Simulation & Testbench

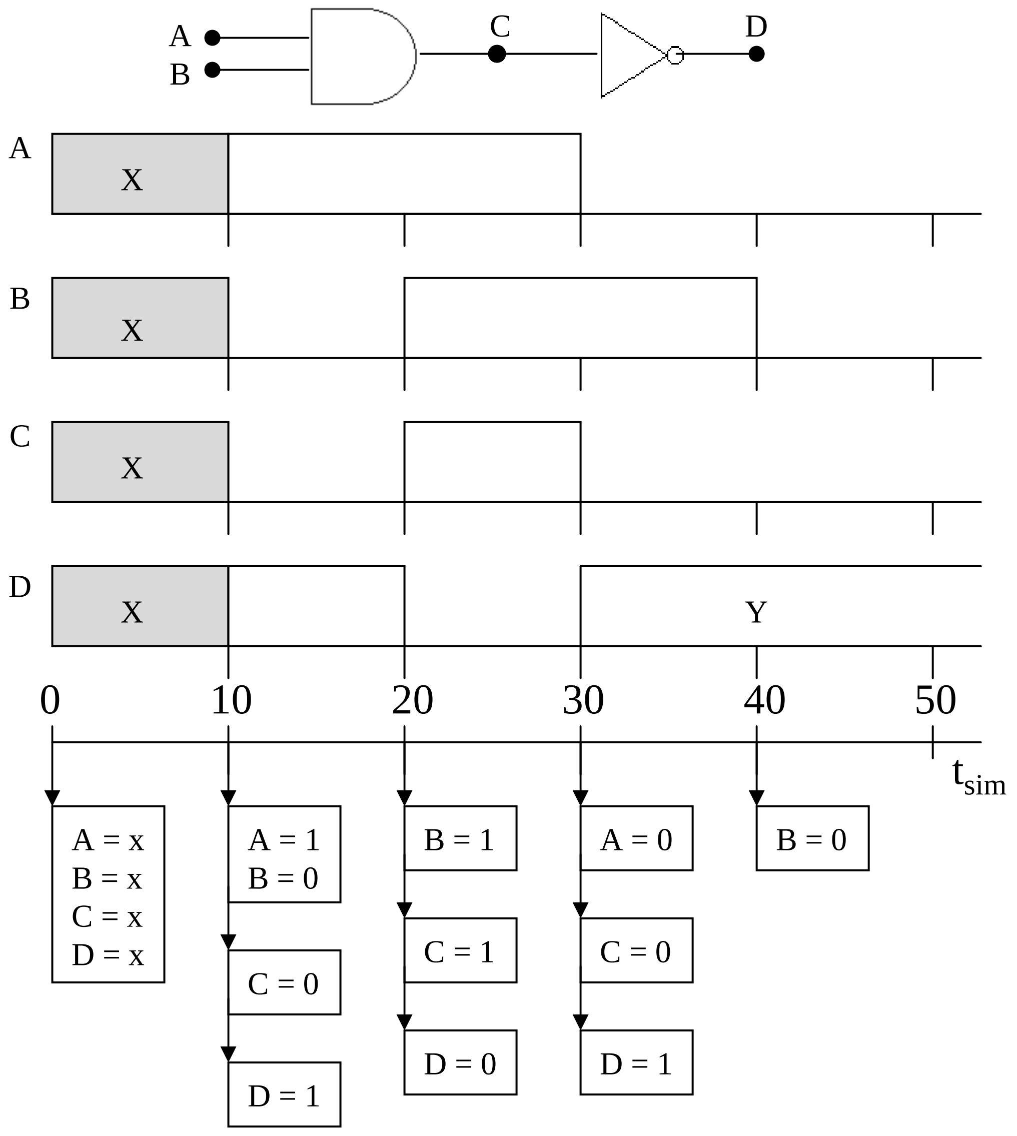
Event-Driven Simulation

- Verilog signal values
  - {0, 1, x, z}
  - x: unknown, ambiguous
  - z: high impedance, open circuit
- An event occurs when a signal changes in value
- Simulation is event-driven if new values are computed
  - only for signals affected by events that have already occurred, and
  - only at those times when changes can occur

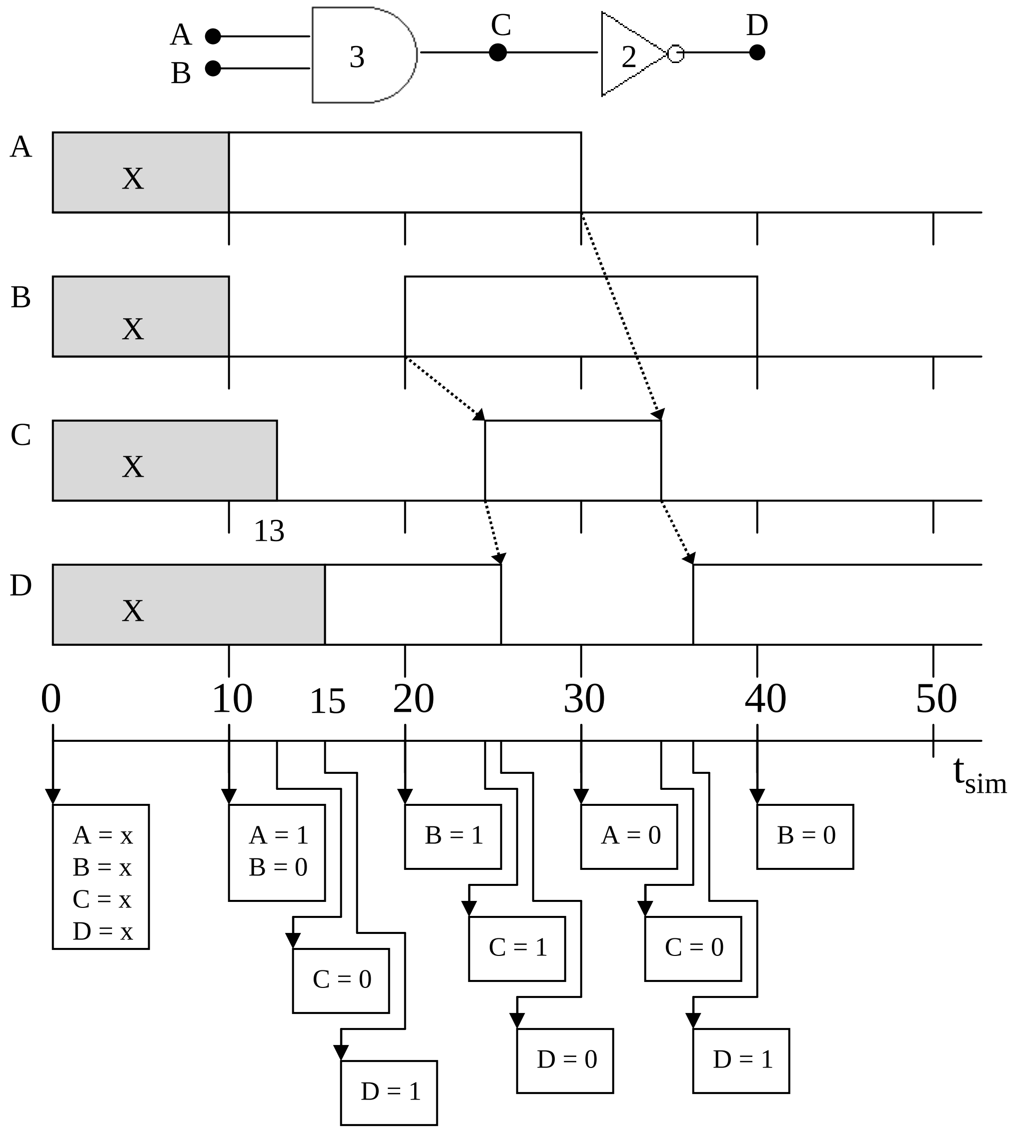
Event-Driven Simulation

- Operation of a simulator depends on a time-ordered event list.
- Initial events on the list consist of input changes. These changes cause events to be scheduled (and placed on the list) for execution at a later time.
- Simulation stops when the event list becomes empty.

Simulation without Delay

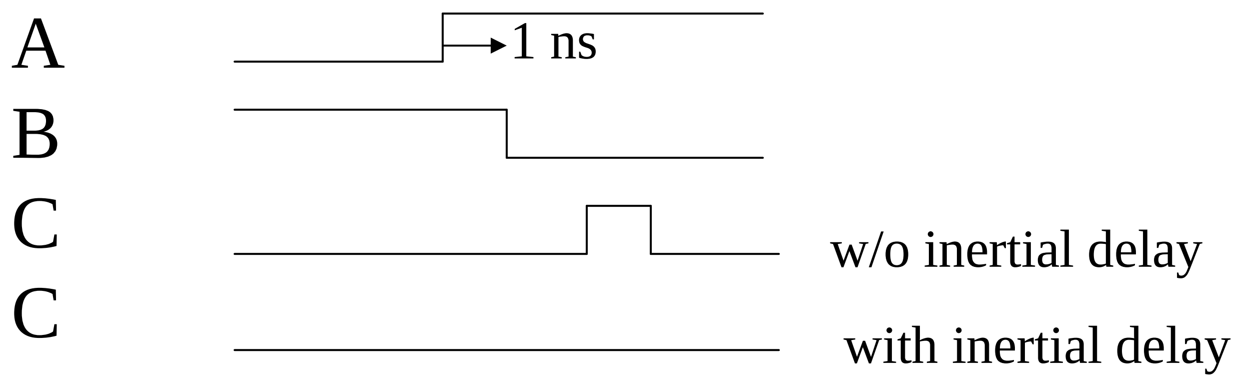


Simulation with Delay

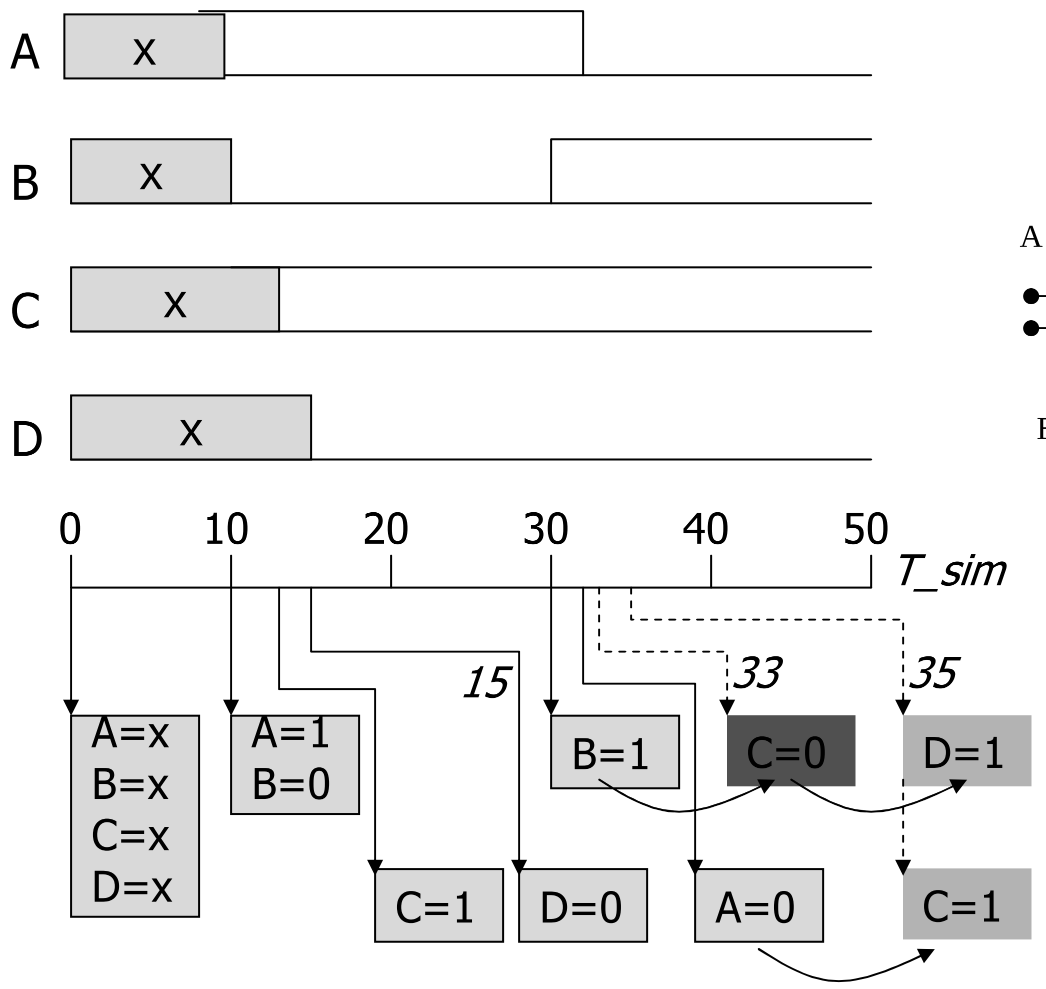


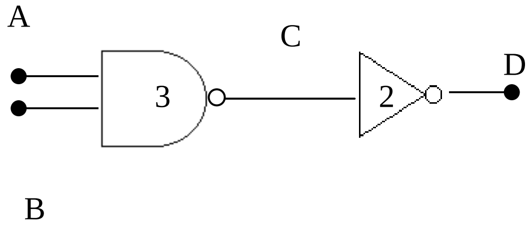
Simulation with Inertial Delay

- Inertial delay: amount of time that input pulse must endure
- Verilog uses the propagation delay as the inertial delay.
- Multiple events cannot occur on the output in a time period less than the inertial delay.
- Example: AND with delay = 2ns



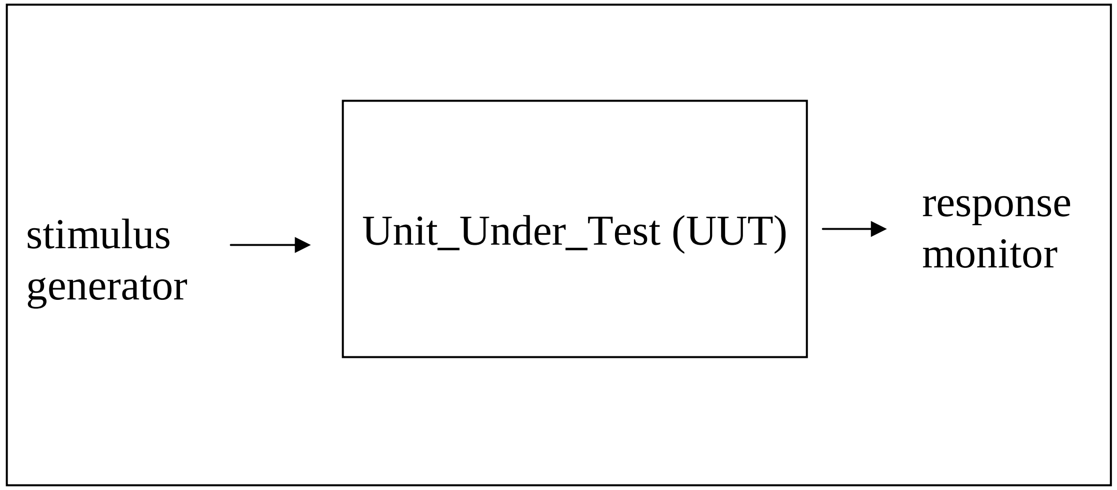
Event De-scheduling





Testbench

- Use Verilog module to produce testing environment including stimulus generation and response monitoring.



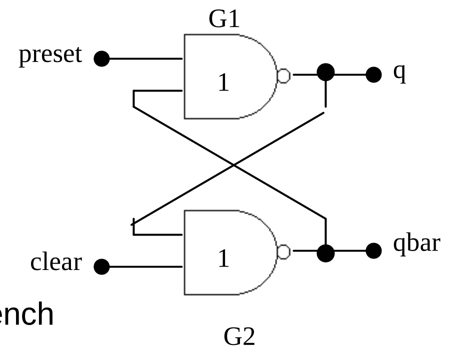
Design_Unit_Test_Bench

initial and Some System Tasks

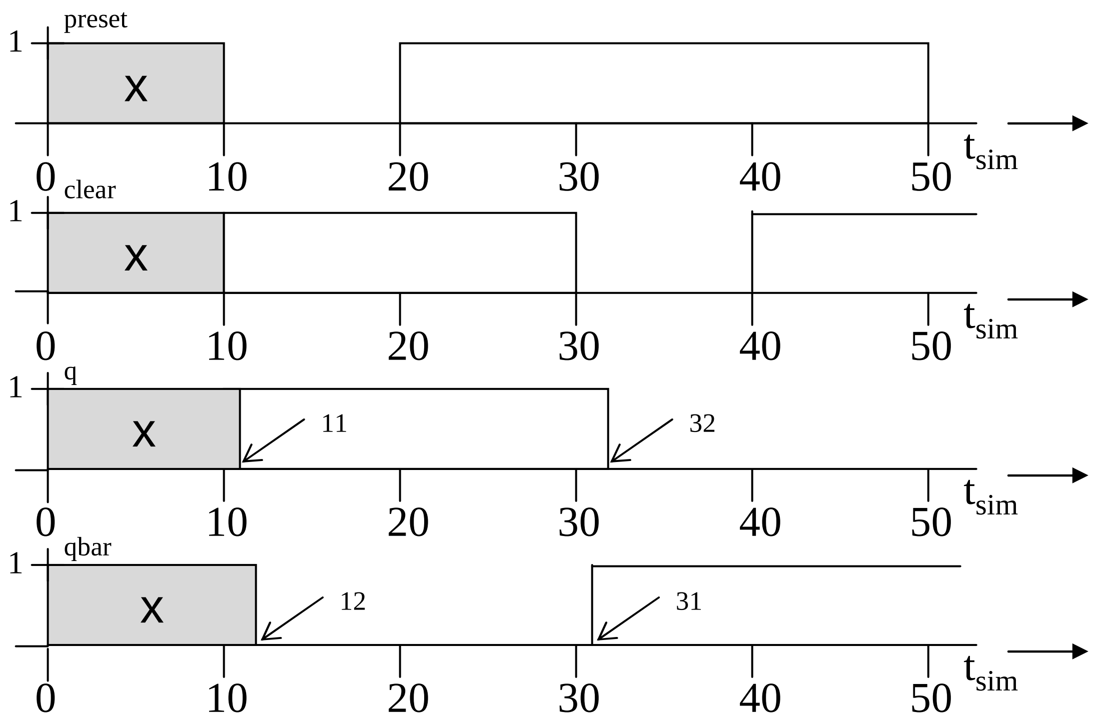
- **initial** declares one-shot behaviors
- **$monitor** is used to observe events
- **$time** returns simulation time
- **$stop** stops execution and wait for interactive input
- **$finish** returns control to operating system

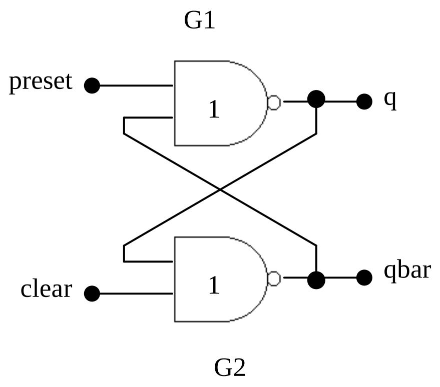
Testbench in Verilog

```
module Nand_Latch_1 (q, qbar, preset, clear);
  output        q, qbar;
  input         preset, clear;

  nand #1       G1 (q, preset, qbar),
                G2 (qbar, clear, q);
endmodule

module test_Nand_Latch_1;               // Design Unit Testbench
  reg           preset, clear;
  wire          q, qbar;

  Nand_Latch_1 M1 (q, qbar, preset, clear);          // Instantiate UUT

  initial                    // Create DUTB response monitor
    begin
      $monitor ($time, "preset = %b clear = %b q = %b qbar = %b", preset, clear, q, qbar);
    end

  initial
    begin                    // Create DUTB stimulus generator
      #10       preset       =0;          clear = 1;
      #10       preset       =1;          $stop;          // Enter . to proceed
      #10       clear        =0;
      #10       clear        =1;
      #10       preset       =0;
    end

  initial
    #60         $finish;     // Stop watch
endmodule
```



Simulation Results





0	preset = x	clear = x	q = x	qbar = x
10	preset = 0	clear = 1	q = x	qbar = x
11	preset = 0	clear = 1	q = 1	qbar = x
12	preset = 0	clear = 1	q = 1	qbar = 0
20	preset = 1	clear = 1	q = 1	qbar = 0
30	preset = 1	clear = 0	q = 1	qbar = 0
31	preset = 1	clear = 0	q = 1	qbar = 1
32	preset = 1	clear = 0	q = 0	qbar = 1
40	preset = 1	clear = 1	q = 0	qbar = 1

Logic System, Data Types and Operators

Variables

- Nets: structural connectivity
- Registers: abstraction of storage (may or may not be physical storage)
- Both nets and registers are informally called signals, and may be either scalar or vector.

Logic Values

- Verilog signal values
  - 0: logical 0 or a FALSE condition
  - 1: logical 1, or a TRUE condition
  - x: an unknown value
  - z: a high impedance condition
- May have associated strengths for switch-level modeling of MOS devices

Example

and	0	1	z	x
0	0	0	0	0
1	0	1	x	x
z	0	x	x	x
x	0	x	x	x

Net Data Types

- **wire** (default): only to establish connectivity
- **tri**: same as wire, but explicitly state that it is tri-stated
- **wand, wor**: wired AND and OR with multiple drivers
- **triand, trior**: tri-stated wired AND or OR with multiple drivers
- **supply0, supply1**: connected to Gnd and Vdd
- **tri0, tri1**: resistive pull-down and pull-up nets
- **trireg**: a charge-stored net

Resolution Rules (Same Driving Strength)

wire/tri	0	1	x	z
0	0	x	x	0
1	x	1	x	1
x	x	x	x	x
z	0	1	x	z

triad / wand	0	1	x	z
0	0	0	0	0
1	0	1	x	1
x	0	x	x	x
z	0	1	x	z

trior/wor	0	1	x	z
0	0	1	x	0
1	1	1	1	1
x	x	1	x	x
z	0	1	x	z

Net Value Assignment

- Value explicitly assigned by
  - continuous assignment
  - **force** … **release** procedural continuous assignment
- Value implicitly assigned by
  - being connected to an output terminal of a primitive
  - being connected to an output port of a module

Net Examples

- **wire** x;
- **wire** [15:0] data;
  - data [5]
  - data [5:3]
- **wire** scalared [0:7] control_a;
- **wire** vectored [0..7] control_b;
- **wand** a;
- **wire** a = b + c;

Initial Value & Undeclared Nets

- At time $t_{sim} = 0$
  - Nets driven by primitives, module or continuous assignment are determined by their drivers, which defaults to *x*
  - Nets without drivers have default value *z*
- Undeclared nets
  - default type: **wire**
  - **‘defaultnettype** compiler directive can specify others except for **supply0** and **supply1**

Register Data Types

- **reg**: store a logic value
- **integer**: support computation
- **time**: store time as 64-bit unsigned quantity
- **real**: store values (e.g., delay) as real numbers
- **realtime**: store time as real numbers

Register Examples

- **reg** a, b;
- **reg** [15:0] counter, shift_reg;
- **integer** c;

Register Value Assignment

- In simulation, a register variable has initial value x
- An undeclared identifier is assumed as a net, which is illegal within a behavior
- A register may be assigned value only within
  - a procedural statement
  - a user-defined sequential primitive
  - a task
  - a function
- A **reg** object may never be
  - the output of a primitive gate
  - the target of a continuous assignment

Addressing Net and Register Variables

- MSB of a part-select of a register = leftmost array index
- LSB = rightmost array index
- If index of part-select is out of bounds, x is returned
- If *word* [7:0] = 8'b00000100
  - *word* [3:0] = 4
  - *word* [5:1] = 2

Variables and Ports

Variable type	Input port	Output port	Inout port
Net	Yes	Yes	Yes
Register	No	Yes	No

Memory

- Memory is a collection of registers
  - **reg** [31:0] cache [0:1023];
    - 1 k memory of 32-bit words
  - **reg** [31:0] one_word;
  - **reg** one_bit;
- Individual bits cannot be addressed directly
  - one_word = cache[988];
  - one_bit = one_word[3];

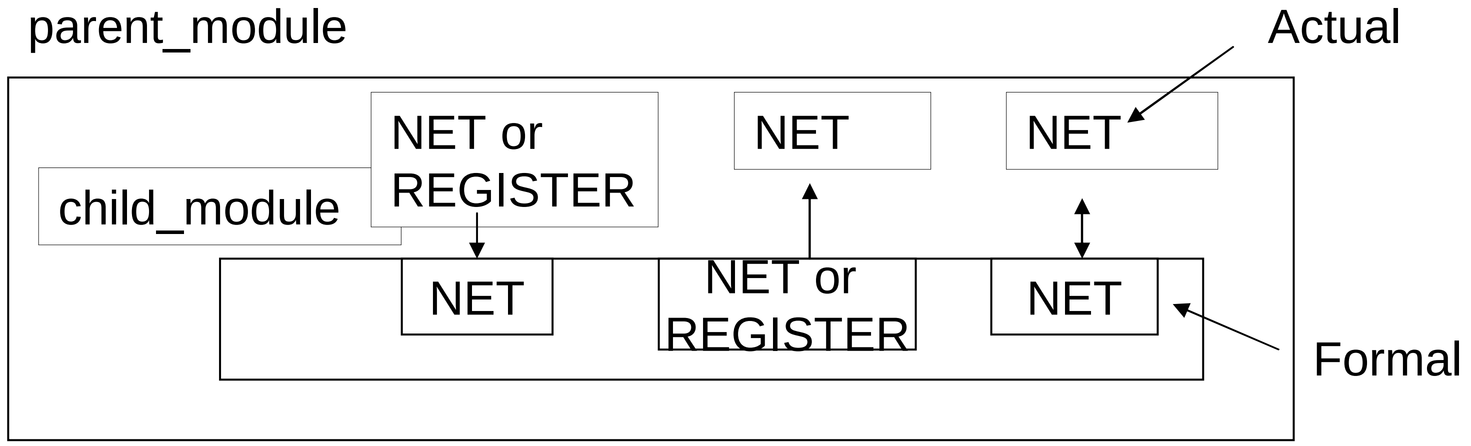
Other Data Types

- **integer**
  - Negative integers stored in 2’s complement format
  - Represented internally to the wordlength (at least 32 bits) of a host machine
  - Example:

    integer Array_of_Ints [1:100];
- **real**
  - Stored in double precision, typically 64-bit value
  - May not be connected to a port or terminal of a primitive
- **time**
  - Stored as unsigned 64-bit value
  - May not be used in a module port or an input (or output) of a primitive
  - Example:
    - **time** T_samples [1:100];
- **realtime**
  - Time values stored in real number format

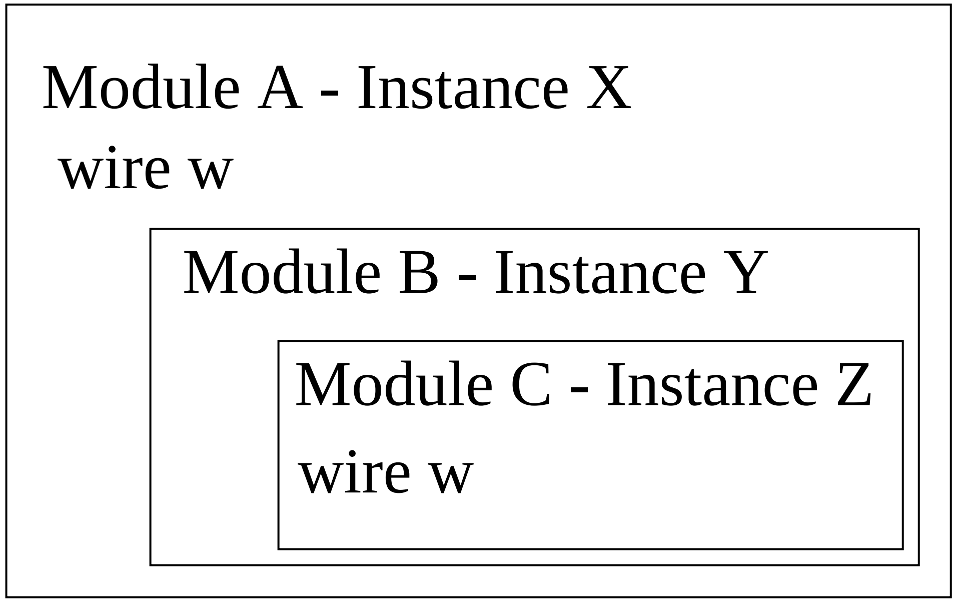
Scope of a Variable

- The scope of a variable is the module, task, function or named procedural block (**begin**…**end**) in which it is declared.
- A variable may be referenced directly by its identifier within the scope in which it is declared.



Hierarchical De-Referencing

- To reference a variable defined inside an instantiated module
- Supported by a variable's hierarchical path name
  - X.w
  - X.Y.Z.w



Example

```
module test_Add_rca_4();
        reg                    [3:0]        a, b;
        reg                                 c_in;
        wire                   [3:0]        sum;
        wire                                c_out;
        initial
          begin
               $monitor ($time, “c_out= %b c_in4=%b c_in3= %b
                             c_in2= %b c_in= %b”,
                              c_out, M1.c_in4, M1.c_in3, M1.c_in2, c_in);
          end
        initial
          begin
              // Stimulus patterns go here
          end
        Add_rca_4 M1 (sum, c_out, a, b, c_in); // module declaration
endmodule

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_in4, c_in3, c_in2;

        Add_full G1 (sum[0], c_in2, a[0], b[0], c_in);
        Add_full G2 (sum[1], c_in3, a[1], b[1], c_in2);
        Add_full G3 (sum[2], c_in4, a[2], b[2], c_in3);
        Add_full G4 (sum[3], c_out, a[3], b[3], c_in4);
endmodule
```

Strings

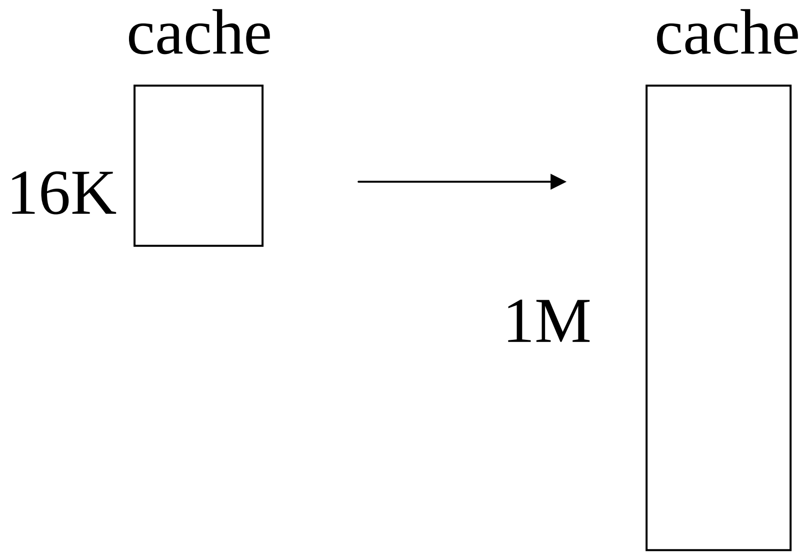
- No explicit data type
- Must be stored in properly sized **reg** (array)
  - **reg** [15:0] string_holder; //store 2 characters
- If an assignment to an array consists of less characters than the array will accommodate, zeros are filled in the unused positions, beginning at MSB.

Constants

- Declared with **parameter**
  - **parameter** size = 16;
    - **reg** [size-1:0] a;
  - **parameter** b = 2’b01;
  - **parameter** av_delay = (min_delay + max_delay) / 2;
- Value may not be changed during simulation
- Value can be changed by direct substitution or indirect substitution during compilation

Module with Parameters

- Sometimes, the function of a block does not change over designs
  - Only the size changes
  - Design once and re-use



Direct Substitution

```
module modXnor (y_out, a, b);
      parameter size = 8, delay = 15;
      output [size-1:0] y_out;
      input [size-1:0] a, b;
      wire   [size-1:0] #delay y_out = a~^b;   // bitwise xnor
endmodule

module Param;
      wire   [7:0] y1_out;
      wire   [3:0] y2_out;
      reg    [7:0] b1, c1;
      reg    [3:0] b2, c2;

      modXnor G1 (y1_out, b1, c1);              // use default parameters
      modXnor #(4, 5) G2 (y2_out, b2, c2);      // override parameters

endmodule
```

Notes: a module instantiation may not have delay associated with it;
a UDP declaration may not contain parameter declarations;
parameters may not be associated with a primitive gate.

Indirect Substitution

```
module hdref_Param;                    // a top level module
      wire    [7:0] y1_out;
      wire    [3:0] y2_out;
      reg     [7:0] b1, c1;
      reg     [3:0] b2, c2;

      modXnor G1 (y1_out, b1, c1),  G2 (y2_out, b2, c2);          // instantiation
endmodule

module annotate;                                    // a separate annotation module
      defparam
            hdref_Param.G2.size = 4,             // parameter assignment
            hdref_Param.G2.delay = 5;            // hierarchical reference name
endmodule

module modXnor (y_out, a, b);
      parameter size = 8, delay = 15;
      output [size-1:0] y_out;
      input [size-1:0]    a, b;
      wire [size-1:0]     #delay y_out = a~^b;  // bitwise xnor
endmodule
```

Verilog Operators

Operator	Number of Operands	Result
Arithmetic	2	Binary word
Bitwise	2	Binary word
Reduction	1	Bit
Logical	2	Boolean value
Relational	2	Boolean value
Shift	1	Binary word
Conditional	3	Expression

Arithmetic Operators

- Binary: +, -, *, /, %
- Unary: +, -
- Examples:
  - **assign** sum = A + B;
  - **assign** diff = A – B;
  - **assign** neg = -A;

Bitwise Operators

- ~, &, |, ^, ~^, ^~
- Shorter operand will extend to the size of longer operand by padding bits with 0
- Examples:

expression	result
~(1010)	0101
(01) & (11)	01
(01) | (11)	11
(01) ^ (11)	10
(01) ~^ (11)	01

Reduction Operators

- Unary operators
- Return single-bit value
- &, ~&, |, ~|, ^, ~^, ^~
- Examples:

expression	result
&(0101)	0
|(0101)	1
&(01xx)	0
|(01xx)	1

Logical Operators

- !, &&, ||, ==, !=, ===, !==

= =	0	1	z	x
0	1	0	x	x
1	0	1	x	x
z	x	x	x	x
x	x	x	x	x

= = =	0	1	z	x
0	1	0	0	0
1	0	1	0	0
z	0	0	1	0
x	0	0	0	1

- Examples:
  - if (b != c) && (index == 0) …
  - if (inword == 1) || (a != d) …

Other Operators

- Relational: <, <=, >, >=
  - e.g., if (a < size –1) || (b >= 3) …
- Shift: <<, >>:
  - e.g., result = (a << 3);
- Conditional: ?:
  - e.g., y= (a==b) ? a : b;
- Concatenation: {,}
  - e.g., {a,b}…
  - e.g., {4{a}}… (equal to {a, a, a, a})

More on Conditional Operator

```
wire [1:0] select;
wire [15:0] D1, D2, D3, D4;
wire [15:0] bus = (select == 2′b00) ? D1 :
                  (select == 2′b01) ? D2 :
                  (select == 2′b10) ? D3 :
                  (select == 2′b11) ? D4 : 16′bx
```

? :	0	1	x
0	0	x	x
1	x	1	x
x	x	x	x

- “z” is not allowed in *conditional_expression.*
- If *conditional_expression* is ambiguous, both *true_expression* and *false_expression* are evaluated, and the result is calculated on a bitwise basis according to the truth table.

Expressions and Operands

- Expressions combine operands with operators to produce resultant values

  - Examples

    - **assign** THIS_SIG = A_SIG ^ B_SIG;

    - @ (SET **or** RESET) **begin** … **end**

- A operand may be compose of
  - Nets
  - Registers
  - Constants
  - Numbers
  - Bit-select of a net or a register
  - Part-select of a net or a register
  - Memory element
  - A function call
  - Concatenation of any of the above

Operator Precedence

Highest	unary
	multiplication, division, modulus
	add, subtract
	shift
	relational
Lowest	conditional

If unsure, use parentheses!