N26F300 VLSI SYSTEM DESIGN (GRADUATE LEVEL)

Fall 2022

Verilog/SystemVerilog (I)

Outline

History of Verilog

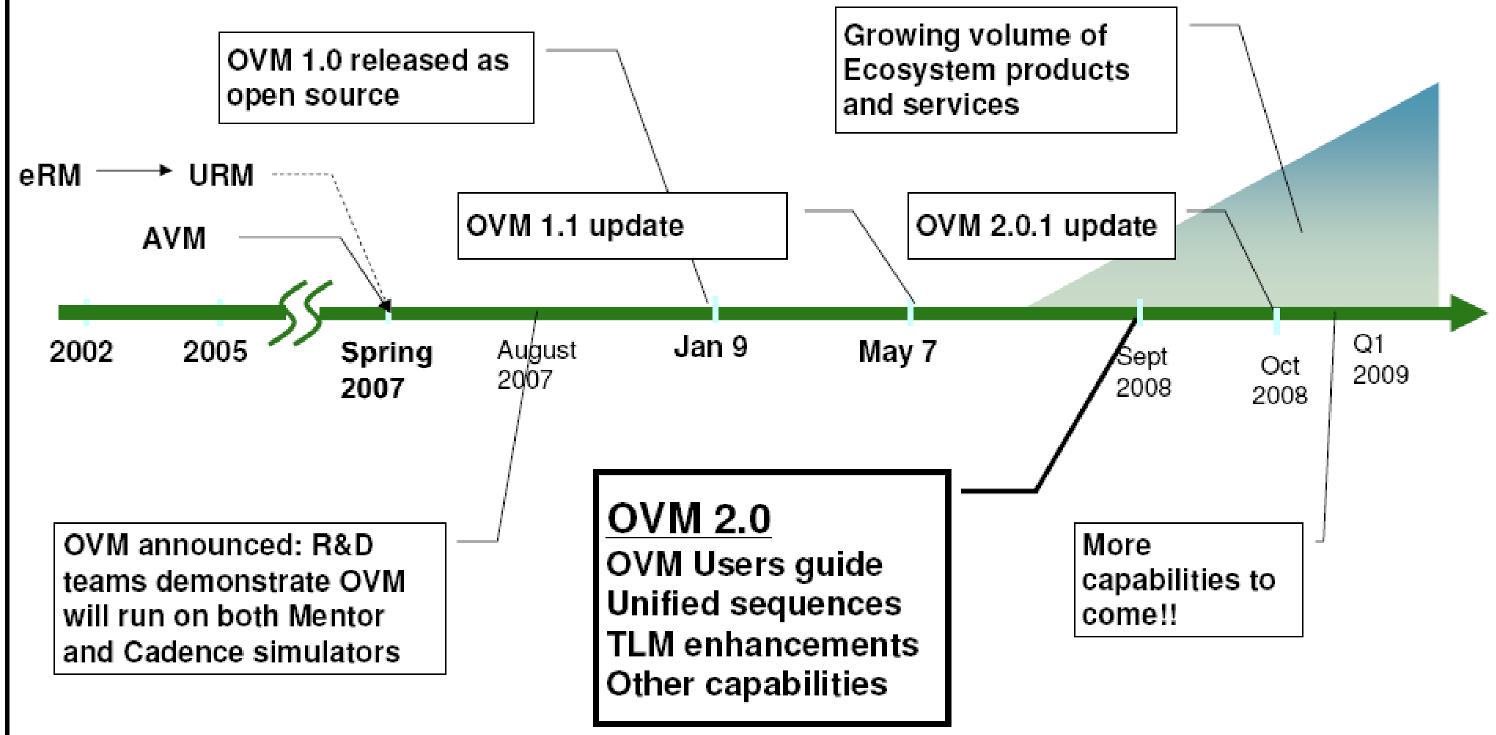
Hardware Description Languages (HDL)

- Similarity and Uniqueness
  - Similar to general-purpose languages like C
  - Additional features
    - Modeling and simulation of the functionality of combinational and sequential circuits
    - Parallel vs. sequential behaviors
- Two competitive forces
  - Verilog: C-like language
  - VHDL: ADA-like language

Hardware Verification Language (HVL)

- Verify the designs of electronic circuits
- Include features of a high-level programming language like C++ or Java as well as features for easy bit-level manipulation
- Provide constrained random stimulus generation, and functional coverage constructs to assist with complex hardware verification.
- Examples
  - SystemVerilog, OpenVera, e(specman), PSL, and SystemC are the most commonly used HVLs.
  - SystemVerilog attempts to combine HDL and HVL constructs into a single standard.

Evolution of HVLs



OVM 1.0 released as open source
Growing volume of Ecosystem products and services
eRM
URM
AVM
OVM 1.1 update
OVM 2.0.1 update
2002
2005
Spring 2007
August 2007
Jan 9
May 7
Sept 2008
Oct 2008
Q1 2009
OVM announced: R&D teams demonstrate OVM will run on both Mentor and Cadence simulators
OVM 2.0
OVM Users guide
Unified sequences
TLM enhancements
Other capabilities
More capabilities to come!!

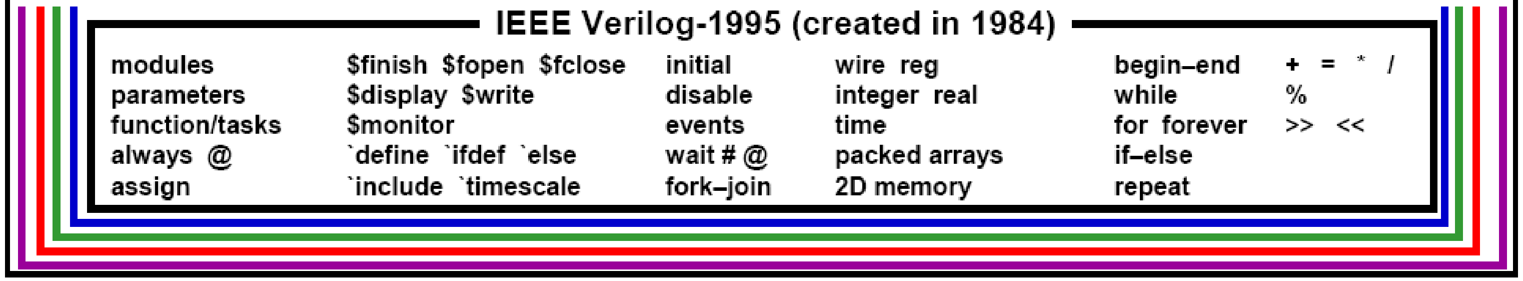
What HDL can do?

- Simulate the behavior of a circuit before it is actually realized.
  - Describe models in common language
  - Serve as a medium for exchanging designs between designers
  - Execute the models as if they are hardware
  - Be translated into gate-level designs
  - Increase design productivity
- Enable designers to verify their designs effectively
  - Help to shorten the verification period
  - Reduce the number of bugs

Development of Verilog

1984	Gateway Design Automation, Phil Moorby
1986	Verilog-XL: an efficient gate-level simulator
1988	Verilog logic synthesizer, Synopsys
1989	Cadence Data System Inc. acquired Gateway
1990	Verilog HDL is released to public domain
1991	Open Verilog International (OVI)
1994	IEEE 1364 Working group
1995	Verilog becomes an IEEE standard (IEEE Std. 1364)
2001	Verilog-2001, significant upgrade
2005	Verilog-2005, minor corrections, branching Verilog-ASM
2009	SystemVerilog, a superset of Verilog-2005, add design verification and design modeling
2013	SystemVerilog-2012, stable version

SystemVerilog extends Verilog



IEEE Verilog-1995 (created in 1984)

modules	\$finish \$fopen \$fclose	initial	wire reg	begin–end	+ = * /
parameters	\$display \$write	disable	integer real	while	%
function/tasks	\$monitor	events	time	for forever	>> <<
always @	\`define \`ifdef \`else	wait # @	packed arrays	if–else	
assign	\`include \`timescale	fork–join	2D memory	repeat	

Why?

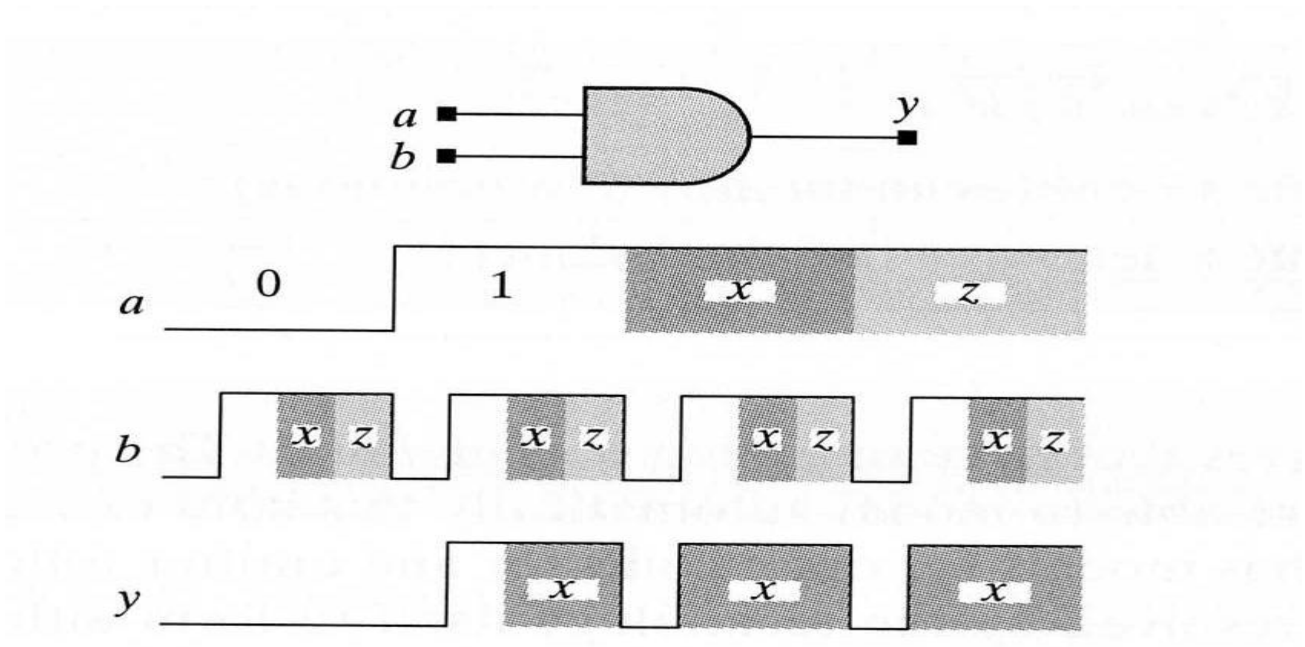
http://www.sutherland-hdl.com/

Source: C. M. Huang / SystemVerilog

Logic Values

Logic values of the signal

- 4 valued logic: 0, 1, x, and z
  - x: unknow
  - z: high impedance



Four Logic Values

- **1** or High, also **H**, usually representing TRUE.
- **0** or Low, also **L**, usually representing FALSE.
- **X** representing "Unknown", "Don't Know", or "Don't Care".
- **Z** representing "high impedance", or a disconnected input.
- Note
  - X only exists in simulators, not in real hardware

Unknown and High-impedance

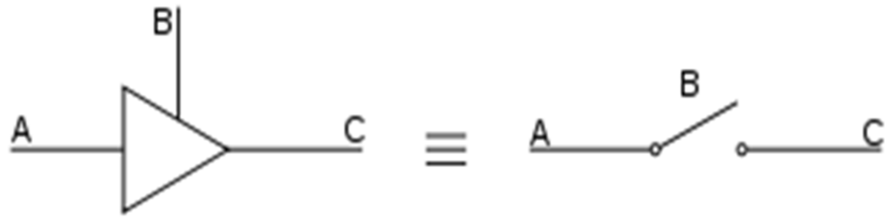
- Unknown (x)
  - Value of signal can not be determined
  - Causes of Value unknown
    - uninitialized
    - Design error → two driving sources to the same signal
    - Design error → unknown state in case selector
    - Design error → MSB of memory address went unknown
- High impedance (z)
  - Floating or tri-stated

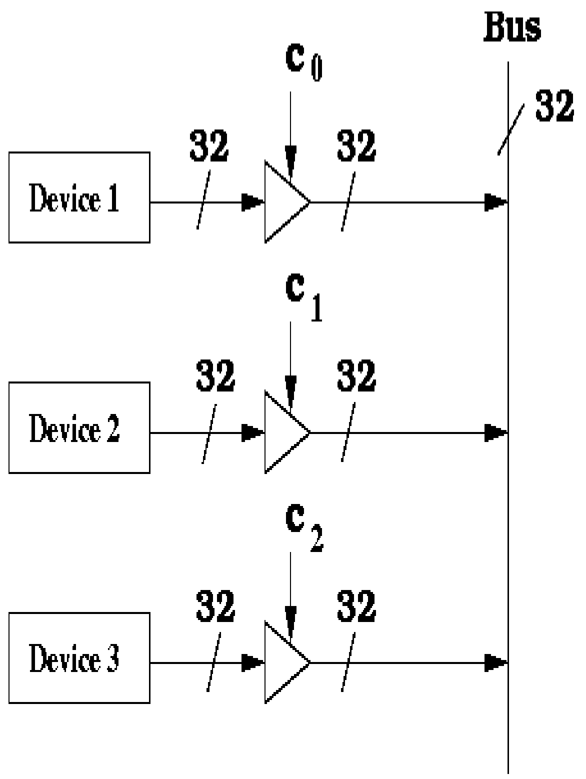
Unknown, X

- Result of a design error
  - Two or more sources driving the same net at the same time
  - Stable output of a flip-flop have not been reached
- Uninitialized memory values or input values before their real values are asserted

High Impedance, Z

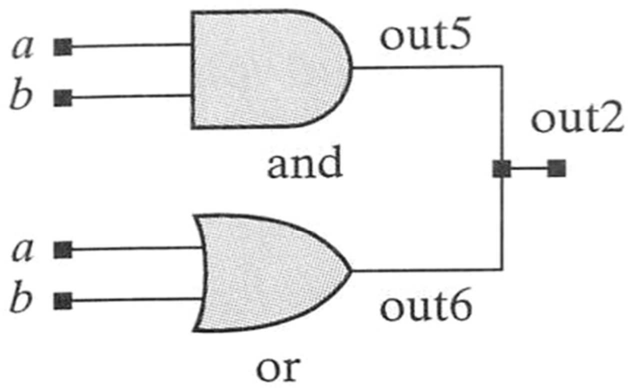
- A disconnected output
- Usually used in bus, a collection of wires in parallel
  - Several devices can communicate one at a time by the same channel.

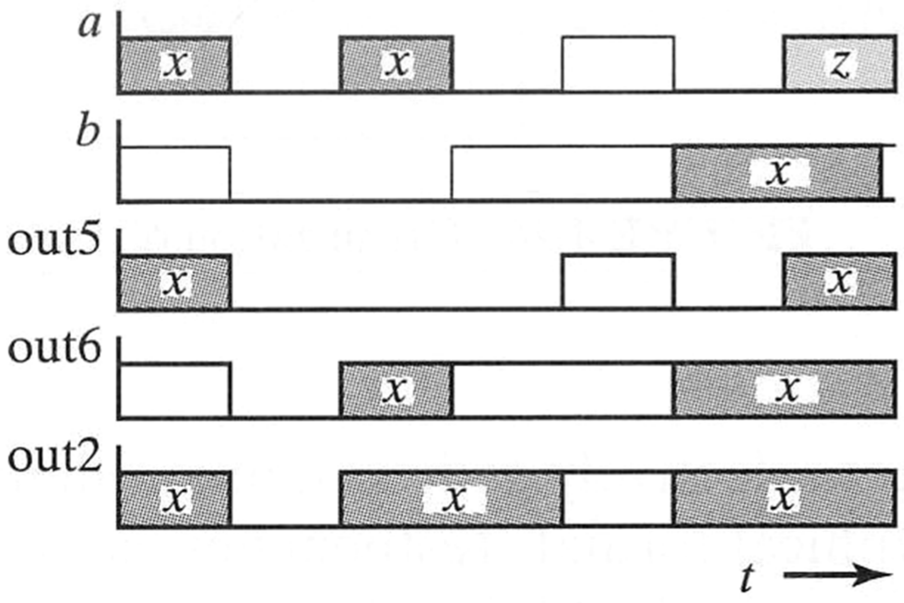




Source: www.wikipedia.org

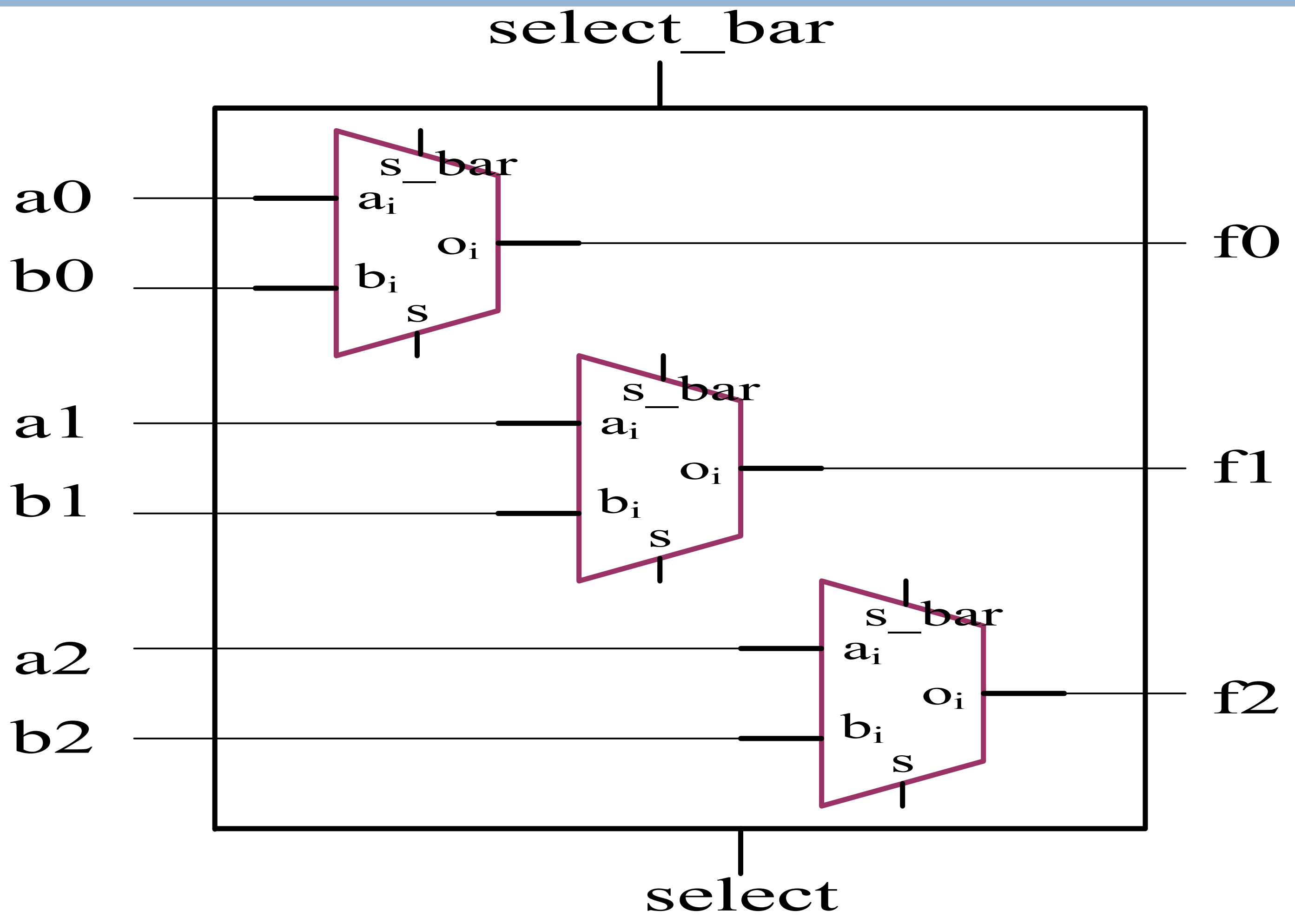
Signal Contention and Resolution



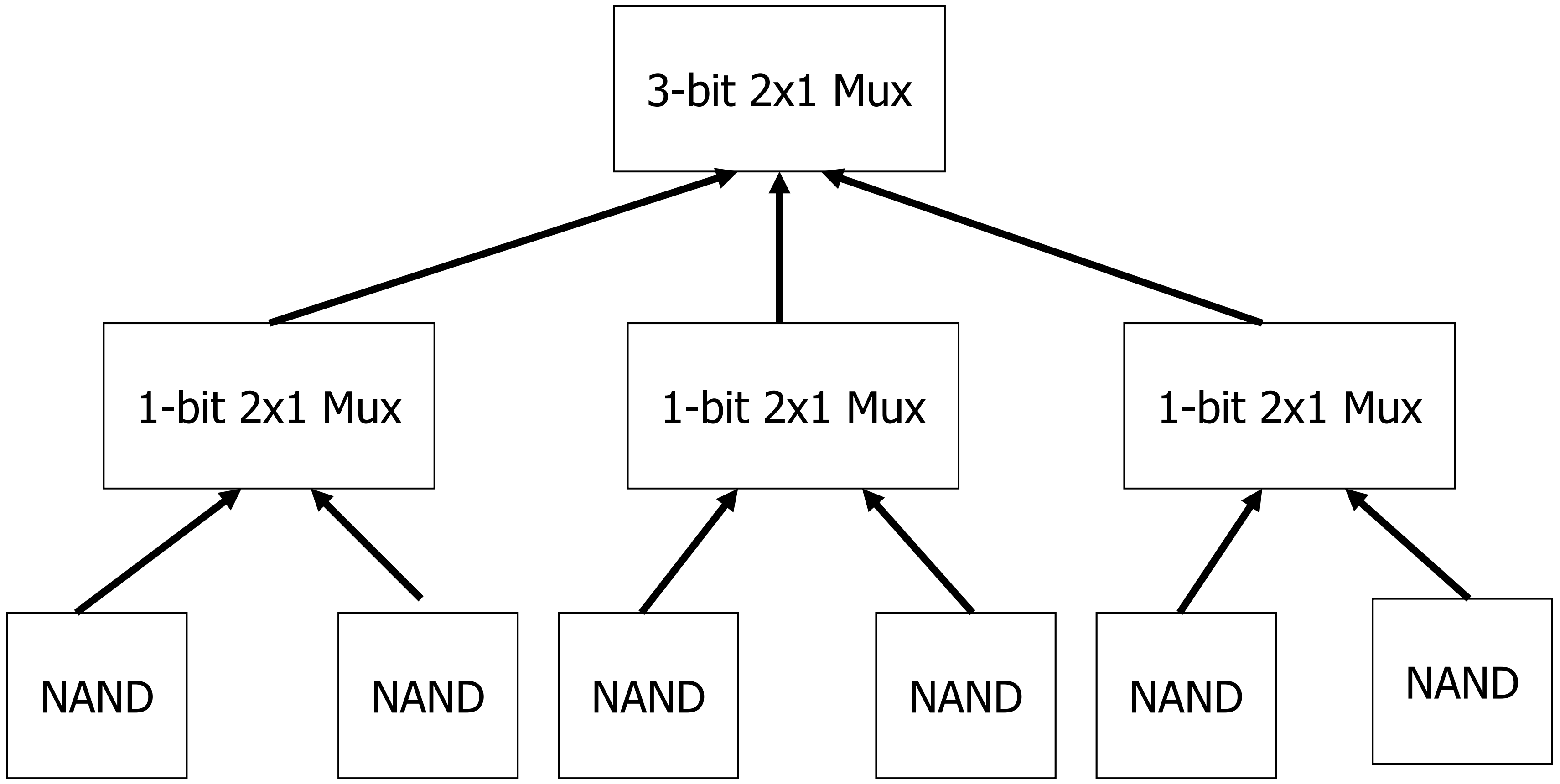


Structure Style of Modeling

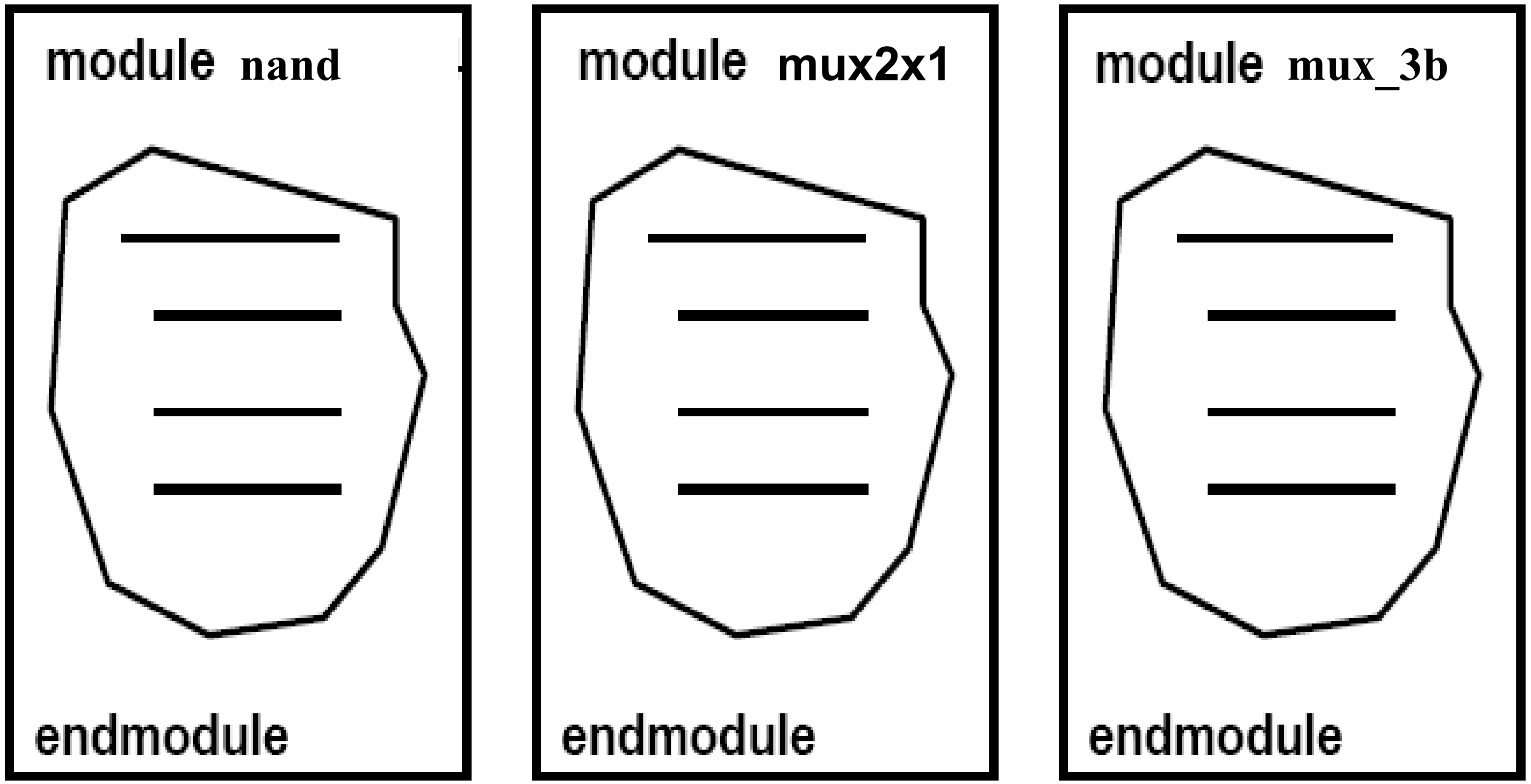
3-bit mux



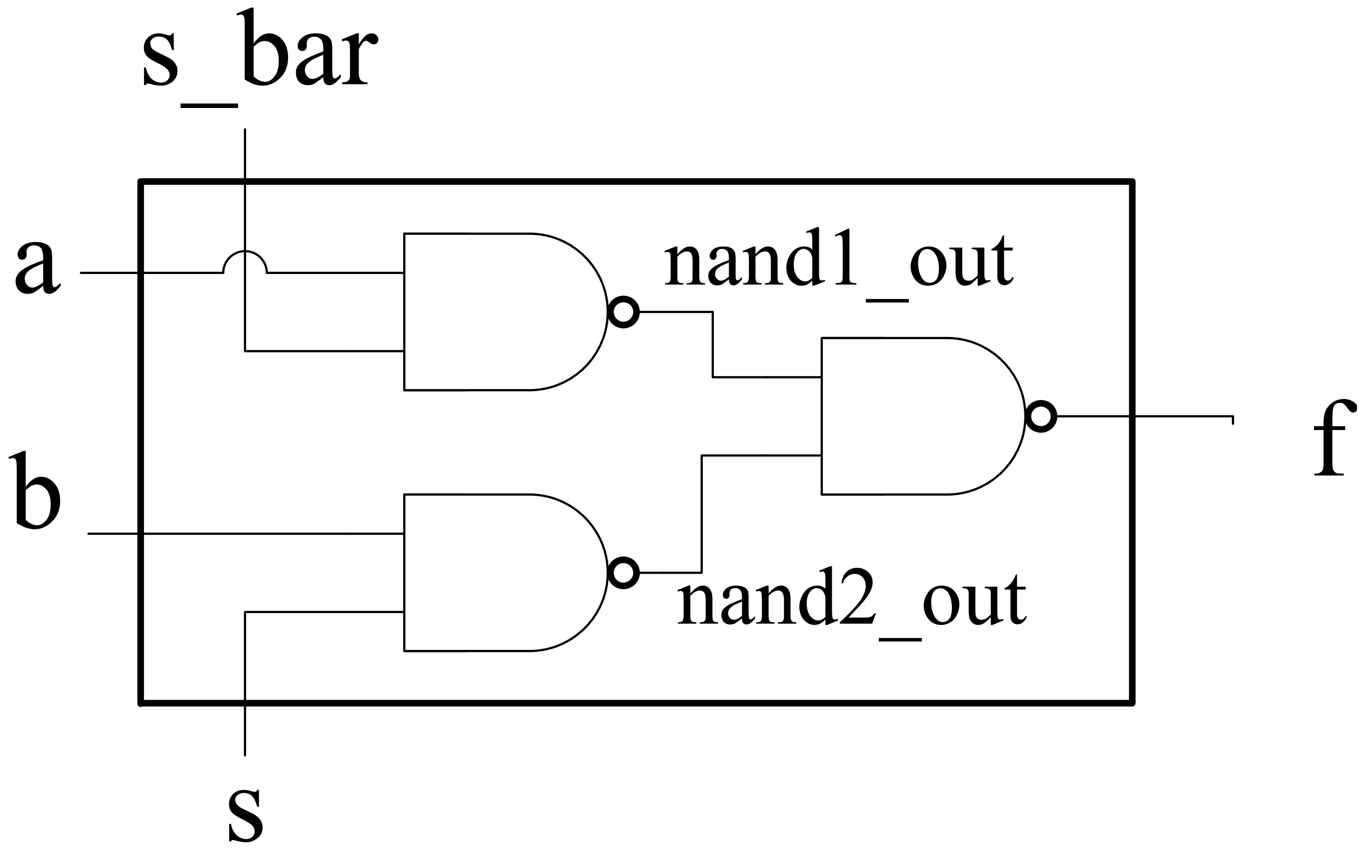
3-bit 2x1 Multiplexier



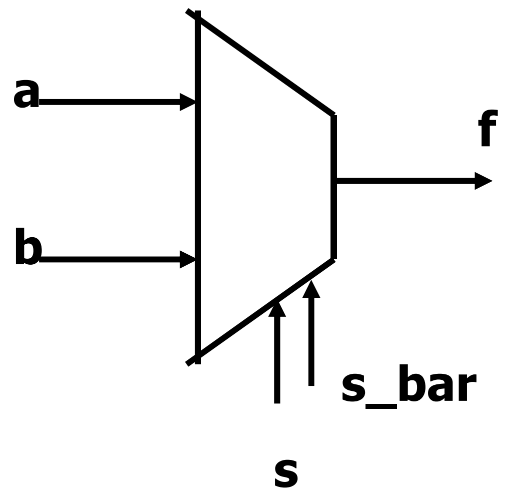
Concepts of Modules



One-bit mux



Mux2x1



```
module mux2x1(f, s ,s_bar, a, b);
   output  f;
   input  s, s_bar;
   input  a, b;
   wire  nand1_out, nand2_out;
  // boolean function
   nand(nand1_out, s_bar, a);
   nand(nand2_out, s, b);
   nand(f, nand1_out, nand2_out);
endmodule
```

Verilog Implementation

```
module mux_3b (f2,f1,f0,a0,b0,a1,b1,a2,b2,
select,select_bar);

   input a0,b0,a1,b1,a2,b2;
   input select, select_bar;
   output f2,f1,f0;

   mux2x1   M0(f0,select, select_bar, a0,b0);
   mux2x1   M1(f1,select, select_bar, a1,b1);
   mux2x1   M2(f2,select, select_bar, a2,b2);

endmodule
```

Data Structure

Wire and Register Revisited

- Net: ***wire***
  - Acts like wires in a physical circuit
  - Connects design objects
  - Needs for a driver

- Register: ***reg, integer***
  - Acts like variables in ordinary procedural languages
  - Stores information while the program executes
  - No needs for a driver, changes its value as wish

Types of Nets

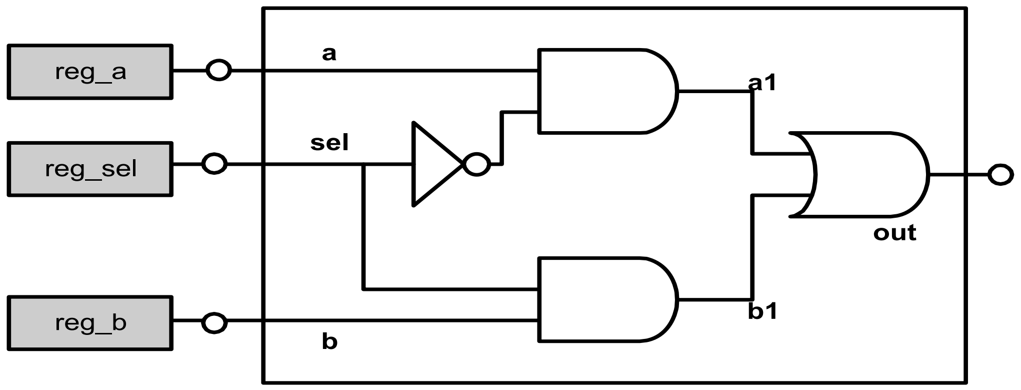
Net Types	Functionality
wire, tri	for standard interconnection wires (default)
wor, trior	for multiple drivers that are Wire-ORed
wand, triand	for multiple drivers that are Wire-ANDed
trireg	for nets with capacitive storage
tri1	for nets which pull up when not driven
tri0	for nets which pull down when not driven
supply1	for power rails
supply0	for ground rails

Nets that are defaulted to single bit nets of type wire. This can be overridden by using the following compiler directive.

'default_nettype <nettype>

Registers

- A register is merely a variable, which holds its value until a new value is assigned to it. It is different from a hardware register.
- Registers are used extensively in behavioral modeling and in applying stimuli.
- Values are applied to registers using behavioral constructs.

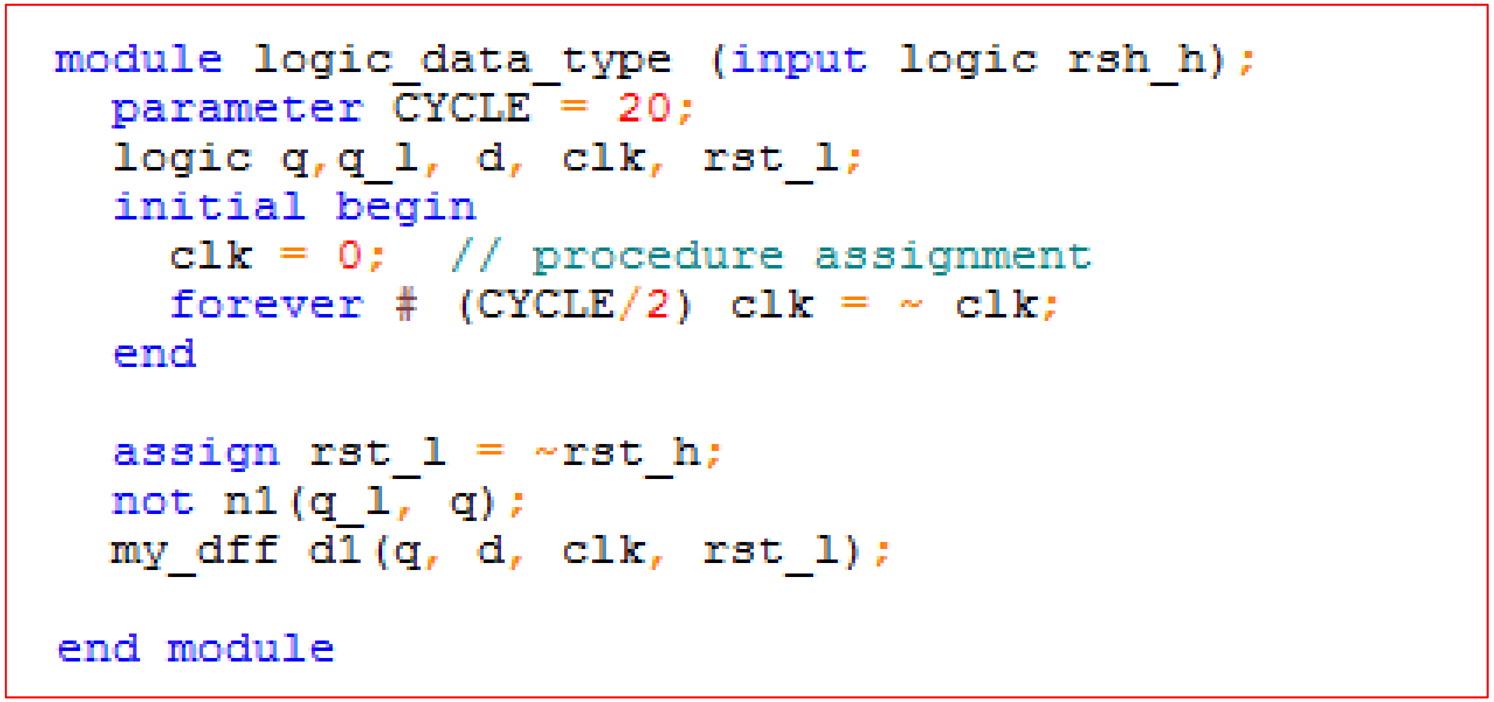


Types in SystemVerilog-2012

Type	Values	Width	New
reg	{0,1,X,Z}	1+	
logic	{0,1,X,Z}	1+	✓
integer	{0,1,X,Z}	32	
bit	{0,1}	1+	✓
byte	{0,1}	8	✓
shortint	{0,1}	16	✓
int	{0,1}	32	✓
longint	{0,1}	64	✓

- *reg* & *logic* are now the same: allowed in continuous or procedural assignment, however, *logic* type cannot be used to create tri-state lines.
- *logic* is preferred than *reg* because of the name.

Using the logic type



```
module logic_data_type (input logic rsh_h);
  parameter CYCLE = 20;
  logic q,q_l, d, clk, rst_l;
  initial begin
    clk = 0;   // procedure assignment
    forever # (CYCLE/2) clk = ~ clk;
  end

  assign rst_l = ~rst_h;
  not n1(q_l, q);
  my_dff d1(q, d, clk, rst_l);

end module
```

2-State Data Types

- Use for improving simulator performance and reduce memory usage over 4-state types
- byte, shortint, int, longint
- Use carefully in distinguish unsigned and signed as well as issues related to connecting to 4-state variables

```
bit x;               // 2-state, single-bit
bit [31:0]  x32;     // 2-state, 32-bit unsigned integer
int i;               // 2-state, 32-bit signed integer
byte  x8;            // 2-state, 8-bit signed integer

integer i4;          // 4-state, 32-bit signed integer
time t;              // 4-state, 64-bit unsigned integer
real  r;             // 2-state, double precision floating point
```

Other Types

Type	Description	Note
time	64-bit unsigned	
shortreal	Like float in C	
real	Like double in C	
realtime	Identical to time	

String Data Type

```
1: byte     c = “A”;      //assign c with “A”
2: bit  [1:4][7:0]        h = “hello”;  // assign h with  “ello”
```

Enumerated Type

```
1: enum   {FETCH, DECODE, EXEC}     STATE;
2: initial begin
3:      STATE = FETCH;
4:      #1 STATE = 3; // This will produce and error
5: end
```

2-state and 4-state Data Types

- The difference between `int` and `integer` is that `int` is a 2-state type and `integer` is a 4-state type.
- The 4-state values have additional bits, which encode the `X` and `Z` states.
- The 2-state data types can simulate faster, take less memory, and are preferred in some design styles.
- The keyword `reg` does not always accurately describe user intent, as it could be perceived to imply a hardware register.
- The keyword `logic` is a more descriptive term. `logic` and `reg` denote the same type.

Signed and Unsigned Data Types

- Integer types when applied to arithmetic can be signed and unsigned, watch out for operators like “<“, “>”, etc.

```
1:  int        unsigned        ui;
2:  int        signed  si;
```

- Byte, shortint, int, integer and longint are default to be signed type
- Bit, reg, and logic are default to unsigned. The same to their extended types – arrays
- Negative numbers are in 2’s complement form

Fixed-size array and Initialization

- Fixed-size array declaration

```
int  fixed_array[0:15];   // 16 ints  [0]..[15]
int  c_style_array[15];   // 16 ints  [0]..[15]

int  Xarray1[0:7][0:15];    // both are equal
int  Xarray2[8][16];
array2[7][3] = 1;           // set array element
```

- Fixed-size array initialization

```
int  fixed_array[0:15];   // 16 ints  [0]..[15]
int  c_style_array[15];   // 16 ints  [0]..[15]

int  Xarray1[0:7][0:15];    // both are equal
int  Xarray2[8][16];
array2[7][3] = 1;           // set array element
```

Using arrays with for- and foreach loops

```
initial begin
  bit [31:0]  src[3], dst[3];
  int mdArray[2][4] = '{'{1,2,3,4}, '{5,6,7,8,9}}

  for (int i=0; i<$size(src); i++)
    src[i] = i;                     // Initialize src array

  foreach (dst[j])
    dst[j] = src[j] * 4;            // Set dst array to 4 * src

  foreach (mdArray[x,y])            // This is the right syntax
    $display ("mdArray[%0d][%0d] = %0d", x, y, mdArray[x][y]);

end
```

Enumerated Types

- Provide a means to declare an abstract variable with a valid value, yet synthesiazble

Verilog style

```
`define FETCH   3’h0
`define WRITE   3’h1
`define ADD     3’h2
. . .
module controller(……)
   parameter   WAIT = 0; LOAD = 1; STORE = 2; // state assignment
   reg [1:0]  State, NextState; // 2 bits for 3 states above
 always @(posedge clk, negedge resetN)
   … …           // finite state machine statements
```

SystemVerilog style

```
`define …
module controller (……)
 enum {WAIT, LOAD, STORE}State, NextState;//default enum base type int
 always @(posedge clk, negedge resetN)
 … …             // finite state machine statements
```

Primitive Data Type: String

- The string data type is an ordered collection of characters.
- The length of a string variable is the number of characters in the collection.
- Variables of type string are dynamic as their length may vary during simulation.
- A single character of a string variable may be selected for reading or writing by indexing the variable.
- A single character of a string variable is of type byte.

```
    string mystring = "hello world";
```

String Data Type   String Variable   String Literal

Summary

- Know the evolution of Verilog/SystemVerilog
- Logic vs register type
- 2-state vs 4-state data types
- Signed vs unsigned number representation
- Compact declaration
- Enumerated type
- String type