EE2026 Digital Design

NUMBER SYSTEMS & VERILOG INTRO

Chua Dingjuan elechuad@nus.edu.sg

NUMBER SYSTEMS

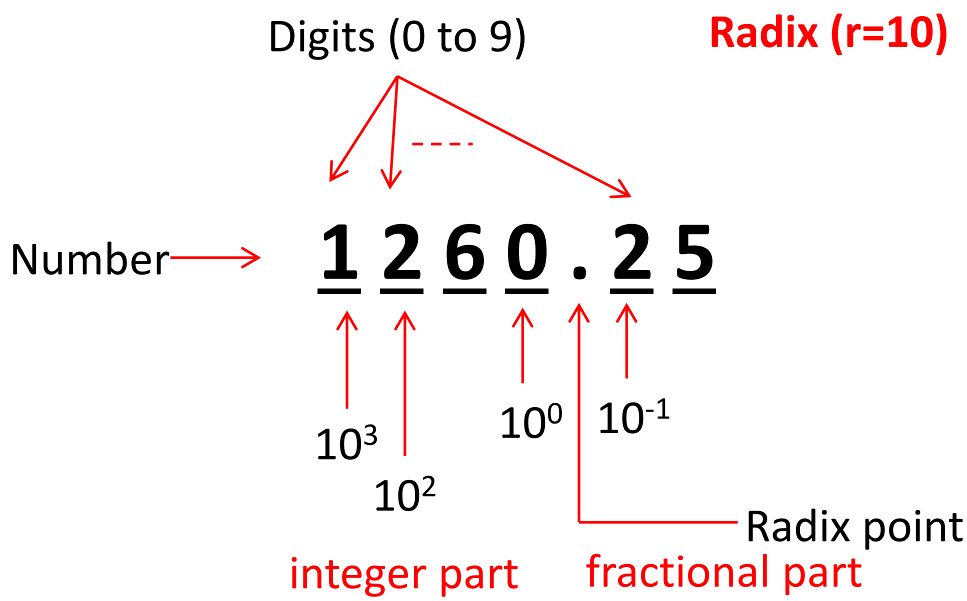
Positional Number Systems : Decimal, Binary, Hex
Binary Arithmetic, Signed Number Representations
Introduction to Verilog

Positional Number System (Decimal)

Decimal number:

Terminologies

- Radix (or base)
- Digits and a numeral (0 → radix-1)
- Radix point
- Place value (or weight) is in the power of the base (positive on the left and negative on the right side of the radix point



$$N = 1\times10^3 + 2\times10^2 + 6\times10^1 + 0\times10^0 + 2\times10^{-1} + 5\times10^{-2} = 1260.25$$

*Weighted sum of each digit (each digit is weighted by its place value)

Radix r and its Decimal Equivalent

General form of Number of radix r:

Radix point

$$A_r = (a_n a_{n-1} \ldots a_o . a_{-1} \ldots a_{-m})_r$$

$$where\ a_n, a_{n-1}, \ldots, a_0, ..a_{-m} \in \{0, \ldots (r-1)\}$$ (Integer only)

Decimal equivalent:

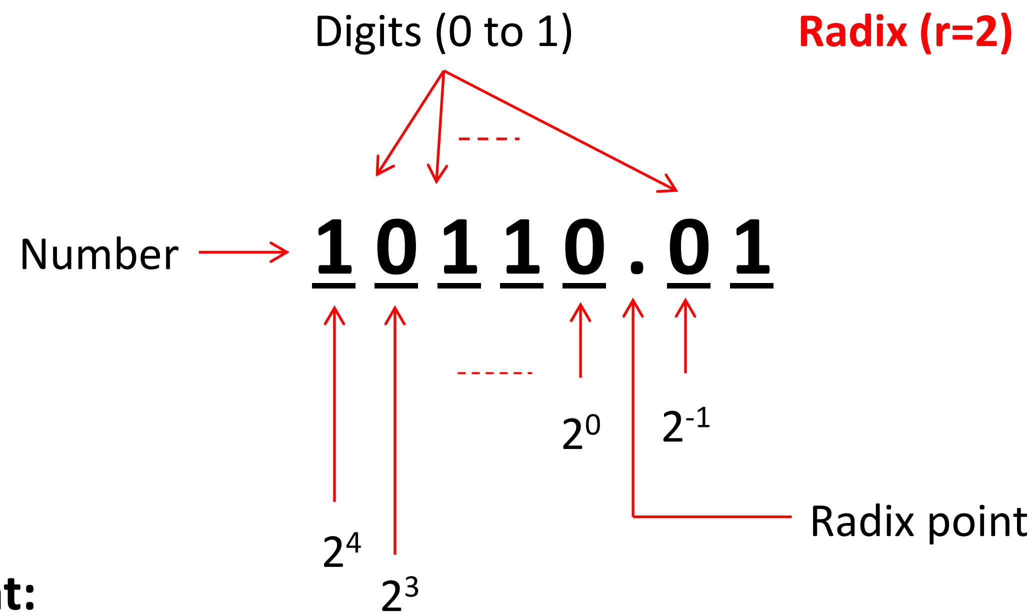
Radix point is here

$$\begin{aligned} A_r &= (a_n a_{n-1} \ldots a_o . a_{-1} \ldots a_{-m})_r \\ &= a_n \times r^n + a_{n-1} \times r^{n-1} + \ldots a_o \times r^0 + a_{-1} \times r^{-1} + \ldots a_{-m} \times r^{-m} \\ &= \sum_{i=-m}^{n} a_i r^i \end{aligned}$$

***Weighted sum of all digits**

Binary Number

Radix (r=2)



Decimal Equivalent:

$$N_{10} = 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2}$$

$$= 16 + 0 + 4 + 2 + 0 + 0 + \frac{1}{4}$$

$$= 22.25$$

$(10110.01)_2 = (22.25)_{10}$

MSB and LSB of a Binary Number

MSB

- Most significant bit

LSB

- Least significant bit

<u>1</u> <u>0</u> <u>1</u> <u>1</u> <u>0</u> <u>0</u> <u>1</u>

(Left-most bit) MSB

LSB (Right-most bit)

***For integer binary number only**

Range

- 0 to $2^n - 1$, where n is the number of bits (2^n values)

Hexadecimal number

$$(\underline{1}\,\underline{1}\,\underline{0}\,\underline{1}\;.\;\underline{0}\,\underline{1}\,\underline{1}\,\underline{0})_2$$

$$(\;\underline{D}\;.\;\underline{6}\;)_{16}$$

Radix (r=16)

Digits (0 to 15)

Number → **$\underline{1}\ \underline{8}\ \underline{F}\ \underline{4}\ .\ \underline{2}\ \underline{A}$**

16^3 16^2 ------ 16^0 16^{-1} 16^{-2}

Radix point

Decimal Equivalent:

$$N_{10} = 1\times16^3 + 8\times16^2 + F\times16^1 + 4\times16^0 + 2\times16^{-1} + 10\times16^{-2}$$

$$= 4096 + 2048 + 240 + 4 + \frac{2}{16} + \frac{10}{256}$$

$$= 6388 + \frac{21}{128}$$

$$\approx 6388.16$$

$(18F4.2A)_{16} \cong (6388.16)_{10}$

Hex	Dec	Bin
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

$$(\ \underline{1}\ \underline{0}\ \underline{1}\ .\ \underline{0}\ \underline{1}\ \underline{1}\ )_2$$

$$(\ \ \underline{5}\ \ .\ \ 3\ \ )_8$$

Octal number

Radix (r=8)

Digits (0 to 7)

Number → $\underline{7}\ \underline{5}\ \underline{4}\ .\ \underline{2}$

8^2 8^0 8^{-1}

Radix point

$(754.2)_8 = (520.25)_{10}$

Oct	Dec
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
?	8
?	9
?	10

Decimal Equivalent:

$$N_{10} = 7\times8^2 + 5\times8^1 + 4\times8^0 + 2\times8^{-1}$$

$$= 448 + 40 + 4 + \frac{2}{8}$$

$$= 492.25$$

Radix Conversion

Three types of conversions:

- Radix r ($r \neq 10$) $\rightarrow$ Decimal
- Decimal $\rightarrow$ Radix r ($r \neq 10$)
- Conversion among Binary, Octal and Hex numbers

Radix r ($r \neq 10$) → Decimal ($r = 10$)

Binary → Decimal **$(10110.01)_2 = (??)_{10}$**

$$(10110.01)_2 \rightarrow 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} = (22.25)_{10}$$

Hex → Decimal **$(18F4.2A)_{16} = (??)_{10}$**

$$\begin{aligned}(18F4.2A)_{16} &= 1\times 16^3 + 8\times 16^2 + F\times 16^1 + 4\times 16^0 + 2\times 16^{-1} + 10\times 16^{-2} \\ &\approx (6388.16)_{10}\end{aligned}$$

***Compute the weighted sum of all digits**

$$\begin{aligned}A_r &= (a_n a_{n-1} \ldots a_o . a_{-1} \ldots a_{-m})_r \\ &= a_n \times r^n + a_{n-1} \times r^{n-1} + \ldots a_o \times r^0 + a_{-1} \times r^{-1} + \ldots a_{-m} \times r^{-m} \\ &= \sum_{i=-m}^{n} a_i r^i\end{aligned}$$

Decimal (r = 10) → Radix r (r ≠ 10)

Decimal → Binary **$(102)_{10} = (??)_2$**

$$
\begin{aligned}
(102)_{10} &= A_2 = (a_n a_{n-1} \ldots a_o . a_{-1} \ldots a_{-m})_r \\
&= a_n \times 2^n + a_{n-1} \times 2^{n-1} + \ldots + a_1 \times 2^1 + a_o \\
&= \underbrace{(a_n \times 2^n + a_{n-1} \times 2^{n-1} + \ldots + a_1 \times 2^1)}_{\text{Integer multiple of 2}} + a_o
\end{aligned}
$$

(Assume integer)

$$\frac{(102)_{10}}{2} \rightarrow$$

Continue dividing quotient by 2

quotient $a_n \times 2^{n-1} + a_{n-1} \times 2^{n-2} + \ldots + a_1$

$$2 \overline{\big)\, a_n \times 2^n + a_{n-1} \times 2^{n-1} + \ldots + a_1 \times 2 + a_o}$$

$$\underline{a_n \times 2^n + a_{n-1} \times 2^{n-1} + \ldots + a_1 \times 2}$$

$$a_o$$

Remainder is a_0

$$a_n \times 2^{n-2} + a_{n-1} \times 2^{n-3} + \ldots + a_1$$

$$2 \overline{\big)\, a_n \times 2^{n-1} + a_{n-1} \times 2^{n-2} + \ldots + a_1}$$

$$\underline{a_n \times 2^{n-1} + a_{n-1} \times 2^{n-2} + \ldots}$$

$$a_1$$

Remainder is a_1

Decimal → Radix r ($r \neq 10$) – cont.

Decimal → Binary $(102)_{10} = (??)_2$

Division	Quotient	Remainder
102/2	51	→ a_0
51/2	25	→ a_1
25/2	12	→ a_2
12/2	6	→ a_3
6/2	3	→ a_4
3/2	1	→ a_5
1/2	0	→ a_6

Stop when the quotient = 0

$(102)_{10} = (\quad)_2$

Check:

$$N_{10} = a_6 \times 2^6 + a_5 \times 2^5 + a_4 \times 2^4 + a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$
$$= 1\times 2^6 + 1\times 2^5 + 0\times 2^4 + 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0$$
$$= 64 + 32 + 0 + 0 + 4 + 2 + 0$$
$$= 102$$

How about Fractional Numbers?

Decimal → Binary **$(0.58)_{10} = (??)_2$**

$$(0.58)_{10} = A_2 = (0.a_{-1}a_{-2} \ldots a_{-m+1}a_{-m})_r$$
$$= a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2} + \ldots + a_{-m+1} \times 2^{-m+1} + a_{-m} \times 2^{-m}$$

Multiply by 2:

$$(0.58)_{10} \times 2 = \underset{\substack{\uparrow \\ \text{Integer part is } a_{-1}}}{a_{-1}} + \underbrace{a_{-2} \times 2^{-1} + \ldots + a_{-m+1} \times 2^{-m+2} + a_{-m} \times 2^{-m+1}}_{\text{fractional part}}$$

How about Fractional Numbers? – cont.

Decimal → Binary **$(0.58)_{10} = (??)_2$**

Multiply by 2	Product	Integer Part
0.58x2	1.16	→ a_{-1}
0.16x2	0.32	→ a_{-2}
0.32x2	0.64	→ a_{-3}
0.64x2	1.28	→ a_{-4}
0.28x2	0.56	→ a_{-5}
0.56x2	1.12	→ a_{-6}
0.12x2	0.24	→ a_{-7}
0.24x2	0.48	→ a_{-8}

$(0.58)_{10} = (\qquad)_2$

Check:

$$
\begin{aligned}
N_{10} &= 1\times 2^{-1} + 1\times 2^{-4} + 1\times 2^{-6} \\
&= \frac{1}{2} + \frac{1}{16} + \frac{1}{64} \\
&= 0.578125 \\
&\approx 0.58
\end{aligned}
$$

- The conversion process may never end.
- Where to stop depends on the required precision
- The process only ends when fractional part = 0

Numbers with Different Radixes: Summary

Numbers with Different Radixes

Decimal (radix 10)	Binary (radix 2)	Octal (radix 8)	Hexadecimal (radix 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Binary Arithmetic

ADDITION, SUBTRACT, MULTIPLICATION, DIVISION

Addition

Addition table:

0 + 0 = 0
0 + 1 = 1
1 + 1 = 10

“1” is the carry to the next higher bit

Example:

10111 + 110 = 11101

```
   1  1      <------ Carry
  1 0 1 1 1
+     1 1 0
-----------
  1 1 1 0 1
```

Multiplication

Multiplication table:

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

Example:

10111 x 110 = 10001010

```
           1 0 1 1 1      ← Multiplicand
     x       1 1 0        ← Multiplier
       ---------------
           0 0 0 0 0   ⎫
         1 0 1 1 1     ⎬  ← Partial products
   +   1 0 1 1 1       ⎭
   ---------------------
     1 0 0 0 1 0 1 0      ← Product
```

Multiplication:
→ Shift then Add
→ Only need "add" operation

Subtraction

Subtraction table:

0 - 0 = 0

1 - 0 = 1

1 - 1 = 0

0 – 1 = 1 ← with a borrow from the next (higher) bit

Example:

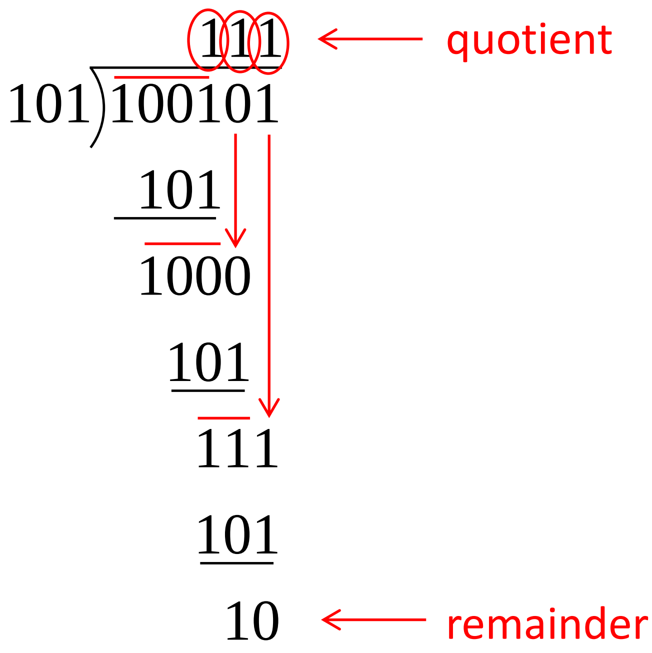
11011 - 110 = 10101

```
   1 1 0 1 1
 -     1 1 0
---------------
   1 0 1 0 1
```

Division

Division (shift and subtract)
→ Shift then subtraction
→ Only need "subtract" operation

100101/101 = ?



Check in decimal

- **Set** quotient to 0
- Align leftmost digits in dividend and divisor
- **Repeat**
  - **If** that portion of the dividend above the divisor is greater than or equal to the divisor
    - **Then** subtract divisor from that portion of the dividend and
    - Concatenate 1 to the right hand end of the quotient
    - **Else** concatenate 0 to the right hand end of the quotient
  - Shift the divisor one place right
- **Until** dividend is less than the divisor
- quotient is correct, dividend is remainder
- **STOP**

Arithmetic

- Only addition, subtraction and shifting are needed for 4 binary arithmetic operations
- Subtraction can be performed by adding a negative number
- Thus, a computer may only use **adders and shifters** to perform all binary arithmetic operations
- This requires an appropriate representation of the negative binary numbers

Signed Binary numbers

Three ways to represent the signed binary numbers

- Signed Magnitude (Sign + magnitude)
- 1’s complement
- 2’s complement

Unsigned Binary number

Unsigned binary number (*n* bits)

1	1	0	1	…	…	1	0

Magnitude

(No sign, always positive)

Range of unsigned binary number:

Max value of a 4-bit number:

$$1111 = \overset{2^4}{1}\,\overset{2^3}{0}\,\overset{2^2}{0}\,\overset{2^1}{0}\,\overset{2^0}{0} - 1 \rightarrow (2^4)_{10} - 1$$

Example:

Decimal	Unsigned binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Max value of *n*-bit unsigned number in decimal → $2^n - 1$. Range: 0 ~ (2^n-1)

Signed Binary – Signed Magnitude (S-M)

Signed binary number (*n* bits)

	1	0	1	…	…	1	0

Magnitude

Sign bit
- **0 – represents a positive number**
- **1 – represents a negative number**

MSB is a sign bit

Example:

Decimal	S-M
3	011
2	010
1	001
+0	000
-0	**100**
-1	**101**
-2	**110**
-3	**111**

Note:
Two zeros

Negative numbers

"1" in MSB position for all negative numbers

Signed Magnitude – cont.

More examples: $00111010 = +0111010 = (58)_{10}$

$11100101 = -\ 1100101 = (-101)_{10}$

$10000001 = -\ 0000001 = (-1)_{10}$

$01111111 = +1111111 = (+127)_{10}$

Range of binary number represented by S-M:

For a *n*-bit Signed binary (S-M), its magnitude is (*n*-1) bits

Max magnitude: $(2^{n-1}-1)_{10}$
Range: $-(2^{n-1}-1)_{10} \sim +(2^{n-1}-1)_{10}$

Arithmetic using Binary Numbers (S-M) ?

Computer performs binary arithmetic operations using only

- Adders
- Multipliers

Subtraction is performed by adding a negative number

Examples of subtraction using S-M binary representation:

```
    3          011              3           011              2          010
 -  2    ⇨   + 110           -  1    ⇨   +  101           -  1    ⇨  +  101
-------      -------         -------       -------        -------     -------
    1         1001  (1)10       2          1000  (0)10        1          111  (-3)10
              ↑       ✓                    ↑        ✗                          ✗
          Discarded                   Discarded
```

***S-M representation cannot be used for addition of two number with opposite signs or subtraction when using a simple adder**
(dedicated hardware is needed for all possible sign combinations)

Complement Representation

Complement representations of a number

- Radix complements
- Diminished complements

Definitions:

- ***Radix Complement***
  of a n-digit integer number A with radix (***r***):

$$A^* = r^n - A$$

- ***Diminished radix complement***
  of a n-digit integer number A with radix (***r***):

$$A^* = r^n - A - 1$$

Diminished Radix Complement

$$A^* = r^n - A - 1 \quad \text{or} \quad A^* = (r^n - 1) - A$$

Examples:

Decimal Operation:

A = 237 → $A^* = (1000_{10} - 1) - 237_{10}$
$= 999_{10} - 237_{10}$
$= 762_{10}$

$$\begin{array}{r} 888 \\ -\ 237 \\ \hline 651 \\ \hline \end{array} \iff \begin{array}{r} 888 \\ +\ (-237) \\ \hline \\ \hline \end{array} \iff \begin{array}{r} {\scriptstyle 1\ 1\ 1} \\ 888 \\ +\ 762 \\ \hline 650 \\ +\ 1 \\ \hline 651 \\ \hline \end{array}$$

Binary number:

A = 0011 → $A^* = (10000_2 - 1) - 0011_2 = 1111_2 - 0011_2 = 1100_2$

A = 1100 → $A^* = (10000_2 - 1) - 1100_2 = 1111_2 - 1100_2 = 0011_2$

Shortcut! →
Diminished **radix 2** complement
can be found by reversing the bits =)
This is also called 1's Complement.

1’s Complement

“1’s Complement” is the *diminished radix complement* of binary numbers

1’s complement of a *n-bit* number is **A* = (2^n – 1) - A**

1’s complement of a binary number can be obtained by reversing the bits, i.e. “1” → “0” and “0” → “1”, since

$$(2^n - 1)_{10} = \underbrace{1000\ldots000}_{n+1 \text{ bits}} - 1 = \underbrace{111\ldots111}_{n \text{ bits}}$$

Binary number (n=8): 0101 1100

1’s Complement: 1111 1111 - 0101 1100 = 1010 0011

Reversing the bits

1's Complement representation of signed binary number

No change for positive numbers and use 1's complement for negative numbers

Decimal	1's Complement
3	011
2	010
1	001
+0	000
-0	
-1	
-2	
-3	

Two Zeroes?

Magnitude range: $-(2^{n-1}-1) \sim (2^{n-1}-1)$

3 – 2 = 3 + (-2) = 1

```
      011        3
  +   101    + (-2)
  --------   -------
   (1)000
  +      1
  --------   -------
      001      + 1
             -------
```

- 3 + 1 = - 3+ 1 = -2

```
      100       -3
  +   001    + ( 1)
  --------   -------
   (0)101
  +      0
  --------   -------
      101      - 2
             -------
```

*** Subtraction can be performed by adding the carry!**

2’s Complement of a Binary Number

“2’s Complement” is the *radix complement* of binary numbers

2’s complement of a *n-bit* number can be obtained by adding “1” to its **1’s complement** (reversing all the bits), i.e.,

$$\begin{aligned} A^* &= 2^n - A \\ &= (2^n - A - 1) + 1 \end{aligned}$$

Binary number (n=8): 01011100

2’s Complement: $\underbrace{\underline{10100011}}_{\text{1's complement}} + 1 = \underbrace{10100100}_{\text{2's complement}}$

2’s Complement Arithmetic

No change for positive numbers and use 2’s complement for negative numbers

Decimal	2’s Complement
3	011
2	010
1	001
0	000
-1	
-2	
-3	
-4	

Only one zero

3 – 2 = 3 + (-2) =1

```
     011        3
  +  110     + (-2)
  --------   -------
  (1)001       + 1
                -------
```

Carry ignored

-3 + 1 = - 3 + 1 = -2

```
     101       -3
  +  001     + (  1)
  --------   -------
  (0)110       -  2
                -------
```

Carry ignored

- **Subtraction can be done!**
- **Carry is discarded (there is NO NEED to shift and add the carry, thus more hardware efficient)**

Magnitude range: $-(2^{n-1}) \sim (2^{n-1}-1)$

Signed Binary Number (Recap)

Sign+Magnitude

- Two zero representations (+/- zeros)
- It cannot correctly perform subtraction
- Magnitude range: $-(2^{n-1}-1) \sim (2^{n-1}-1)$

1's Complement (Diminished radix complement)

- Defined as: $\mathbf{A^* = (2^n - 1) - A}$
- 1's complement can be obtained by reversing the bits
- Two zero representations (+/- zeros)
- It can correctly perform subtraction, but needs to shift and add the carry
- Magnitude range: $-(2^{n-1}-1) \sim (2^{n-1}-1)$

2's Complement (Radix complement)

- Defined as: $\mathbf{A^* = 2^n - A}$
- One zero representation
- It can correctly perform subtraction by just ignoring the carry
- 2's complement can be obtained by adding "1" to its 1's complement
- Magnitude range: $-(2^{n-1}) \sim (2^{n-1}-1)$

Positive numbers are same in all 3 signed binary number representations

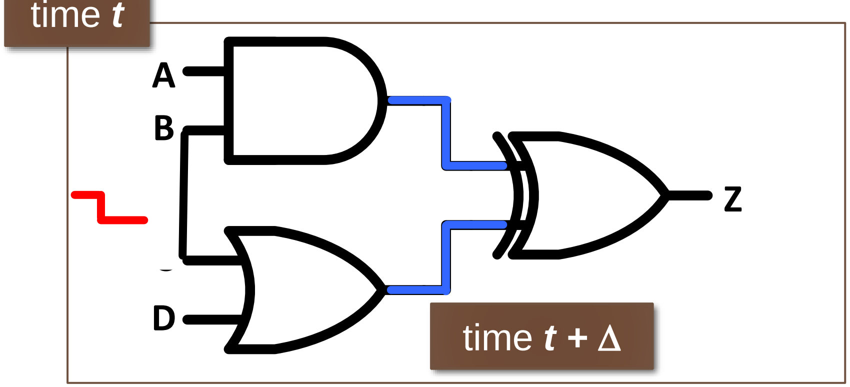
Introduction to Verilog

`Hdl, module, I/Os, wires, reg, operators`

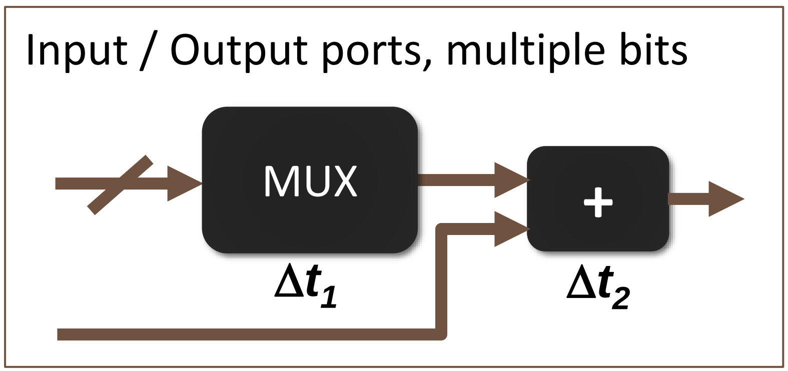
What are HDLs?

Hardware **Description Languages** (HDLs) are programming languages for describing digital circuits and systems.

Concurrency



Structure & Time



Today, ***Verilog* and *VHDL*** are the two leading HDLs.

Verilog code is used to describe RTL (Register Transfer Level) designs.

Virtually every chip (FPGA, ASIC, etc.) is designed in part using one of these two languages.

Xilinx and Altera are the two largest FPGA manufacturers.
(AMD) (Intel)

Verilog…

Verilog is an IEEE 1364 Standard → link here

Used for ***Modeling, Simulation*** and ***Synthesis*** of digital circuits.

Focus on synthesizable logic in this module.

Advantages :

- Reduces Design Time → Cost
- Improves Design Quality
- Vendor and Technology Independence
- Easy Design Management

Disadvantages :

- Cost (Including training you and me!)
- Debugging

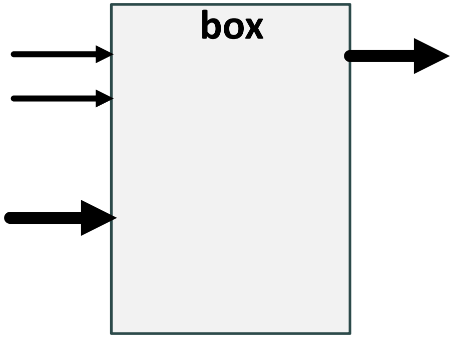
The Module

A piece of hardware with inputs & outputs : *module*

Module Name

Port Declaration

```
module box(   input a, b
              input [1:0] c,
              output [3:0] y );
// Here is where the magic
happens!
endmodule
```



y : y[3] y[2] y[1] y[0]

- Verilog is CasE-SeNSitiVe….
- Module Name : No spaces, No starting with numbers (1box), use meaningful names (box)
- Port Direction : `input`, `output`, `inout` (bidirectional)
- Port Bitwidth : `input a,b ; input [1:0] c ; output [3:0] y`

  By default, signals are one bit!

  Input c is a 2-bit bus / vector (little endian)

  Output y is a 4-bit vector
- Don't forget the _____!

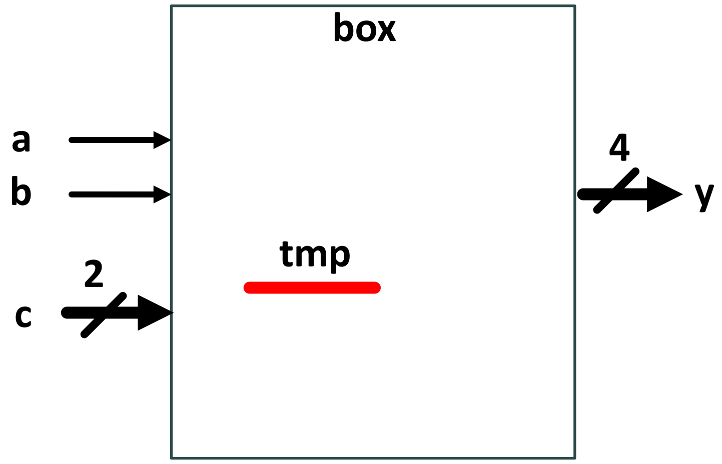
Data Types – Net / `wire`

```
module box(   input   a, b
              input   [1:0] c,
              output  [3:0] y );
wire tmp;

assign tmp = a;

assign y[3:0] = 4'hA;
//We can also write assign y = 4'hA;
//What is the driver?

endmodule
```



y = 4'hA

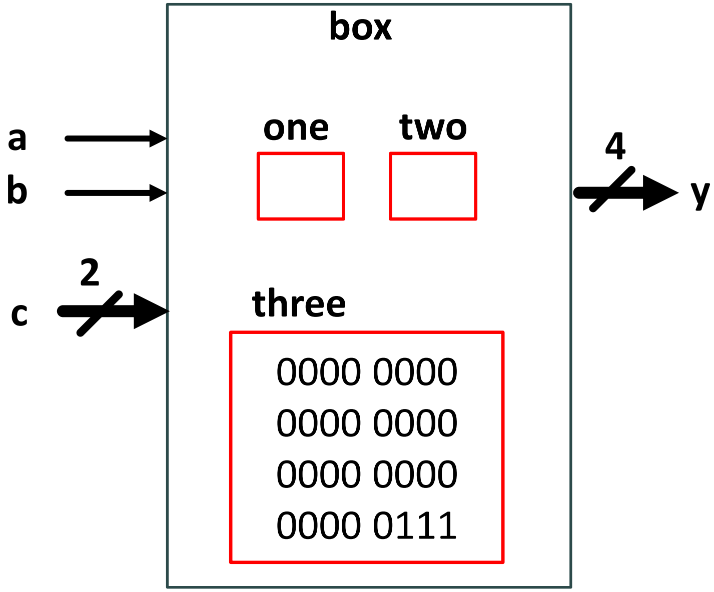
- Verilog HDL values consists of four basic values – `0` , `1`, `X` (unknown), `Z` (high impedance)
- Input and output ports default to the `wire` or net type.
- Nets do not store a value, and its value is determined by its driver (just like a wire!)
- If no driver is connected to a net, the value shall be high-impedance Z.
- Nets are connected to drivers via `assign` statements.

Data Types – Variables / `reg`

```
module box(  input   a, b
             input   [1:0] c,
             output  [3:0] y );

reg [1:0] one = 2'b1_1;
reg two;
integer three = 7;
//7 interpreted as decimal

endmodule
```



- Variables is an abstraction of a data storage element and is of `reg` type.
- When uninitialized, the value will be `X` (unknown).
- `reg` variables can be used to model both combinatorial or sequential logic.
- Assignments to `reg` are made via procedural assignments (always @)
- Registers cannot be connected to nets
- `integer` is a <u>general purpose</u> variable for manipulating quantities and not regarded as hardware. When the size is undefined, it is by default 32-bit.

Numerical Values (IEEE Std 1364-2001 p9)

Example 1—Unsized constant numbers

- 659 // is a decimal number
- ’h 837FF // is a hexadecimal number
- ’o7460 // is an octal number
- 4af // is illegal (hexadecimal format requires ’h)

Example 2—Sized constant numbers

- 4’b1001 // is a 4-bit binary number
- 5 ’D 3 // is a 5-bit decimal number
- 3’b01x // is a 3-bit number with the least significant bit unknown
- 12’hx // is a 12-bit unknown number
- 16’hz // is a 16-bit high-impedance number

Example 3—Using sign with constant numbers

- 8 ’d -6 // this is illegal syntax
- -8 ’d 6 // this defines the two’s complement of 6, held in 8 bits—equivalent to -(8’d 6)
- 4 ’shf // this denotes the 4-bit number ‘1111’, to be interpreted as a 2’s complement number, or ‘-1’. This is equivalent to -4’h 1
- -4 ’sd15 // this is equivalent to -(-4’d 1), or ‘0001’

Numerical Values (IEEE Std 1364-2001 p9)

Example 4—Automatic left padding

```
reg [11:0] a, b, c, d;
initial begin
a = ’h x; // yields xxx
b = ’h 3x; // yields 03x
c = ’h z3; // yields zz3
d = ’h 0z3; // yields 0z3
end
```

Example 5—Using underscore character in numbers

- `27_195_000`
- `16’b0011_0101_0001_1111`
- `32 ’h 12ab_f001`

Useful Operators

High

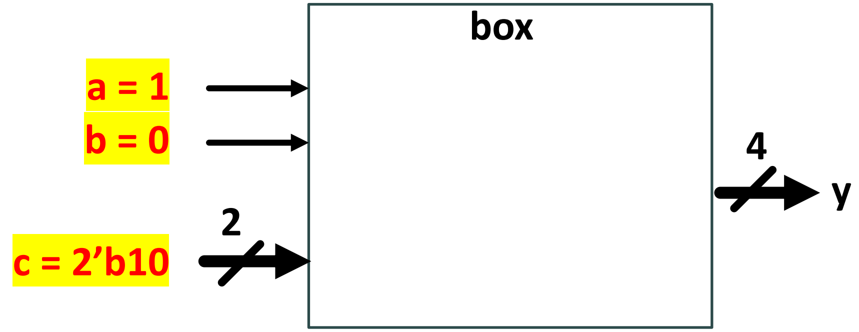
Precedence

Low

<table>
<tr><th>Operator</th><th>Description</th><th>Examples: a = 4’b1010, b=4’b0000</th></tr>
<tr><td>!, ~</td><td>Logical negation, Bit-wise NOT</td><td>!a = 0, !b =1, ~a=4’b0101, ~b=4’b1111</td></tr>
<tr><td>&, |, ^</td><td>Reduction (Outputs 1-bit)</td><td>&a = 0, |a=1, ^a = 0</td></tr>
<tr><td>{___,___}</td><td>Concatenation</td><td>{b, a} = 8’b00001010</td></tr>
<tr><td>{n{____}}</td><td>Replication</td><td>{2 {a} } = 8’b10101010</td></tr>
<tr><td>*, /, %,</td><td>Multiply, *Divide, *Modulus</td><td>3 % 2 = 1, 16 % 4 = 0</td></tr>
<tr><td>+, -</td><td>Binary addition, subtraction</td><td>a + b = 4’b1010</td></tr>
<tr><td><< , >></td><td>Shift Zeros in Left / Right</td><td>a << 1 = 4’b0100, a >> 2 = 4’b0010</td></tr>
<tr><td><, <=, >, >=</td><td>Logical Relative (1-bit output)</td><td>(a > b) = 1</td></tr>
<tr><td>==, !=</td><td>Logical Equality (1-bit output)</td><td>(a == b)= 0 (a != b)= 1</td></tr>
<tr><td>&, ^, |</td><td>Bit-wise AND, XOR, OR</td><td>a&b = a|b =</td></tr>
<tr><td>&&, ||</td><td>Logical AND, OR (1-bit output)</td><td>a&&b = a||b =</td></tr>
<tr><td>?:</td><td>Conditional Operator</td><td><out> = <condition> ? If_ONE : if_ZERO</td></tr>
</table>

What is happening here?

- Let's assume that a, b and c are being provided these values as shown →




```
module box(  input   a, b
             input   [1:0] c,
             output  [3:0] y );
wire tmp;
reg [1:0] one = 3;
reg two;
integer three = 1;

assign y[3] = one[0];

assign y[2:1] = a + c;

assign y[0] = ( a > b ) ;

endmodule
```

Net / Variable Name	Number of bits?	Value in dec / bin
a		
b		
c		
tmp		
one		
two		
three		
y		

Summary

We have covered :

- Positional number system (radix 10, 2, 8 and 16)
- Conversion among decimal, binary, octal and hex
- Binary arithmetic
- Signed binary number representations, SM, 1’s C, 2’s C
- Arithmetic using SM, 1’s C, 2’s C

We have covered:

- Introduction to Verilog
- Module, input and output ports (Single Bit and Multi-Bit signals)
- Data Types (wire/net vs reg)
- Numerical values (Hexa/Decimal/Binary/Octal/Integer)
- Addition / Subtraction / Conditional Operators