Introduction to Verilog

Nanditha Rao

Verilog - References

- Verilog HDL: A Guide to Digital Design and Synthesis, By Samir Palnitkar
- Verilog® HDL Quick Reference Guide – by Sutherland
- Quick reference guide: https://web.stanford.edu/class/ee183/handouts_win2003/VerilogQuickRef.pdf

Verilog Contents

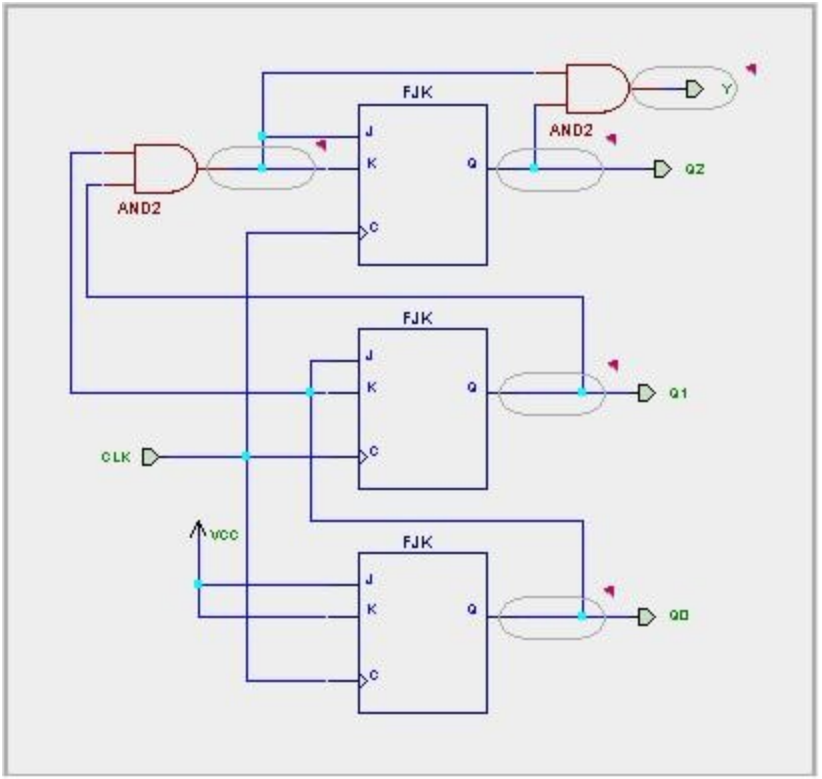
Modules and ports

Structural design

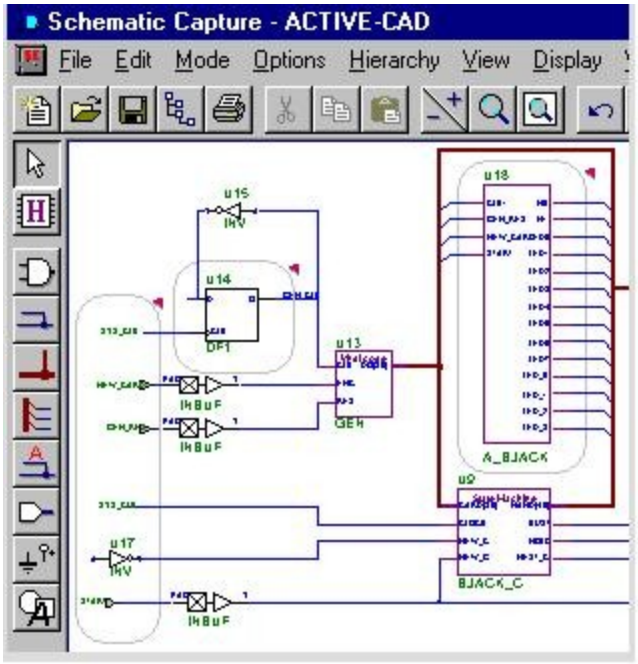
Behavioral design

Synthesis

Traditional Design approaches

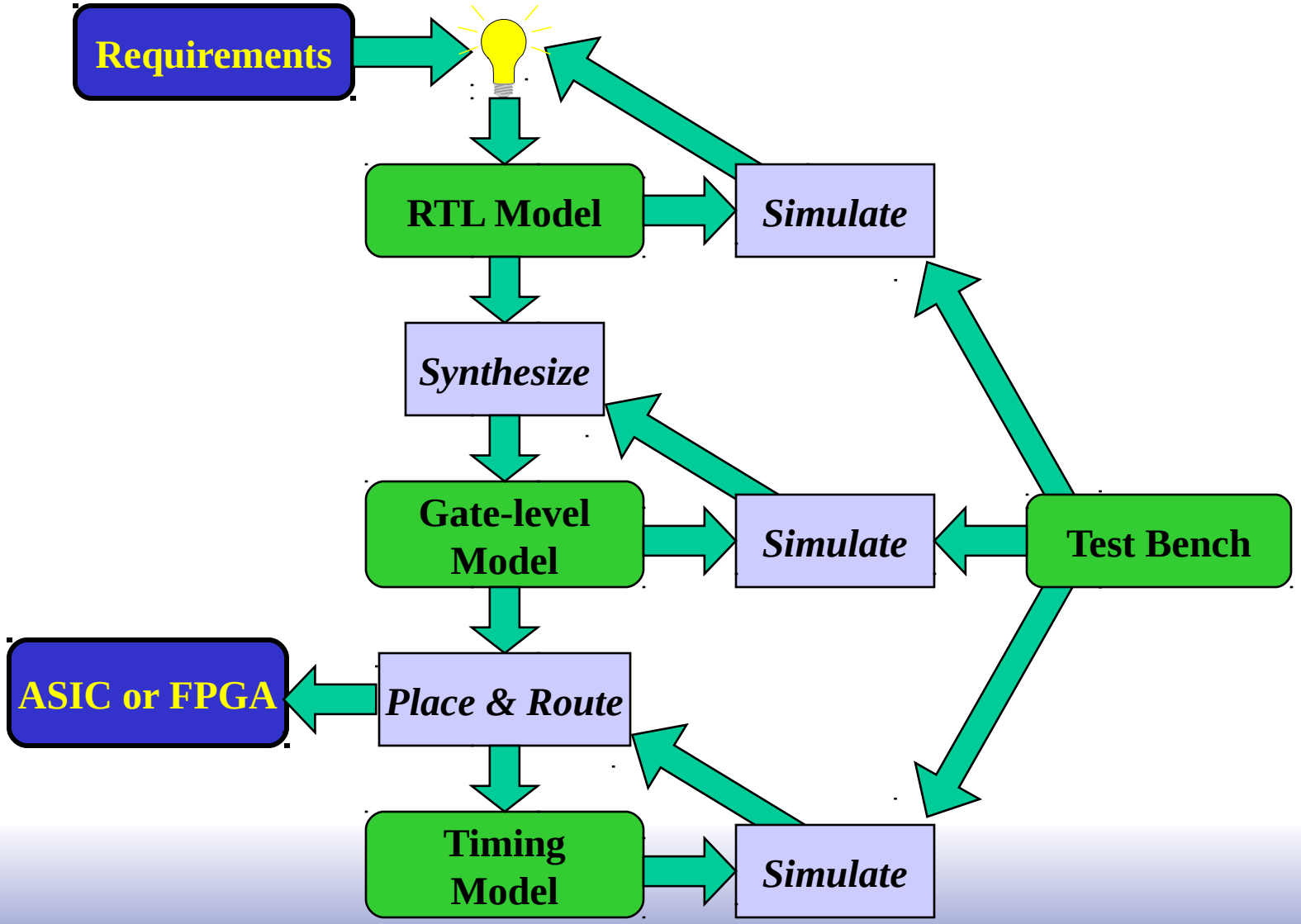


Gate Level Design



Schematic Design

Basic Design Methodology



Verilog/VHDL

- The “standard” languages
- Very similar
  - Many tools provide front-ends to both
  - Verilog is “simpler”
    - Simple syntax, fewer constructs
  - VHDL supports large, complex systems
    - Better support for modularization

Definition of Module

- Entire code is within the module
- Each port in the port list is defined as input, output, or inout

name

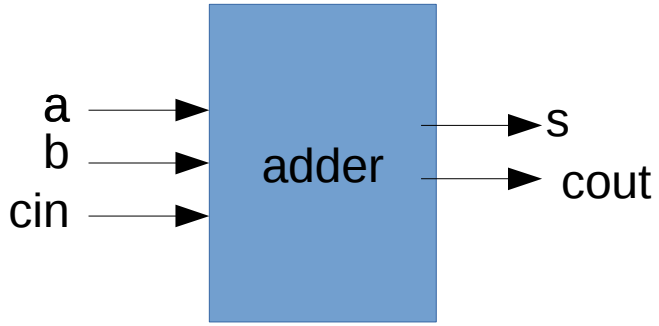
port list

```
module adder (a, b, cin, s, cout);
  input     a, b, cin;
  output    s, cout;

module body

endmodule
```

keywords

port declarations (input, output, or inout)

These are one-bit ports: 0 or 1



Verilog Introduction

- the **`module`** describes a component in the circuit
- Two ways to describe:
  - Structural Verilog
    - List of components and how they are connected
    - Just like schematics, but using text
    - Hard to write, hard to decode
    - Useful if you don't have integrated design tools
  - Behavioral Verilog
    - Describe *what* a component does, not *how* it does it
    - Synthesized into a circuit that has this behavior

Structural Model - XOR example

```
module xor_gate ( out, a, b );
  input     a, b;
  output     out;
   wire     aBar, bBar, t1, t2;

  not invA (aBar, a);
  not invB (bBar, b);
  and and1 (t1, a, bBar);
  and and2 (t2, b, aBar);
  or  or1 (out, t1, t2);

endmodule
```

module name

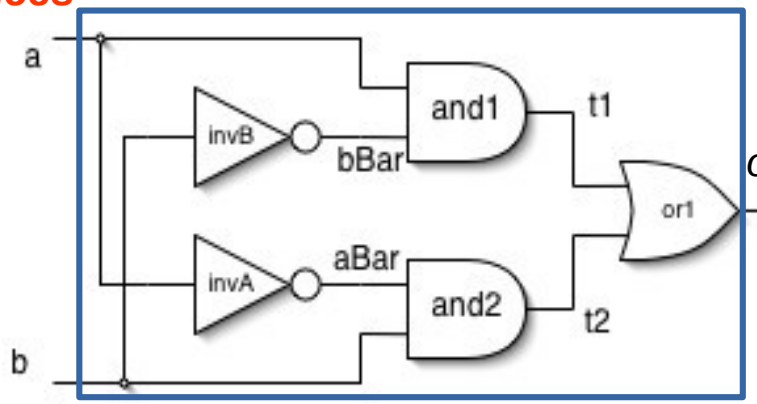
port list

port declarations

Wires: internal signals or connections

Built-in gates

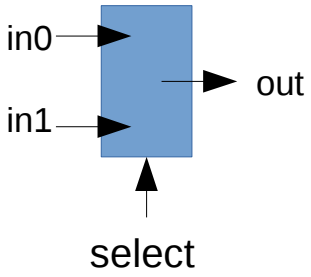
instances

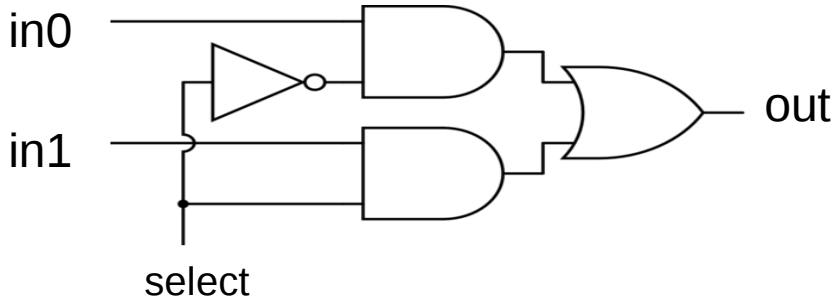


Interconnections (note output is first)

Instance name

Structural Example: 2-to1 mux

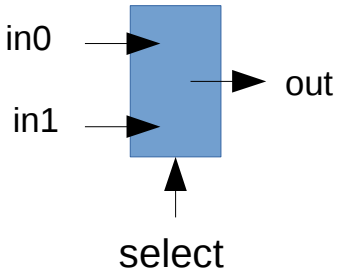


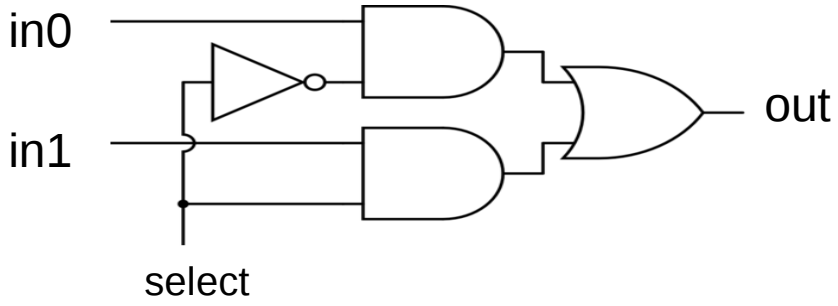



```
/* 2-input multiplexor in gates */
module mux2 (in0, in1, select, out);
   input in0,in1,select;
    output out;
   wire s0,w0,w1;

   _____________
   _____________
   ___________________
   _____________

endmodule // mux2
```

comments

Structural Example: 2-to1 mux






```
/* 2-input multiplexor in gates */
module mux2 (in0, in1, select, out);
   input in0,in1,select;
    output out;
   wire s0,w0,w1;

   not n1 (s0, select);
   and a1 (w0, s0, in0);
   and A2 (w1, select, in1);
   Or  e2 (out, w0, w1);

endmodule // mux2
```

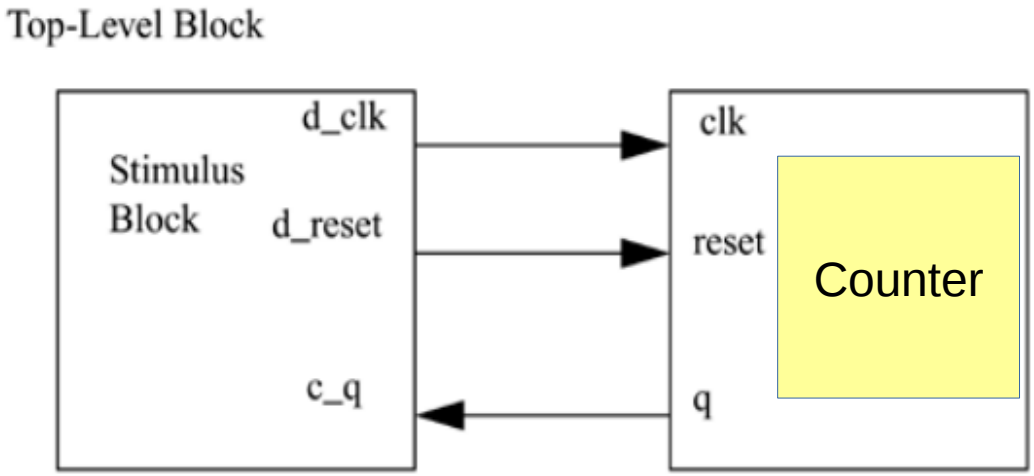
Built-ins gates can have > 2 inputs. Ex:

```
and (w0, a, b, c, d);
```

Concurrent

- All statements in Verilog are concurrent (executed simultaneously)- unless they are inside a sequential block
- All modules inside a file run concurrently

Testbench or Stimulus



- Connect the design to a test module
- Test module or test bench provides input signals to verify the design
- The design module cannot be described inside a testbench
- Testbench is only meant to provide stimulus/ test inputs

Class of Signals

- Nets: interconnection between hardware elements
- nets have values continuously driven on them by the outputs of device
- Eg - wire a
- If a net has no driver, it gets the value z.

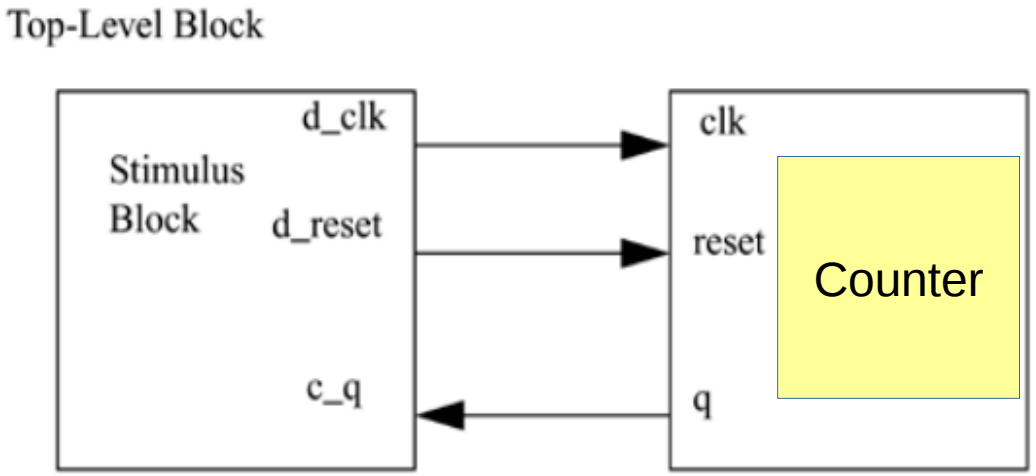
Reg

- Reg- retains value until another value is placed onto them.
- Do not confuse the term registers in Verilog with hardware registers built from flipflops in real circuits.
- Reg is a variable that can hold a value. Unlike a net, a register does not need a driver.

Eg of reg with delays

```
reg reset; // declare a variable reset that can hold its value
initial
begin
reset = 1'b1; //initialize reset to 1 to reset the digital circuit.
#100 reset = 1'b0; // after 100 time units reset is deasserted.
end
```

Reg and Wire



In a testbench

inputs are reg type: Values are assigned to it. Needs to hold values

outputs are wire type: Connects to the output port. Must be driven by a driver

`timescale: Time unit/precision unit

```
`timescale 10ns/1ps
module tim();
reg i;
initial
        begin
        i=0;
        #7.7;
        i=1;
        #10;
        i=0
        end
endmodule
```

```
`timescale 10ns/2ns
module tim();
reg i;
initial
        begin
        i=0;
        #7.7;
        i=1;
        #1.5;
        i=0
        end
endmodule
```

7.7*10 =77 --> Round out to nearest multiple of 2ns --> 78ns

Broad classification

- Verilog
  - Structural verilog design
    - Concurrent, not sequential
    - Does not contain any signal assignment statements. Based purely on netlist structure
  - Behavioral or sequential verilog design
    - Sequential
    - Based purely on signal assignment statements such as if/else, for, while loops etc

Verilog Assignments

- Assignments always happen to reg, not to wires
- a=1, b=0 etc
- Assignments can be made only inside
  - Initial block: Executed only at time t=0
  - always block: Executed always
  - These are sequential statements

Initial block

- Starts at time 0, executes exactly once during a simulation
- Contents inside the block are sequential
- Multiple initial blocks--> each block starts to execute concurrently at time 0

```
module stimulus;
initial
begin
#5 a = 1'b1;
#25 z = 1'b0;
end
endmodule
```

Assignments always happen to reg

Assignments always happen in initial/always blocks

Simple Behavioral Model: the `always` block

- `always` block
  - starts at time 0 and executes continuously in a loop
  - Always waiting for a change to a trigger signal
    - Then executes the body

```
module and_gate (out, in1, in2);
  input   in1, in2;
  output  out;
  reg     out;

  always @(in1 or in2) begin
     out = in1 & in2;
  end
endmodule
```

Not a real register!!
A Verilog register
Needed because of assignment in always block

Trigger-→ Specifies when block is executed I.e., triggered by which signals

`always` Block

- Procedure that describes the function of a circuit
  - Can contain many statements including `if`, `for`, `while`, `case`
  - Statements in the `always` block are executed *sequentially*
    - (Continuous assignments <= are executed in *parallel*)
  - `begin`/`end` used to group statements

Vectors – like arrays (multiple bit widths)

- wire a; // scalar net variable, default
- wire [7:0] bus; // 8-bit bus
- wire [31:0] busA,busB,busC; // 3 buses of 32-bit width.
- reg [255:0] data1; //Little endian notation
- reg [0:255] data2; //Big endian notation
- reg [7:0] byte;

Specific commands for iverilog installation in linux

Install using:

sudo add-apt-repository ppa:team-electronics/ppa

sudo apt-get update

sudo apt-get install iverilog gtkwave

Check if iverilog is already installed using command;

> iverilog

Compile and Run

- //compile using the following command:
- iverilog -o mux_out.out lab1-mux2.v lab1-mux2_tb.v

- //Run the binary using the following command:
- vvp mux_out.out
- Or using ./mux_out.out
-

- //Alternately you can just use the following command to run iverilog
- iverilog lab1-mux2.v lab1-mux2_tb.v --> in this case, a.out is created
- // Run this using the command ./a.out
-

- //View the file using the following command:
- gtkwave mux_out.vcd

Practice

1. Simulate the 2: 1 mux example with the testbench given to you
2. Create a 4:1 mux from the 2:1 mux and test it
3. Write a testbench to the full adder provided to you and simulate it with iverilog

   To be done later:

4. Create a 4 bit adder instantiating the full adder available
5. Write the Verilog code for a 2: 1 Mux which takes 16 bit inputs and produces a 16 bit output

Next session

- We will try to do debugging online!