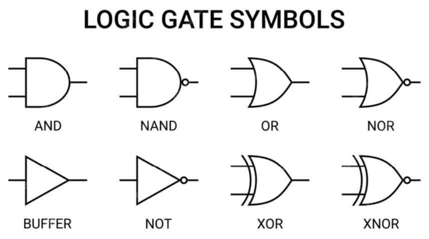
EES4725 Digital Circuits and FPGA Design

Chua Dingjuan
elechuad@nus.edu.sg

Lecture 1

Module Organization



- Week 1
  - Review of Combinational Logic
  - Introduction to Verilog
    - Dataflow, Structural and Behavioural Styles
  - Lab 1, Lab 2, Lab 3

三种建模方式

assign

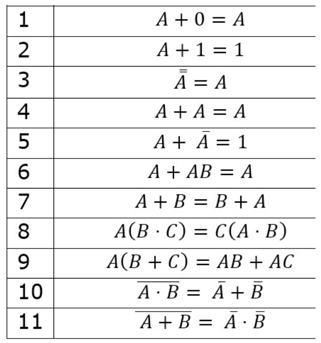
模块/门的"连线"

and, xor, not mux21

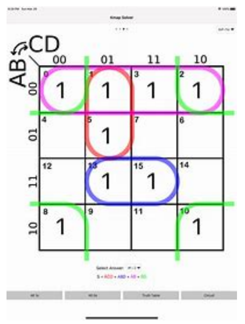
always、if/case、过程化描述

→ 组合、时序

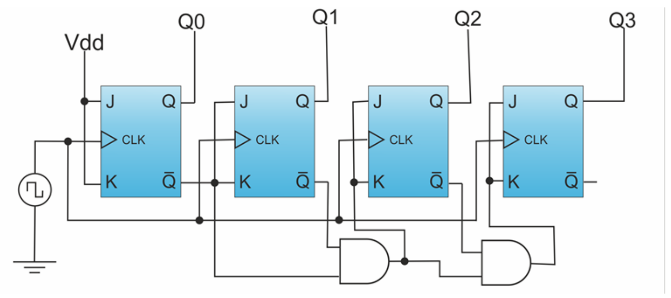
(*) (posedge clk)



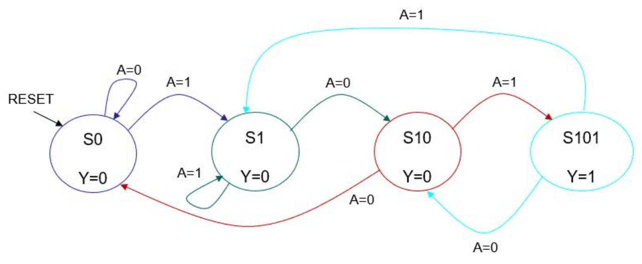
1	$A + 0 = A$
2	$A + 1 = 1$
3	$\bar{\bar{A}} = A$
4	$A + A = A$
5	$A + \bar{A} = 1$
6	$A + AB = A$
7	$A + B = B + A$
8	$A(B \cdot C) = C(A \cdot B)$
9	$A(B + C) = AB + AC$
10	$\overline{A \cdot B} = \bar{A} + \bar{B}$
11	$\overline{A + B} = \bar{A} \cdot \bar{B}$



- Week 2
  - Review of Sequential Logic – Flip flops, Synchronous Counters
  - Introduction to Design of Finite State Machines
  - Modeling sequential logic and FSMs in Verilog
  - Lab 4 and Project Work



- Week 3
  - Project Work
  - Quiz



Software Installation

- Software installation guide → EE4725_Lab0_Vivado_Installation_Instructions.pdf
- Software installation can take 3-4 hours depending on connection / laptop.
- Please make sure to install the software as soon as possible!

Expected Learning Outcomes

- *Be able to perform conversion* between binary, octal, hexadecimal and decimal number systems, 数制转换
- *Be able to express functions and manipulate in* Boolean Algebra
- *Be able to design simple combinational logic circuits* based on Truth table and Karnaugh Map
- *Be able to design combinational logic circuits* for simple practical problems / applications
- *Be able to describe simple sequential logic circuits* based on state transition diagrams

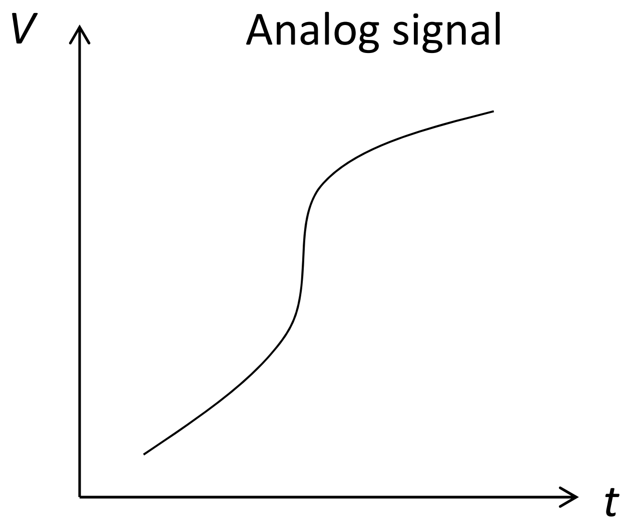
- *Be able to design complex logic circuits* using Hardware Description Languages (Verilog) and/or sequential/combinational building blocks/IPs
- *Be able to simulate complex logic blocks* and verify their proper functionality through behavioural simulation
- *Be able to design complex logic circuits* for practical problems/applications

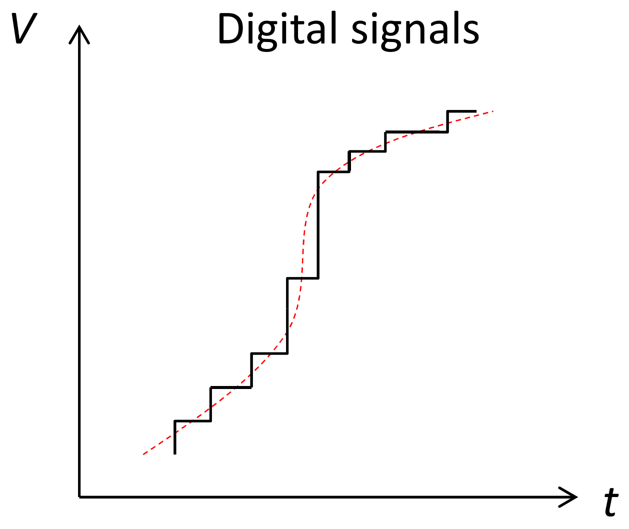
INTRODUCTION

有限状态+离散时间

Analog vs. Digital Circuit

- Analog circuit deals with continuous signals
- Digital circuit deals with signals having discrete levels



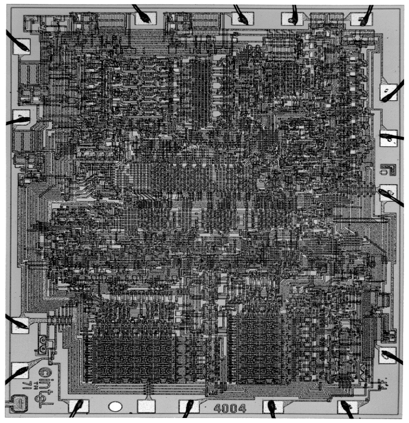


易受感染的

- Analog circuit is more susceptible to noise
- Digital circuit is a binary system which is much more robust

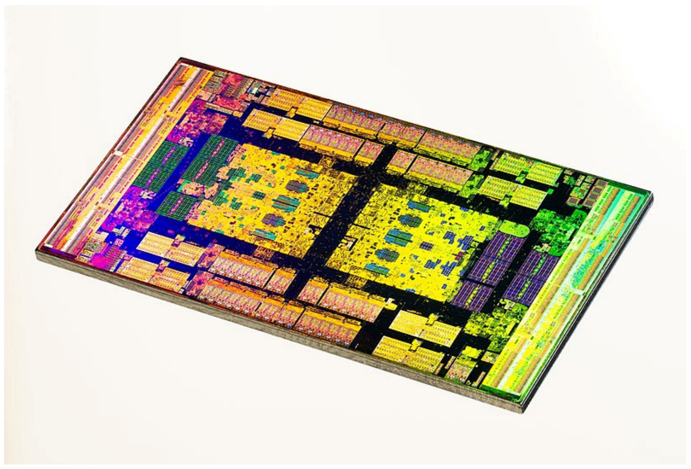
鲁棒性

Technology Scaling (cont.)



1971:

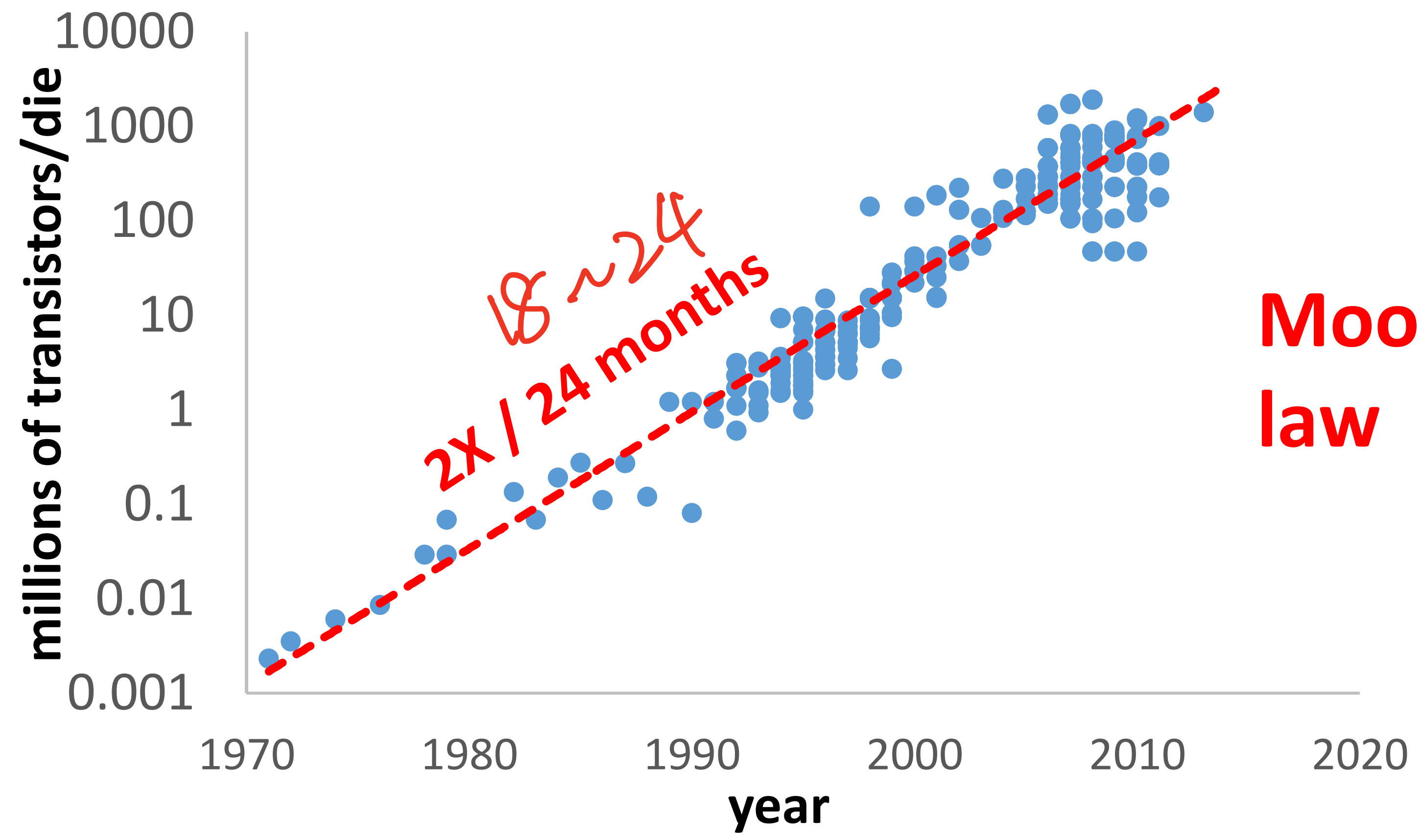
- Intel 4-bit processor in 10 μm PMOS process with 2300 transistors
- Initial clock speed of 108 kHz
- 10μm pMOS technology



2020:

- AMD Epyc Rome 7 nm processor (64 cores, 256MB L3, Zen 2 arch.) 40B transistors
- IBM z15 5.2 GHz clock freq., 12 cores in 14 nm FinFET, 9.2B transistors
- Intel Xeon Platinum 8180 in 14nm CMOS (28 cores), 3.6 GHz, 205 W, 8B transistors
- nVIDIA Ampere, 7nm FinFET, 5 PFLOPS, 54B transistors

Technology Scaling (cont.)

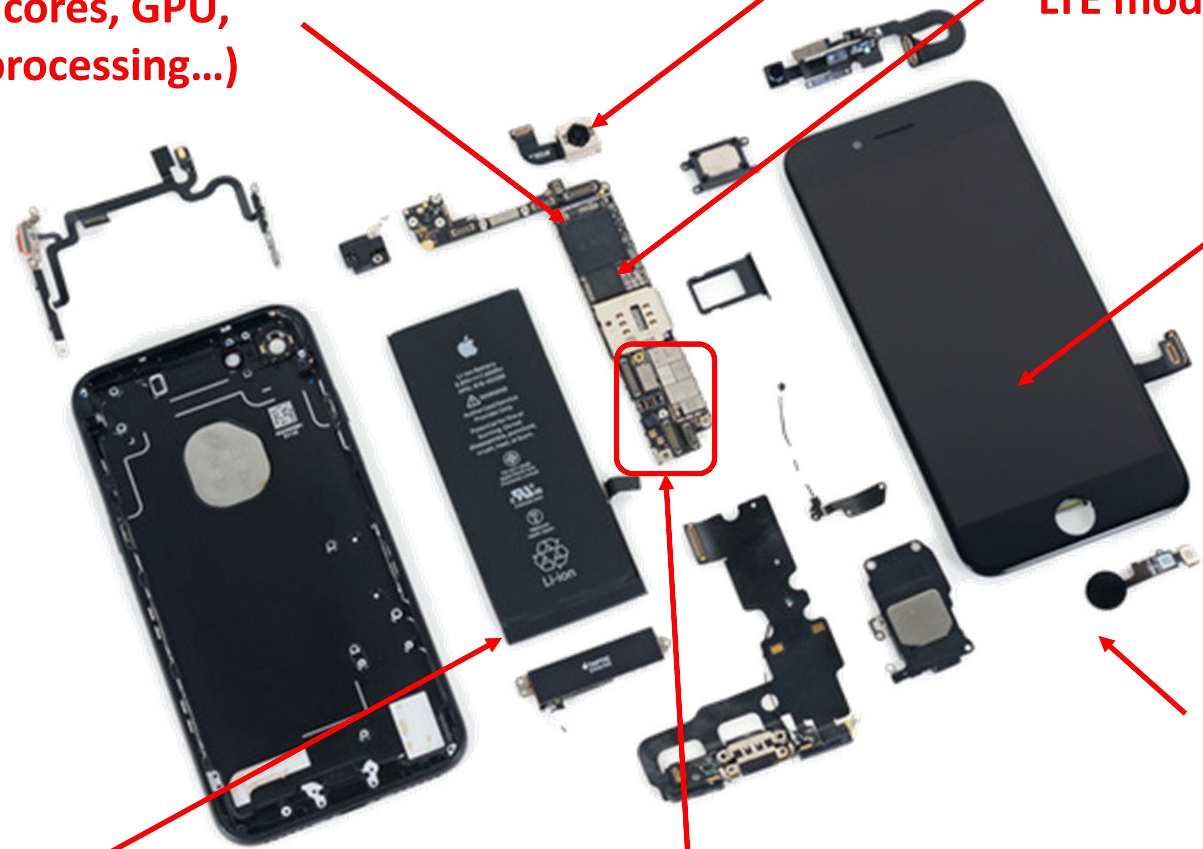


Moore's law

As more and more transistors can be integrated on a single chip,

- functionality is increased
- for the same functionality: lower chip area, lower cost per transistor

Example in Your Pocket (Today)


Application Processor (AP) (microprocessor cores, GPU, memory, video processing…)
Image sensor + pre-processing
LTE modem
Touchscreen controller
Image sensor + pre-processing
smart battery (charge, wearing, genuineness)
GPS, WiFi, DRAM, Flash, RF transceiver, Power Management, NFC, audio, display power management, FPGA, battery charger, compass, other sensors

NUMBER SYSTEMS

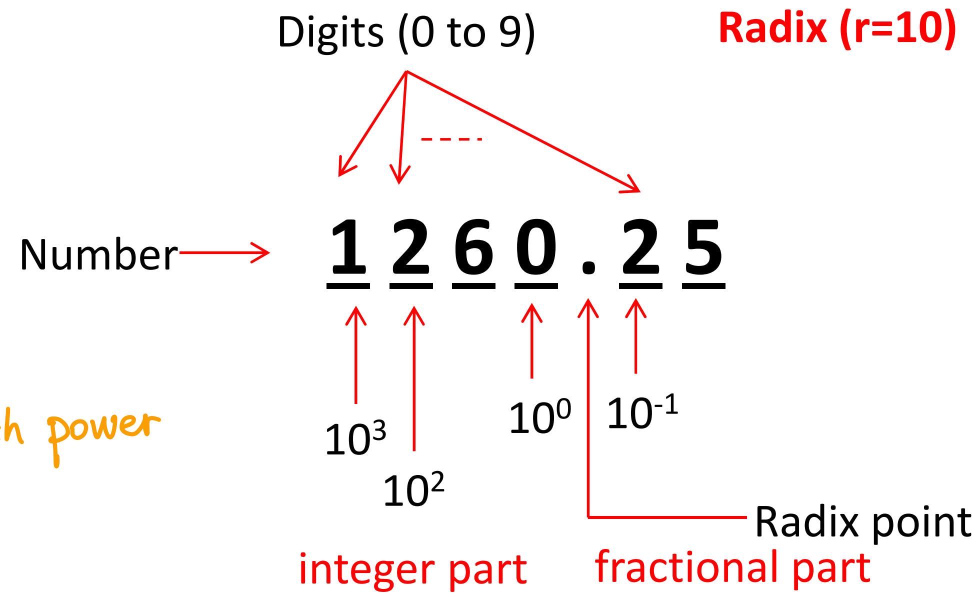
Decimal, Binary, Hex, Arithmetic, 2’s, Magnitude Ranges

Positional Number System

Decimal number:

Terminologies 术语

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



$$N = 1\times 10^3 + 2\times 10^2 + 6\times 10^1 + 0\times 10^0 + 2\times 10^{-1} + 5\times 10^{-2} = 1260.25$$

*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

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

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

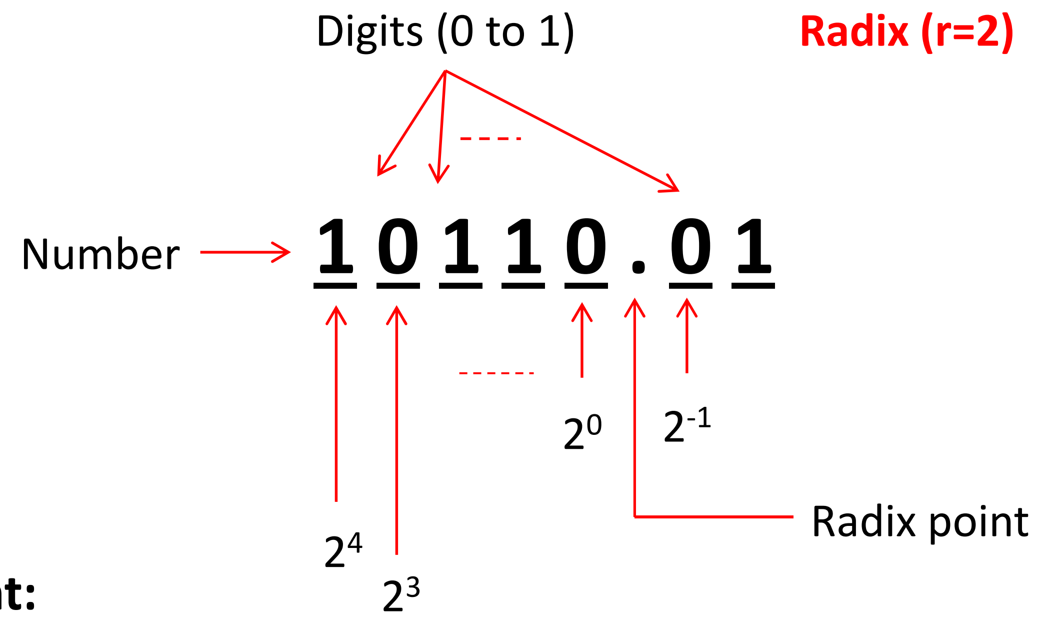
Decimal equivalent:

Radix point is here

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

***Weighted sum of all digits**

Binary Number



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

Little-Endian Format --

MSB

- Most significant bit

LSB

- Least significant bit

Range

- 0 to 2^n-1 where n represents the number of bits

1 0 1 1 0 0 1

(Left-most bit) MSB

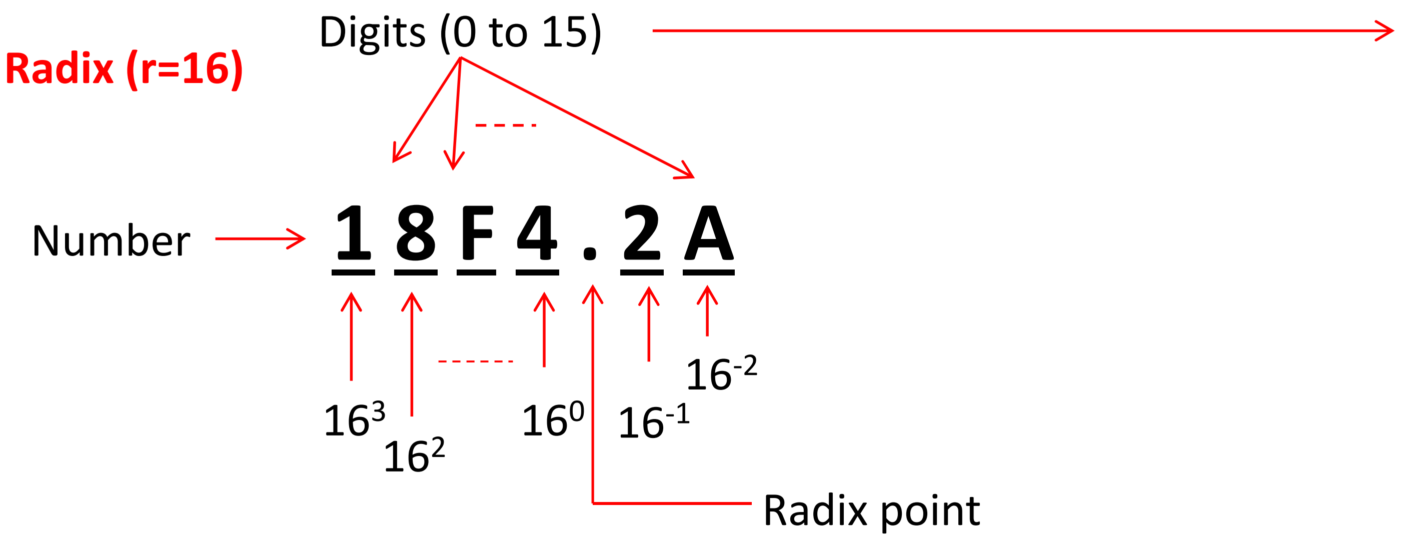
LSB (Right-most bit)

***For integer binary number only**

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

Hexadecimal number $(\quad \underline{D} \quad . \quad \underline{6} \quad)_{16}$

Radix (r=16)



Decimal Equivalent:

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

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

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

$$\approx 6388.16$$

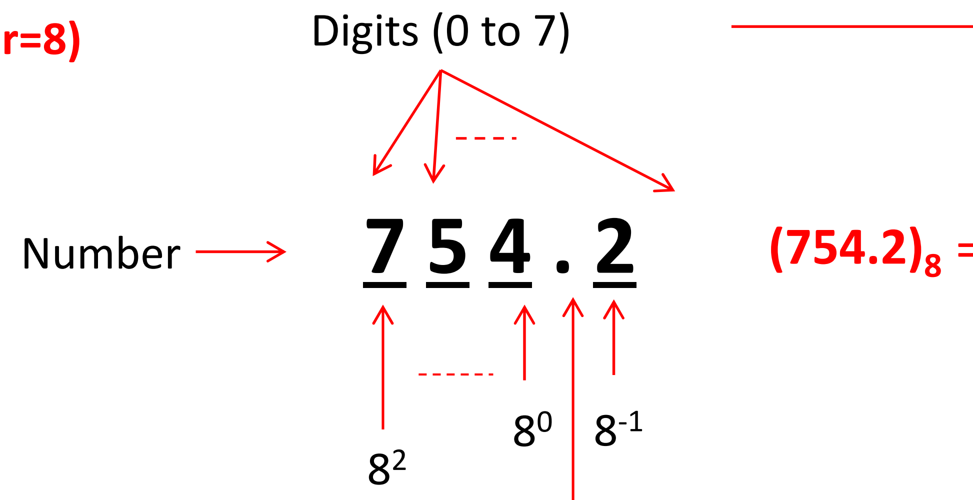
$(18F4.2A)_{16} \cong (6388.16)_{10}$

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

Octal number

$$(\underline{1}\,\underline{0}\,\underline{1}\,.\,\underline{0}\,\underline{1}\,\underline{1})_2$$
$$(\underline{5}\,.\,3)_8$$

Radix (r=8)



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

Decimal Equivalent:

$$N_{10} = 7\times8^2 + 5\times8^1 + 4\times8^0 + 2\times8^{-1}$$
$$= 448 + 40 + 4 + \frac{2}{8}$$
$$= 492.25$$

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

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

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

$$(10110.01)_2 \rightarrow 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0 + 0\times2^{-1} + 1\times2^{-2} = (22.25)_{10}$$

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

$$(18F4.2A)_{16} = 1\times16^3 + 8\times16^2 + F\times16^1 + 4\times16^0 + 2\times16^{-1} + 10\times16^{-2}$$
$$\approx (6388.16)_{10}$$

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

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

Decimal (r = 10) → Radix r (r ≠ 10)

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

$$(102)_{10} = A_2 = (a_n a_{n-1} \ldots a_o . a_{-1} \ldots a_{-m})_r$$

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

(Assume integer)

$$= \underbrace{(a_n \times 2^n + a_{n-1} \times 2^{n-1} + \ldots + a_1 \times 2^1)}_{\text{Integer multiple of 2}} + a_o$$

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

Continue dividing quotient by 2
商

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

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

Remainder is a_0

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

Remainder is a_1

Decimal → Radix r (r ≠ 10) – cont.

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

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

Stop when the quotient = 0

$(102)_{10} = (1100110)_2$

Check:

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

$\boxed{\text{Bit ?}}$ $2^N \geq 102 \Rightarrow N \geq \boxed{\log_2 102}$ $= 6.6 \rceil$ 7

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:

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

How about Fractional Numbers? – cont.

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

Multiply by 2	Product	Integer Part
0.58x2	1.16	1 → a_{-1}
0.16x2	0.32	0 → a_{-2}
0.32x2	0.64	0 → a_{-3}
0.64x2	1.28	1 → a_{-4}
0.28x2	0.56	0 → a_{-5}
0.56x2	1.12	1 → a_{-6}
0.12x2	0.24	0 → a_{-7}
0.24x2	0.48	0 → a_{-8}

$(0.58)_{10} = (0.100101)_2$

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 $\boxed{\text{the required precision}}$
- The process only ends when fractional part = 0

Binary Arithmetic

Addition

Addition table:

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

“1” is the carry to the next higher bit

Example:

10111 + 110 =

```
                  ←——— Carry
    1 0 1 1 1
+       1 1 0
---------------
```

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  被乘数
               x      1 1 0   ←———  Multiplier    乘数
              ---------------
                  0 0 0 0 0   }
                              }  ←———  Partial products
          +                   }
         ----------------------
                                ←———  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:

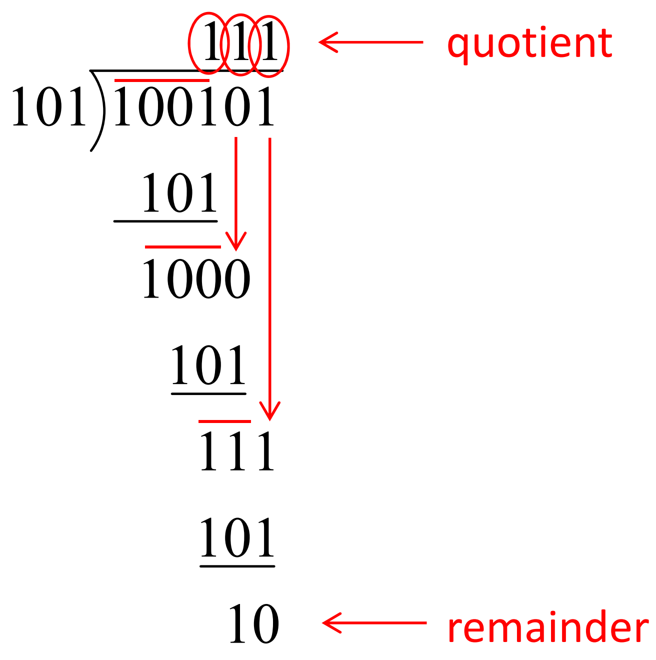
11011 - 110 =

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

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 被除数 除数
- **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
    - 联系 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** $\boxed{\text{dividend}}$ 被除数 is less than the $\boxed{\text{divisor}}$ 除数
- quotient is correct, dividend is remainder
- **STOP**

Arithmetic

- Only addition and subtraction are needed for 4 binary arithmetic operations
- Subtraction needs more elements than addition in hardware
- 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

Unsigned Binary number

$$\underbrace{\overbrace{\boxed{1 \quad 1 \quad 0 \quad 1 \quad \dots \quad \dots \quad 1 \quad 0}}^{\textbf{Unsigned binary number } (n \textbf{ bits})}}_{\textbf{Magnitude}}$$

(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: $0 \sim (2^n-1)$

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.,

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

Binary number (n=8): 01011100 $\rightarrow (92)_{10}$

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

1’s complement **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	**111**
-2	**110**
-3	**101**
-4	**100**

Only one zero

100 011

3 – 2 = 3+(-2)=1

```
   011
+  110
--------
(1)001
 ↑
Carry ignored
```

3 – 1 = 3+(-1) = 2

```
   011
+  111
--------
(1)010
 ↑
Carry ignored
```

- **No problem in performing subtraction**
- **Carry is discarded (there is NO NEED to shift and add the carry, thus more hardware efficient)**

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

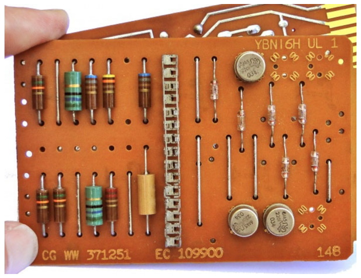
↑ 范围不对称(少一个正数)

Introduction to Verilog

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

Background…

An IBM Standard Modular System (SMS) printed-circuit card from the early-1960s. This particular card implemented three simple logic gates.



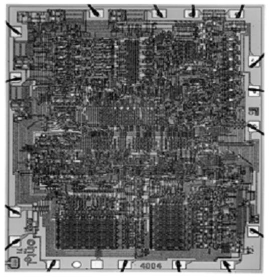
1947

The First Transistor

1950s – 1960s

Rapid Development of Technology

1971



Intel 4-bit Processor 2300 Transistors ~ 400 gates

1980s

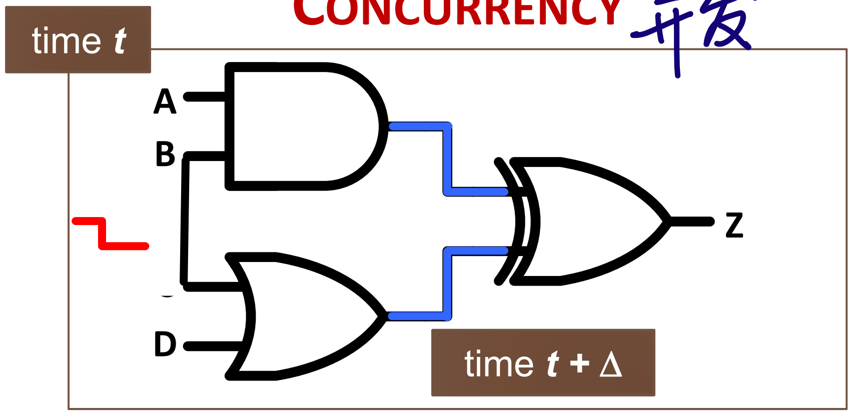
Hardware Description Languages : VHDL & Verilog

Robert, G. (2013) IBM Standard Modular System [Photograph] Retrieved 1 Dec, 2014, from IEEE Global History Network, Early Popular Computers, 1950 - 1970 http://www.ieeeghn.org/wiki/index.php/File:EPC_-_fig_4.jpg

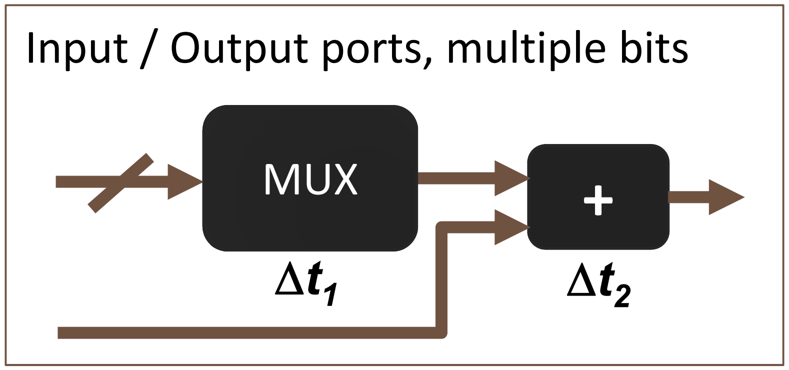
What are HDLs?

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

CONCURRENCY 并发



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

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.

综合化

Advantages :

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

Disadvantages :

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

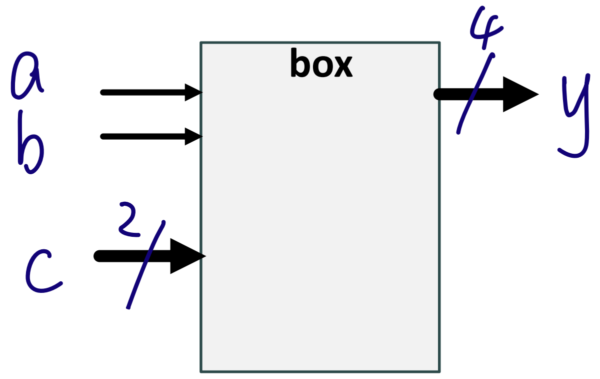
The Module

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
```



- Verilog is CasE-SeNSitiVe.... 大小写敏感
- 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!

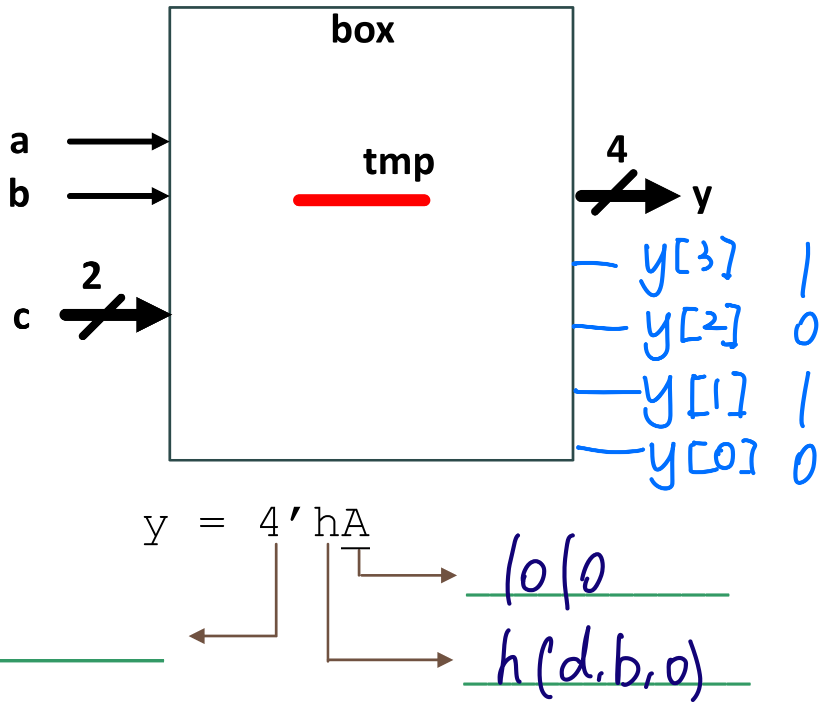
Input c is a 2-bit vector (little endian)

Output y is a 4-bit vector

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

Data Types – Net / wire

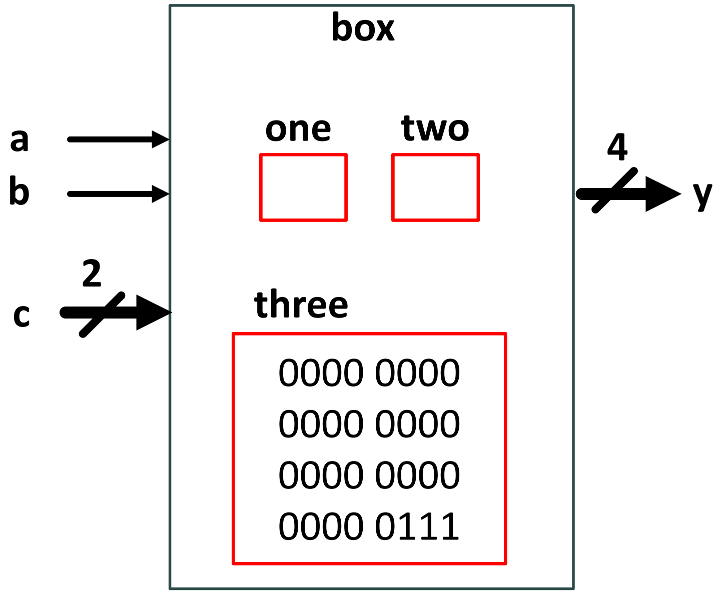
```
module box(  input   a, b
             input   [1:0] c,
             output  [3:0] y );
wire tmp;

assign tmp = a;

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


endmodule
```



- Input and output ports default to the `wire` or net type.
- In Verilog, a 1-bit net can take 3 basic values – 0 , 1, Z (high impedance).
- 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 @)
- `integer` is a general purpose 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

- Operators are extremely useful in Verilog, let's start with some simple ones!

High

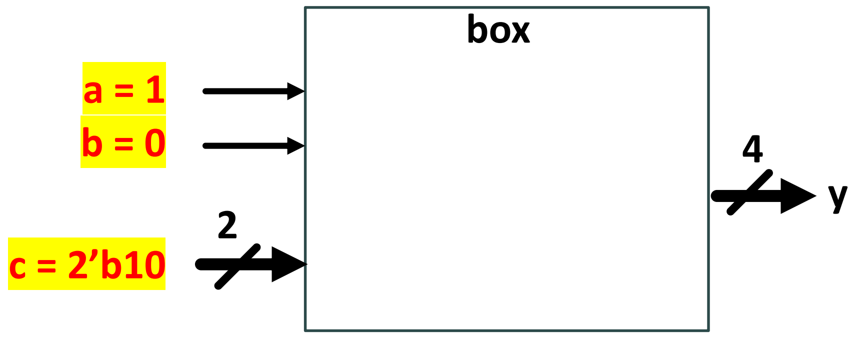
Precedence

Low

Operator	Description	Examples: a = 4'b1010, b=4'b0000
`!, ~`	Logical negation, Bit-wise NOT	**!a = 0, !b =1, ~a=4'b0101, ~b=4'b1111**
`&, |, ^`	Reduction (Outputs 1-bit)	**&a = 0, |a=1, ^a = 0**
`{___,___}`	Concatenation	{b, a} = 8'b00001010
`{n{____}}`	Replication	{2 {a} } = 8'b10101010
`*, /, %,`	Multiply, *Divide, *Modulus	3 % 2 = 1, 16 % 4 = 0
`+, -`	Binary addition, subtraction	a + b = 4'b1010
`<< , >>`	Shift Zeros in Left / Right	a << 1 = 4'b0100, a >> 2 = 4'b0010
`<, <=, >, >=`	Logical Relative (1-bit output)	(a > b) = 1'b1
`==, !=`	Logical Equality (1-bit output)	(a == b) = 1'b0 (a != b)= 1'b1
`&, ^, |`	Bit-wise AND, XOR, OR	a&b = a|b =
`&&, ||`	Logical AND, OR (1-bit output)	a&&b = a||b =
`?:`	Conditional Operator	<out> = <condition> ? If_ONE : if_ZERO

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];                 1

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

assign y[0] = ( a > b ) ;             1

endmodule
```

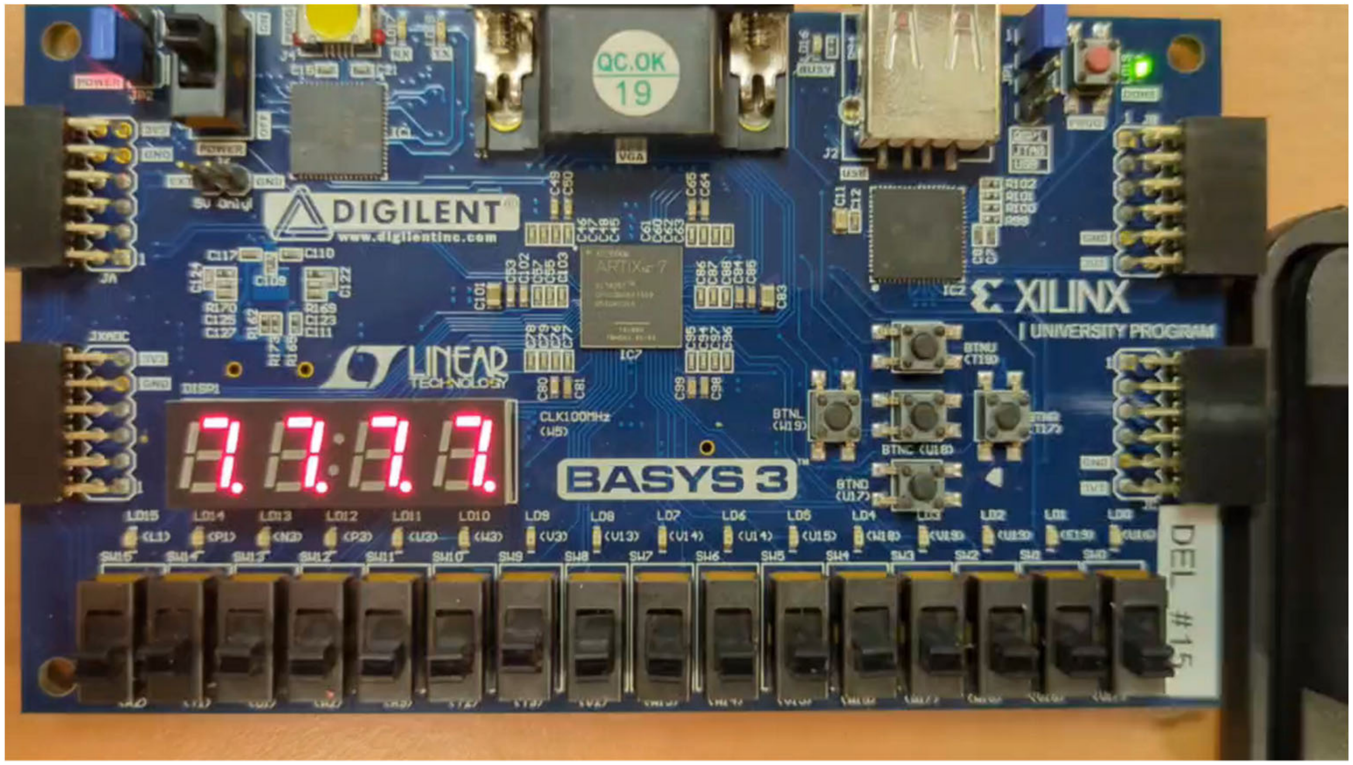
Net / Variable Name	Number of bits?	Value in dec / bin
a	1	1 / 1'b1
b	1	0 / 1'b0
c	2	2 / 2'b10
tmp	1	Z / 1'bZ
one	2	3 / 2'b11
two	1	X / 1'bX
three	32	1 / 32'b1
y	4	15 / 4'b1111

Let’s see it in action! - 1

```
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
```

These increasing numbers are part of the bootloaded program on the FPGA!

Downloading this code ← onto the FPGA….



Summary

We have covered :

- Positional number system (radix 10, 2, 8 and 16)
- Conversion among decimal, binary, octal and hex
- Binary arithmetic

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