

ECE 351

Verilog and FPGA Design

Week 2_1: Literals, nets, and vars Simulating digital systems

Roy Kravitz

Electrical and Computer Engineering Department

Maseeh College of Engineering and Computer Science



1

Literals, nets, and vars

Source material drawn from:

- Alex P., Mark F. and Roy K. ECE 571 lecture slides
- Roy K. ECE 351 and Verilog Workshop lecture slides
- *Logic Design and Verification Using SystemVerilog (Revised)*
by Donald Thomas
- *RTL Modeling with SystemVerilog for simulation and
Synthesis* by Stuart Sutherland

2

Literals

3

- **sized** or **unsized**: `<size>'[s]<radix><number>`
 - **Sized** example: `3'b010` = 3-bit wide binary number
 - The prefix (3) indicates the size of number
 - **Unsize**d example: `123` = 32-bit wide decimal number by default
 - **Signed** example: `-8'sd39` = 8-bit wide signed number
 - **Defaults**
 - No specified `<base format>` defaults to **decimal**
 - No specified `<size>` defaults to **32-bit** number
 - No `"s"` defaults to unsigned
- **Radix**:
 - Decimal ('d or 'D) `16'd255` = 16-bit wide decimal number
 - Hexadecimal ('h or 'H) `8'h9a` = 8-bit wide hexadecimal number
 - Binary ('b or 'B) `'b1010` = 32-bit wide binary number
 - Octal ('o or 'O) `'o21` = 32-bit wide octal number

3

Literals (cont'd)

4

- `'_'` (underscore): used for readability (ignored)
 - Example: `32'h21_65_bc_fe` = 32-bit hexadecimal number
- `'x'` or `'X'` (unknown value)
 - Example: `12'hxxx` = 12-bit hexadecimal number, unknown state
- `'z'` or `'Z'` (high impedance value)
 - Example: `1'bz` = 1-bit high impedance number
- `'?'` can be used in place of `'z'` or `'Z'` to represent hi-impedance and don't care

4

Mismatched size and values

5

- ☐ Legal to specify a literal integer with a bit-width != number of bits required:
 - 4'hFACE // 4-bit width, 16-bit unsigned value
 - 16'sh8 // 16-bit size, 4 bit signed value, MSB set
 - 32'bz // 32-bit width, 1-bit unsigned valued
- ☐ SystemVerilog will adjust to match according to these rules:
 - When size < #bits than value, left-most bits truncated
 - When size > #bits than value, value is left-extended
 - ☐ if left-most bit of value is 0 or 1, additional bits filled w/ 0
 - ☐ If left-most bit of value is Z, upper bits are filled w/ Z
 - ☐ If left-most bit of value is X, upper bits are filled w/ X
 - e.g. Value is not sign extended even is literal integer is specified as signed. Sign extension occurs when signed literal value is used in operations and assignment statements

5

Mismatched size and values

6

Your turn:

- ☐ 4'hFACE //
- ☐ 16'sh8 //
- ☐ 32'bz //

- ☐ 4'hFACE // truncates to 4'hE
- ☐ 16'sh8 // extends to 16'hsh0008
- ☐ 32'bz // extends to 32'hZZZZZZZZ

6

Vector fill and floating-point literal values

7

- SystemVerilog provides a special form of unsized literals:
 - Sets all bits of vector of any size to 0, 1, X, or Z
 - Vector size is automatically determined based on context
 - '0 fills all bits on the LHS w/ 0
 - '1 fills all bits on the LHS w/ 1
 - 'z or 'Z fills all bits on the LHS w/ z
 - 'x or 'X fills all bits on the LHS w/ x
 - Important construct for modeling scalable designs (i.e. different vector widths for different design configurations)
- Floating-point literal values (real numbers):
 - Represented using 64-bit double precision floating point values
 - Examples: 3.1567, 5.0, 0.5
 - Real numbers not typically supported by synthesis tools

7

Vectors and part select

8

- Verilog net and variable data types more than one bit wide are called *vectors*
 - Examples:

```
wire [255:0] bus;           // 256-bit wide bus
tri [0:127] backward_bus;  // bit 0 is MSB
logic[31:24] whatever;    // don't need bit 0
```
- Select part of a vector
 - Examples:

```
my_bit = bus[152];
my_bit = bus[i];           // selects ith bit of bus
my_byte = bus[87:80];
my_byte = bus[80+:8];      // selects bits [87:80]
my_byte = bus[87-:16];     // selects bits [87:72]
my_byte = bus[byteNum*8 -: 8]; // variable part select
my_byte = bus[120:127];    // Illegal!
```

8

Vector part select

9

- Part select can include expressions of variables
`my_byte = word[n*8+7:n*8]; //selects nth byte of word`

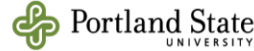
- Part select increment and decrement function
[<starting_bit>+:<width>] increments from starting_bit
[<starting_bit>-:<width>] decrements from starting_bit

- Examples:

```
logic [127:0] vector_le;  
logic [0:127] vector_be;  
my_byte = vector_le[31 -: 8]; //selects vector_le[31:24]  
my_byte = vector_le[24 +: 8]; //selects vector_le[31:24]  
my_byte = vector_be[31 -: 8]; //selects vector_be[24:31]  
my_byte = vector_be[24 +: 8]; //selects vector_be[24:31]  
my_byte = vector_le[n*8+: 8]; //selects nth byte of vector_le  
my_byte = vector_be[n*8+: 8]; //selects nth byte of vector_be
```

Source: Verilog 2001 A guide to the New Features of the Verilog Hardware Description Language by Stuart Sutherland

ECE 351 Verilog and FPGA Design



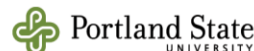
9

Types and data types

10


- SystemVerilog introduces notion of **type** and **data type**
 - **type** indicates whether identifier is a net or a variable
 - **data type** indicates the value system (0/1 for two-state logic, 0/1/X/Z for four-state logic)
- Data types are used in RTL modeling to indicate desired silicon behavior
 - ex: should an adder performed signed or unsigned arithmetic
- Type:
 - Nets are used to connect design blocks together
 - Transfers data values from a source (driver) to a destination (receiver)
 - SystemVerilog supports several net types
 - Net types are always 4-state data types
 - Variables provide temporary storage for simulation
 - Required on LHS of procedural block assignments
 - Can be either 2-state or 4-state data types

ECE 351 Verilog and FPGA Design

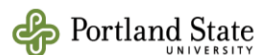


10

11

- 
- Portland State
UNIVERSITY

12



6

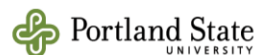
Synthesizable variable data types

13

Type	Representation
reg	An obsolete general purpose 4-state variable of a user-defined vector size; equivalent to var logic
logic	Usually infers a general purpose var logic 4-state variable of a user-defined vector size, except on module input/inout ports, where wire logic is inferred
integer	A 32-bit 4-state variable; equivalent to var logic [31:0]
bit	A general purpose 2-state var variable with a user-defined vector size; defaults to a 1-bit size if no size is specified
int	A 32-bit 2-state variable; equivalent to var bit [31:0]; synthesis compilers treat int as the 4-state integer type
byte	An 8-bit 2-state variable; equivalent to var bit [7:0]
shortint	A 16-bit 2-state variable; equivalent to var bit [15:0]
longint	A 64-bit 2-state variable; equivalent to var bit [63:0]

ECE 351 Verilog and FPGA Design

Source: Sutherland, Table 3-1



13

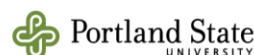
SystemVerilog 2-State variables

14

- Useful for modeling at higher levels of abstraction than RTL (e.g. system, transaction)
 - Higher level models seldom require **z**, **x** which take up more memory and are slower to simulate
- Most of the 2-state types are signed by default
 - Can override w/ the **unsigned** keyword

```
int s_int; // signed 32-bit variable
int unsigned u_int; // unsigned 32-bit variable
```
- Common uses:
 - Bus functional models
 - Interfacing Verilog models to C/C++ models using SystemVerilog Direct Programming I/F
 - Loop variables (aren't synthesized, don't require **z**, **x** states)

ECE 351 Verilog and FPGA Design



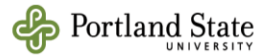
14

SystemVerilog 2-State variables (cont'd)

15

- 2-state types:
 - bit** 1-bit 2-state integer
 - byte** 8-bit 2-state integer (similar to C **char**)
 - shortint** 16-bit 2-state integer (similar to C **short**)
 - int** 32-bit 2-state integer (similar to C **int**)
 - longint** 64-bit 2-state integer (similar to C **longlong**)
- Synthesis treats 4-state and 2-state variables the same way
- 2-state data types are initialized to 0 in simulation but not in synthesized hardware
- 4-state variables can be assigned to 2-value data types but be careful...

ECE 351 Verilog and FPGA Design



15

SystemVerilog 2-State variables (cont'd)

16

4-state variables can be assigned to 2-value data types but **use care when connecting 2-state variables in testbench to design under test, especially outputs.**

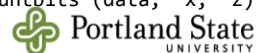
- 4-state variables driving **x** or **z** to a 2-state variable will be (silently) converted to 2-state value (0) and your testbench will never know
- Use `$isunknown()` which returns 1 if any bit of the expression is **x** or **z**.

```
if ($isunknown(iport) == 1)
    $display("@%0t: 4-state value detected on iport %b",
            $time, iport);
```

- Use `$countbits()` to count number of bits having a specified set of values (0, 1, **x**, **z**)

```
assert (!$isunknown(data)
    else $error("data has %0d bits with x or z", $countbits (data, 'x, 'z'));
```

ECE 351 Verilog and FPGA Design



16

Variable assignment rules

17

- A variable can receive a value from any one of these ways:
 - Assigned a value from single `always_comb`, `always_ff`, or `always_latch` procedural block
 - Assigned a value from single continuous assignment statement
 - As the result of an assignment operator such as `++`
 - As an input to a module, task or function
 - As a connection to an output port of a module instance, task or function instance or primitive (gate level) instance
- A variable can only be assigned by a single source, however multiple assignments to a variable in the same procedural block is treated as a single driver

```
logic [15:0] q; //16-bit 4-state unsigned variable
```

```
always_ff @(posedge clk)
  if (!rstN)
    q <= '0;
  else
    q <= d
```

ECE 351 Verilog and FPGA Design



17

Variable assignment rules (cont'd)

18

var(iables) cannot be driven by multiple sources but nets can

```
module add_and_increment (output logic [63:0] sum,
                          output logic      carry,
                          input logic [63:0] a, b );

  always @(a, b)
    sum = a + b;           // procedural assignment to sum

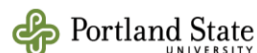
  assign sum = sum + 1;    // ERROR! sum is already being
                          // assigned a value

  look_ahead i1 (carry, a, b); // module instance drives carry
  overflow_check i2 (carry, a, b); // ERROR! 2nd driver of carry
endmodule

module look_ahead (output wire      carry,
                  input logic [63:0] a, b);
  ...
endmodule

module overflow_check (output wire      carry,
                     input logic [63:0] a, b);
  ...
endmodule
```

ECE 351 Verilog and FPGA Design



18

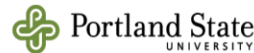
Non-synthesizable variable data types

19

Type	Representation
real	A double precision floating-point variable
shortreal	A single precision floating-point variable
time	A 64-bit unsigned 4-state variable with timeunit and timeprecision attributes
realtime	A double precision floating-point variable, identical to real
string	A dynamically sized array of byte types that can store a string of 8-bit ASCII characters
event	A pointer variable that stores a handle to a simulation synchronization object
class handle	A pointer variable that stores a handle to a class object (the declaration type is the name of a class, not the keyword class)
chandle	A pointer variable that stores pointers passed into simulation from the SystemVerilog Direct Programming Interface
virtual interface	A pointer variable that stores a handle to an interface port (the interface keyword is optional)

ECE 351 Verilog and FPGA Design

Source: Sutherland, Table 3-2



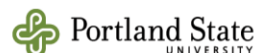
19

Net types

20

- Used to connect design elements together
 - ex: connecting the output port of one module to the input port of another module
- Nets differ from variables in these significant ways:
 - Nets reflect the current value of the driver(s) on the net (i.e. no temporary storage)
 - Nets can resolve the results of multiple drivers
 - Nets reflect both a driver value (0, 1, Z, or X) and a driver strength
 - Strength is represented in steps from 0 (weak, low drive) to 7 (strong, high drive)
 - Default drive strength is 6 (strong)
 - Strengths are NOT used in RTL modeling (dependent on target technology)

ECE 351 Verilog and FPGA Design



20

Synthesizable net types

21

Type	Representation
real	A double precision floating-point variable
shortreal	A single precision floating-point variable
time	A 64-bit unsigned 4-state variable with timeunit and timeprecision attributes
realtime	A double precision floating-point variable, identical to real
string	A dynamically sized array of byte types that can store a string of 8-bit ASCII characters
event	A pointer variable that stores a handle to a simulation synchronization object
class handle	A pointer variable that stores a handle to a class object (the declaration type is the name of a class, not the keyword class)
chandle	A pointer variable that stores pointers passed into simulation from the SystemVerilog Direct Programming Interface
virtual interface	A pointer variable that stores a handle to an interface port (the interface keyword is optional)

ECE 351 Verilog and FPGA Design

Source: Sutherland, Table 3-3



21

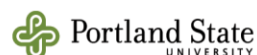
(Generally) non-synthesizable net types

22

Type	Representation
uwire	An interconnecting net that does not permit or resolve multiple drivers
pull0	An interconnecting net that has the behavior of having a pull-down resistor tied to the net
pull1	An interconnecting net that has the behavior of having a pull-up resistor tied to the net
wand	An interconnecting net that resolves multiple drivers by ANDing the driven values
triand	A synonym for wand , and identical in all ways; can be used to emphasize nets that are expected to have tri-state values
wor	An interconnecting net that resolves multiple drivers by ORing the driven values
trior	A synonym for wor , and identical in all ways; can be used to emphasize nets that are expected to have tri-state values
triereg	An interconnecting net with capacitance; if all drivers are at high-impedance, the capacitance reflects the last resolved driven value

ECE 351 Verilog and FPGA Design

Source: Sutherland, Table 3-4



22

Net assignment and connection rules

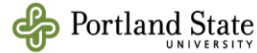
23

- Nets can receive a value from two types of sources:
 - As a connection to an output or inout port
 - As the LHS of a continuous assignment (and assign statement)
- Any changes on the RHS of an assignment cause the LHS to be re-evaluated and LHS updated
 - LHS can be a variable or a net

```
wire [15:0] sum;  
assign sum = a + b;
```
 - An implicit continuous assignment combines the declaration of a net and the assignment

```
wire [15:0] sum = a + b;
```
- Rules for resolving port/connection mismatches:
 - port size < net or variable size – leftmost bits truncated
 - Port size > net or variable size – value of net is left-extended
 - If either port or net/variable is unsized – zero-extended
 - If both port and net/variable are signed – sign-extended

ECE 351 Verilog and FPGA Design



23

Simulating Digital Systems

Source material drawn from:

- *SystemVerilog for Design: Overview* by Donald Thomas
- *RTL Modeling with SystemVerilog for Simulation and Stimulus* by Stuart Sutherland

24

Some basic terms

25

□ System (or model) state

- Includes variables and things intended to become nets and registers
- Every variable deep in instantiated modules is uniquely named (hierarchical naming)
- Statically defined

□ Events and their flavors

- Events are tuples:
 - something to do, and
 - a time to do it
- **Update Event** — a value-change scheduled to occur at a given time
 - e.g., b is set to 1 at time 15
- **Evaluation Event** — (sometimes “execution event”) a model (always, initial...) to resume executing at a given time

ECE 351 Verilog and FPGA Design



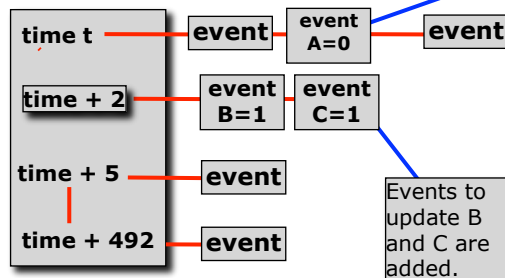
25

Event list

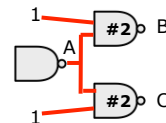
26

□ A time-ordered list of events is maintained

- All events for a given time are kept together in an event list ordered by time
- Events are handled at the given time and possibly schedule their output to change at a later time (a new event)
- Here, these are update events

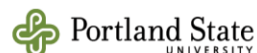


This event says that A goes to 0 at time t .



Events to update B and C are added.

ECE 351 Verilog and FPGA Design



26

Event regions

27

- Events in a begin-end statement group are scheduled into an event region and executed in the order in which the statements are listed
- Events from concurrent processes are scheduled into the event region in an arbitrary order chosen by the simulator
- Simulators will execute all scheduled events in a region before transitioning to the next region
- As events are processed, they are permanently removed from the event list
- Each region will be empty before simulation proceeds to the next region
- As events are processed in a later region, they can possibly schedule new events in a previous region
- The transition from one event region to the next is referred to as a delta. Each iteration through all event regions is referred to as a delta cycle

ECE 351 Verilog and FPGA Design

Sutherland Section 1.5.3.5.



27

Pseudo Code: Event-Driven Simulation

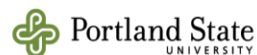
28

```
put all initial and always blocks in the event list
while something in time-ordered event list {
    advance simulation time to first event's time
    retrieve all events e for this time
    update state from all update-event values
```

One traversal of the while loop is a *simulation cycle*.

```
For each event e in arbitrary order {
    If it's an update event {
        follow fanout, evaluate gates there
        If an output changes
            schedule update event for it
    }
    else // it's an evaluation event
        evaluate the model
}
```

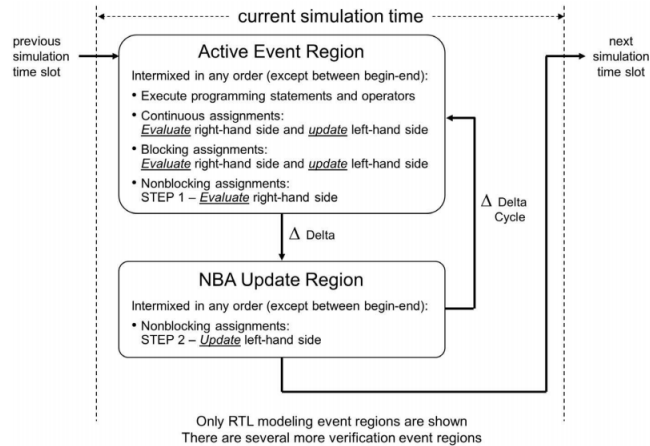
ECE 351 Verilog and FPGA Design



28

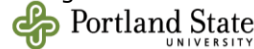
Event Scheduling

29



- Blocking assigns (assign $a = b$) – model combinational logic
- Non-blocking assigns ($a <= b$) – model sequential logic

ECE 351 Verilog and FPGA Design

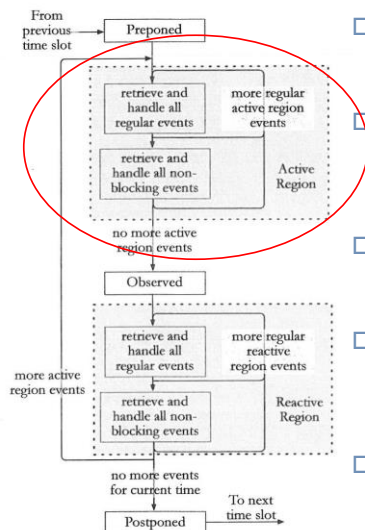


Sutherland Figure 1-8

29

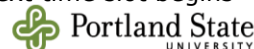
FYI: The “Whole” Simulation Kernel

30



- **Preponed** – time has been updated but event handling hasn’t started. Values sampled for assertions
- **Active** – handling all regular and non-blocking events in the active region of event list. Events all have to do with design being modeled
- **Observed** – Assertions are executed using the values sampled in the *preponed* region
- **Reactive** – handling all regular and non-blocking events in the reactive region. Events all have to do with execution of program blocks
- **Postponed** – monitor events are handled. After this simulation time is advanced and the next time slot begins

ECE 351 Verilog and FPGA Design



Roy’s ECE 571 slide – you won’t be tested on this

30

Sutherland Example 1-7

31

```
//////////////////////////////////////////
// Design module with RTL model of a D-type register
//////////////////////////////////////////
//`begin_keywords "1800-2012"
module d_reg (
    input logic      clk,
    input logic [7:0] d,
    output logic [7:0] q
);

timeunit 1ns; timeprecision 1ns;

always_ff @(posedge clk)
    q <= d;

endmodule: d_reg
//`end_keywords
```

ECE 351 Verilog and FPGA Design



31

Sutherland Example 1-7 (cont'd)

32

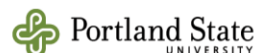
```
//////////////////////////////////////////
// Top module with clk oscillator
//////////////////////////////////////////
//`begin_keywords "1800-2012"
module top;
    timeunit 1ns; timeprecision 1ns;

    logic      clk;
    logic [7:0] d;
    logic [7:0] q; test i1 (.*); // connect top module to test module

    d_reg i2 (.*); // connect top module to d_reg module

    initial begin
        // clk oscillator
        clk <= 0; // initialize clk at time 0
        forever #5 clk = ~clk; // toggle clk every 5ns
    endendmodule: top//`end_keywords
```

ECE 351 Verilog and FPGA Design



32

Sutherland Example 1-7 (cont'd)

33

```

////////////////////////////////////
// Test module with stimulus generator
////////////////////////////////////
//`begin_keywords "1800-2012"
module test (
    input  logic      clk,
    output logic [7:0] d,
    input  logic [7:0] q
);
    timeunit 1ns; timeprecision 1ns;

    initial begin
        d = 1;
        #7 d = 2;
        #10 d = 3;
        #10 $display("\n%m: No output-- checking that example compiles\n");
        #10 $finish;
    endendmodule: test
//`end_keywords

```

ECE 351 Verilog and FPGA Design

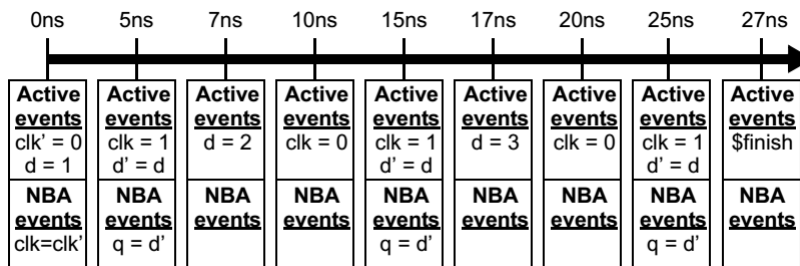


33

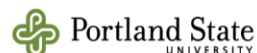
Simulation timeline

34

[Sutherland: Example 1_7](#)



ECE 351 Verilog and FPGA Design



Sutherland Figure 1-9

34

Next Time

35

- Topics:
 - Simulation w/ QuestaSim
 - Graded In-class exercise (implementing and simulating a RCA)
- You should:
 - Read Sutherland Ch 3
 - Prepare to write and simulate code
- Homework, projects and quizzes
 - Homework #1 will be assigned Tue, 13-Apr