Review of Digital Design with VHDL
Digital World

- Digital world is a world of ‘0’ and ‘1’
  - Each binary digit is called a bit
  - Eight consecutive bits are called a byte

- Hexadecimal (base 16) representation for convenience
  - 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
  - Each hexadecimal digital represents 4 binary bits

- Basic operations for values in hexadecimal representation
  - Addition
  - Subtraction
Overview

Digital Component
VHDL structure

- Library
  - Definitions, constants
- Entity
  - Interface
- Architecture
  - Implementation, function
Libraries

- Library ieee;
- Use ieee.std_logic_1164.all;
- Use ieee.std_logic_arith.all;
- Use ieee.std_logic_signed.all;
- Use ieee.std_logic_unsigned.all;
VHDL - Library

- Include library

```vhdl
library IEEE;
```

- Define the library package used

```vhdl
use IEEE.STD_LOGIC_1164.all;
```

- Define the library file used

- For example, STD_LOGIC_1164 defines ‘1’ as logic high and ‘0’ as logic low

  ```vhdl
  output <= '1'; -- Assign logic high to output
  ```
VHDL - Entity

- Interface for communication among different modules / components
- Define the signal port modes (INPUT and OUTPUT)

This is a black box implemented by the statements in Architecture
VHDL - Entity

- Input port can only be read inside architecture
  - `input1 <= temp;` -- This statement is NOT allowed

- Output port can only be written inside architecture
  - `temp <= output1;` -- This statement is NOT allowed

```
output <= inputa and inputb;
```

- output is assigned to be inputa AND inputb
Entity

- Define inputs and outputs
- Example:

```vhdl
Entity test is
Port( A,B,C,D: in std_logic;
    E: out std_logic);
End test;
```

Digital Component

Inputs and Outputs
Architecture

- Define functionality of the component

- $X \leftarrow A \text{ AND } B$;
- $Y \leftarrow C \text{ AND } D$;
- $E \leftarrow X \text{ OR } Y$;
Signal

- All internal variables

Signal X,Y : std_logic;

Diagram of a component with inputs A, B, C, D and outputs X, Y, and E, showing the flow of signals.
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY TEST IS
PORT (A,B,C,D : IN STD_LOGIC;
      E   : OUT STD_LOGIC);
END TEST;

ARCHITECTURE BEHAVIOR OF TEST IS
SIGNAL X,Y : STD_LOGIC;
BEGIN
  X <= A AND B;
  Y <= C AND D;
  E <= X OR Y;
END BEHAVIOR;
VHDL features

- Case insensitive
  - inputa, INPUTA and InputA are refer to the same variable
- Comments
  - ‘--’ until end of line
  - If you want to comment multiple lines, ‘--’ need to be put at the beginning of every single line
- Statements are terminated by ‘;’
- Signal assignment:
  - ‘<=’
- Variable assignment:
  - ‘:=’
- User defined names:
  - letters, numbers, underscores (‘_’)
  - start with a letter
Different ways to describe a digital system in VHDL

- Description of a complex system is typically a mix of both behavioral description and structural description.

![Diagram showing behavioral and structural descriptions](image-url)
Behavior Description

- \( \text{WARNING} \leq (\text{not DOOR and IGNITION}) \) or \( (\text{not SBELT and IGNITION}) \)
Structural Description

architecture structural of BUZZER is

-- Declarations

component AND2

  port (in1, in2: in std_logic;
        out1: out std_logic);

end component;

component OR2

  port (in1, in2: in std_logic;
        out1: out std_logic);

end component;

component NOT1

  port (in1: in std_logic;
        out1: out std_logic);

end component;

-- declaration of signals used to interconnect gates
signal DOOR_NOT, SBELT_NOT, B1, B2: std_logic;

begin

  -- Component instantiations statements
  U0: NOT1 port map (DOOR, DOOR_NOT);
  U1: NOT1 port map (SBELT, SBELT_NOT);
  U2: AND2 port map (IGNITION, DOOR_NOT, B1);
  U3: AND2 port map (IGNITION, SBELT_NOT, B2);
  U4: OR2 port map (B1, B2, WARNING);

end structural;
Chip1 : Chip_A
Port map (A,B,C,X,Y);

Chip2 : Chip_B
Port map (X,Y,D,E);
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY TEST IS
PORT (A,B,C,D : IN STD_LOGIC;
       E       : OUT STD_LOGIC);
END TEST;

ARCHITECTURE BEHAVIOR OF TEST IS
SIGNAL X,Y : STD_LOGIC;

COMPONENT Chip_A
PORT (L,M,N : IN STD_LOGIC;
      O,P   : OUT STD_LOGIC);
END COMPONENT;

COMPONENT Chip_B
PORT (Q,R,S : IN STD_LOGIC;
      T : OUT STD_LOGIC);
END COMPONENT;

BEGIN

Chip1 : Chip_A
PORT MAP (A,B,C,X,Y);
Chip2 : Chip_B
PORT MAP (X,Y,D,E);

END BEHAVIOR;
Port Mapping

Two ways to associate the port signals and actual signals in an entity

- **Positional association**
  - Ordering matters
    - \((\text{actual}_0, \text{actual}_1, \text{actual}_2, \ldots, \text{actual}_n)\)

- **Named association**
  - Ordering does not matter
    - \((\text{port}_0=>\text{actual}_0, \text{port}_1=>\text{actual}_1, \text{port}_2=>\text{actual}_2, \ldots, \text{port}_n=>\text{actual}_n)\)
Structural Modeling

- Declare a list of components being used
- Declare signals which define the nets (i.e., wires) that interconnect components
- Label multiple instances of the same component so that each instance is uniquely defined

```vhdl
architecture architecture_name of NAME_OF_ENTITY is
  -- Declarations
  component declarations
  signal declarations
  begin
  -- Statements
  component instantiation and connections
  :
  :
end architecture_name;
```
Component Declaration

Before components can be instantiated they need to be declared in the architecture declaration section

```
component component_name [is]
  [port ( port_signal_names: mode type;
    port_signal_names: mode type;
    ...
    port_signal_names: mode type);]
end component [component_name];
```

```
component OR2
  port (in1, in2: in std_logic;
    out1: out std_logic);
end component;
```
Component Instantiation and Interconnections

The component instantiation statement references a component that can be:

- Previously defined at the current level of the hierarchy
- Defined in a technology library (e.g., vendor’s library)

```
instance_name : component_name
    port map (port1=>signal1, port2=> signal2,… port3=>signaln);
```

```
instance_name : component_name
    port map (signal1, signal2,… signaln);
```

```
component NAND2
    port (in1, in2: in std_logic;
          out1: out std_logic);
end component;

signal int1, int2, int3: std_logic;
:
U1: NAND2 port map (A,B,int1);
U2: NAND2 port map (in1=>C, in2=>D, out1=>int2);
U3: NAND3 port map (int1, int2, Z);
```
Data Objects

- **Constant**
  - A constant can have a single value of a given type and cannot be changed
  - Constants declared at the start of an architecture
    - can be used anywhere within the architecture
  - Constants declared within a process
    - can only be used inside that specific process

```plaintext
constant list_of_name_of_constant: type [ := initial value] ;

constant    RISE_FALL_TME: time := 2 ns;
constant    DELAY1: time := 4 ns;
constant    RISE_TIME, FALL_TIME: time:= 1 ns;
constant    DATA_BUS: integer:= 16;
```
Data Objects

- **Variable**
  - can be updated using a variable assignment statement
    
    Variable_name := expression;
  - is updated without any delay as soon as the statement is executed
  - must be declared *inside* a process
    - is local to the process

```
variable list_of_variable_names: type [ := initial value] ;

variable  CNTR_BIT: bit :=0;
variable  VAR1: boolean :=FALSE;
variable  SUM: integer range 0 to 256 :=16;
variable  STS_BIT: bit_vector (7 downto 0);
```
Data Objects

■ Signal

■ Represent wires and storage elements
■ Declared inside the architecture, outside the process
■ Updated when the signal assignment statement is executed, \textit{after a certain delay}

\[
\text{signal}\_\text{name} <= \text{expression};
\]

\textbf{signal list_of_signal_names: type [ := initial value];}

\textbf{signal SUM, CARRY: std\_logic;}
\textbf{signal CLOCK: bit;}
\textbf{signal TRIGGER: integer :=0;}
\textbf{signal DATA\_BUS: bit\_vector (0 to 7);}
\textbf{signal VALUE: integer range 0 to 100;}

Data Objects

Signal

- Initial value for signals representing wires
  - Not necessary
- Initial value for signals representing storage elements
  - Use explicit control signal, e.g., “reset” to initialize all storage elements to a pre-determined state
Difference between variables and signals

Example of a process using Variables

```
architecture VAR of EXAMPLE is
  signal TRIGGER, RESULT: integer := 0;
begin
  process
    variable variable1: integer :=1;
    variable variable2: integer :=2;
    variable variable3: integer :=3;
    begin
      wait on TRIGGER;
      variable1 := variable2;
      variable2 := variable1 + variable3;
      variable3 := variable2;
      RESULT <= variable1 + variable2 + variable3;
    end process;
  end process;
end VAR
```

- What are the values of “variable1”, “variable2”, “variable3”, and “RESULT” after the process is executed?
Difference between variables and signals

Example of a process using Signals

```vhdl
architecture SIGN of EXAMPLE is
  signal TRIGGER, RESULT: integer := 0;
  signal signal1: integer :=1;
  signal signal2: integer :=2;
  signal signal3: integer :=3;
begin
  process
  begin
    wait on TRIGGER;
    signal1 <= signal2;
    signal2 <= signal1 + signal3;
    signal3 <= signal2;
    RESULT  <= signal1 + signal2 + signal3;
  end process;
end SIGN;
```

What are the values of “signal1”, “signal2”, “signal3”, and “RESULT” after the process is executed?
Data Types

- **bit** values: '0', '1'
- **boolean** values: TRUE, FALSE
- **integer** values: \(-2^{31}\) to \(+2^{31} - 1\)

- **std_logic** values: 'U', 'X', '1', '0', 'Z', 'W', 'H', 'L', '-'
  - 'U' = uninitialized
  - 'X' = unknown
  - 'W' = weak 'X'
  - 'Z' = floating
  - 'H'/'L' = weak '1'/'0'
  - '-' = don't care

- **Std_logic_vector** (n downto 0);
- **Std_logic_vector** (0 upto n);
Concurrency in the Architecture

architecture architecture_name of NAME_OF_ENTITY is
  -- Declarations
  -- components declarations
  -- signal declarations
    :

begin
  -- Statements
  -- Processes
    :

end architecture_name;

- A statement will be executed when one or more of the signals on the right hand side change their value
- A process will be executed if there is any change in the value of the signals in the sensitivity list
Process

- All statements in a process occur sequentially
  - The values of signals are assigned after a delay
- Processes typically have sensitivity list
  - If the sensitivity list is not specified, one has to include a `wait` statement to make sure that the process will halt
  - Cannot include both a sensitivity list and a wait statement
- Processes are used to model both sequential circuits and combinational circuits

```vhdl
Process (A,B,C)
Begin
  statements;
End process;
```
Process Statement

[process_label:] process [ (sensitivity_list) ] [is] [ process_declarations]

begin

list of sequential statements such as:
signal assignments
variable assignments
case statement
exit statement
if statement
loop statement
next statement
null statement
procedure call
wait statement

end process [process_label];
Process

Example: Sequential Circuit

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity DFF_CLEAR is
  port (CLK, CLEAR, D : in std_logic;
        Q : out std_logic);
end DFF_CLEAR;

architecture BEHAV_DFF of DFF_CLEAR is
begin
  DFF_PROCESS: process (CLK, CLEAR)
  begin
    if (CLEAR = '1') then
      Q <= '0';
    elsif (CLK'event and CLK = '1') then
      Q <= D;
  end if;
  end process;
end BEHAV_DFF;
```
Process

Example: Combinational Circuit

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity FULL_ADDER is
  port (A, B, Cin : in std_logic;
        Sum, Cout : out std_logic);
end FULL_ADDER;

architecture BEHAV_FA of FULL_ADDER is
  signal int1, int2, int3: std_logic;
begin
  -- Process P1 that defines the first half adder
  P1: process (A, B)
  begin
    int1<= A xor B;
    int2<= A and B;
  end process;
  -- Process P2 that defines the second half adder and the OR -- gate
  P2: process (int1, int2, Cin)
  begin
    Sum <= int1 xor Cin;
    int3 <= int1 and Cin;
    Cout <= int2 or int3;
  end process;
end BEHAV_FA;
```
Process

- Test
  - How to use process to model a multiplexer?
VHDL language elements

VHDL is composed of language **building blocks** that consist of more than *75 reserved words* and about *200 descriptive words* or *word combinations*.
Reserved VHDL keywords

<table>
<thead>
<tr>
<th>ABS</th>
<th>ACCESS</th>
<th>AFTER</th>
<th>ALIAS</th>
<th>ALL</th>
<th>AND</th>
<th>ARCHITECTURE</th>
<th>ARRAY</th>
<th>ASSERT</th>
<th>ATTRIBUTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>BLOCK</td>
<td>BODY</td>
<td>BUFFER</td>
<td>BUS</td>
<td>CASE</td>
<td>COMPONENT</td>
<td>CONFIGURATION</td>
<td>CONSTANT</td>
<td></td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>DOWNTO</td>
<td>ELSE</td>
<td>ELSIF</td>
<td>END</td>
<td>ENTITY</td>
<td>EXIT</td>
<td>FILE</td>
<td>FOR</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>GENERATE</td>
<td>GENERIC</td>
<td>GROUP</td>
<td>GUARDED</td>
<td>IF</td>
<td>IMPURE</td>
<td>IN</td>
<td>INERTIAL</td>
<td>INOUT</td>
<td>IS</td>
</tr>
<tr>
<td>LABEL</td>
<td>LIBRARY</td>
<td>LINKAGE</td>
<td>LITERAL</td>
<td>LOOP</td>
<td>MAP</td>
<td>MOD</td>
<td>NAND</td>
<td>NEW</td>
<td>NEXT</td>
</tr>
<tr>
<td>PACKAGE</td>
<td>PORT</td>
<td>POSTPONED</td>
<td>PROCEDURE</td>
<td>PROCESS</td>
<td>PURE</td>
<td>RANGE</td>
<td>RECORD</td>
<td>REGISTER</td>
<td>REM</td>
</tr>
<tr>
<td>RETURN</td>
<td>SELECT</td>
<td>SEVERITY</td>
<td>SIGNAL</td>
<td>SHARED</td>
<td>SLA</td>
<td>SLL</td>
<td>SRA</td>
<td>SRL</td>
<td>SUBTYPE</td>
</tr>
<tr>
<td>THEN</td>
<td>TO</td>
<td>TRANSPORT</td>
<td>TYPE</td>
<td>UNAFFECTED</td>
<td>UNITS</td>
<td>UNTIL</td>
<td>USE</td>
<td>VARIABLE</td>
<td>WAIT</td>
</tr>
</tbody>
</table>
## VHDL Operators

<table>
<thead>
<tr>
<th>Class</th>
<th>and</th>
<th>or</th>
<th>nand</th>
<th>nor</th>
<th>xor</th>
<th>xnor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Logical operators</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Relational operators</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Shift operators</td>
<td>sll</td>
<td>srl</td>
<td>sla</td>
<td>sra</td>
<td>rol</td>
<td>ror</td>
</tr>
<tr>
<td>4. Addition operators</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Unary operators</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Multiplying op.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Miscellaneous op.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Logical operators are bitwise
reference

- VHDL Tutorial
  - Jan Van der Spiegel, University of Pennsylvania
  - http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html