Assignment 1
Introduction to Verilog

ECN 102 - Digital Logic Design
Department of Electronics and Communication Engineering
Indian Institute of Technology, Roorkee

November 29, 2017

Hardware Description Languages

Hardware description language (HDL) is a convienient, device independent way of representing digital logic. HDLs are helpful describing, simulating and verifying digital circuits.

Why not use C/C++, Java...?

C/C++, Java are programming languages which are very good at what they were designed for, that is programming. However describing digital circuits in programming language is difficult and often confusing, which led to development of HDLs.

Which HDL to use?

Today many Hardware description languages are available, each of them are different from each other in terms of functionality, semantic and grammar. In this course however we will be using Verilog 2001.

Using Verilog 2001

As a begininer we are going to describe and simulate simple combinational circuit with Verilog.

Design Flow

Verilog Syntax

Modules

Module is the basic building block in Verilog, modules helps in organizing and structuring designs in logical and human readable form. Module can be considered as analogus to functions in programming languages.

Listing 1: Sample module indicating its structure

```
module module_name( ...module_input_and_outputs... );
  ...
  // Module functionality
  ...
endmodule;
```

An example of AND gate would be

Listing 2: Illustrative AND gate module

```
module andGate (inputA, inputB, result);
      ...
      // AND gate functionality
      ...
endmodule
```

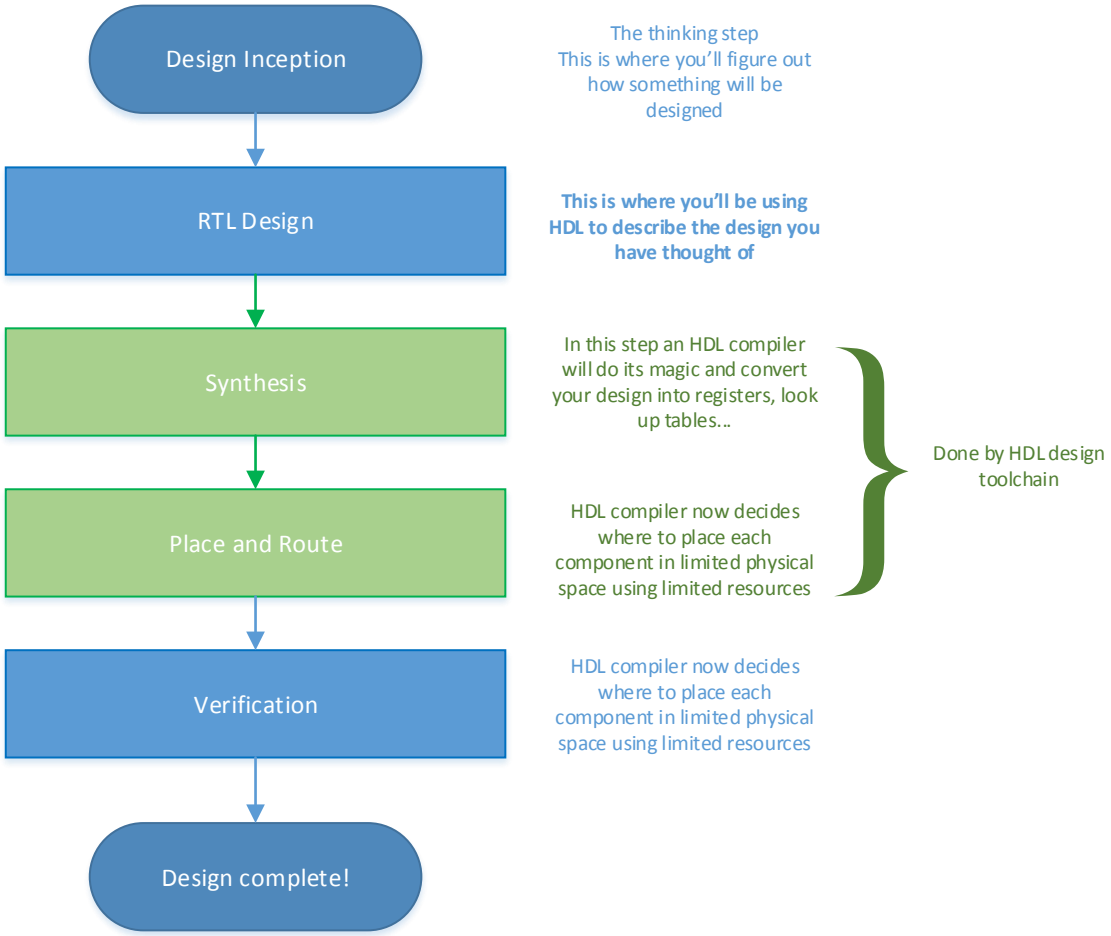

Figure 1: Example binary search tree constructed using names from Table **??** with insertion order and output port information.

Instantiating modules

The process of creating objects of modules is called instatiation in Verilog. Modules are a

Listing 3: Illustrative AND gate module

```
1  // Define the top-level module called AndGate3 which is a three
2  // input AND gate. It instantiates 2 2-input AND gates.
3  // Interconnections will be explained later
4  module AndGate3(input1, input2, input3, result);
5    input input1, input2, input3; // Declare variables as input
6    output result; // declare result as an output
7
8    wire result1; // To be explained later
9
10    // Instatiation of 2-input AND gates
11   AndGate2(input1, input2, result1);
12   AndGate2(result1, input3, result);
13 endmodule
14
15 // Define a module AndGate2 which is a 2-input AND gate.
16 module AndGate2(input1, input2, result);
17   // Assignment would be explained later
18   assign result = input1 & input2;
19 endmodule
```

Comments

Comments in verilog are of two kinds:

Single Line Comment

Two forward slashes represents begining of a single line comment in verilog, any charater after the two forward slashes will be ignored by verilog compiler until the end of line.

Listing 4: Single line comment

```
...
// This is a single line comment
...
```

Block Comment

Block comments in verilog are used to comment more than one line, the start with '/*' and ends with '*/'. Anything between these two character sequence is ignored by the compiler.

Listing 5: Block comment

```
...
/*
 This is a block comment
 */
...
```

Numerical Literals

Sized Numbers

To represent digital circuits accurately Verilog allows defining numbers of fixed size. These numbers have following format:

<size>'<character for base><number>

For example:

Listing 6: Example of sized numbers

```
6'b010010 // (18) 32-bit number in binary
24'hc0ffee // (12648430) 24-bit number
16'd255 // (255) 16-bit decimal number
```

Unisized Numbers

Verilog also includes support for unsized numbers. These numbers are assumed to be of a particular size depending upon the compiler/machine.

Constants

Global constants can be decalered in Verilog. When Verilog code is processed all these constants will be replaced by their respective values. NOTE: Verilog constants always starts with backtick ' ` '.

Listing 7: Declaration and use of constants

```
`define A 2'200 // NOTE: Constant declarations do
`define B 2'b01 // not end with semicolon!
`define C 2'210
`define D 2'b11

// Using constants
wire [A:0] wire_a = `A; // Notice the backtick
```

```
8   wire [B:0] wire_b = `B;
9   wire [C:0] wire_c = `C;
10  wire [D:0] wire_d = `D;
```

Wires

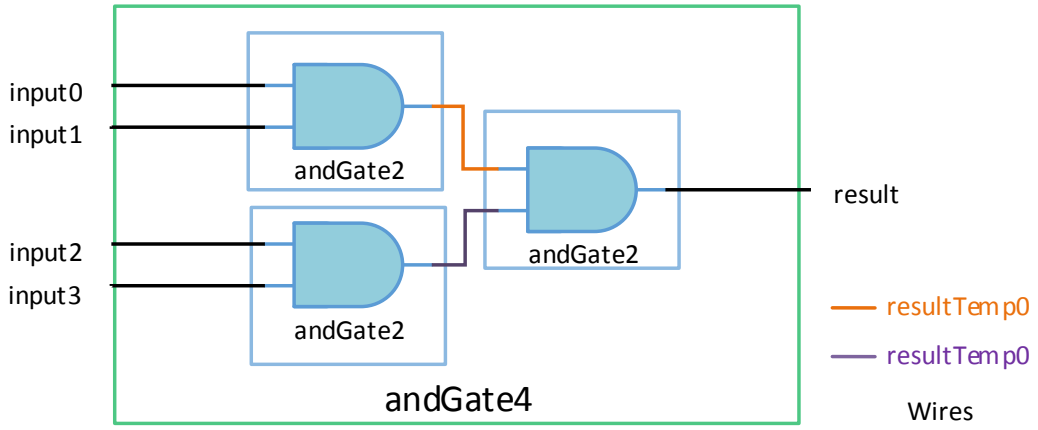

Figure 2: Gate level representation of code in Listing 8.

Wires are something very important in verilog, use of wires in Verilog is for what wires are used in real life ... to connect two things electrically! Wires will be used extensively in future assignments to connect two modules, registers and even wires together. This concept is explained using Listing 7 where wires are used to connect output of two 2-input AND gates to inputs of single 2-input AND gates. Gate level diagram of which is shown in Fig. 2.

Listing 8: Use of wires to connect output of one module to input of another

```
1   // andGate4 is a four input AND gate that uses 3 2-input AND
2   // gates to produce its result. resultTemp0 and resultTemp1 wires
3   // are used to connect output of two andGate2 to a another andGate2
4   // This module uses andGate2 module from previous listing.
5   module andGate4(input0, input1, input2, input3, result);
6     // Declare input and output
7     input input0, input1, input2, input3;
8     output result;
9
10    // Declare wire for connecting gates
11    wire resultTemp0;
12    wire resultTemp1;
13
14    // Declare instances of 2-input AND gate
15    andGate2(input0, input1, resultTemp0);
16    andGate2(input2, input3, resultTemp1);
17
18    andGate2(resultTemp0, resultTemp1, result);
19  endmodule
```

Registers

Always Block

If and Case Statement