Assignment - 2
Verilog Modelling Techniques

Department of Electronics and Communication Engineering
Indian Institute of Technology, Roorkee

ECN 104 Digital Logic Design

Introduction

Verilog supports wide variety of modelling techniques. Different modelling techniques allows hardware description at different level of abstraction, starting from switch level modelling (PMOS/NMOS) and all the way up to behavioural modelling (algorithmic description). Each one of them have their own benefits and use cases. In this assignment we will discuss three of them, namely Gate-level modelling, structural modelling and behavioural modelling.

Gate-level modelling

Gate-level modelling in Verilog is used to describe a circuit only using logic gates. This approach is used to describe critical parts of a design, like adders and multipliers. Using gate-level implementation allows greater control over the design than other techniques. Gate-level modelling is only used for small scale design, due to its complexity other modelling techniques are commonly used to abstract gate level implementation.

Gate Primitives in Verilog

Verilog support following gates:

- AND
- NAND
- OR
- NOR
- XOR
- XNOR
- NOT

Gates in Verilog are available as primitives and can be instantiated similar to modules, Listing 1 is an example gate level circuit.

Listing 1: Example module using Gate-level modelling

```verilog
/* Example using gate-level modelling, gate is a built-in
 * primitive in Verilog. Gates are instantiated in a way
 * similar to modules. Gates can be of single input or
 * multiple input
 */
module gateLevelExample(input1, input2, input3, result);
   input input1, input2, input3;
   output [8:0] result;

   // Single input gates
   not  g0(result[0], input1);           // = ~input1

   // Two input gates
   and  g1(result[1], input1, input2);   // =   input1 & input2
   nand g2(result[2], input1, input2);   // = ~(input1 & input2)
   or   g3(result[3], input1, input2);   // =   input1 | input2
   nor  g4(result[4], input1, input2);   // = ~(input1 | input2 )
   xor  g5(result[5], input1, input2);   // =   input1 ^ input2
   xnor g6(result[6], input1, input2);   // = ~(input1 ^ input2)

   // Gates with more than two input
   xnor g7(result[7], input1, input2, input3);
   and  g8(result[8], input1, input2, input3, input1);
endmodule // gateLevelExample
```

Verilog also supports instantiating gates without a instance name demonstrated in Listing 2.

Listing 2: Instantiating unnamed gates

```verilog
/*
 * Gates in Verilog can be instantiated without a name,
 * such instantiation in Verilog is legal.
 */
module unnamedGate(input1, input2, result);
   input input1, input2;
   output result;

   and(result, input1, input2); // Unnamed gate
endmodule // unnamedGate
```

Delay specification

All the circuits we have studied so far have no delay associated with them, these are called 0-delay circuits. Real circuits however, always have a delay between their input and output. Verilog allows modelling of delays at various level of abstraction using delay statements.

Syntax for specifying a delay is:

```
<gate_primitive> #(<delay>) <inst_name>(...ports...)
```

For example:

```verilog
/* 2-input AND gate with 1 time unit delay */
and #(1) a1(result, input1, input2);
```

To specify unit of delay we'll write a compiler directive, this is typically written at the begining of the description.

```verilog
`timescale 1ns/1ps
```

Here 1ns is the time unit while 1ps is the time resolution. Which will be explained in later assignment. A complete example using delay statements is given in Listing 3.

Listing 3: Example usage of delays statement to specify propagation delay of logic gates.

```verilog
/* Compiler directive to specify time unit, which will be
 * used for assigning propogation delays of logic gates.
 */
`timescale 1ns/1ps

module delayExample(input1, input2, result);
   input input1, input2;
   output result;

   // Specify a NAND gate having propogation delay of 2ns
   nand #(2) nd1(result, input1, input2);
endmodule // delayExample
```

Data-flow Modelling

Data flow modelling is a higher level of abstraction. Describing a circuit using data-flow modelling does not require knowledge of gates level circuit, thus it is easier than gate-level modelling when description of large scale circuits are written. All the examples from Assignment 1 used data flow modelling.

Continuous Assignment

Continuous assignment in Verilog are used for data flow modelling, these assignment starts with `assign` keyword. Continuous assignment drives value into a net (`wire`). Following example describes use of continuous assignment:

Note: Verilog is concurrent language unlike programming languages such as C,C++ or Java. All the continuous assignments are evaluated at the same time.

Listing 4: Example usage of continuous assignment.

```
/*
 * Following module uses continuous assignement to model
 * an AND gate
 */
module continuousAssignment(input1, input2, result);
   input input1, input2;
   output result;

   /* Use continuous assignment to set result
    * This is equalent to:
    *     and (result, input1, input2);
    */
   assign result = input1 & input2;
endmodule // continuousAssignment
```

Continuous assigment in Verilog can also be done implicitly, which assigning value on declaration of a net (`wire`). Implicit delcaration of Verilog is down as follows:

```
wire new_wire = input1 & input2;
```

Assignment Delays

Similar to gate-level modelling, Verilog allows specifying delays in assignment to model real circuits. Assignment delay specify the delay between the change of LHS and RHS of a continuous assignment. Listing 5 shows example usage of assignment delay while Fig. 1 shows simulation result of Listing 5.

Listing 5: Using assignment delay in Verilog.

```
`timescale 1ns/1ps

module assignmentDelay(input1, input2, result);
   input input1, input2;
   output result;

   // Defines a circuit which will have output value
   // input1 | input2 15ns after changing input
   assign #15 result = input1 | input2;
endmodule // assignmentDelay
```

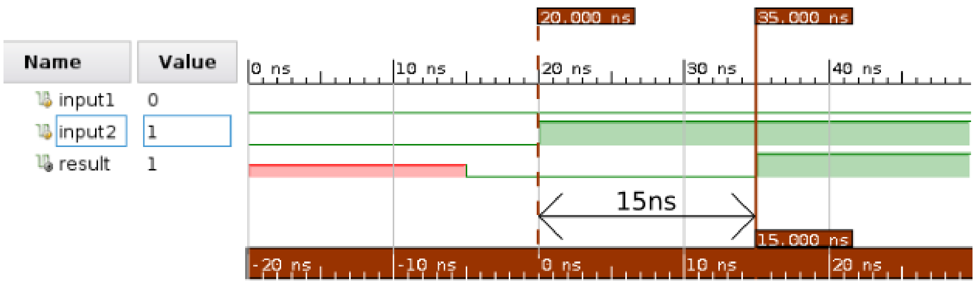

Figure 1: Simulation result for Listing 5, output changes 15ns after the change in input.

References

- http://inst.eecs.berkeley.edu/~cs150/fa08/Documents/Always.pdf

- Aenean in sem ac leo mollis blandit.