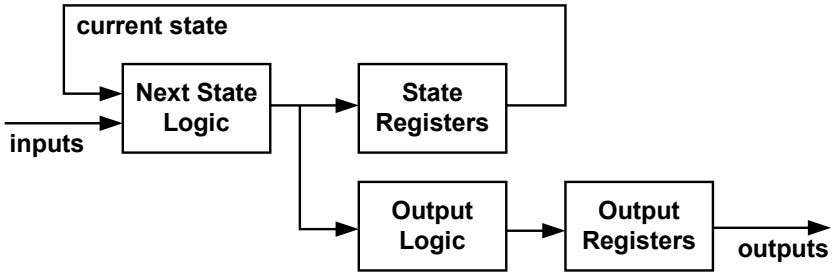
```
when decision =>
        oe<='0';
        we<='0';
        if rw ='1'  then next_state<= read;
                    else next_state<= write;
        end if;
when read =>
        oe<='1';
        we<='0';
        if ready ='1' then next_state<= idle;
                      else next_state<= read;
        end if;
```

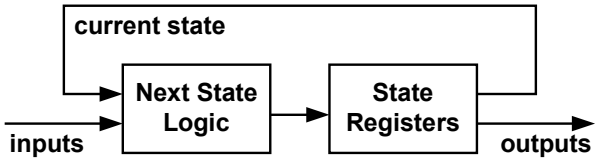
- … State-Machine Design

```
    when write => oe<='0'; we<='1';
            if ready ='1' then next_state<= idle;
                          else next_state<= write;
            end if;
    when others => oe<='0'; we<='0';
                   next_state <= idle;
    end case;
end process;
state_clock : process(clk)
begin
    if clk'event and clk ='1' then
            present_state <= next_state;
    end if;
end process;
end state_machine;
```

- … State-Machine Design
  - Outputs are decoded from state bits combinatorially
    - Clock to output delay is high.
  - Outputs are decoded in parallel output registers
    - Clock to output delay is minimum
    - Operating frequency is reduced.
  - Outputs are encoded within state bits
    - Clock to output delay is minimum
    - Operating frequency is maximum
    - Less macro-cells
    - Time and effort are needed for encoding
    - Debugging is harder







- … State-Machine Design
  - One-Hot Encoding
    - Technique that uses n flip-flops to represent a state-machine with n states.
    - Each state has its own flip-flop.
    - Only one flip-flop is "hot" (holds a '1') at any given time.
    - Depending on the target device architecture, it may require less macro-cells.
    - More flip-flops are required.
    - Simple and faster output decoding logic.
  - Example: One-Hot encoding

state	sequential encoded	One-Hot encoded
a	000	00000001
b	001	00000010
c	010	00000100
…		