

Lecture 1: Verilog HDL Introduction

1

What is Verilog HDL?

- Verilog Hardware Description Language(HDL)?
 - A high-level computer language can model, represent and simulate digital design
 - Hardware concurrency
 - Parallel Activity Flow
 - Semantics for Signal Value and Time
 - Design examples using Verilog HDL
 - Intel Pentium, AMD K5, K6, Atheon, ARM7, etc
 - Thousands of ASIC designs using Verilog HDL

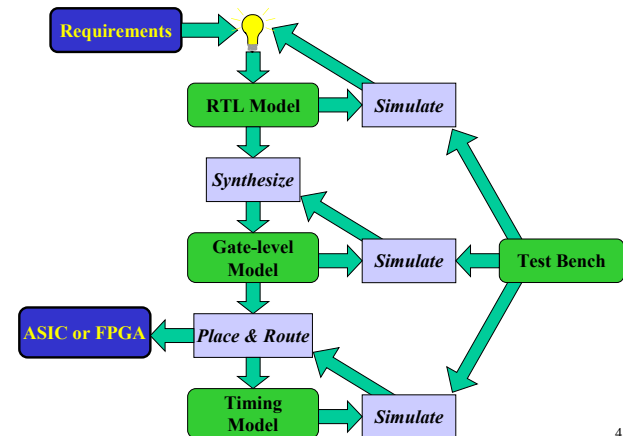
2

What is VHDL?

- VHDL represents another high level language for digital system design.
- In this course we study Verilog HDL
 - reason:
 - used more often in electronic and computer industry
 - programming style is very similar to C programming language

3

Basic Design Methodology



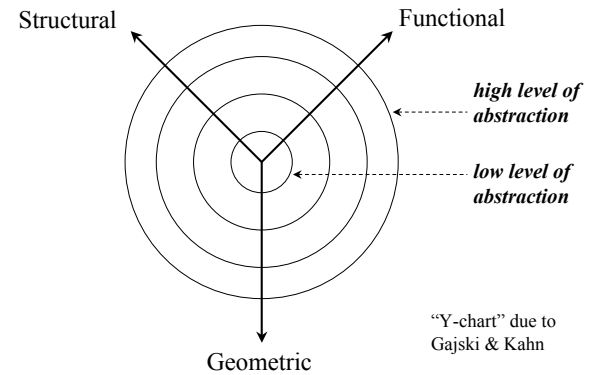
4

Modeling Digital Systems

- Verilog HDL is for writing models of a system
- Reasons for modeling
 - requirements specification
 - documentation
 - testing using simulation
 - formal verification
 - synthesis
- Goal
 - most reliable design process, with minimum cost and time
 - avoid design errors!

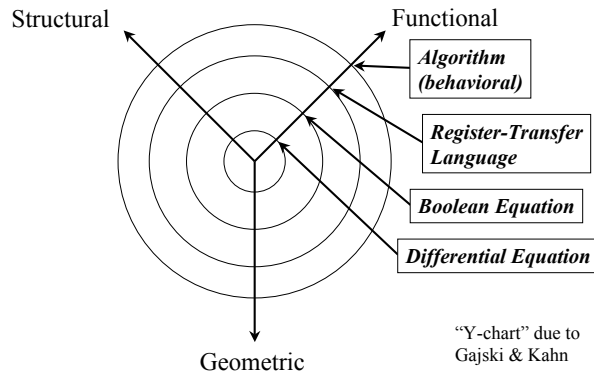
5

Domains and Levels of Modeling



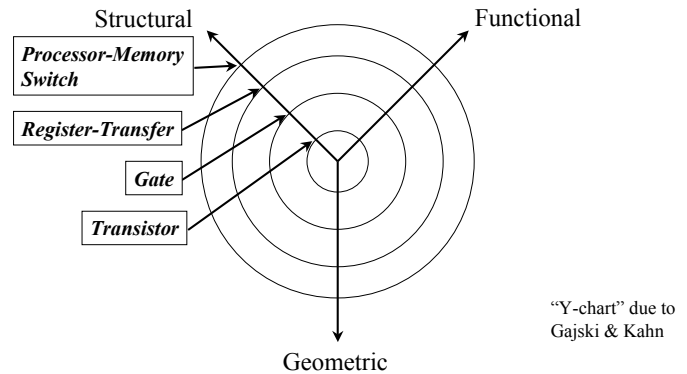
6

Domains and Levels of Modeling



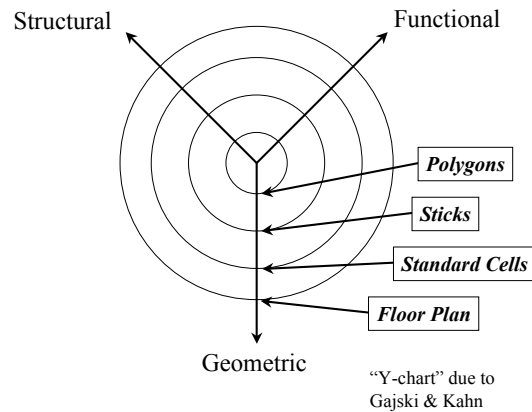
7

Domains and Levels of Modeling



8

Domains and Levels of Modeling



9

Verilog HDL Models

- HDL model specifies the relationship between input signals and output signals
- HDL uses special constructs to describe hardware concurrency, parallel activity flow, time delays and waveforms

Verilog code for a AND gate

```
module and_gate(y, x1, x2);
  input x1, x2;
  output y;
  and(y, x1, x2);
endmodule
```

10

Verilog Examples

```
module Add_half(sum, c_out, a, b);
  output sum, c_out;
  input a, b;
  wire c_out_bar;
  xor G1(sum, a, b);
  nand G2(c_out_bar, a, b);
  not G3(c_out, c_out_bar);
endmodule
```

* the instance name of Verilog primitives is optional.

11

Verilog Example: A D Flip Flop

```
module Flip_Flop(q, data_in, clk, rst);
  input data_in, clk, rst;
  output q;
  reg q;
  always @(posedge clk)
  begin
    if (rst == 1) q = 0;
    else q = data_in;
  end
endmodule
```

12

Verilog Example: behavioral model

```
module adder_4_RTL (a, b, c_in, sum, c_out);
    output [3:0] sum;
    output c_out;
    input [3:0] a, b;
    input c_in;

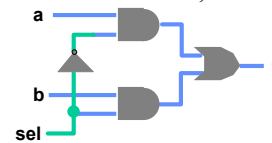
    assign {c_out, sum} = a + b + c_in;

endmodule
```

13

Verilog Example: Structural Models

- Structural models
 - Are built from gate primitives and/or other modules
 - They describe the circuit using logic gates — much as you would see in an implementation of a circuit.
 - You could describe your assignment1 circuit this way
- Identify
 - Gate instances, wire names, delay from *a* or *b* to *f*.



```
module mux (f, a, b, sel);
    output f;
    input a, b, sel;

    and #5 g1 (f1, a, nsel),
    g2 (f2, b, sel);
    or #5 g3 (f, f1, f2);
    not g4 (nsel, sel);
endmodule
```

14

Verilog Example: Gate-Level Models

- Need to model the gate's:
 - Function
 - Delay
- Function
 - Generally, HDLs have built-in gate-level primitives
 - Verilog has NAND, NOR, AND, OR, XOR, XNOR, BUF, NOT, and some others
 - The gates operate on input values producing an output value
 - typical Verilog gate instantiation is:

```
and #delay optional instance-name (out, in1, in2, in3, ...);
```

15

How to build and test a module

Construct a “test bench” for your design

- Develop your hierarchical system within a module that has input and output ports (called “**design**” here)
- Develop a separate module to generate tests for the module (“**test**”)
- Connect these together within another module (“**testbench**”)

16

```

module testbench ();
  wire  l, m, n;

  design d (l, m, n);
  test   t (l, m);

  initial begin
    //monitor and display
    ...
  end

```

```

module design (a, b, c);
  input  a, b;
  output c;
  ...

```

```

module test (q, r);
  output q, r;

  initial begin
    //drive the outputs with signals
    ...
  end

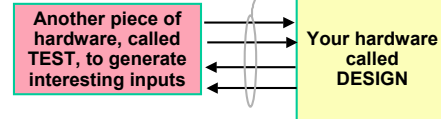
```

17

Another view of this

- 3 chunks of verilog, one for each of:

TESTBENCH is the final piece of hardware which connect DESIGN with TEST so the inputs generated go to the thing you want to test...



18

Verilog Examples

Module testAdd generated inputs for module halfAdd and displayed changes. Module halfAdd was the *design*

```

module tBench;
  wire  su, co, a, b;

  halfAdd ad(su, co, a, b);
  testAdd tb(a, b, su, co);
endmodule

```

```

module halfAdd (sum, cOut, a, b);
  output sum, cOut;
  input  a, b;

  xor #2 (sum, a, b);
  and #2 (cOut, a, b);
endmodule

```

```

module testAdd(a, b, sum, cOut);
  input  sum, cOut;
  output a, b;
  reg   a, b;

  initial begin
    $monitor ($time,,
      "a=%b, b=%b, sum=%b, cOut=%b",
      a, b, sum, cOut);
    a = 0; b = 0;
    #10 b = 1;
    #10 a = 1;
    #10 b = 0;
    #10 $finish;
  end
endmodule

```

19

The test module

- It's the test generator
- \$monitor
 - prints its string when executed.
 - after that, the string is printed when one of the listed values changes.
 - only one monitor can be active at any time
 - prints at end of current simulation time

```

module testAdd(a, b, sum, cOut);
  input  sum, cOut;
  output a, b;
  reg   a, b;

  initial begin
    $monitor ($time,,
      "a=%b, b=%b, sum=%b, cOut=%b",
      a, b, sum, cOut);
    a = 0; b = 0;
    #10 b = 1;
    #10 a = 1;
    #10 b = 0;
    #10 $finish;
  end
endmodule

```

20

The test module (continued)

- Function of this tester
 - at time zero, print values and set a=b=0
 - after 10 time units, set b=1
 - after another 10, set a=1
 - after another 10 set b=0
 - then another 10 and finish

```
module testAdd(a, b, sum, cOut);
  input  sum, cOut;
  output a, b;
  reg    a, b;

  initial begin
    $monitor ($time,,
      "a=%b, b=%b, sum=%b, cOut=%b",
      a, b, sum, cOut);
    a = 0; b = 0;
    #10 b = 1;
    #10 a = 1;
    #10 b = 0;
    #10 $finish;
  end
endmodule
```

21

Other things you can do

- More than modeling hardware
 - \$monitor — give it a list of variables. When one of them changes, it prints the information. Can only have one of these active at a time.
 - e.g. ...
 - \$monitor (\$time,, "a=%b, b=%b, sum=%b, cOut=%b", a, b, sum, cOut);
- extra commas
print a spaces
- %b is binary (also, %h, %d and others)
- The above will print:
 - 2 a=0, b=0, sum=0, cOut=0<return>
- newline automatically included
- \$display() — sort of like printf()
 - \$display ("Hello, world — %h", hexvalue)
- display contents of data item called "hexvalue" using hex digits (0-9,A-F)

22

Structural vs Behavioral Models

- Structural model
 - Just specifies primitive gates and wires
 - i.e., the structure of a logical netlist
 - You basically know how to do this now.
- Behavioral model
 - More like a procedure in a programming language
 - Still specify a module in Verilog with inputs and outputs...
 - ...but inside the module you write code to tell what you want to have happen, NOT what gates to connect to make it happen
 - i.e., you specify the behavior you want, not the structure to do it
- Why use behavioral models
 - For testbench modules to test structural designs
 - For high-level specs to drive logic synthesis tools

23

Summary

- Model hardware at different levels of abstraction
- Mix different levels of abstraction in description and simulation
- Able to model hardware concurrency
- Support Hierarchical decomposition
- Availability of ASIC Foundry Support

24