CS4161 Digital IP Authoring Fall 2008

Summary for IEEE Verilog 1363-2001

Chih-Tsun Huang (黃稚存)

Department of Computer Science
National Tsing Hua University

Original Lecture from Prof. Juinn-Dar Huang, NCTU

Verilog-2001

- The official standard is IEEE Std. 1364-2001
- "Verilog-2001 – a guide to the new features of the Verilog HDL," S. Sutherland, KAP, 2002
  - outlines 45 major enhancements
    - some for better synthesizable RTL writing
    - some for convenient testbench construction
    - some for gate-level simulation improvement

Combined Port/Type Declarations

```
module adder(sum, co, a, b, ci);
  output [31:0] sum;
  output        co;
  input  [31:0] a, b;
  input         ci;

  reg    [31:0] sum;
  reg           co;
  wire   [31:0] a, b;
  wire          ci;
```

Verilog-1995

```
module adder(sum, co, a, b, ci);
  output reg  [31:0] sum;
  output reg         co;
  input  wire [31:0] a, b;
  input  wire        ci;
```

Verilog-2001

ANSI-C Style Port List

```
module adder(
  output [31:0] sum,
  output        co,
  input  [31:0] a, b,
  input         ci
);

  reg    [31:0] sum;
  reg           co;
  wire   [31:0] a, b;
  wire          ci;
```

Verilog-2001

```
module adder(
  output reg  [31:0] sum,
  output reg         co,
  input  wire [31:0] a, b,
  input  wire        ci
);
```

Verilog-2001

Can apply to **task** and **function** declarations

Module Port Parameter List

```
module adder(sum, co, a, b, ci);
  parameter MSB = 31,
            LSB = 0;

  output [MSB:LSB] sum;
  output           co;
  input  [MSB:LSB] a, b;
  input            ci;

  reg    [MSB:LSB] sum;
  reg              co;
  wire   [MSB:LSB] a, b;
  wire             ci;
```

Verilog-1995

```
module adder
  #(parameter MSB = 31, LSB = 0)
( output reg  [MSB:LSB] sum,
  output reg            co,
  input  wire [MSB:LSB] a, b,
  input  wire           ci
);
```

Verilog-2001

Variable Declaration with Initialization

```
module test;

  reg clock;

  initial
    clock = 0;
```

Verilog-1995

```
module test;

  reg clock = 0;
  // initialized once
  // at time 0
```

Verilog-2001

Automatic(Re-Entrant) Task/Function

- In Verilog-1995, tasks and functions are static
  - allocated storage are static
  - prevent re-entrance and recursion

```
function automatic [12:0] factorial
  input wire [2:0] n;

  if(n==0 || n==1)
    factorial = 1;
  else
    factorial = n * factorial(n-1);
    // recursive call

endfunction
```

Verilog-2001

Constant Function

```
module ram (...);
parameter SIZE = 4096;
parameter ADDR = 12;
input [ADDR-1:0] addr_bus;
...
```

Verilog-1995

```
module ram (...);
parameter SIZE = 4096;
input [log2(SIZE)-1:0] addr_bus;
...

function integer log2;
  input integer depth;
  begin
    for(log2=0; depth>0; log2=log2+1)
      depth = depth >> 1;
  end
endfunction
```

Verilog-2001

Comma Separated Sensitivity List

```
always @(a or b or ci)
  sum = a + b + ci;

always @(posedge clk
         or negedge rst_n)
  if(rst_n = 1'b0)
    q <= 0;
  else
    q <= d;
```

Verilog-1995

```
always @(a, b, ci)
  sum = a + b + ci;

always @(posedge clk,
         negedge rst_n)
  if(rst_n = 1'b0)
    q <= 0;
  else
    q <= d;
```

Verilog-2001

Combinational Logic Sensitivity List

```
always @(a or b or sel)
  case(sel)
    1'b0: y = a;
    1'b1: y = b;
  endcase
```

Verilog-1995

```
always @* // @(*) is also OK
  case(sel)
    1'b0: y = a;
    1'b1: y = b;
  endcase
```

Verilog-2001

Vector Part Select

```
[<starting_bit>+:<width>] part-select increments from the starting bit
[<starting_bit>-:<width>] part-select decrements from the starting bit


reg [63:0] vector1;  // little-endian
reg [0:63] vector2;  // big-endian

byte = vector1[31-:8];  // selects vector1[31:24]
byte = vector1[24+:8];  // selects vector1[31:24]
byte = vector2[31-:8];  // selects vector2[24:31]
byte = vector2[24+:8];  // selects vector2[24:31]
```

Verilog-2001

Multi-Dimensional Array

```
reg [31:0] array_1D[0:127];         // also valid in Verilog-1995
reg [31:0] array_2D[0:127][0:127];  // only valid in Verilog-2001

reg [31:0] data;

data = array_2D[3][8];
```

Verilog-2001

Arrays of Net and Real

```
// In Verilog-1995,
// only 1D arrays of reg, integer and time are allowed


// In Verilog-2001
wire [31:0] array_1D[0:127];
real        array_2D[0:127][0:127];
```

Verilog-2001

Array Bit and Part Select

```
reg [31:0] ram [0:255];
reg  [7:0] high_byte;
reg [31:0] temp;

temp = ram[5];
high_byte = temp[31:24];
```

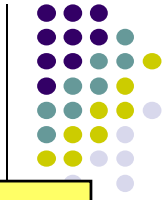
Verilog-1995

```
reg [31:0] ram [0:255];
reg  [7:0] high_byte;


high_byte = ram[5][31:24];
```

Verilog-2001

Signed Arithmetic Extensions



* reg and net data types can be declared as signed

```
reg  signed [31:0] data;
wire signed [11:0] address;
```

* Function returns can be declared as signed

```
function signed [63:0] alu;
```

* Literal integer numbers can be declared as signed

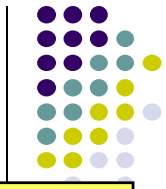
```
16'shC501 //a signed 16-bit hex value
```

* New arithmetic shift operators, <<<, >>>, maintain the sign of a value

* New $signed() and $unsigned() system functions can "cast" a value to signed or unsigned

Verilog-2001

Width Extension Beyond 32 Bits



```
reg [63:0] d1, d2, d3;
integer r1;  // 32-bit signed
reg [31:0] r2;

r1 = -1;  // 32'hffffffff
r2 = -1;  // 32'hffffffff

d1 = r1;  // 64'hffffffffffffffff
d2 = r2;  // 64'h00000000ffffffff
d3 = 'bz; // 64'h00000000zzzzzzzz
```

Verilog-1995

```
reg [63:0] d1, d2, d3;
integer r1;  // 32-bit signed
reg [31:0] r2;

r1 = -1;  // 32'hffffffff
r2 = -1;  // 32'hffffffff

d1 = r1;  // 64'hffffffffffffffff
d2 = r2;  // 64'hffffffffffffffff
d3 = 'bz; // 64'hzzzzzzzzzzzzzzzz
```

Verilog-2001

Caution! Not Backward Compatible

Power Operator **

```
module ram (...);
parameter ADDR_WIDTH = 12;
...
reg [0:(2**ADDR_WIDTH)-1] ram_data;
...
```

* the return value is unsigned if both operands are unsigned
* or, it will be `real`

Verilog-2001

Attribute

```
// one-hot encoding
always @(state)
  case(state) // synopsys full_case parallel_case
    3'b001: next_state = 3'b010;
    3'b010: next_state = 3'b100;
    3'b100: next_state = 3'b001;
  endcase
```

Verilog-1995

```
// one-hot encoding
always @(state)
  (* full_case, parallel_case *) case(state)
    3'b001: next_state = 3'b010;
    3'b010: next_state = 3'b100;
    3'b100: next_state = 3'b001;
  endcase
```

Verilog-2001

Attributes are not standardized yet

Sized and Typed Parameter

```
module fsm(...);
  parameter IDLE = 3'd0,
            S1   = 3'd1,
            S2   = 3'd2,
            S3   = 3'd3,
            S4   = 3'd4;
```

Verilog-1995

```
module fsm(...);
  parameter [2:0]
            IDLE = 3'd0,
            S1   = 3'd1,
            S2   = 3'd2,
            S3   = 3'd3,
            S4   = 3'd4;

* Available formats
parameter list_para_assign;
parameter [msb:lsb] list_para_assign;
parameter signed list_para_assign;
parameter signed [msb:lsb] list_para_assign;
parameter integer list_para_assign;
parameter time list_para_assign;
parameter real list_para_assign;
parameter realtime list_para_assign;
```

Verilog-2001

Explicit In-Line Parameter Passing

```
module ram (...);
parameter SYNC = 1;
// 0 for async
parameter WIDTH = 10;
parameter SIZE = 1024;
...

ram mem1 #(1, 12, 4096) (...);
```

Verilog-1995

```
module ram (...);
parameter SYNC = 1;
// 0 for async
parameter WIDTH = 10;
parameter SIZE = 1024;
...

ram mem1 #(.SIZE(4096),
           .WIDTH(12)) (...);
```

Verilog-2001

Fixed Local Parameter

```
module multiplier (a, b, product);
parameter a_width = 8, b_width = 8;
localparam product_width = a_width + b_width;
input [a_width-1:0] a;
input [b_width-1:0] b;
output [product_width-1:0] product;
...
```

parameter : can be redefined by defparam or in-line parameter redefinition
localparam : cannot be redefined

Verilog-2001

Enhanced File I/O and String Tasks

*** Open up to 2^{30} files**

*** New built-in file I/O tasks:**
$fgetc, $ungetc, $fgets
$fscanf, $fread,
$ftell, $fseek, $rewind, $fflush
$ferror

*** New string manipulation tasks:**
$swrite, $swriteb, $swriteh, $swriteo, $sformat
$sscanf

Verilog-2001

Enhanced Invocation Option Testing

```
initial
begin
  if($test$plusargs(“test1”))
    $readmemh(“test1.dat”, vector);
  else if($test$plusargs(“test2”))
    $readmemh(“test2.dat”, vector);
  else
    $display(“Error”);
end

-- command line
verilog +test1
```

Verilog-1995

```
initial
begin
  if($value$plusargs(“test=%s”, fs))
    $readmemh(fs, vector);
  else
    $display(“Error”);
end


-- command line
verilog +test=test1.dat
```

Verilog-2001

Enhanced Conditional Compilation

In Verilog 1995,
only `ifdef, `else, `endif, and `undef are supported

In Verilog 2001, two more are added
`ifndef and `elsif

Verilog-2001

Generate

- Use *for* loops to generate any number of instances of:
  - modules, primitives, procedures, continuous assignments, tasks, functions, variables, nets
- Use *if*–*else* and *case* decisions to control what instances are generated
  - provides greater control than the VHDL generate
- New reserved words added:
  - *generate*, *endgenerate*, *genvar*

Generate Example (1/2)

```
generate
  genvar i;

  for(i = 0; i <= 7; i = i + 1)
  begin: u
    adder8 add(sum[(i*8)+:8], co[i+1],
               a[(i*8)+:8], b[(i*8)+:8], ci[i]);
  end
endgenerate
```

u[0].add, u[1].add, …, u[7].add are generated

Verilog-2001

Generate Example (2/2)

```
module multiplier (a, b, product);
  parameter a_width = 8, b_width = 8;
  localparam product_width = a_width + b_width;
  input [a_width-1:0] a;
  input [b_width-1:0] b;
  output [product_width-1:0] product;

  generate
    if ((a_width < 8) || (b_width < 8))
      CLA_multiplier #(a_width, b_width) u1 (a, b, product);
    else
      WALLACE_multiplier #(a_width, b_width) u1 (a, b, product);
  endgenerate

endmodule
```

Verilog-2001

Configuration

Verilog Design

```
module test;
  ...
  myChip dut (...);
  ...
endmodule
```

```
module myChip(...);
  ...
  adder a1 (...);
  adder a2 (...);
  ...
endmodule
```

Configuration Block (part of Verilog source code)

```
/* define a name for this configuration */
config cfg4

  /* specify where to find top level modules */
  design rtlLib.top

  /* set the default search order for finding
     instantiated modules */
  default liblist rtlLib gateLib;

  /* explicitly specify which library to use
     for the following module instance */
  instance test.dut.a2 liblist gateLib;
endconfig
```

Library Map File (separate from Verilog source)

```
/* location of RTL models (current directory) */
library rtlLib "./*.v";

/* Location of synthesized models */
library gateLib "./synth_out/*.v";
```

More…

Several additional changes are not listed here

- Library vendor enhancements for Deep Sub-micron
- Programming Language Interface (PLI) enhancements
- Value Change Dump (VCD) file enhancements
- Enhanced timing check capability
- Support for the latest Standard Delay File (SDF) standard
- Errata, clarifications and minor changes