Experiment # 05

Introduction to HDL Based Designing

1. OBJECTIVES:
   - To familiarize students with HDL based designing.
   - To familiarize students with different IDE tools for HDL based designing.

2. Resources Required
   - A Computer
   - Veriwell
   - ModelSim

3. Introduction

   HDL (Hardware Description Language) is any language from a class of computer languages, specification languages, or modelling languages for formal description and design of electronic circuits, and most-commonly, digital logic. It can describe the circuit's operation, its design and organization, and tests to verify its operation by means of simulation. The two most popular HDLs are Verilog and VHDL. Verilog due to its similarity to C language is easier to understand so has become most widely used HDL in educational institutions.

   Design Methodologies

   There are two basic types of digital design methodologies: a top-down design methodology and a bottom-up design methodology.

   In a top-down design methodology, we define the top-level block and identify the sub blocks necessary to build the top-level block. We further subdivide the sub-blocks until we come to leaf cells, which are the cells that cannot further be divided.

   In a bottom-up design methodology, we first identify the building blocks that are available to us. We build bigger cells, using these building blocks. These cells are then used for higher-level blocks until we build the top-level block in the design.
Module:

Verilog provides the concept of a module. A module is the basic building block in Verilog. A module can be an element or a collection of lower-level design blocks.

In Verilog, a module is declared by the keyword `module`. A corresponding keyword `endmodule` must appear at the end of the module definition. Each module must have a `module_name`, which is the identifier for the module, and a `module_terminal_list`, which describes the input and output terminals of the module.

```
module <module_name> (<module_terminal_list>); //outputs first then inputs
...
<module internals>   // Also called module body
...
endmodule
```

Lexical Conventions:

The basic lexical conventions used by Verilog HDL are similar to those in the C programming language. Verilog contains a stream of tokens. Tokens can be comments, delimiters, numbers, strings, identifiers, and keywords. Verilog HDL is a case-sensitive language. All keywords are in lowercase.

Whitespace:

Blank spaces (\b) , tabs (\t) and newlines (\n) comprise the whitespace. Whitespace is ignored by Verilog except when it separates tokens. Whitespace is not ignored in strings.

Comments:

Comments can be inserted in the code for readability and documentation. There are two ways to write comments. A one-line comment starts with "//". Verilog skips from that point to the end of line. A multiple-line comment starts with "/*" and ends with "*/". Multiple-line comments cannot be nested. However, one-line comments can be embedded in multiple-line comments.

```
a = b && c; // This is a one-line comment

/* This is a multiple line
   comment */
```
Operators:

Operators are of three types: unary, binary, and ternary. Unary operators precede the operand. Binary operators appear between two operands. Ternary operators have two separate operators that separate three operands.

\[ a = \sim b; \] // \sim \text{ is a unary operator. } b \text{ is the operand} \\
\[ a = b \&\& c; \] // \&\& \text{ is a binary operator. } b \text{ and } c \text{ are operands} \\
\[ a = b \text{ ? } c : d; \] // ?: \text{ is a ternary operator. } b, c \text{ and } d \text{ are operands}

Number Specification:

There are two types of number specification in Verilog: sized and unsized.

a) Sized numbers

Sized numbers are represented as \(<size> \langle base \text{ format} \rangle <number>\).

\(<size>\) is written only in decimal and specifies the number of bits in the number. Legal base formats are decimal (’d or ’D), hexadecimal (’h or ’H), binary (’b or ’B) and octal (’o or ’O). The number is specified as consecutive digits from 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f. Only a subset of these digits is legal for a particular base. Uppercase letters are legal for number specification.

\[ 4'b1111 \] // This is a 4-bit binary number \[ 12'habc \] // This is a 12-bit hexadecimal number \[ 16'd25 \] // This is a 16-bit decimal number.

b) Unsized numbers

Numbers that are specified without a \(<base \text{ format}>\) specification are decimal numbers by default. Numbers that are written without a \(<size>\) specification have a default number of bits that is simulator- and machine-specific (must be at least 32).

\[ 23456 \] // This is a 32-bit decimal number by default \\
\[ 'hc3 \] // This is a 32-bit hexadecimal number \\
\[ 'o21 \] // This is a 32-bit octal number

Strings:

A string is a sequence of characters that are enclosed by double quotes. The restriction on a string is that it must be contained on a single line, that is, without a carriage return. It cannot be on multiple lines. Strings are treated as a sequence of one-byte ASCII values.

"Hello Verilog World" // is a string \\
"a / b" // is a string
Identifiers and Keywords:

Keywords are special identifiers reserved to define the language constructs. Keywords are in lowercase. Identifiers are names given to objects so that they can be referenced in the design. Identifiers are made up of alphanumeric characters, the underscore ( _ ), or the dollar sign ( $ ). Identifiers are case sensitive. Identifiers start with an alphabetic character or an underscore. They cannot start with a digit or a $ sign (The $ sign as the first character is reserved for system tasks).

reg value; // reg is a keyword; value is an identifier

input clk; // input is a keyword, clk is an identifier

Simulation:

Once a design block is completed, it must be tested. The functionality of the design block can be tested by applying stimulus and checking results. We call such a block the stimulus block. It is good practice to keep the stimulus and design blocks separate. The stimulus block can be written in Verilog. A separate language is not required to describe stimulus. The stimulus block is also commonly called a test bench. Different test benches can be used to thoroughly test the design block.

Stimulus block is usually of the form:

module Stimulus;

reg <inputs> //all inputs become reg

wire <outputs> //all outputs becomes wire

<main_module instance> //Instantiation will be discussed in more detail in future labs

<Simulation conditions with timings> //which give specific outputs related to design

endmodule

HDL Coding:

Verilog is both a behavioural and a structural language. Internals of each module can be defined at four levels of abstraction, depending on the needs of the design. The module behaves identically with the external environment irrespective of the level of abstraction at which the module is described. The internals of the module are hidden from the environment. Thus, the level of abstraction to describe a module can be changed without any change in the environment. The levels are defined below.

Behavioural or algorithmic level:

This is the highest level of abstraction provided by Verilog HDL. A module can be implemented in terms of the desired design algorithm without concern for the hardware implementation details. Designing at this level is very similar to C programming.
Dataflow level:

At this level, the module is designed by specifying the data flow. The designer is aware of how data flows between hardware registers and how the data is processed in the design.

Gate level:

The module is implemented in terms of logic gates and interconnections between these gates. Design at this level is similar to describing a design in terms of a gate-level logic diagram.

Switch level:

This is the lowest level of abstraction provided by Verilog. A module can be implemented in terms of switches, storage nodes, and the interconnections between them. Design at this level requires knowledge of switch-level implementation details.

Main module:
/* A simple AND gate
File: and.v */

module andgate (y, a, b);
input a, b;
output y;
and and1(y,a,b);  //assign y = a & b;
endmodule

Stimulus:

module stimulus;
wire y;
reg a, b;
andgate my_gate( y,a,b);
```verilog
initial
begin
Smonitor(a, b, y);     // displays every time one of its parameters changes
    a = 1'b0; b = 1'b0;
    #5 a = 1'b0; b = 1'b1;
    #5 a = 1'b1; b = 1'b0;
    #5 a = 1'b1; b = 1'b1;
    #5 $stop;         // halts the simulator and puts it in the interactive mode
    #5 $finish;      // exits the simulator back to the operating system.
end
endmodule
```

**Simulation in Veriwell:**

1. Open Veriwell. Go to Project. Create a new Project named `andgate`.
2. Go to File. Select New and write the code mentioned above in the file. You can ignore the comments in the code.
3. Save the file with name `and_gate`.
4. Again go to File. Select New and write the code mentioned above in the file.
5. Save the file with name `stimulus`.
6. Again go to Project menu and select Add File and add both files created in step 2 to 5.
7. Again go to Project menu and select run or press Ctrl+R to run the simulation.
8. From the window menu select Veriwell Console.
9. Again go to project menu and select continue.
10. Verify the results in the Veriwell Console.

**Lab Task:**

1. Simulate OR, NOT, NAND, NOR and XOR in Veriwell and show the results to the Instructor.
1) What is HDL? Give its examples.

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

2) What is the basic building block of Verilog? What are its components?

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

3) What is Stimulus module? Also write its function.

……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………