CT Verilog Series

Verilog Overview Part 2
Dataflow Modeling

黃稚存

Chih-Tsun Huang

cthuang@cs.nthu.edu.tw

國立清華大學
資訊工程學系

04

聲明

- 本課程之內容 (包括但不限於教材、影片、圖片、檔案資料等)，僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容(例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter, Google Drive, Dropbox 等等)。如有侵權行為，需自負法律責任。

Outline

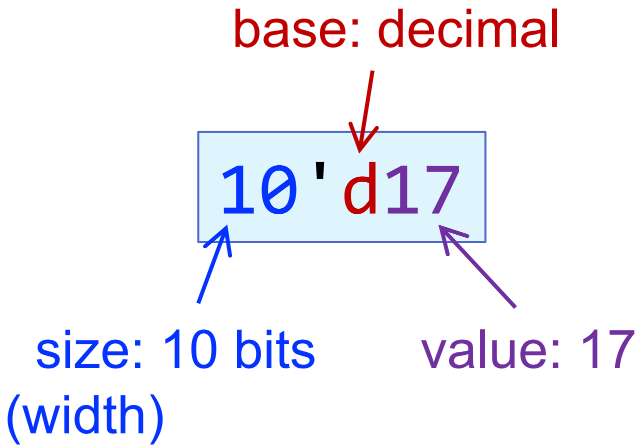
- Numbers
- Data Types
- Operators
- Compiler Directives
- Dataflow Modeling

- Part 1 Recap
  - Background
  - Digital Switches and Logic Gates
  - Tri-state Gates
  - Structural Modeling
  - Delay Model

Numbers

Number Representation (1/2)

- Sized: <size>'<base><value>
  - <size>: number of bits
  - <base>: binary (b), octal (o), decimal (d) or hexadecimal (h)
  - <value>: any legal number in the selected base
    - `0123456789abcdef_`
    - i.e., `'b0010_0111`
  - Default type is 32-bit decimal number
- Unsized: '<base> <value>
  - <base> is optional for decimal
  - Default: 32 bits



`10'd17`

Number Representation (2/2)

- Negative: -<size>’<base><value>
  - Unsigned vs. signed
  - Default numbers are unsinged

```
reg [7:0] address;
wire signed [3:0] constant;
assign constant = -4'd5;
```

- Unknown (x) and high impedance (z)
  - `12'h3z  // 0000 0011 zzzz`
  - `12'b0   // 0000 0000 0000`
  - `7'b1    // 0000001`
  - `7'bz    // zzzzzzz`
  - `7'bx    // xxxxxxx`

Examples: Sized Numbers

Number	# Bits	Base	Decimal	Stored
`2'b10`	2	Binary	2	`10`
`8'b0001_0000`	8	Binary	16	`00010000`
`3'd5`	3	Decimal	5	`101`
`3'o5`	3	Octal	5	`101`
`8'h5`	8	Hex	5	`00000101`
`3'b5`	Invalid!	--	--	--

Signed vs. Unsigned? Subtraction in Verilog

- Using two's complement

```
diff = a + ~b + 1'b1;
```

- Using - operator

```
diff = a - b;
```

  - Do you need to define all the signals as signed numbers?
  - You will probably get a warning like:
    Signed to unsigned assignment occurs. (VER-318)

Signed vs. Unsigned?
signed Data Type

```
module test;
reg signed [7:0] a, sum;
reg signed [3:0] b;
reg [3:0] c;

initial begin
  a = 8'd5;
  b = -7'h3;
  c = 4'b1011;
  #10;
  sum = a + b + c;
  $display("a = %b b = %b c = %b : sum = %d", a, b, c, sum);
  #10;
  sum = a + b + $signed(c);
  $display("a = %b b = %b signed c = %b : sum = %d", a, b, c, sum);
end
endmodule
```

00000101
11111101
1011

00001011

11111011

$$\begin{array}{r} 00000101 \\ 00001101 \\ +\,00001011 \\ \hline 00011101 \\ (\;29\;)_{10} \end{array} \qquad \begin{array}{r} 00000101 \\ 11111101 \\ +\,11111011 \\ \hline 11111101 \\ (\;-3\;)_{10} \end{array}$$

$signed(c)

```
a = 00000101 b = 1101 c = 1011 : sum =    29
a = 00000101 b = 1101 signed c = 1011 : sum =   -3
```

Data Types

Variables

- Syntax

  ```
  <data_type> [<MSB>:<LSB>] <list_of_identifier>
  ```

- Net type: structural connectivity
  - Physical connections between ports
- Register type: abstraction of data storage
  - May or may not be physical storage
  - Only these types of signals can be on the left-hand side of assignments in initial/always blocks)
- Both nets and registers are informally called signals
  - May be either scalar or vector

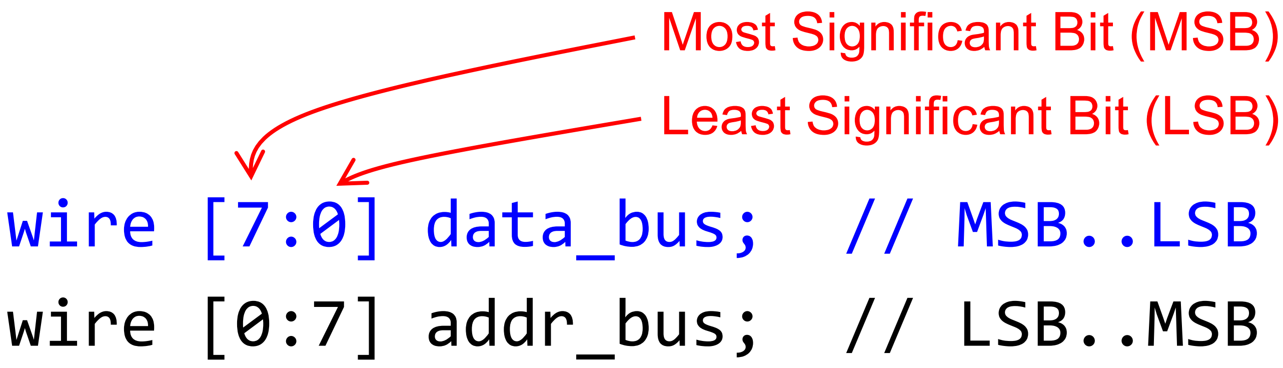
Net Data Types

Most popular type of output/internal variables (signals) in your Verilog design!

- **wire** (default): only to establish connectivity (most popular type of input/internal signals in your Verilog code)
- **tri**: same as wire, but explicitly state that it is tri-stated
- **wand, wor**: wired AND and OR
- **triand, trior**: tri-stated wired AND or OR
- **supply0, supply1**: connected to Gnd and Vdd

Net Examples

- `wire x;`
- `wire [15:0] data;`
  - Bit select: `data [5]`
  - Part select: `data [5:3]`
- Vector representation



Most Significant Bit (MSB)

Least Significant Bit (LSB)

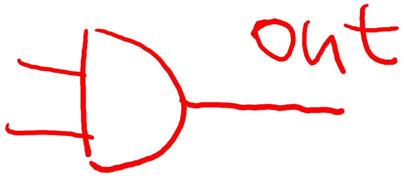
```
wire [7:0] data_bus;  // MSB..LSB
wire [0:7] addr_bus;  // LSB..MSB
```

Net Value Assignment

- Value explicitly assigned by
  - continuous assignment

```
wire [7:0] out;
assign out = a+b;
```

- Value implicitly assigned by
  - being connected to an output terminal of an instance



Initial Value & Undeclared Nets

- At simulated time $t_{sim} = 0$
  - Nets driven by primitives, modules or continuous assignments are determined by their drivers (maybe ***x*** by default)
  - Nets without drivers have default value ***z***
- Undeclared nets
  - default type: 1-bit **`wire`**

Register Data Types

Most popular type of output/internal variables (signals) in your Verilog design!

- **reg**: store a logic value
- **integer**
  - 32-bit two's complement number format
  - Represented internally to the wordlength (at least 32 bits) of a host machine
  - `integer Array_of_Ints [1:100];`
- **real**
  - Double precision, typically 64-bit value
  - May not be connected to a module port or IO of a primitive
- **time**
  - Stores time as unsigned 64-bit value
  - May not be used in a module port or IO of a primitive
  - `time T_samples [1:100];`
- **realtime**
  - Stores time as real number

Register Examples

```verilog
reg a, b;
reg [15:0] counter, shift_reg;
integer c;
reg [31:0] one_word;
reg one_bit;

reg [1:0] state = 2'b00,
          next_state = 2'b00;
```

Not a good coding style: synthesizable with Vivado, but not with other IC synthesis tools!

Vectors

- Vector representation

```
reg  [31:0] busA;
```

  - Vector part-select

```
busA[2] = 0;
data = data_bus[2:0];
```

- Variable Vector Part-Select (Verilog 2001)

```
reg [31:0] data1;
byte = data1[31-:8]  // data1[31:24]
byte = data1[24+:8]  // data1[31:24]

// [7:0], [15:8]… [255:248]
for (j=0; j<=31; j=j+1)
  byte = data1[(j*8)+:8];

// With parameter: byteNum=1 ->data1[15:8]
data1[(byteNum*8)+:8] = 8’b0;
```

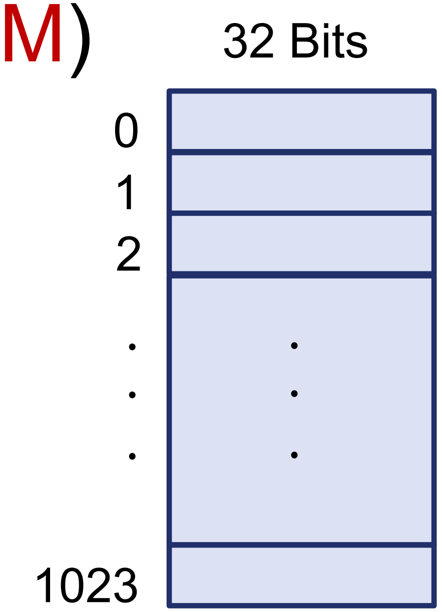
More on Part-Select of Net and Register Variables

- MSB of a part-select of a register ➔ the leftmost array index
- LSB ➔ the 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  (0100)`
  - `word[5:1] // -> 2 (00010)`
  - `word[9]   // -> X`

Arrays (1/2)

- Array is a collection of storage (it is not an SRAM)
  - **reg** `[31:0] cache [0:1023];`
    - 1K array of 32-bit words
    - Array access
      `array[8] = 8'bff;`
- Individual bits can be obtained as follows
  - `one_word = cache[988];`
  - `one_bit = one_word[3];`
- 2-D Array

```
reg [7:0] memory [0:3] [0:7];
```



Arrays (2/2)

- Arrays are allowed for reg, wire, integer and real data types

```
reg  a1 [11:0];
wire [0:7] a2 [3:0]; // 8-bit vector net with a depth of 4
reg  [7:0] a3 [0:1][0:3]; // 2D (2x4) array of 8-bit vectors
```

- Assignment

```
a1 = 0;              // Illegal
a2[0] = 8'ha2;       // Assign 0xa2 to index=0
a2[2] = 8'h1c;       // Assign 0x1c to index=2
a3[1][2] = 8'hdd;    // Assign 0xdd to rows=1 cols=2
a3[0][0] = 8'haa;    // Assign 0xaa to rows=0 cols=0
```

https://www.chipverify.com/verilog/verilog-arrays

Register Value Assignment

- In simulation, a register-type (reg) variable has the initial value X
  - ~~With the simulator of Vivado, the initial value is 0~~
  - **Since Vivado of version 2019, the initial value is X!**
- 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

wire out;

assign out= ...

reg out;

always/initial

out= ...

Strings

- Verilog has no explicit string type
- Must be stored in properly sized **reg** (as an 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.

Operators

Operators

- Arithmetic
- Bitwise
- Reduction
- Logical
- Equality
- Relational
- Shift
- Conditional
- Concatenation
- Replication

Arithmetic Operators

- Binary (two operands): +, -, *, /, %
- Unary (one operand): +, -
- Example
  - A + B
  - A - B
  - -A

Bitwise Operators

- Unary: ~
- Binary: AND &, OR |, XOR ^, XNOR ~^ or ^~
- Shorter operand will extend to the size of longer operand by padding bits with 0
- Example

Expression	Result
~(1010)	0101
(01) & (11)	01
(01) | (11)	11
(01) ^ (11)	10
(01) ~^ (11)	01

```
a & b   // and
        // 4'b1101 & 4'b1011 => 4’b1001
a | b   // or
a ^ b   // xor
```

Reduction Operators

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

Expression	Result
&(0101)	0
|(0101)	1
&(01xx)	0
|(01xx)	1

```
&a
|a   // if a = 4'b1101, |a => 1
^a
```

Logical Operators

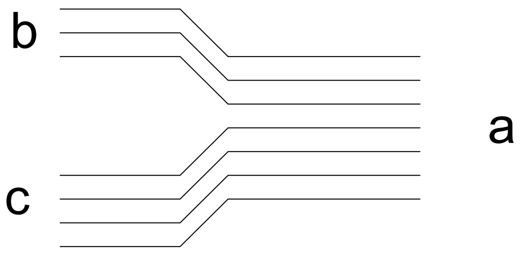
- Unary: `!` `// not ( !a vs. ~a )`
- Binary: `&&, ||`
- Example
  - `!A`
  - `A && B` `// 4'b1101 && 4'b0010 => 1`

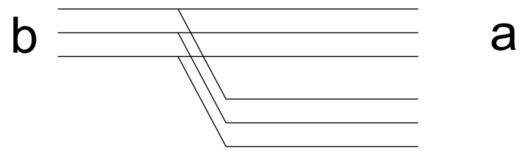
    true true
  - `A || B`

Other Operators

- Relational: <, <=, >, >=
  - e.g., a < size –1, b >= 3
- Shift: <<, >>:
  - e.g., a << 3, a >> 1
- Conditional: ?:
  - e.g., c ? a : b;
- Concatenation: {,}
  - `a = {b, c};`
  - `e[3:0] = {d[2:0], 1'b0};`
  - `{cout, s} = a + b + cin;`
- Replication:
  - `a = {2 {b}};`
  - {4{a}} equal to {a, a, a, a}

d<< 1





Equality 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

- Example
  - b != c
  - index == 0
  - a === d

Expressions and Operands

- Expressions combine operands with operators to produce resultant values
  - E.g., `A_SIG ^ B_SIG`
- An operand may be composed of
  - Nets
  - Registers
  - Constants
  - Numbers
  - Bit-select of a net or a register (e.g., `data[3]`)
  - Part-select of a net or a register (e.g., `data[5:2]`)
  - Memory element
  - A function call
  - Concatenation of any of the above

Operator Precedence

Highest

- unary
- multiplication, division, modulus
- add, subtract
- shift
- relational
- equality
- reduction
- logical
- conditional

Lowest

- If unsure, use parentheses!

Compiler Directives

Compiler Directives (1/2)

- Similar to the compiler directives in C/C++
- `<keyword>

```
`include header.v
`define WORD_SIZE 32
reg [`WORD_SIZE - 1:0] data;

`define S $stop;    $finish;

`define WORD_REG reg [31:0]
`WORD_REG data_in;
```

Compiler Directives (2/2)

- For readability

```
`define VECTOR reg [31:0]
`VECTOR data;
```

- Handy to parameterize 2^n, $2^n + 1$, $2^n – 1$, or parameters related to the power of 2, by the shift operator

```
`define RANGE (1 << `WIDTH) - 1
```

  or

```
parameter 2_POW_n = 1 << n;
```

- Now Verilog supports power operator, i.e., `2**n`
- Verilog also supports `` `ifdef `else `endif ``

Dataflow Modeling

- Or data-flow modeling

Continuous Assignment: **assign**

```
module nand2_1 ( y, x1, x2);
   input     x1, x2;
   output    y;
   wire y;
   assign y = ~ (x1 & x2);
endmodule
```

Explicit continuous assignment

```
module nand2_2 ( y, x1, x2);
   input     x1, x2;
   output    y;
   wire y = ~ (x1 & x2);

endmodule
```

Implicit continuous assignment

(Left-Hand Side)

- *Continuous assignment: static binding between LHS and RHS*
- *No mechanism to eliminate or alter the binding*

(Right-Hand Side)

Coding Style: assign

```
assign n_data =
  (s2==1) ? (s1==1) ?
  (s0==1) ? data : n_data : n_data :
  (s1==0) ? (s0==1) ? R1: R0 :
  (s0==0) ? R2 : R3;
```

Half Adder with Dataflow Modeling

```
module half_adder (sum, c_out, a, b);
  input  a, b;
  output sum, c_out;

  assign {c_out, sum} = a + b;

endmodule
```

Concatenation