

LAB 3

INTRODUCTION TO VHDL

CMP301A: COMPUTER ARCHITECTURE COURSE

EXERPTED FROM DR. ADNAN SHAOUT- *THE UNIVERSITY OF MICHIGAN-
DEARBORN*

OBJECTIVES

- Behavioral Modeling
- Review about Conditional statements.
- Combinational logic vs. registered logic processes.
- Variables, Signals, Constants.

REVIEW

- Code written maps to Hardware
- General Code written in VHDL is Concurrent
- Only inside a Process the code is sequential
- Every entity could have one Architecture or More
- Architecture can be modeled as dataflow, Structural or Mixed or ,... let's see

Next

MUX ... THE BEHAVIORAL WAY

```
ENTITY mux IS  
  PORT( a,b,c,d : IN std_logic;  
        s : IN std_logic_vector(1 DOWNTO 0);  
        x : OUT std_logic);  
END mux;
```

MUX ... THE BEHAVIORAL WAY

```
ARCHITECTURE behav OF mux IS  
BEGIN
```

```
    PROCESS (s0,s1,a,b,c,d) ← sensitivity list  
    BEGIN
```

```
        IF s0 = '0' AND s1 = '0' THEN
```

```
            y <= a;
```

```
        ELSIF s0 = '0' AND s1 = '1' THEN
```

```
            y <= b;
```

```
        ELSIF s0 = '1' AND s1 = '0' THEN
```

```
            y <= c;
```

```
        ELSE
```

```
            y <= d;
```

```
        END IF;
```

```
    END PROCESS;
```

```
END behav;
```

sensitivity list

Sequential statements
Executes one by one
because process is a
special case

VHDL - Architecture course

IF ... ELSE / CASE ... WHEN

```
ARCHITECTURE a_mux_seq OF
mux_seq
BEGIN
    PROCESS(a,b,c,d,s)
    BEGIN
        IF(s = "00") THEN
            x <= a ;
        ELSIF (s = "01") THEN
            x <= b;
        ELSIF (s = "10") THEN
            x <= c;
        ELSE
            x <= d;
        END IF;
    END PROCESS;
END a_mux_seq;
```

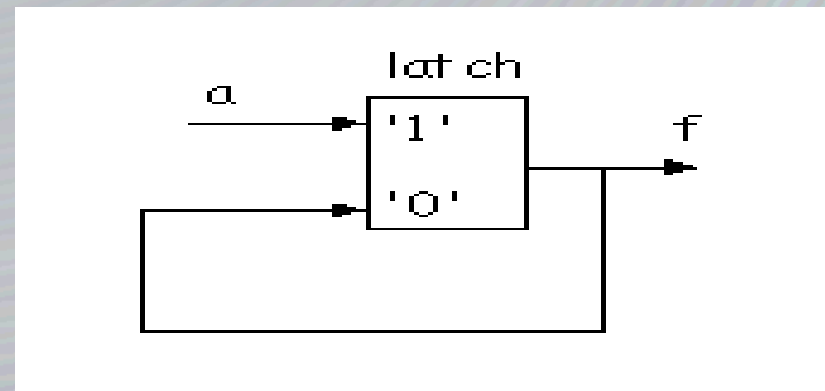
```
ARCHITECTURE b_mux_seq OF
mux_seq
BEGIN
    PROCESS(a,b,c,d,s)
    BEGIN
        CASE s IS
            WHEN "00" =>
                x <= d ;
            WHEN "01" =>
                x <= c;
            WHEN "10" =>
                x <= b;
            WHEN OTHERS =>
                x <= a;
        END CASE;
    END PROCESS;
END b_mux_seq;
```


IMPORTANT NOTES ABOUT PROCESSES

- Process is concurrent with respect to each other.
- Process is concurrent with other concurrent statements.
- Processes are only triggered on the change of the parameters in its sensitivity list
- Process with no sensitivity list loops forever (except ... to be mentioned later)
- To generate Combinational circuit using processes, All ports/signals on the right-hand side or in the conditions should be in the sensitivity list.

IF ... ELSE / CASE ... WHEN

- (IF ... ELSE) & (CASE ... WHEN) statements are used only inside a process
- Outside process we use (WHEN ... ELSE) or (WITH ... SELECT)
 - Look for (WITH ... SELECT) syntax in VHDL, Its similar to
- What happens if we missed a case ?!
 - A latch will be formed to save previous value



REVIEW CONDITIONAL STATEMENTS...1

```
ARCHITECTURE a_mux_seq OF mux IS  
BEGIN  
  PROCESS(a,b,c,d,s)  
    BEGIN  
    IF (s = "00") THEN  
      x <= a;  
    ELSIF (s = "01") THEN  
      x <= b;  
    ELSIF (s = "10") THEN  
      x <= c;  
    ELSE x <= d;  
    END IF;  
  END PROCESS;  
END a_mux_seq;
```

```
. ARCHITECTURE a_mux_con OF mux IS  
. BEGIN  
.   x <= a WHEN s = "00"  
.   ELSE b WHEN s = "01"  
.   ELSE c WHEN s = "10"  
.   ELSE d;  
. END a_mux_con ;
```

REVIEW CONDITIONAL STATEMENTS...2

```
ARCHITECTURE b_mux_seq OF mux IS  
BEGIN  
  PROCESS(a,b,c,d,s)  
    BEGIN  
    CASE s IS  
    WHEN "00" =>  
      x <= a;  
    WHEN "01" =>  
      x <= b;  
    WHEN "10" =>  
      x <= c;  
    WHEN OTHERS =>  
      x <= d;  
    END CASE;  
  END PROCESS;  
END b_mux_seq;
```

```
ARCHITECTURE b_mux_con OF mux IS  
BEGIN  
  WITH s SELECT  
    x <= a WHEN "00",  
    b WHEN "01",  
    c WHEN "10",  
    d WHEN OTHERS;  
END b_mux_con ;
```

Note: it is a comma not
semi colon

REVIEW CONDITIONAL STATEMENTS...3

- *If no “else” or “default” block exist , it automatically generates a **latch** to reserve the value .*

The background features a series of concentric circles in shades of light blue and grey, creating a ripple effect. In the corners, there are stylized circuit board traces with small circular nodes, resembling electronic components or data paths.

WHAT IS A REGISTER ?!

REGISTERED LOGIC...EXAMPLE _1

```
LIBRARY IEEE;  
USE IEEE.std_logic_1164.all;  
ENTITY my_DFF IS  
    PORT( d,clk,rst : IN std_logic;  q : OUT std_logic);  
END my_DFF;
```

```
    ARCHITECTURE a_my_DFF OF my_DFF IS  
    BEGIN  
    PROCESS(clk,rst)  
    BEGIN  
    IF(rst = '1') THEN  
        q <= '0';  
    ELSIF clk'event and clk = '1' THEN  
        q <= d;  
    END IF;  
    END PROCESS;  
    END a_my_DFF;
```

REGISTERED LOGIC...EXAMPLE _1

```
LIBRARY IEEE;  
USE IEEE.std_logic_1164.all;  
ENTITY my_DFF IS  
    PORT( d,clk,rst : IN std_logic;  q : OUT std_logic);  
END my_DFF;
```

```
    ARCHITECTURE a_my_DFF OF my_DFF IS  
    BEGIN  
    PROCESS(clk,rst)  
    BEGIN  
    IF(rst = '1') THEN  
        q <= '0';  
    ELSIF rising_edge(clk) THEN  
        q <= d;  
    END IF;  
    END PROCESS;  
    END a_my_DFF;
```

Last if doesn't have an "else" part so it will be automatically generate the behavior as "else q <=q;"

REGISTERED LOGIC...EXAMPLE _2

```
ENTITY my_nDFF IS  
GENERIC ( n ← integer := 16);  
PORT( Clk,Rst : IN std_logic;  
        d : IN std_logic_vector(n-1 DOWNT0 0);  
        q : OUT std_logic_vector(n-1 DOWNT0 0));  
END my_nDFF;
```

Remember the generic
data

REGISTERED LOGIC...EXAMPLE _2

```
ARCHITECTURE a_my_nDFF OF my_nDFF
IS
BEGIN
  PROCESS (Clk,Rst)
    BEGIN
      IF Rst = '1' THEN
         $q \leq (\textbf{OTHERS} \Rightarrow '0');$ 
      ELSIF rising_edge(Clk) THEN
         $q \leq d;$ 
      END IF;
    END PROCESS;
  END a_my_nDFF;
```

Notice the use of others
when size is unknown or
too big to write

```
ARCHITECTURE b_my_nDFF OF
  my_nDFF IS
  . COMPONENT my_DFF IS
  .   PORT( d,Clk,Rst : IN std_logic;
  .     q : OUT std_logic);
  . END COMPONENT;
  . BEGIN
  .   loop1: FOR i IN 0 TO n-1 GENERATE
  .     fx: my_DFF PORT MAP(d(i),Clk,Rst,q(i));
  .   END GENERATE;
  . END b_my_nDFF;
```

Remember the
for..generate

The background features a series of concentric circles in shades of light blue and grey, creating a ripple effect. In the corners, there are stylized circuit board traces with small circular nodes, some of which are highlighted in a light blue color.

VARIABLE, SIGNAL, CONSTANT

CONSTANT DECLARATION

- *A constant can have a single value of a given type.*
- *A constant's value cannot be changed during the simulation.*
- *Constants declared at the start of an architecture can be used anywhere in the architecture.*
- *Constants declared in a process can only be used inside the specific process.*

```
CONSTANT constant_name : type_name := value;
```

```
CONSTANT rise_fall_time : TIME := 2 ns;
```

```
CONSTANT data_bus : INTEGER := 16;
```

VARIABLE DECLARATION

- *Variables are used for local storage of data.*
- *Variables are generally not available to multiple components or processes.*
- *All variable assignments take place immediately.*
- *Variables are more convenient than signals for the storage of (temporary) data.*

```
VARIABLE variable_name : type_name [:=value];  
  
VARIABLE opcode : BIT_VECTOR(3 DOWNT0 0) := "0000";  
VARIABLE freq : INTEGER;
```

SIGNAL DECLARATION

- *Signals are used for communication between components.*
- *Signals are declared outside the process.*
- *Signals can be seen as real, physical signals.*
- *Some delay must be incurred in a signal assignment.*

```
SIGNAL signal_name : type_name [:=value];
```

```
SIGNAL brdy : BIT;
```

```
SIGNAL output : INTEGER := 2;
```


SIGNAL VS VARIABLE

CAN BE USED ANYWHERE

$X \leq Y$

*CHANGES CAN'T BE SEEN UNLESS
AFTER A DELAY*

```
. SIGNAL s : bit := '0';  
.  
.  
.  
.  
. PROCESS(x)  
. BEGIN  
.   s <= '1';  
.   a <= s;  
. END PROCESS;
```

a = '0'

ONLY USED INSIDE A PROCESS

$X := Y$

CHANGES CAN BE SEEN IMMEDIATELY

```
PROCESS(x)  
  VARIABLE s : bit := '0';  
BEGIN  
  s := '1';  
  a <= s;  
END PROCESS;
```

a = '1'

SIGNAL VS VARIABLE

	Signals	Variables
Assignment Operator	<code><=</code> e.g. <code>Sig1 <= a and b;</code> The left side must be signal	<code>:=</code> e.g. <code>Var1 := a and b;</code> The left side must be variable
Use	Represents circuit interconnect	Represents local storage
Declaration Region	Signals may only be declared in the declarative part of an Architecture (i.e. before BEGIN)	Variables may only be declared in the declarative part of the process (i.e. before the BEGIN).
Scope	Global Signals, seen by all the concurrent statements.	Local to process
Behavior	Updated at end of process execution(new value not immediately available)	Updated immediately

BEFORE REQUIREMENTS

- If a signal or a port got to be open circuit we put on it 'Z' high impedance
- Port mode (direction) has three modes :
 - From outside we see it as :
 - In \leftarrow input port can be used to feed data **to** this entity only
 - Out \leftarrow output port can be used to get data **from** this entity only
 - Inout \leftarrow can be used in both cases
 - From inside we see it as :
 - In \leftarrow input port can be used to get data from outside only (can't write to)
 - Out \leftarrow output port can be used to show output only (can't read from)
 - Inout \leftarrow can be used in both cases

