
Simplify, clarify, verify!

*CSE/EE 40462: VLSI Design
Fall 2006*

HDL-Based Methodology for Designing VLSI Systems

Mike Ciletti

CSE/EE 40462 HDL-Based Methodology.1

Copyright 2006 Michael D. Ciletti, ND - 2006

Learning Objectives

- Acquire a basic knowledge of the Verilog HDL
 - Syntax and lexical conventions
 - Data types, operators, expressions, and assignments
 - Structural primitives
 - Structural and behavioral models
 - Testbenches for simulation and verification
- Read and write simple Verilog models of with basic constructs of the Verilog HDL
- Synthesize ASICs and FPGAs from HDL models
- Learn a methodology for designing, verifying, and synthesizing a FSM controller for a datapath in a digital system
- Write race-free, latch-free synthesizable models

CSE/EE 40462 Overview and Comb Logic.2

Copyright 2006 Michael D. Ciletti, ND - 2006

Roadmap

Part 1: Modeling, Verification and Synthesis of Combinational Logic

- Design Flow for HDL-Based Design Methodology
- Modeling, Verification, and Synthesis of Combinational Logic
- Structural decomposition and top-down design
- Verilog language constructs
- Test plan, test bench, simulation, and verification

- Part 2: Modeling, Verification, and Synthesis of Sequential Logic
 - RTL data flow constructs and operators
 - Control flow constructs
 - Sequential (=) vs. concurrent (<=) assignments
- Part 3: Synchronous Finite State Machines and Datapath Controllers
 - Mealy, Moore machines
 - Behavioral modeling for latch-free synthesis
 - Race-Free Synthesis of Datapath Controllers

CSE/EE 40462 Overview and Comb Logic.3

Copyright 2006 Michael D. Ciletti, ND - 2006

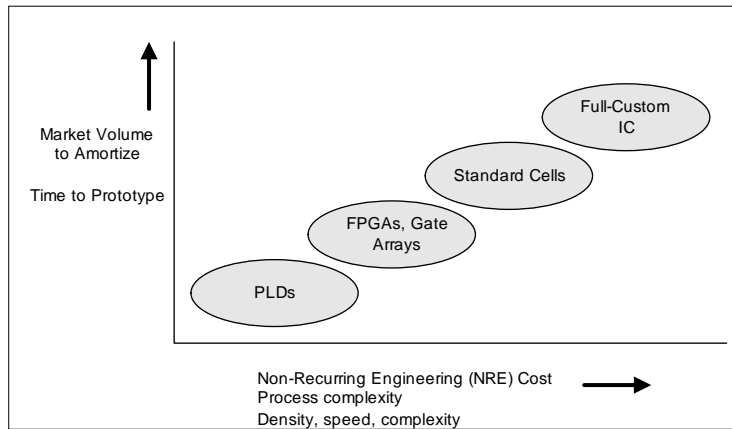
References

1. Palnitkar, S. 1996. *Verilog HDL: A Guide to Digital Design and Synthesis*. SunSoft Press (A Prentice Hall Title).
2. Bhasker, J. 1997. *A Verilog HDL Primer*. Allentown, PA: Star Galaxy Press.
3. Thomas, D. E., and P. R. Moorby. 1998. *The VeriLog Hardware Description Language* 4th ed. Boston: Kluwer Academic Publishers.
4. Bhasker, J. 1998. *Verilog HDL Synthesis*. Allentown, PA: Star Galaxy Press.
5. Ciletti, M. D. 1999. *Modeling, Synthesis, and Rapid Prototyping with Verilog HDL*. Upper Saddle River, NJ: Prentice Hall.
6. Ciletti, M. D. 2003. *Advanced Digital Design with the Verilog HDL*. Upper Saddle River, NJ: Prentice Hall.
7. Ciletti, M. D. 2004. *Starter's Guide to Verilog 2001*. Upper Saddle River, NJ: Prentice Hall.

CSE/EE 40462 Overview and Comb Logic.4

Copyright 2006 Michael D. Ciletti, ND - 2006

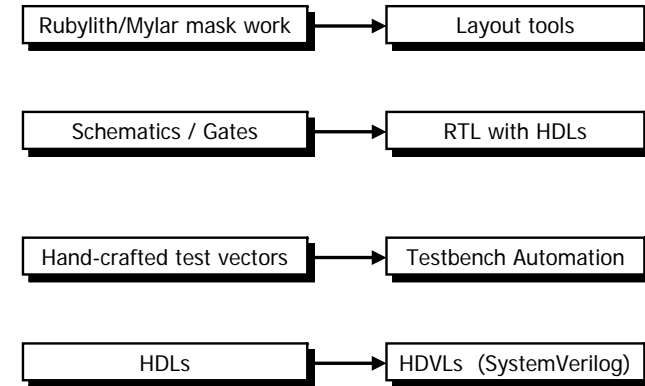
Technology Tradeoffs



CSE/EE 40462 Overview and Comb Logic.5

Copyright 2006 Michael D. Ciletti, ND - 2006

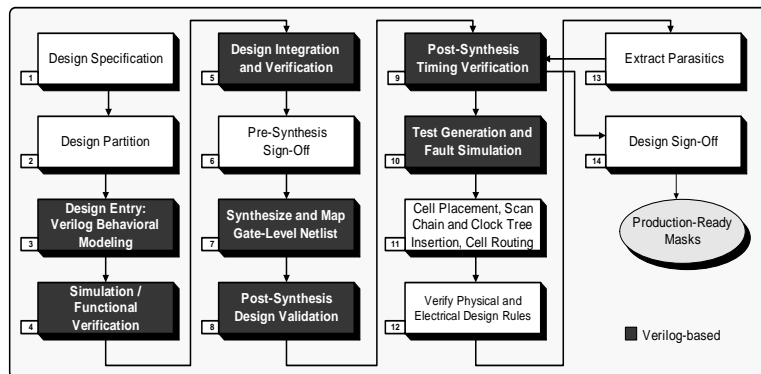
Paradigm Shifts



CSE/EE 40462 Overview and Comb Logic.6

Copyright 2006 Michael D. Ciletti, ND - 2006

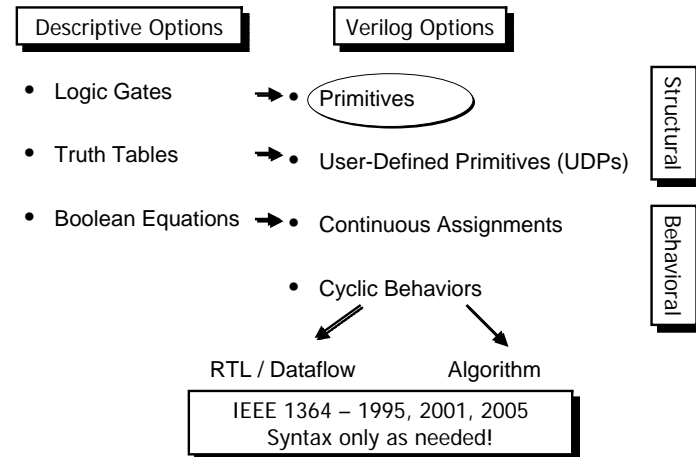
HDL-Based ASIC Design Flow



CSE/EE 40462 Overview and Comb Logic.7

Copyright 2006 Michael D. Ciletti, ND - 2006

Verilog Models: Combinational and Sequential Logic



CSE/EE 40462 Overview and Comb Logic.8

Copyright 2006 Michael D. Ciletti, ND - 2006

HDL Styles of Models

Example: Compare two 2-bit binary words:

$$A_lt_B = A1' B1 + A1' A0' B0 + A0' B1 B0$$

$$A_gt_B = A1 B1' + A0 B1' B0' + A1 A0 B0'$$

$$A_eq_B = A1' A0' B1' B0' + A1' A0 B1' B0 + A1 A0 B1 B0 + A1 A0' B1 B0'$$

- Classical approach: use K-maps to reduce the logic and produce the schematic
- Structural HDL approach: Connect primitives to describe the functionality implied by the schematic
- Behavioral HDL approach: Write an RTL/algorithm description of the functionality, then synthesize a physical implementation

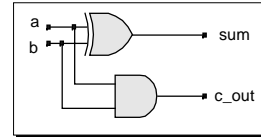
CSE/EE 40462 Overview and Comb Logic:9

Copyright 2006 Michael D. Ciletti, ND - 2006

HDL Example: Half Adder - Structural Model

Verilog primitives encapsulate pre-defined functionality of common logic gates.

- The counterpart of a schematic is a structural model composed of Verilog primitives
- The model describes relationships between outputs and inputs



```

module Add_half (sum, c_out, a, b);
  input      a, b;
  output    c_out, sum;
  xor      (sum, a, b);
  and      (c_out, a, b);
endmodule

```

```

module Add_half (output sum, c_out, input a, b); // ANSI C style
  xor (sum, a, b);
  and (c_out, a, b);
endmodule

```



CSE/EE 40462 Overview and Comb Logic:10

Copyright 2006 Michael D. Ciletti, ND - 2006

Primitives

Verilog has 26 built-in primitives (combinational):

n-Input	n-Output, 3-state
and	buf
nand	not
or	bufif0
nor	bufif1
xor	notif0
xnor	notif1

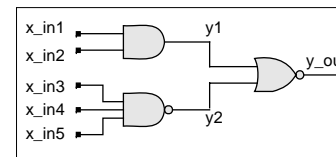
CSE/EE 40462 Overview and Comb Logic:11

Copyright 2006 Michael D. Ciletti, ND - 2006

Modeling with Primitives

Model structural detail by instantiating and connecting primitives.

Example:



```

wire y1, y2;
nor (y_out, y1, y2);
and (y1, x_in1, x_in2);
nand (y2, x_in3, x_in4, x_in5);

```

Modeling Tip

The output port of a primitive must be first in the list of ports.

CSE/EE 40462 Overview and Comb Logic:12

Copyright 2006 Michael D. Ciletti, ND - 2006

Design Encapsulation

Encapsulate structural and functional details in a **module**.

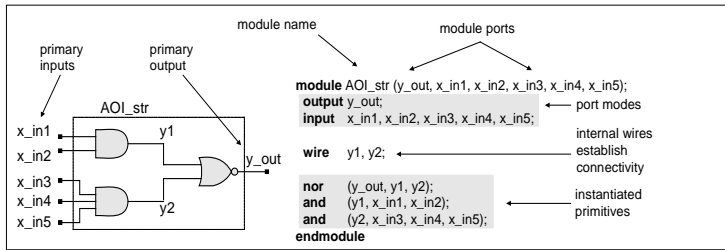
```

module my_design (module_ports);

... // Declarations here
... // Structural and functional details go here

endmodule
    
```

Encapsulation makes the model available for instantiation in other modules.



CSE/EE 40462 Overview and Comb Logic.13

Copyright 2006 Michael D. Ciletti, ND - 2006

Language Rules

- Verilog is a case sensitive language (with a few exceptions)
- Identifiers (space-free sequence of symbols)
 - upper and lower case letters from the alphabet
 - digits (0, 1, ..., 9)
 - underscore (_)
 - \$ symbol (only for system tasks and functions)
 - Max length of 1024 symbols
- Terminate lines with semicolon (;)
- Single line comments: // A single-line comment goes here
- Multi-line comments:

```
/* Do not /* nest multi-line comments*/ like this */
```

CSE/EE 40462 Overview and Comb Logic.14

Copyright 2006 Michael D. Ciletti, ND - 2006

Representation of Numbers

- Sized numbers specify the number of bits that are to be stored for a value
- Base specifiers:
 - b or B binary
 - d or D decimal (default)
 - o or O octal
 - h or H hexadecimal
- **Examples:**
 - 8'b1001_1101 // Use underscore for readability
 - 32'HABCD // Pads 0s
- Note Unsized numbers are stored as integers (at least 32 bits)

CSE/EE 40462 Overview and Comb Logic.15

Copyright 2006 Michael D. Ciletti, ND - 2006

Data Types

- Two families of data types for variables:
- Nets: **wire**, **tri**, **wand**, **triand**, **wor**, **trior**, **supply0**, **supply1**
- Registers: **reg**, **integer**, **real**, **time**, **realtime**
- Nets establish structural connectivity
- Register variables act as storage containers for the waveform of a signal
- Default size of a net or reg variable is a signal bit
- An **integer** is stored at a minimum of 32 bits
- **time** is stored as 64 bit integer
- **real** is stored as a real number
- **realtime** stores the value of time as a real number

CSE/EE 40462 Overview and Comb Logic.16

Copyright 2006 Michael D. Ciletti, ND - 2006

Verilog Keywords

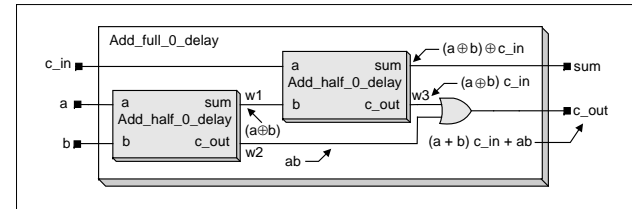
always	end	initial	output	rtran	tranif1
and	endcase	inout	parameter	rtranif0	tri
assign	endfunction	input	pmos	rtranif1	tri0
begin	endmodule	integer	posedge	scalared	tri1
buf	endprimitive	join	primitive	small	triand
bufif0	endspecify	large	pull0	specify	trior
bufif1	endtable	macromodule	pull1	specparam	trireg
case	endtask	medium	pulldown	strength	vectored
casex	event	module	pullup	strong0	wait
casez	for	nand	rcmos	strong1	wand
cmos	force	negedge	real	supply0	weak0
deassign	forever	nmos	realtime	supply1	weak1
default	fork	nor	reg	table	while
defparam	function	not	release	task	wire
disable	highz0	notif0	repeat	time	wor
edge	highz1	notif1	rnmos	tran	xnor
else	if	or	rmos	tranif0	xor

CSE/EE 40462 Overview and Comb Logic.17

Copyright 2006 Michael D. Ciletti, ND - 2006

Structural Decomposition – Top-Down Design (1 of 2)

Model complex structural detail by instantiating modules within modules



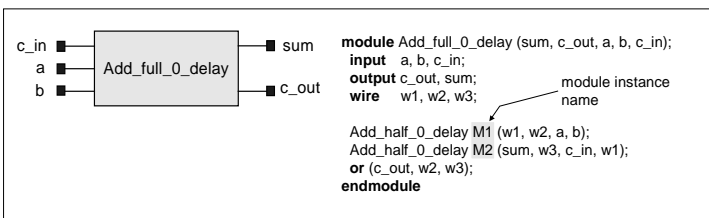
Modeling Tip

Use nested module instantiations to create a top-down design hierarchy.

CSE/EE 40462 Overview and Comb Logic.18

Copyright 2006 Michael D. Ciletti, ND - 2006

Structural Decomposition – Top-Down Design (2 of 2)



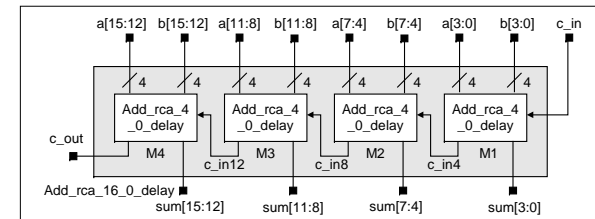
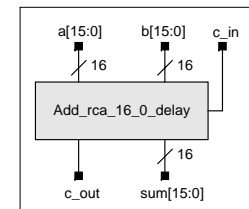
Modeling Tip

The ports of a module may be listed in any order.
The instance name of a primitive is optional.

CSE/EE 40462 Overview and Comb Logic.19

Copyright 2006 Michael D. Ciletti, ND - 2006

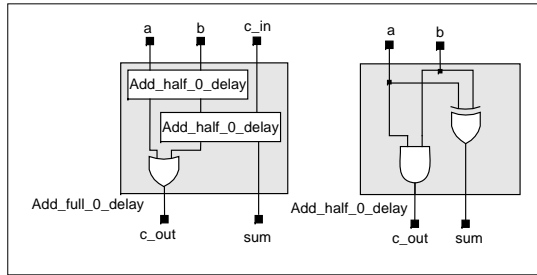
Example: 16-Bit Adder (1 of 2)



CSE/EE 40462 Overview and Comb Logic.20

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: 16-Bit Adder (2 of 2)



CSE/EE 40462 Overview and Comb Logic.21

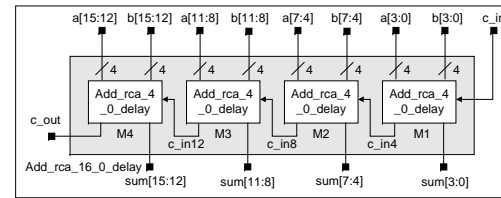
Copyright 2006 Michael D. Ciletti, ND - 2006

HDL Model: 16-Bit Adder (1 of 3)

```

module Add_rca_16_0_delay (
    output [15: 0] sum, output c_out, input [15: 0] a, b, input c_in
);
    wire
        c_in4, c_in8, c_in12, c_out;
    Add_rca_4_0_delay M1 (sum[3:0], c_in4, a[3:0], b[3:0], c_in);
    Add_rca_4_0_delay M2 (sum[7:4], c_in8, a[7:4], b[7:4], c_in4);
    Add_rca_4_0_delay M3 (sum[11:8], c_in12, a[11:8], b[11:8], c_in8);
    Add_rca_4_0_delay M4 (sum[15:12], c_out, a[15:12], b[15:12], c_in12);
endmodule
    
```

Verilog
2001



CSE/EE 40462 Overview and Comb Logic.22

Copyright 2006 Michael D. Ciletti, ND - 2006

HDL Model: 16-Bit Adder (2 of 3)

```

module Add_rca_4 (sum, c_out, a, b, c_in);
    output [3: 0] sum;
    output c_out;
    input [3: 0] a, b;
    input c_in;
    wire
        c_in2, c_in3, c_in4;
    Add_full_0_delay M1 (sum[0], c_in2, a[0], b[0], c_in);
    Add_full_0_delay M2 (sum[1], c_in3, a[1], b[1], c_in2);
    Add_full_0_delay M3 (sum[2], c_in4, a[2], b[2], c_in3);
    Add_full_0_delay M4 (sum[3], c_out, a[3], b[3], c_in4);
endmodule
    
```

CSE/EE 40462 Overview and Comb Logic.23

Copyright 2006 Michael D. Ciletti, ND - 2006

HDL Model: 16-Bit Adder (3 of 3)

```

module Add_full_0_delay(output sum, c_out, input a, b,
    c_in);
    wire
        w1, w2, w3;
    Add_half_0_delay M1 (w1, w2, a, b);
    Add_half_0_delay M2 (sum, w3, c_in, w1);
    or
        M3 (c_out, w2, w3);
endmodule

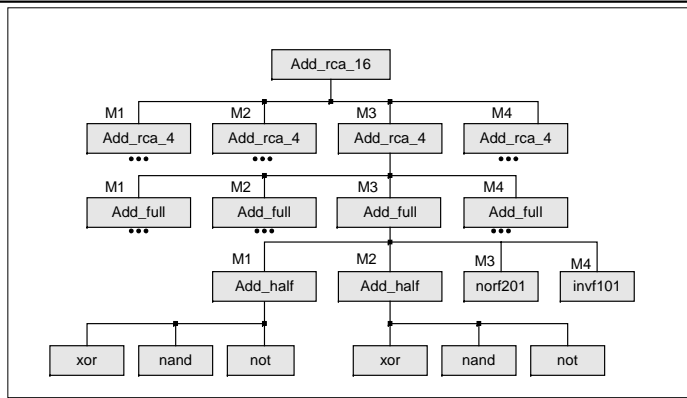
module Add_half_0_delay (output sum, c_out, input a, b);
    xor
        M1 (sum, a, b);
    and
        M2 (c_out, a, b);
endmodule
    
```

Verilog
2001

CSE/EE 40462 Overview and Comb Logic.24

Copyright 2006 Michael D. Ciletti, ND - 2006

Design Hierarchy: 16-Bit Adder



CSE/EE 40462 Overview and Comb Logic.25

Copyright 2006 Michael D. Ciletti, ND - 2006

Structural Connectivity

- Wires in Verilog establish connectivity between primitives and/or modules
- Data type: nets (Example: **wire**)
- The logic value of a **wire** (net) is determined dynamically during simulation by what is connected to the wire.

Modeling Tip

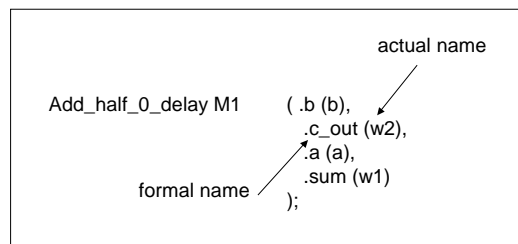
Use wires to establish structural connectivity.
An undeclared identifier is treated as a wire.

CSE/EE 40462 Overview and Comb Logic.26

Copyright 2006 Michael D. Ciletti, ND - 2006

Port Connection by Name

Connect ports by name in modules that have several ports.



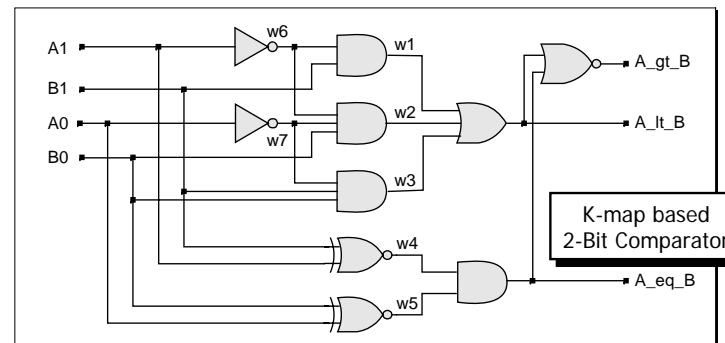
CSE/EE 40462 Overview and Comb Logic.27

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: 4-Bit Comparator (1 of 5)

Bottom-up design builds complex models from lower-level structural units.

Example: 4-Bit Comparator built from 2-Bit Comparators



CSE/EE 40462 Overview and Comb Logic.28

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: 4-Bit Comparator (2 of 5)

```

module compare_2_str (A_gt_B, A_lt_B, A_eq_B, A0, A1, B0, B1);
output A_gt_B, A_lt_B, A_eq_B;
input A0, A1, B0, B1;
// Note: w1, w2, ... are implicit wires

nor (A_gt_B, A_lt_B, A_eq_B);
or (A_lt_B, w1, w2, w3);
and (A_eq_B, w4, w5);
and (w1, w6, B1);
and (w2, w6, w7, B0);
and (w3, w7, B0, B1); // Note: interchanging w7, B0 and B1 has no effect
not (w6, A1);
not (w7, A0);
xnor (w4, A1, B1);
xnor (w5, A0, B0);
endmodule

```

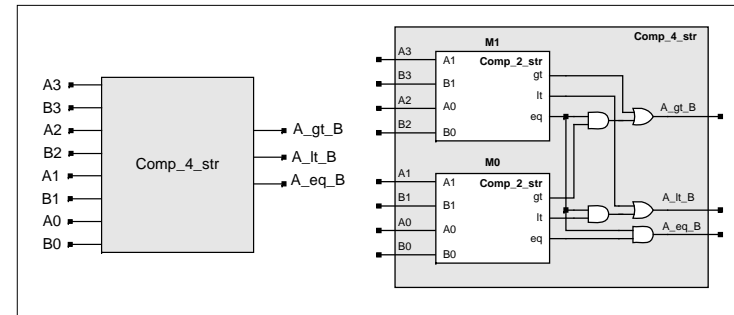
Note: n inputs are accommodated automatically.

CSE/EE 40462 Overview and Comb Logic.29

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: 4-Bit Comparator (3 of 5)

- Using 2-bit comparators, construct a 4-bit comparator.
- Note:** A strict inequality in the higher order bit-pair determines the relative magnitudes of the 4-bit words; if the higher-order bit-pairs are equal, the lower-order bit-pairs determine the output.



CSE/EE 40462 Overview and Comb Logic.30

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: 4-Bit Comparator (4 of 5)

Verilog Model:

```

module Comp_4_str (
output A_gt_B, A_lt_B, A_eq_B,
input A3, A2, A1, A0, B3, B2, B1, B0
);
wire w1, w0;
Comp_2_str M1 (A_gt_B_M1, A_lt_B_M1, A_eq_B_M1, A3, A2, B3, B2);
Comp_2_str M0 (A_gt_B_M0, A_lt_B_M0, A_eq_B_M0, A1, A0, B1, B0);
or (A_gt_B, A_gt_B_M1, w1);
and (w1, A_eq_B_M1, A_gt_B_M0);
and (A_eq_B, A_eq_B_M1, A_eq_B_M0);
or (A_lt_B, A_lt_B_M1, w0);
and (w0, A_eq_B_M1, A_lt_B_M0);
endmodule

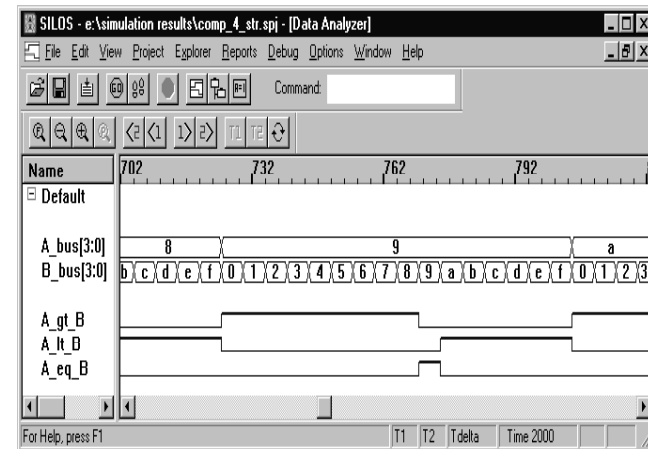
```

Verilog
2001

CSE/EE 40462 Overview and Comb Logic.31

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: 4-Bit Comparator (5 of 5)



CSE/EE 40462 Overview and Comb Logic.32

Copyright 2006 Michael D. Ciletti, ND - 2006

Logic System

- Four values: 0, 1, x or X, z or Z // Not case sensitive here
- Primitives have built-in 4-value logic
- Simulators describe 4-value logic

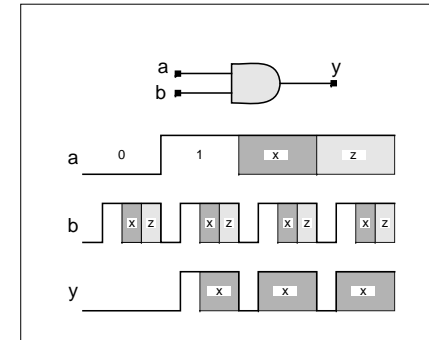
Modeling Tip

Logic value x denotes an unknown (ambiguous) value.
Logic value z denotes a high impedance.

CSE/EE 40462 Overview and Comb Logic.33

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: 4-Valued Logic Gate

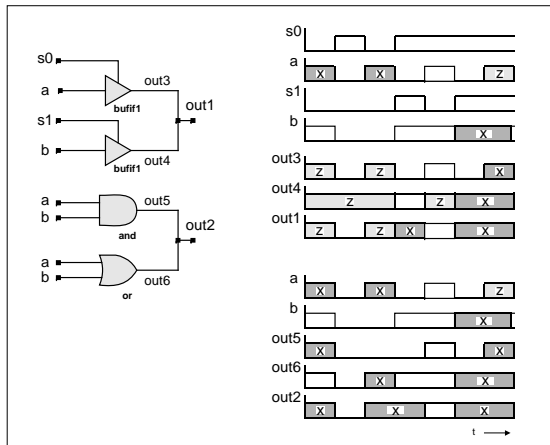


CSE/EE 40462 Overview and Comb Logic.34

Copyright 2006 Michael D. Ciletti, ND - 2006

Resolution of Contention

The value on a **wire** having multiple drivers in contention may be x.



CSE/EE 40462 Overview and Comb Logic.35

Copyright 2006 Michael D. Ciletti, ND - 2006

Wired Logic

How does Verilog treat multiple drivers?

- The family of nets includes the types **wand** and **wor**

A **wand** net type resolves multiple driver as wired-and logic (open-collector)

A **wor** net type resolves multiple drivers as wired-or logic (emitter-coupled)

How does Verilog model power and ground?

- The family of nets includes **supply0** and **supply1**

supply0 has a fixed logic value of 0 to model a ground connection
supply1 has a fixed logic value of 1 to model a power connection

CSE/EE 40462 Overview and Comb Logic.36

Copyright 2006 Michael D. Ciletti, ND - 2006

Model Simulation and Verification

Is it correct? Statically? Dynamically?

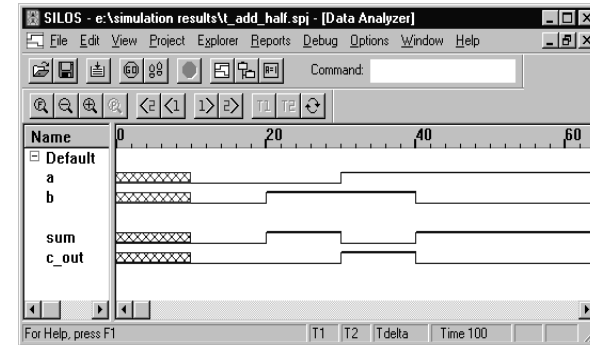
- Develop a test plan
 - What functional features will be tested?
 - How will they be tested?
 - How complete is the test?
- Execute a test plan
 - Simulation
 - Formal Verification

Hardware prototyping is not feasible.

CSE/EE 40462 Overview and Comb Logic.37

Copyright 2006 Michael D. Ciletti, ND - 2006

Event-Driven Simulation

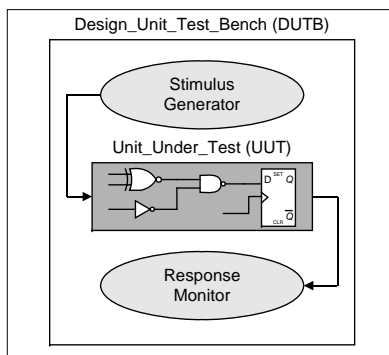


CSE/EE 40462 Overview and Comb Logic.38

Copyright 2006 Michael D. Ciletti, ND - 2006

Simulation

- Detect syntax violations in source code
- Simulate behavior
- Monitor results
- Verify functionality



CSE/EE 40462 Overview and Comb Logic.39

Copyright 2006 Michael D. Ciletti, ND - 2006

Preview: Behavioral Modeling with Verilog

- Three types of behaviors for composing abstract models
 - Continuous assignment (Keyword: **assign**) – Boolean logic
 - Single pass behavior (Keyword: **initial**) – Use only in testbenches
 - Cyclic behavior (Keyword: **always**) – Level-sensitive (combinational logic) and edge-sensitive (synchronous logic)
- Single pass and cyclic behaviors execute procedural statements like those executed by a programming language (**case**, **if**, **for**, **repeat** ...)
- Procedural assignment statements execute sequentially (=) or concurrently (<=), depending on the assignment operator
- A single pass behavior expires after the last statement executes
- A cyclic behavior executes again, repeatedly, after the last statement executes

CSE/EE 40462 Overview and Comb Logic.40

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: Testbench for Half Adder

```

module t_Add_half ( );
wire      sum, c_out;
reg      a, b;    // Storage containers for stimulus waveforms

Add_half_0_delay M1 (sum, c_out, a, b);    // UUT

initial begin #100 $finish; end           // Optional stopwatch

initial begin                               // Statements execute in sequence
    #10 a = 0; b = 0;                          // Execution delays accumulate
    #10 b = 1;
    #10 a = 1;
    #10 b = 0;                                // Executes at tsim = 40
end
endmodule
    
```

Delay control operator

Behavioral description of signals

Example: Alternative Testbench for Half Adder

```

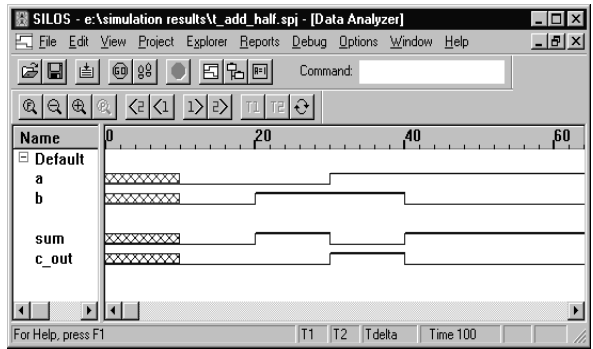
module t_Add_half ( );
wire      sum, c_out;
reg      a, b;    // Storage containers for stimulus waveforms

Add_half_0_delay M1 (sum, c_out, a, b);    // UUT

initial begin #100 $finish; end           // Optional stopwatch

initial fork                               // Statements execute in parallel
    #40 b = 0;                                // Executes at tsim = 40
    #10 a = 0; b = 0;
    #30 a = 1;
    #20 b = 1;                                // Executes at tsim = 20
    #50 begin a = 0; # 5 b = 1; end
join                                       // Expires at tsim = 55
endmodule
    
```

Event-Driven Simulation



- Modeling Tip
- Event-driven simulators update logic values only when signals changes.
 - Simulator assigns x to all **reg** type variables.
 - A **wire** inherits the value of its driver.

Abstract Signal Generators

- Use single-pass and cyclic behaviors to describe stimulus generators
- Statements in a behavior may be grouped in blocks
 - **begin ... end** // Sequential execution
 - **fork ... join** // Parallel execution
- Execution of behaviors begins at tsim = 0
- Delay control operator (#) temporarily suspends execution of a statement for a specified amount of time
- Event control operator (@) suspends execution of a statement until an event occurs
- The operator = denotes blocked procedural assignment (A blocked assignment cannot execute until the statement before it completes execution)

Modeling Tip

Use procedural assignments to describe stimulus patterns.

Template for Testbenches

```

module t_DUTB_name ( );           // Substitute the name of the UUT
  reg ...;                         // Declare regs for inputs of the UUT
  wire ...;                        // Declare wires for outputs of the UUT
  parameter time_out = 1;         // Provide a value

  UUT_name M1_instance_name ( UUT ports go here);

  initial $monitor ( );           // Specify signals to be displayed as text

  initial #time_out $finish;       // (Also $stop) Stopwatch

  initial                               // Stimulus patterns
  begin                                 // Alternative: fork ... join block
  // ...                               // Behavioral statements
  end                                   // Alternative: join
  ...
endmodule

```

CSE/EE 40462 Overview and Comb Logic.45

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: Clock Generator

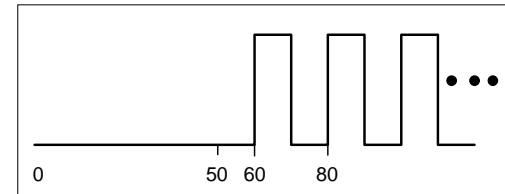
```

...
parameter latency = 50, half_cycle = 10;
...
initial #latency forever # half_cycle clock = ~ clock;
...

```

Single pass behavior

Unconditional loop



CSE/EE 40462 Overview and Comb Logic.46

Copyright 2006 Michael D. Ciletti, ND - 2006

Propagation Delay

- Gate propagation delay specifies the time between a change in an input and the resulting change in an output
- Transport delay describes the time-of-flight of a signal transition
- Verilog uses an inertial delay model for gates and transport delay for nets
- Inertial delay suppresses short pulses (width less than the propagation delay value)
- Primitives and nets have a default delay of 0

CSE/EE 40462 Overview and Comb Logic.47

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: Unit Delay Simulation (1 of 2)

Simulation with unit-delay models reveals the chain of events.

```

module Add_full (sum, c_out, a, b, c_in);
  output      sum, c_out;
  input      a, b, c_in;
  wire       w1, w2, w3;
  Add_half M1 (w1, w2, a, b);
  Add_half M2 (sum, w3, w1, c_in);
  or        #1 M3 (c_out, w2, w3); // Primitive with unit delay
endmodule

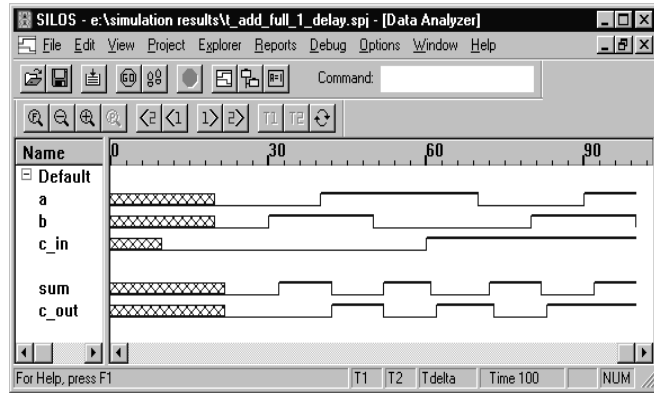
module Add_half (sum, c_out, a, b);
  output      sum, c_out;
  input      a, b;
  xor       #1 M1 (sum, a, b); // Single delay value format
  and      #1 M2 (c_out, a, b); // Others are possible
endmodule

```

CSE/EE 40462 Overview and Comb Logic.48

Copyright 2006 Michael D. Ciletti, ND - 2006

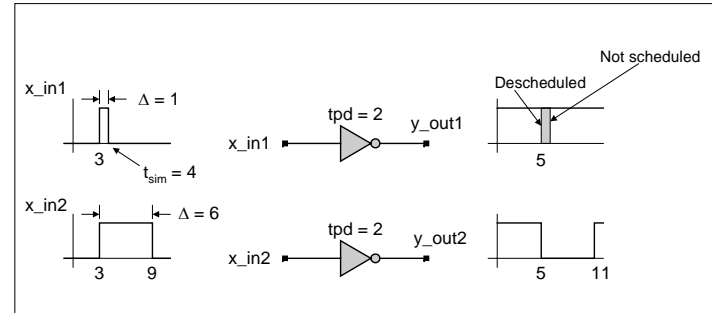
Example: Unit Delay Simulation (2 of 2)



CSE/EE 40462 Overview and Comb Logic.49

Copyright 2006 Michael D. Ciletti, ND - 2006

Inertial Delay



Note: The falling edge of x_{in1} occurs before the response to its rising edge occurs.

CSE/EE 40462 Overview and Comb Logic.50

Copyright 2006 Michael D. Ciletti, ND - 2006

Simulation with ASIC Standard Cells (1 of 2)

```

`timescale 1ns / 1 ps // time scale directive for units and resolution
module Add_full_ASIC (sum, c_out, a, b, c_in);
  output      sum, c_out;
  input       a, b, c_in;
  wire        w1, w2, w3;
  wire        c_out_bar;
  Add_half_ASIC M1 (w1, w2, a, b);
  Add_half_ASIC M2 (sum, w3, w1, c_in);
  norf201     M3 (c_out_bar, w2, w3);
  invf101     M4 (c_out, c_out_bar);
endmodule

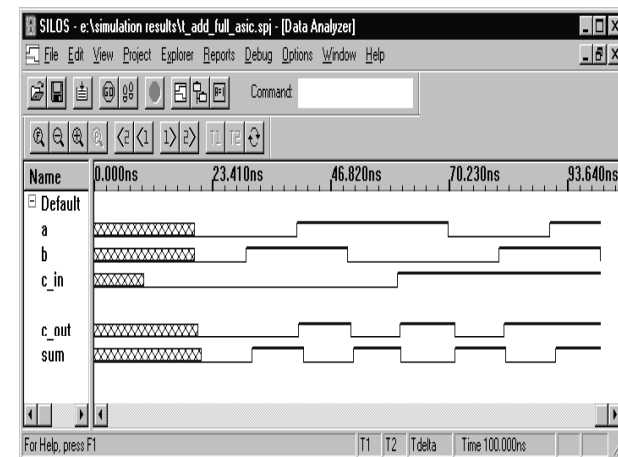
module Add_half_ASIC (sum, c_out, a, b);
  output      sum, c_out;
  input       a, b;
  wire        c_out_bar;
  xorf201     M1 (sum, a, b);
  nanf201     M2 (c_out_bar, a, b);
  invf101     M3 (c_out, c_out_bar);
endmodule
    
```

CMOS Standard cells

CSE/EE 40462 Overview and Comb Logic.51

Copyright 2006 Michael D. Ciletti, ND - 2006

Simulation with Standard Cells (2 of 2)



CSE/EE 40462 Overview and Comb Logic.52

Copyright 2006 Michael D. Ciletti, ND - 2006

Verilog Models: Combinational and Sequential Logic

Descriptive Options

- Logic Gates → • Primitives
- Truth Tables → • User-Defined Primitives (UDPs)
- Boolean Equations → • Continuous Assignments
- Cyclic Behaviors
 - ↙ RTL / Dataflow
 - ↘ Algorithm

Verilog Options

Structural

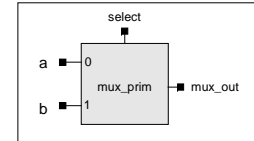
Behavioral

CSE/EE 40462 Overview and Comb Logic.53

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: User-Defined Primitive (UDP)

```
primitive mux_prim (mux_out, select, a, b);
output mux_out;
input select, a, b;
table
// sel a b : mux_out
0 0 0 : 0;
0 0 1 : 0;
0 0 x : 0;
0 1 0 : 1;
0 1 1 : 1;
0 1 x : 1;
1 0 0 : 0;
1 1 0 : 0;
1 x 0 : 0;
1 0 1 : 1;
1 1 1 : 1;
1 x 1 : 1;
// Remove pessimism
endtable
endprimitive
```



Single scalar output

Order of columns =
order of input ports

Table may not include z

```
x 0 0 : 0;
x 1 1 : 1;
endtable
endprimitive
```

CSE/EE 40462 Overview and Comb Logic.54

Copyright 2006 Michael D. Ciletti, ND - 2006

UDP Table Symbols

table

```
// ? represents iteration of the table entry over the values 0,1,x.
// i.e., don't care on the input
// sel ab : mux_out
// 0 0 ? : 0; // ? = 0, 1, x shorthand notation.
// 0 1 ? : 1;
// 1 ? 0 : 0;
// 1 ? 1 : 1;
// ? 0 0 : 0;
// ? 1 1 : 1;
```

endtable

CSE/EE 40462 Overview and Comb Logic.55

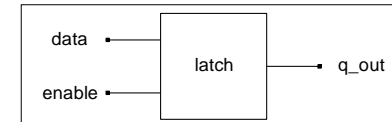
Copyright 2006 Michael D. Ciletti, ND - 2006

UDP Example - Transparent Latch

```
primitive latch (q_out, enable, data);
output q_out;
input enable, data;
reg q_out;
table
// enable data state q_out/next_state
1 1 : ? : 1;
1 0 : ? : 0;
0 ? : ? : -;

// Above entries do not deal with enable = x.
// Ignore event on enable when data = state:
x 0 : 0 : -;
x 1 : 1 : -;
: -;

// Note: The table entry '-' denotes no change of the output.
endtable
endprimitive
```



Additional column for
output (next-state).

The type of the output of a
sequential UDP must be **reg**.

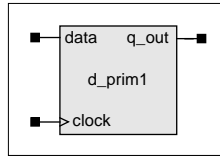
CSE/EE 40462 Overview and Comb Logic.56

Copyright 2006 Michael D. Ciletti, ND - 2006

UDP Example: D-Type Flip-Flop

- UDP notation for rising edge transition: (01), (0x), (x1)
- UDP notation for falling edge transition: (10), 1x), (x0)

```
primitive d_prim1 (q_out, clock, data);
output q_out;
input clock, data;
reg q_out;
```



```
table
// clk data : state : q_out/next_state
(01) 0 : ? : 0; // Rising clock edge
(01) 1 : ? : 1;
(0x) 1 : 1 : 1; // Reduce pessimism
(0x) 0 : 0 : 0;
(?0) ? : ? : -; // Falling or steady clock edge
? (??) : ? : -; // Steady clock, ignore data
// transitions
endtable
endprimitive
```

CSE/EE 40462 Overview and Comb Logic.57

Copyright 2006 Michael D. Ciletti, ND - 2006

Verilog Models: Combinational and Sequential Logic

Descriptive Options

Verilog Options

- Logic Gates → Primitives
 - Truth Tables → User-Defined Primitives (UDPs)
 - Boolean Equations → Continuous Assignments
 - Cyclic Behaviors
- RTL / Dataflow Algorithm

Structural

Behavioral

CSE/EE 40462 Overview and Comb Logic.58

Copyright 2006 Michael D. Ciletti, ND - 2006

Continuous Assignments

- Continuous assignments (Keyword: **assign**) are the Verilog counterpart of Boolean equations
- Hardware is implicit (e.g., combinational logic)

Example:

```
module AOI_5_CA0 (
output y_out, input x_in1, x_in2, x_in3, x_in4, x_in5
);
assign y_out = ~((x_in1 & x_in2) | (x_in3 & x_in4 & x_in5));
endmodule
```

Verilog 2001

The RHS expression is monitored, and the LHS variable is updated automatically when the RHS expression changes value.

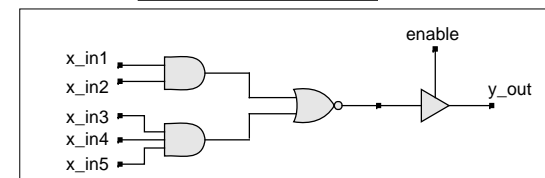
CSE/EE 40462 Overview and Comb Logic.59

Copyright 2006 Michael D. Ciletti, ND - 2006

Conditional Operator (? :)

```
module AOI_5_CA1 (
output y_out, x_in1, x_in2, x_in3, x_in4, x_in5, input enable
);
assign y_out =
enable ? ~((x_in1 & x_in2) | (x_in3 & x_in4 & x_in5)) : 1'bz;
endmodule
```

Verilog 2001

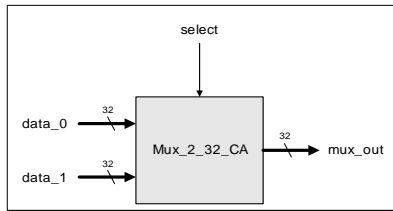


Note: The conditional operator requires both expressions.

CSE/EE 40462 Overview and Comb Logic.60

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: Parameterized/Portable/Reusable Model



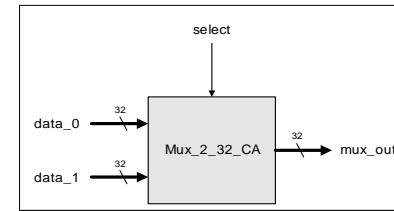
```

module Mux_2_32_CA ( mux_out, data_1, data_0, select);
parameter word_size = 32;
output [word_size -1: 0] mux_out;
input [word_size-1: 0] data_1, data_0;
input select;
assign mux_out = select ? data_1 : data_0;
endmodule
    
```

CSE/EE 40462 Overview and Comb Logic.61

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: V2001 Parameterized Model



```

module Mux_2_32_CA # (parameter word_size = 32) (
output [word_size -1: 0] mux_out,
input [word_size -1: 0] data_1, data_0, input select
);

assign mux_out = select ? data_1 : data_0;
endmodule
    
```

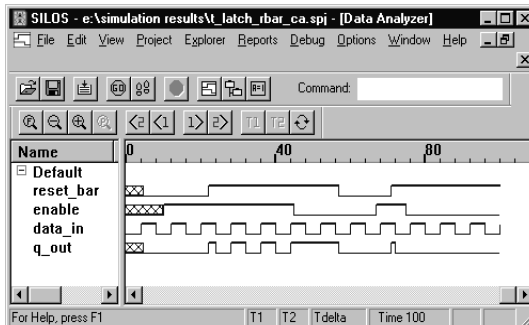
CSE/EE 40462 Overview and Comb Logic.62

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: T-Latch

```

module Latch_Rbar_CA (q_out, data_in, enable, reset_bar);
output q_out;
input data_in, enable, reset_bar;
assign q_out = !reset_bar ? 1'b0 : enable ? data_in : q_out;
endmodule
    
```



CSE/EE 40462 Overview and Comb Logic.63

Copyright 2006 Michael D. Ciletti, ND - 2006

Concurrent Continuous Assignments

Multiple continuous assignments are active concurrently.

Example: 2-Bit Comparator

```

module compare_2_CA0 (
output A_lt_B, A_gt_B, A_eq_B, input A1, A0, B1, B0
);
assign A_lt_B = (~A1) & B1 | (~A1) & (~A0) & B0 | (~A0) & B1 & B0;
assign A_gt_B = A1 & (~B1) | A0 & (~B1) & (~B0) | A1 & A0 & (~B0);
assign A_eq_B = (~A1) & (~A0) & (~B1) & (~B0) | (~A1) & A0 & (~B1)
& B0 | A1 & A0 & B1 & B0 | A1 & (~A0) & B1 & (~B0);
endmodule
    
```



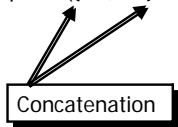
CSE/EE 40462 Overview and Comb Logic.64

Copyright 2006 Michael D. Ciletti, ND - 2006

Alternative Models (1 of 3)

```

module compare_2_CA1 (A_lt_B, A_gt_B, A_eq_B, A1, A0, B1, B0);
input  A1, A0, B1, B0;
output A_lt_B, A_gt_B, A_eq_B;
assign A_lt_B = ({A1, A0} < {B1, B0});
assign A_gt_B = ({A1, A0} > {B1, B0});
assign A_eq_B = ({A1, A0} == {B1, B0});
endmodule
    
```



Alternative Models (2 of 3)

```

module compare_2_CA1 (A_lt_B, A_gt_B, A_eq_B, A, B);
input  [1: 0]  A, B;
output  A_lt_B, A_gt_B, A_eq_B;

assign  A_lt_B = (A < B);      // RHS is true (1) or false (0)
assign  A_gt_B = (A > B);
assign  A_eq_B = (A == B);
endmodule
    
```

Alternative Models (3 of 3)

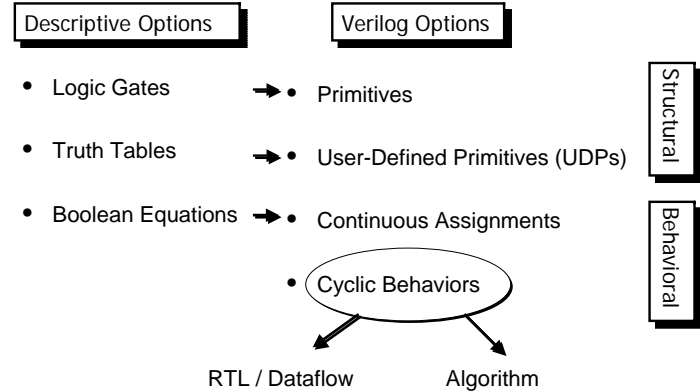
Example: 32-Bit Comparator

```

module compare_32_CA (A_gt_B, A_lt_B, A_eq_B, A, B);
parameter word_size = 32;
input      [word_size-1: 0]  A, B;
output    A_gt_B, A_lt_B, A_eq_B;
assign    A_gt_B = (A > B),    // List of multiple assignments
           A_lt_B = (A < B),
           A_eq_B = (A == B);
endmodule
    
```

The description is clear and the model is portable.
Use a synthesis tool to create the gate-level structure.

Verilog Models: Combinational and Sequential Logic



Modeling Combinational Logic With Cyclic Behaviors

Example: 32-Bit Comparator

```
module compare_32_CA # (parameter word_size = 32) (
  input [word_size-1: 0] A, B, output reg A_gt_B, A_lt_B, A_eq_B
);
```

Event control operator

Sensitivity list

```
always @ (A, B) begin
  A_gt_B = (A > B), // List of multiple procedural assignments
  A_lt_B = (A < B),
  A_eq_B = (A == B);
end
endmodule
```

The target of a procedural assignment in a cyclic behavior must be a declared **reg**.

CSE/EE 40462 Overview and Comb Logic.69

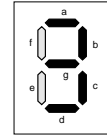
Copyright 2006 Michael D. Ciletti, ND - 2006

Example: Seven-Segment Display (1 of 2)

```
module Seven_Seg_Display (output reg [6: 0] Display, input [3: 0] BCD);
```

```
//          abc_defg

parameter BLANK = 7'b111_1111;
parameter ZERO  = 7'b000_0001; // h01
parameter ONE   = 7'b100_1111; // h4f
parameter TWO   = 7'b001_0010; // h12
parameter THREE = 7'b000_0110; // h06
parameter FOUR  = 7'b100_1100; // h4c
parameter FIVE  = 7'b010_0100; // h24
parameter SIX   = 7'b010_0000; // h20
parameter SEVEN = 7'b000_1111; // h0f
parameter EIGHT = 7'b000_0000; // h00
parameter NINE  = 7'b000_0100; // h04
```



CSE/EE 40462 Overview and Comb Logic.70

Copyright 2006 Michael D. Ciletti, ND - 2006

Example: Seven-Segment Display (2 of 2)

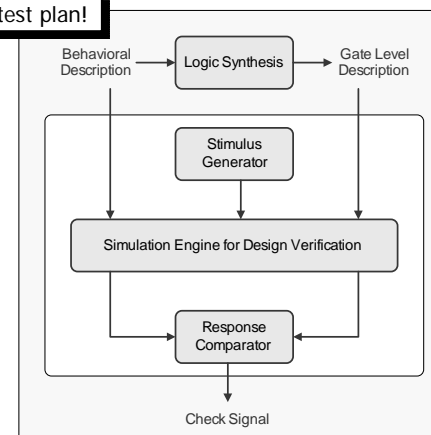
```
always @ ( BCD )
case (BCD)
0:      Display = ZERO;
1:      Display = ONE;
2:      Display = TWO;
3:      Display = THREE;
4:      Display = FOUR;
5:      Display = FIVE;
6:      Display = SIX;
7:      Display = SEVEN;
8:      Display = EIGHT;
9:      Display = NINE;
default: Display = BLANK;
endcase
endmodule
```

CSE/EE 40462 Overview and Comb Logic.71

Copyright 2006 Michael D. Ciletti, ND - 2006

Verification Methodology (1 of 3)

Develop a test plan!



CSE/EE 40462 Overview and Comb Logic.72

Copyright 2006 Michael D. Ciletti, ND - 2006

Verification Methodology (2 of 3)

- Guide the verification activities
- Specify the functional modes to be verified
- Identify the process for testing the functional modes
 - Blocks
 - Interfaces
- Specify the verification approach
 - Top-down, Bottom-up
 - Other
- Specify the verification technology
 - Abstraction levels (e.g. behavioral, gates)
 - Simulation, formal methods
 - Static timing analysis
- Identify the testbench elements (e.g., stimulus generators)
- Other

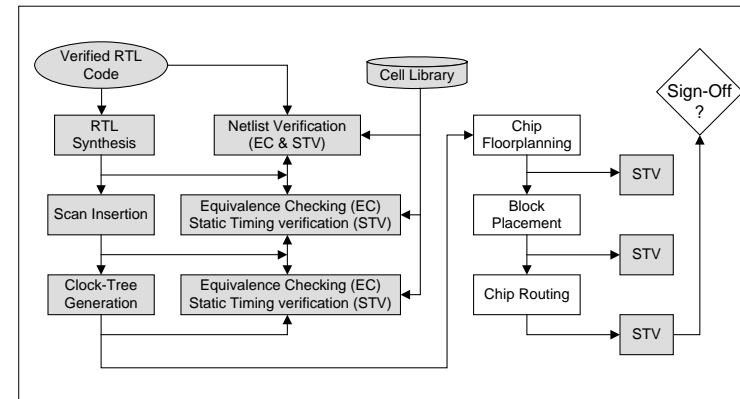
Bugs that reach the field eat profits.

Ref: Rashinagr P., et. al.
System-on-a-chip Verification,
 Boston: Kluwer, 2001

CSE/EE 40462 Overview and Comb Logic.73

Copyright 2006 Michael D. Ciletti, ND - 2006

Verification – The Bigger Picture



CSE/EE 40462 Overview and Comb Logic.74

Copyright 2006 Michael D. Ciletti, ND - 2006