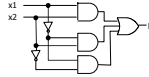


Lecture 2 Intro to Verilog

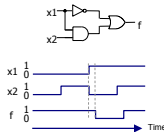
Design Entry with HDLs

- Design entry using HDLs (Hardware Description Language)
 - Similar to a computer program
 - Describes underlying hardware implementation
- Many HDLs exists, two main ones
 - Verilog
 - VHDL (Very High Speed Integrated Circuit HDL)
- Advantages
 - Widely supported
 - Enables portability
 - Underlying implementation can differ without having to change the design specification
 - Text-based
 - Easy to include in documentation
 - Modular implementation possible
 - Enables hierarchical implementation of circuits
 - Sharing and re-use



Introduction to Verilog

- How do we know our design actually works?
 - Functional Simulation
- Method
 - Designer provides input values
 - Functional simulator applies these values to the equations
 - Functional simulator produces correspond outputs
 - Truth table or timing diagram
 - User examines output to verify design
- Gate Delay
 - Functional simulator assumes delay negligible
 - Timing simulator accounts for timing details related to a specific technology



Primitives

- Verilog includes a set of *gate level primitives* corresponding to commonly used logic gates

```
and (y, x1, x2); // 2-input AND gate
```

↑ specifies output, y
↑ specifies input, x1 and x2
↑ everything after // to end of line is a comment
keyword that specifies gate type

```
and (f, a, b, c); // 3-input AND gate
```

```
and (out, in1, in2, in3, in4); // 4-input AND gate
```

More primitives

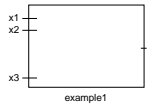
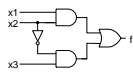
```
or (out, in1, in2);  
not (out, in);  
nand (out, in1, in2);  
nor (out, in1, in2);  
xor (out, in1, in2);  
xnor (out, in1, in2);
```

ECE 474a/575a
Susan Lysdecky

4 of 53

Example 1

- Using primitives let's implement a circuit in Verilog



```
// example1.v  
module example1 (x1, x2, x3, f);  
endmodule
```

module indicates the start of our specification, endmodule indicates the end

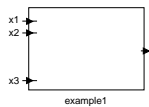
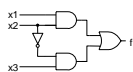
we have a "black box" named example1

"black box" has 4 ports - x1, x2, x3, f

ECE 474a/575a
Susan Lysdecky

5 of 53

Example 1 Continued



```
// example1.v  
module example1 (x1, x2, x3, f);  
input x1, x2, x3;  
output f;  
endmodule
```

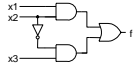
input statement indicates x1, x2, and x3 are inputs to the systems

output statement indicates f is an output of the system

ECE 474a/575a
Susan Lysdecky

6 of 53

Example 1 Continued



```
// example1.v
module example1 (x1, x2, x3, f);
  input x1, x2, x3;
  output f;
  wire g;

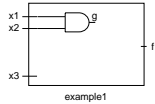
  and (g, x1, x2);
endmodule
```

actual structure of the circuit specified by the primitives

first and statement specifies an AND gate with input x1 and x2, output g

g is internal value, use wire
wire is just a connection, does not store value

CAUTION – Verilog will implicitly declare wire if you don't but it will be a 1-bit wire

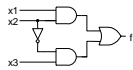


example1

ECE 474a/575a
Susan Lysdecky

7 of 53

Example 1 Continued



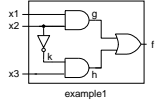
```
// example1.v
module example1 (x1, x2, x3, f);
  input x1, x2, x3;
  output f;
  wire g, k, h;

  and (g, x1, x2);
  not (k, x2);
  and (h, k, x3);
  or (f, g, h);
endmodule
```

not statement specifies a NOT gate with input x2 and output k

next and statement specifies an AND gate with input k and x3, output h

or statement specifies an OR gate with input g and h, output f

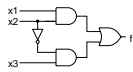


example1

ECE 474a/575a
Susan Lysdecky

8 of 53

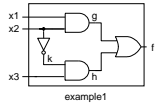
Example 1 Continued



```
// example1.v
module example1 (x1, x2, x3, f);
  input x1, x2, x3;
  output f;
  wire g, k, h;

  and (g, x1, x2);
  not (k, x2);
  and (h, k, x3);
  or (f, g, h);
endmodule
```

Done!
We have implemented our circuit in Verilog



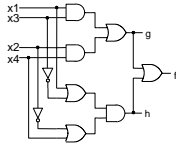
example1

ECE 474a/575a
Susan Lysdecky

9 of 53

Example 2

- Let's try another example



```
// Example 2
// g = x1x3 + x2x4
// h = (x1+x3)(x2+x4)
// f = g +h
```

```
module example2 (x1, x2, x3, x4, f, g, h);
input x1, x2, x3, x4;
output f, g, h;

and (z1, x1, x3);
and (z2, x2, x4);
or (g, z1, z2);
or (z3, x1, ~x3);
or (z4, ~x2, x4);
and (h, z3, z4);
or (f, g, h);

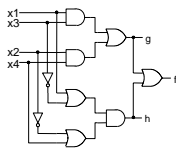
endmodule
```

enclose description
between module
and endmodule

ECE 474a/575a
Susan Lysdecky

10 of 53

Example 2 Continued



```
// Example 2
// g = x1x3 + x2x4
// h = (x1+x3)(x2+x4)
// f = g +h
```

```
module example2 (x1, x2, x3, x4, f, g, h);
input x1, x2, x3, x4;
output f, g, h;

and (z1, x1, x3);
and (z2, x2, x4);
or (g, z1, z2);
or (z3, x1, ~x3);
or (z4, ~x2, x4);
and (h, z3, z4);
or (f, g, h);

endmodule
```

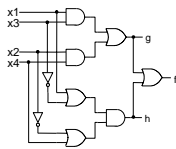
module named
example2

7 port signals
- x1, x2, x3, x4,
f, g, h

ECE 474a/575a
Susan Lysdecky

11 of 53

Example 2 Continued



```
// Example 2
// g = x1x3 + x2x4
// h = (x1+x3)(x2+x4)
// f = g +h
```

```
module example2 (x1, x2, x3, x4, f, g, h);
input x1, x2, x3, x4;
output f, g, h;

and (z1, x1, x3);
and (z2, x2, x4);
or (g, z1, z2);
or (z3, x1, ~x3);
or (z4, ~x2, x4);
and (h, z3, z4);
or (f, g, h);

endmodule
```

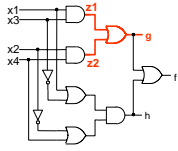
x1, x2, x3, and x4
are inputs to the
system

f, g, h are outputs
of the system

ECE 474a/575a
Susan Lysdecky

12 of 53

Example 2 Continued



```
// Example 2
// g = x1x3 + x2x4
// h = (x1+x3)(x2+x4)
// f = g + h
```

```
module example2 (x1, x2, x3, x4, f, g, h);
  input x1, x2, x3, x4;
  output f, g, h;

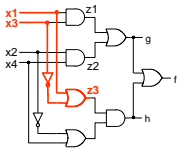
  and (z1, x1, x3);
  and (z2, x2, x4);
  or (g, z1, z2);
  or (z3, x1, ~x3);
  or (z4, ~x2, x4);
  and (h, z3, z4);
  or (f, g, h);
endmodule
```

we define how the circuit is connected

ECE 474e/575e
Susan Lysdecky

16 of 53

Example 2 Continued



notice that the not operation is implemented with "~" symbol instead of a primitive

built in Verilog operator that performs complement

```
// Example 2
// g = x1x3 + x2x4
// h = (x1+x3)(x2+x4)
// f = g + h
```

```
module example2 (x1, x2, x3, x4, f, g, h);
  input x1, x2, x3, x4;
  output f, g, h;

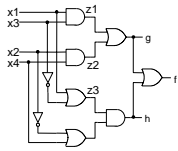
  and (z1, x1, x3);
  and (z2, x2, x4);
  or (g, z1, z2);
  or (z3, x1, ~x3);
  or (z4, ~x2, x4);
  and (h, z3, z4);
  or (f, g, h);
endmodule
```

we define how the circuit is connected

ECE 474e/575e
Susan Lysdecky

17 of 53

Example 2 Continued



continue describing remaining gates/interconnections

```
// Example 2
// g = x1x3 + x2x4
// h = (x1+x3)(x2+x4)
// f = g + h
```

```
module example2 (x1, x2, x3, x4, f, g, h);
  input x1, x2, x3, x4;
  output f, g, h;

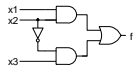
  and (z1, x1, x3);
  and (z2, x2, x4);
  or (g, z1, z2);
  or (z3, x1, ~x3);
  or (z4, ~x2, x4);
  and (h, z3, z4);
  or (f, g, h);
endmodule
```

we define how the circuit is connected

ECE 474e/575e
Susan Lysdecky

18 of 53

Example 3 Continued



```
// Example 3
module example3 (x1, x2, x3, f);
input x1, x2, x3;
output f;
assign f = (x1 & x2) | (~x2 & x3);
endmodule
```

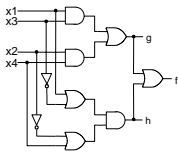
assign keyword indicates anytime something changes on right hand side, re-evaluate left hand side

ECE 474a/575a
Susan Lysdecky

22 of 53

Example 4 - Modeling Circuit With Bit-wise Operators

- Model circuit from example 2 using bit-wise operators



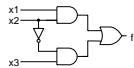
```
// Example 4
module example4 (x1, x2, x3, x4, f, g, h);
input x1, x2, x3, x4;
output f, g, h;
assign g = (x1 & x3) | (x2 & x4);
assign h = (x1 | ~x3) & (~x2 | x4);
assign f = g | h;
endmodule
```

ECE 474a/575a
Susan Lysdecky

23 of 53

Analyze High-level Function of Example 1

- Further analyze circuit from example 1 – what does it really do?
 - If $x_2 = 1$, output equal to value of x_1
 - If $x_2 = 0$, output equal to value of x_3



x1	x2	x3	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

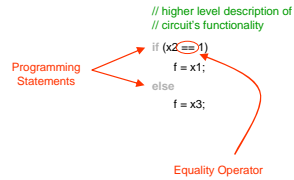
```
// higher level description of
// circuit's functionality
if (x2 == 1)
    f = x1;
else
    f = x3;
```

ECE 474a/575a
Susan Lysdecky

24 of 53

Equality Operators and Programming Statements

- Introduced a couple of new things
 - Equality Operators
 - Programming Statements



ECE 474a/575a
Susan Lysdecky

25 of 53

Equality Operators and Relational Operator Definitions

- Equality Operators
 - Operands are compared bit by bit
 - Zero filling (left side) if the two operands do not have the same length
 - Result is 0 (false) or 1 (true)
- Relational Operators
 - The result 0 if the relation is false, 1 if the relation is true

```
a == b
if a = 00, b = 01, evaluates to 0
if a = 0000, b = 010, evaluates to 0
if a = 101, b = 101, evaluates to 1
```

```
a != b
if a = 00, b = 01, evaluates to 1
if a = 0000, b = 010, evaluates to 1
if a = 101, b = 101, evaluates to 0
```

Equality Operators
== equal
!= not equal

Relational Operators
< less than
> greater than
<= less than or equal to
>= greater than or equal to

ECE 474a/575a
Susan Lysdecky

26 of 53

Programming Statement Definitions

- Programming Statements

```
// Executes the next statement or statement
// group if expression evaluates as true.
```

```
if (expression)
  statement or statement_group
```

```
// Executes the first statement or statement group
// if expression evaluates as true.
// Executes the second statement or statement
// group if expression evaluates as false.
```

```
if (expression)
  statement or statement_group
else
  statement or statement_group
```

```
if (x1 == 0)
  out = 0;
```

```
if (bob == 0) begin
  f = x2 & x3;
  g = x2 | ~x3;
  h = ~x4;
end
```

```
if (x1 == 0) begin
  temp = 1;
  out = 0;
end
else
  out = 1;
```

ECE 474a/575a
Susan Lysdecky

27 of 53

Programming Statement Definitions

Programming Statements (cont')

// A loop that executes a statement or statement group as long as an expression evaluates as true

```
while (expression)
    statement or statement_group
```

```
while (x1 == 0)
    out = 0;
```

// Executes initial_assignment once when the loop starts.
// Executes the statement or statement group as long as the expression evaluates as true
// Executes the step_assignment at the end of each pass through the loop.

```
for (initial_assignment; expression; step_assignment)
    statement or statement_group
```

```
for (x1=0; x<10; x=x+1) begin
    x2 = x1;
end
x4 = x1;
```

ECE 474a/575a
Susan Lysdecky

28 of 53

Programming Statement Definitions

Programming Statements (cont')

// Compares the net, register, or literal value to each case and executes the statement or statement group associated with the first matching case. Executes the default if none of the cases match

```
case (net_or_register_or_literal)
    case_match1 :
        statement or statement_group
    case_match1 :
        statement or statement_group
    ...
default :
    statement or statement_group
endcase
```

```
// assume x equals 2
case ( x ) begin
    0:
        y = 10;

    1:
        y = 5;

    2:
        y = 12;

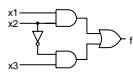
    default:
        y = 0;
endcase
```

ECE 474a/575a
Susan Lysdecky

29 of 53

Example 5 - Modeling Circuit 1 With Programming Statements

Alternative implementation of example 1 using programming statements



// higher level description of
// circuit's functionality

```
if (x2 == 1)
    f = x1;
else
    f = x2;
```

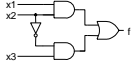
```
// Example 5
// Behavioral Specification
module example5 (x1, x2, x3, f);
    input x1, x2, x3;
    output f;
    reg f;

    always @(x1 or x2 or x3) begin
        if (x2 == 1)
            f = x1;
        else
            f = x3;
        end
    end
endmodule
```

ECE 474a/575a
Susan Lysdecky

30 of 53

Example 5 Continued



notice something new –
always block

```
// Example 5
// Behavioral Specification
module example5 (x1, x2, x3, f);
  input x1, x2, x3;
  output f;
  reg f;
  always @(x1 or x2 or x3) begin
    if (x2 == 1)
      f = x1;
    else
      f = x3;
  end
endmodule
```

ECE 474a/575a
Susan Lysdecky

31 of 53

Always Block

- Always block
 - Sensitivity list associate with always block
 - When one or more value inside sensitivity list changes, code block executed

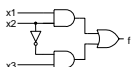
when x1 or x2 or x3 changes
value, code block executes in
the order given

```
// Example 5
// Behavioral Specification
module example5 (x1, x2, x3, f);
  input x1, x2, x3;
  output f;
  reg f;
  always @(x1 or x2 or x3) begin
    if (x2 == 1)
      f = x1;
    else
      f = x3;
  end
endmodule
```

ECE 474a/575a
Susan Lysdecky

32 of 53

Example 5 - Modeling Circuit 1 With Programming Statements



also added another
statement – reg

in a procedural statement if we
want to store a value to signal
must declared it as reg

```
// Example 5
// Behavioral Specification
module example5 (x1, x2, x3, f);
  input x1, x2, x3;
  output f;
  reg f;
  always @(x1 or x2 or x3) begin
    if (x2 == 1)
      f = x1;
    else
      f = x3;
  end
endmodule
```

ECE 474a/575a
Susan Lysdecky

33 of 53

Testbench Continued

```

// example1_tb.v
// Testbench for example1

module Testbench;
  reg x1_t, x2_t, x3_t;
  wire f_t;

  example1 example1_x1(x1_t, x2_t, x3_t, f_t);

  initial
  begin
    // case 0
    x1_t <= 0; x2_t <= 0; x3_t <= 0;
    #1 $display("T = %b", f_t);

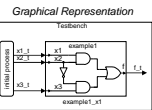
    // case 1
    x1_t <= 0; x2_t <= 0; x3_t <= 1;
    #1 $display("T = %b", f_t);

    // ...

    // case 7
    x1_t <= 1; x2_t <= 1; x3_t <= 1;
    #1 $display("T = %b", f_t);
  end
endmodule
    
```

We declare initial process

Basically everything in between begin & end gets executed **once** when Testbench first executed



ECE 474a/574a
Susan Lynch

40 of 53

Testbench Continued

```

// example1_tb.v
// Testbench for example1

module Testbench;
  reg x1_t, x2_t, x3_t;
  wire f_t;

  example1 example1_x1(x1_t, x2_t, x3_t, f_t);

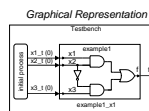
  initial
  begin
    // case 0
    x1_t <= 0; x2_t <= 0; x3_t <= 0;
    #1 $display("T = %b", f_t);

    // case 1
    x1_t <= 0; x2_t <= 0; x3_t <= 1;
    #1 $display("T = %b", f_t);

    // ...

    // case 7
    x1_t <= 1; x2_t <= 1; x3_t <= 1;
    #1 $display("T = %b", f_t);
  end
endmodule
    
```

First case scenario, what happens when inputs are 0, 0, 0?



ECE 474a/574a
Susan Lynch

41 of 53

Testbench Continued

```

// example1_tb.v
// Testbench for example1

module Testbench;
  reg x1_t, x2_t, x3_t;
  wire f_t;

  example1 example1_x1(x1_t, x2_t, x3_t, f_t);

  initial
  begin
    // case 0
    x1_t <= 0; x2_t <= 0; x3_t <= 0;
    #1 $display("T = %b", f_t);

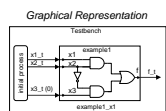
    // case 1
    x1_t <= 0; x2_t <= 0; x3_t <= 1;
    #1 $display("T = %b", f_t);

    // ...

    // case 7
    x1_t <= 1; x2_t <= 1; x3_t <= 1;
    #1 $display("T = %b", f_t);
  end
endmodule
    
```

Wait for a single time unit for the signals to propagate through the module

Verilog language does not define time units such as microseconds, instead these units are specified by the designer in the simulation environment



ECE 474a/574a
Susan Lynch

42 of 53

Testbench Continued

```
// example1_tb.v
// Testbench for example1

module Testbench;
  reg x1_t, x2_t, x3_t;
  wire f_t;

  example1 example1_x1(x1_t, x2_t, x3_t, f_t);

  initial
  begin
    // case 0
    x1_t <= 0; x2_t <= 0; x3_t <= 0;
    #1 $display("f = %b", f_t);

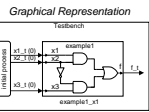
    // case 1
    x1_t <= 0; x2_t <= 0; x3_t <= 1;
    #1 $display("f = %b", f_t);

    // ...

    // case 7
    x1_t <= 1; x2_t <= 1; x3_t <= 1;
    #1 $display("f = %b", f_t);
  end
endmodule
```

ECE 474a/574a
Susan Lysack

View the outcome by displaying the value of the output wire f_t by using \$display("f = %b", f_t)
Value of the f_t wire is displayed in binary as indicated by %b.



43 of 53

Testbench Continued

```
// example1_tb.v
// Testbench for example1

module Testbench;
  reg x1_t, x2_t, x3_t;
  wire f_t;

  example1 example1_x1(x1_t, x2_t, x3_t, f_t);

  initial
  begin
    // case 0
    x1_t <= 0; x2_t <= 0; x3_t <= 0;
    #1 $display("f = %b", f_t);

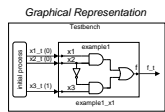
    // case 1
    x1_t <= 0; x2_t <= 0; x3_t <= 1;
    #1 $display("f = %b", f_t);

    // ...

    // case 7
    x1_t <= 1; x2_t <= 1; x3_t <= 1;
    #1 $display("f = %b", f_t);
  end
endmodule
```

ECE 474a/574a
Susan Lysack

Add remaining test cases in a similar fashion



44 of 53

Comments

- Comments
 - "//" begins a single line comment
 - "/*" begins a multi-line comment, terminated by "*/"
- Verilog is case sensitive
 - input1 ≠ Input1 ≠ INPUT1

```
// this is a comment
if (x1 == 0)
  out = 0;
```

```
// this is two single-line
// comments
if (x1 == 0)
  out = 0;
```

```
/* this is a multiple-line
comment */
if (x1 == 0)
  out = 0;
```

ECE 474a/574a
Susan Lysack

45 of 53

Testbench Test Cases

- What should we test in the testbench?
 - Exhaustive
 - Infeasible when large number of inputs
 - 2 inputs – 4 test cases, 3 inputs – 8 test cases, ..., 10 inputs – 1024 test cases
 - Generally, good method is to
 - Border cases
 - Extreme cases - all inputs equal to 0, all input equal to 1
 - Several random cases in-between border cases
 - No definitive number of cases or which cases to indicate what a "good" testbench is

ECE 474a/575a
Susan Lysbecky

46 of 53

Naming Conventions and Logic Values

- Naming
 - Must begin with alphabetic or underscore characters (a-z, A-Z, _)
 - May contain the characters a-z, A-Z, 0-9, _, and \$
 - Cannot use keyword
- Logic Values
 - 0 - zero, low, or false
 - 1 - one, high, or true
 - z or Z - high impedance (tri-stated or floating)
 - x or X - unknown or uninitialized

Sample Identifiers

```
input1 // valid name
1input // cannot begin with number
and // keyword
and2gate // valid name
and2_gate // valid name
```

ECE 474a/575a
Susan Lysbecky

47 of 53

Reserved Keywords

Reserved Keywords

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

ECE 474a/575a
Susan Lysbecky

48 of 53

Timescale Compiler Directive

- Two types of assignment statements in Verilog
 - Blocking "="
 - Non-blocking "<=:"
- Blocking Assignment
 - Evaluation and assignment are immediate
- Non-blocking Assignment
 - All assignments are deferred until all right-hand sides have been evaluated

```
// blocking assignment  
f = a & b;
```

```
// non-blocking assignment  
f <= a & b;
```

```
// a is equal to 0  
// b is equal to 0
```

```
b = -a;  
g = a + b;
```

~a evaluated, b immediately assigned 1

a + b evaluates to 1

```
// a is equal to 0  
// b is equal to 0
```

```
b <= -a;  
g <= a + b;
```

~a evaluated, b not assigned yet (still 0)

a + b evaluates to 0

when all evaluations done then we assign b & g

ECE 474a/575a
Susan Lysdecky

52 of 53

References

- Course Page – Verilog by Example
 - http://www.ece.arizona.edu/~ece474a/resources/verilog_tutorial/index.html
- World of ASIC – Verilog Tutorial and Examples
 - <http://www.asic-world.com/verilog/veritut.html>
- Google

ECE 474a/575a
Susan Lysdecky

53 of 53