

- Describes the hardware functionality of the Digital Systems in the textual form
- It resembles programming language, but is specially developed for describing the functionality of the Digital hardware
- The main difference with other high-level languages is
 - HDL's constructs ability to perform parallel execution
 - Unlike sequential execution nature of other language constructs
- Two HDLs are widely used
 - Verilog HDL
 - VHDL(Very High Speed Integrated Circuit HDL)

- ❑ Verilog is a free-format language
 - Like C language
- ❑ White space (blank, tab, newline) can be used freely
- ❑ Verilog is a case-sensitive language
- ❑ Identifiers
 - User-provided names for Verilog objects in the descriptions
 - Legal characters are “a-z”, “A-Z”, “0-9”, “_”, and “\$”
 - First character has to be a letter or an “_”
 - Example: Count, _R2D2, FIVE\$
- ❑ Keywords
 - Predefined identifiers to define the language constructs
 - All keywords are defined in lower case
 - Cannot be used as identifiers
 - Example: initial, assign, module

❑ Comments: two forms

/* First form:

can extend

over many lines */

// Second form: ends at the end of this line

❑ Strings

– Enclosed in double quotes and must be specified in one line

- “Sequence of characters”

❑ Accept C-liked escape character

- \n = newline
 - \t = tab
 - \\ = backslash
 - \" = quote mark (“)
 - %% = % sign
-

❑ System tasks / function

- Execute the built-in tasks and functions
- Frequently used system tasks / functions
 - \$time: report the current simulation time
 - \$display: display the values of signals
 - \$monitor: continuously monitor the values of signals
 - \$stop: stop the simulation
 - \$finish: quit the simulation

❑ Compiler directive

- Remain active through the rest of compilation until they are overridden or deactivated
- Frequently used compiler directives
 - ``define<name><macro_text>:`
 <name>will substitute<macro_text> at compile time
 - ``include<file_name> :`
 include the contents of the file named as<file_name>

□ Sized integers

- For unsigned integers only
- Representation form: [`<size>`] ' `<base><value>`

➤ where

- `<size>` is the size in bits (can be ignored)
- `<base>` can be b(binary), o(octal), d(decimal), or h(hexadecimal)
- `<value>` is any legal number in selected base and X(unknown), Z(high-impedence) (binary only)
- Example: ``o721` (9-bit octal), `4'd2` (4-bit decimal)
 - `4'bz` (4-bit z, z extended)

□ Unsized integers

- Signed decimal integers in two's complement form
- Example: 32 (decimal 32), -15 (decimal -15)

❑ Decimal notation

11.2

1.572

0.1

❑ Scientific notation

235.1e2

23510.0

3.6E2

360.0 (e is the same as E)

5E-4

0.0005

❑ Must have at least one digit on either side of decimal

❑ Stored and manipulated in double precision (usually 64-bits)

- 0: logic-0 / FALSE
- 1: logic-1 / TRUE
- x: unknown / don't care, can be 0, 1 or z
- z: high-impedance

□Nets

- Connects between structural elements
- Values come from its drivers
 - Continuous assignment
 - Module or gate instantiation
- If no drivers are connected to net, default value is Z

□Registers

- Represent abstract data storage elements
- Manipulated within procedural blocks
- The value in a register is saved until it is overridden
- Default value is X

1. wire, tri: standard net
2. wor, trior: [wired-or](#) net
3. wand, triand: [wired-and](#) net
4. trireg: capacitive
5. If all drivers at z, previous value is retained
6. tri1: pull up (if no driver, 1)
7. tri0: pull down (if no driver, 0)
8. supply0: ground
9. supply1: power
10. A net that is not declared defaults to a 1-bit wire
 - wire reset;
 - wor [7:0] DBUS;
 - supply0 GND;

- ❑ reg: any size, unsigned
- ❑ integer: 32-bit signed (2's complement)
- ❑ time: 64-bit unsigned
- ❑ real, realtime: 64-bit real number
 - Defaults to an initial value of 0
- ❑ Examples:
 - reg CNT;
 - reg [31:0] SAT;
 - integer A, B, C; // 32-bit
 - real SWING;
 - realtime CURR_TIME;
 - time EVENT;

- Constant
- Can be modified at compilation time
 - Use defparam statement
- Examples:
 - parameter LINE_LENGTH = 132, ZLL_X_S = 16'bx;
 - parameter BIT = 1, BYTE = 8, PI = 3.14;
 - parameter SROBE_DELAY = (BYTE + BIT) / 2;
 - parameter TQ_FILE = “/home/jimmy/TEST/add.tq”;
- Common usage
 - Specify delays and widths

- Array of registers
- No multiple dimensions
 - reg [3:0] MY_MEM [0:63]; // 64 4-bit registers
- Entire memory cannot be assigned a value in a single assignment
 - reg [1:5] DIG; // 5-bit register
 - DIG = 00000
 - reg BOG [1:5]; // 5 1-bit register
 - {BOG[1], BOG[2], ..., BOG[5]} = 00000;
- Can load memory by using a system task
 - \$readmem<base>("<filename>",<memory_name>,<start_addr>,<finish_addr>);
 - where <base> can be b(binary) or h(hexadecimal)

- Switch-level modeling
- Gate-level modeling
- Dataflow-level modeling
- Behavioural modeling
- Structural modeling
- Mixed level modeling

SWITCH-LEVEL PRIMITIVES

MOS TRANSISTOR SWITCHES

MOS PULL GATES

MOS BIDIRECTIONAL SWITCHES

nmos

pullup

tran

pmos

pulldown

tranif0

cmos

tranif1

rnmos

rtran

rpmos

rtranif0


rcmos

rtranif1

- **nmos, pmos**
- **rnmos, rpmos** (resistive version)
- Terminal list: (drain, source, gate)
- (i.e., (output,input, control))

Input-output relationship

- L: 0 or z
- H: 1 or z

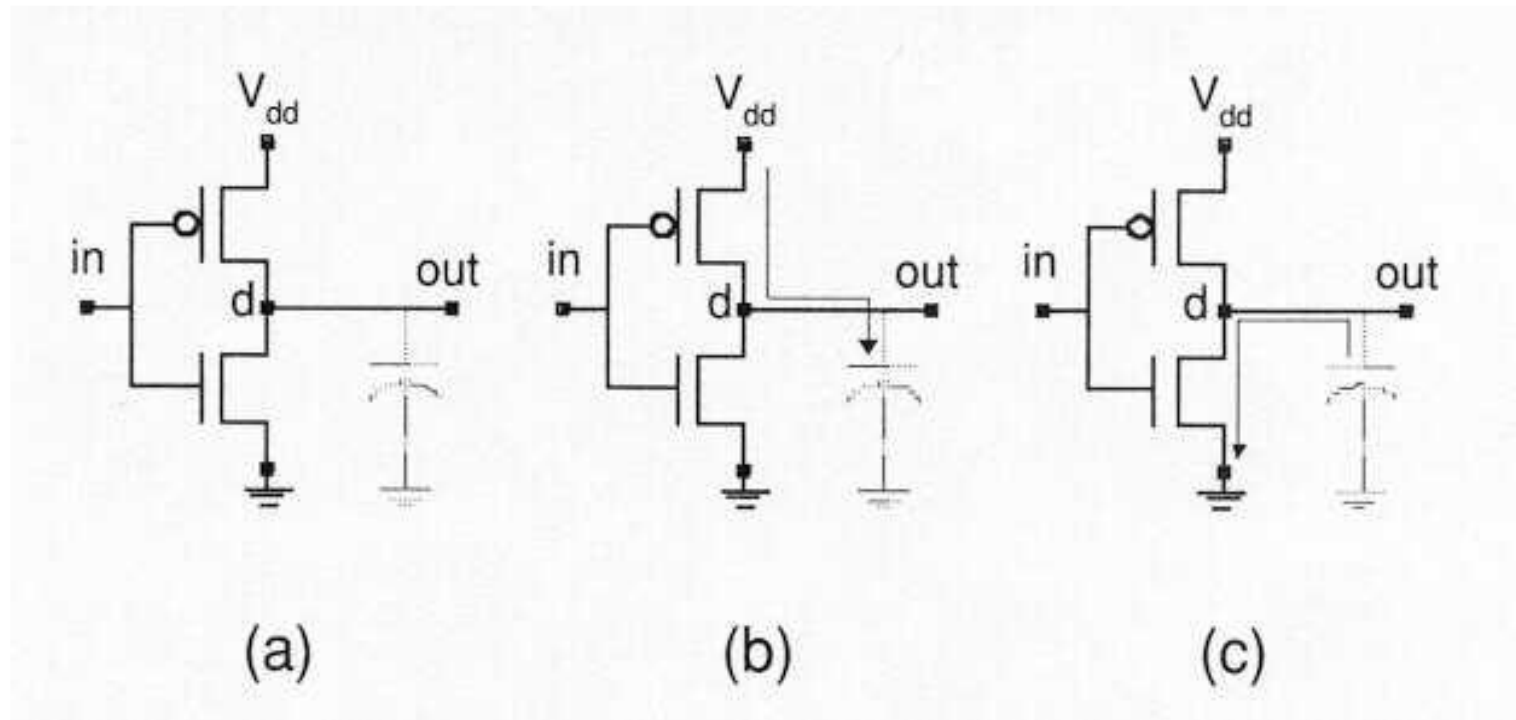


| | | control | | | |
|-------|---|---------|---|---|---|
| | | 0 | 1 | x | z |
| input | 0 | 0 | z | L | L |
| | 1 | 1 | z | H | H |
| | x | x | z | x | x |
| | z | z | z | z | z |

| | | control | | | |
|-------|---|---------|---|---|---|
| | | 0 | 1 | x | z |
| input | 0 | z | 0 | L | L |
| | 1 | z | 1 | H | H |
| | x | z | x | x | x |
| | z | z | z | z | z |

EXAMPLE: INVERTER

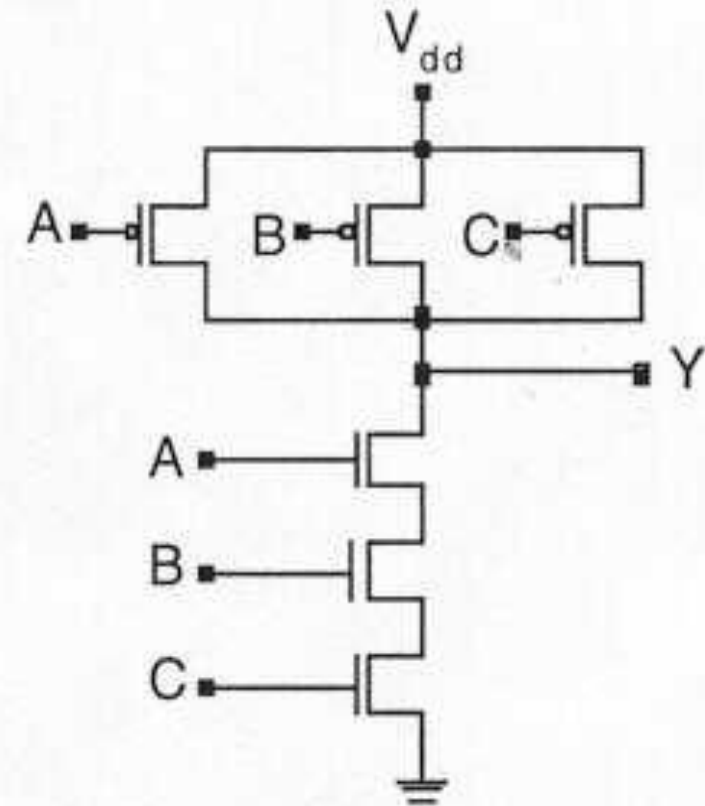
(a) CMOS circuit, (b) $\text{in} = 0$, (c) $\text{in} = 1$




```
1. module cmos_inverter (inv_out, inv_in);  
2.   output inv_out;  
3.   input inv_in;  
4.   supply0 GND;  
5.   supply1 PWR;  
6.   pmos (inv_out, PWR, inv_in);  
7.   nmos (inv_out, GND, inv_in);  
8. endmodule
```

EXAMPLE: 3-INPUT NAND GATE

```
module nand_3 (Y, A, B, C);  
  output      Y;  
  input       A, B, C;  
  supply0     GND;  
  supply1     PWR;  
  wire        w1, w2;  
  
  pmos        (Y, PWR, A);  
  pmos        (Y, PWR, B);  
  pmos        (Y, PWR, C);  
  nmos        (Y, w1, A);  
  nmos        (w1, w2, B);  
  nmos        (w2, GND, C);  
endmodule
```



| Gate-level Primitives | | |
|-----------------------|--------------------|-------------------|
| Multi-input Gates | Multi-output Gates | Three-state Gates |
| and | buf | bufif0 |
| nand | not | bufif1 |
| or | | notif0 |
| nor | | notif1 |
| xor | | |
| xnor | | |

```
1. module allgates(inverter_out, and_out, nand_out, or_out, nor_out, exor_out, exnor_out, vin1, vin2);
2. input vin1, vin2;
3. output inverter_out, and_out, nand_out, or_out,
4. nor_out, exor_out, exnor_out;
5. not inverter1(inverter_out,vin2);
6. and and1(and_out,vin1,vin2);
7. nand nand1(nand_out,vin1,vin2);
8. or or1(or_out,vin1,vin2);
9. nor nor1(nor_out,vin1,vin2);
10.xor xor1(exor_out,vin1,vin2);
11.xnor xnor1(exnor_out,vin1,vin2);
12.endmodule
```

TESTBENCH LOGIC GATES:

1. ``timescale 1 ns / 1 ps`
2. `module test;`
3. `reg vin1, vin2;`
4. `wire inverter_out_vin2, and_out, nand_out, or_out, nor_out, exor_out, exnor_out;`
5. `integer i;`
6. `allgates ag(inverter_out_vin2, and_out, nand_out, or_out, nor_out, exor_out, exnor_out, vin1, vin2);`
7. `initial`
8. `for(i=0;i<4;i=i+1)`
9. `begin`
10. `{vin1,vin2}=i;`
11. `#5 ;`
12. `end`
13. `endmodule`

➤ There are 2 types

1. positional
2. named

1. `//Binary 1-bit Half Adder`
2. `module ha(a,b,sum,co); //actual module`
3. `input a,b;`
4. `output sum,co;`
5. `assign sum = a^b;`
6. `assign co = a&b;`
7. `endmodule`

-
- When you want to instantiate the above module in Testbench or any Higher level module(for example Binary Full Adder using Half Adder)
 - You will be doing the following
 1. module name
 2. module instantiation name
 3. port list, such as shown below
 - module;/*It may be testbench or any other higher level module*/
 - **//x,y,sum_result and carry_out are ports in the current module**
 - **ha h1(x,y,sum_result,carry_out);**//instantiated module
 - endmodule
 - If you observe carefully the port list order is same in instantiated module and in the actual module .
-

2. NAMED

- module; /*It may be testbench or any other higher level module*/
-
-
- **// x, y, sum_result and carry_out are ports in the current module**
- **ha h1(.co(carry_out), .b(y), .sum(sum_result), .a(x));** //instantiated module
-
-
- endmodule

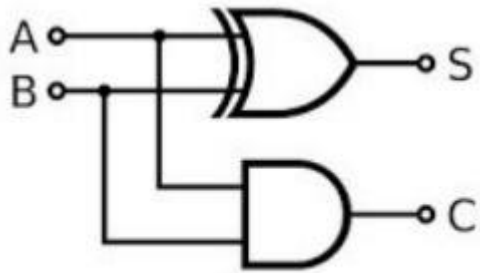


Fig. 1 Binary 1-bit half-adder

```
1. module ha(a,b,sum,co);  
2.   input a,b;  
3.   output sum,co;  
4.   xor (sum,a,b);  
5.   and (co,a,b);  
6. endmodule
```

TESTBENCH: 1-BIT BINARY HALF ADDER

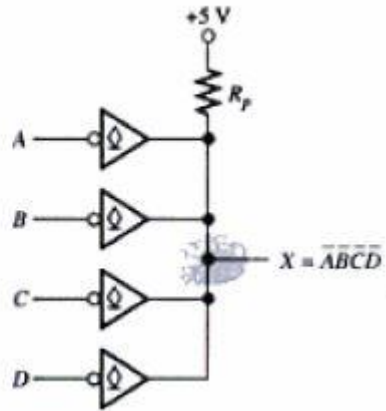
```
1.  `timescale 1ns/1ps
2.  module ha_test;
3.  reg a,b;
4.  wire sum,co;
5.  ha hadder(a,b,sum,co); /*INSTANTIATE THE MODULE NAME(ha) THAT NEEDS BE TESTED WITH INSTANTIATION NAME hadder*/
6.  initial
7.  begin
8.      {a,b} = 2'b00;
9.      #5 {a,b} = 2'b01;
10.     #5 {a,b} = 2'b10;
11.     #5 {a,b} = 2'b11;
12.     #5 $finish;
13. end
14. initial
15. $monitor($time, "a=%b, b=%b, sum=%b, co=%b" , a, b, sum, co);
16. endmodule
```

Operators

| | |
|----------------------------|---------------------------|
| Arithmetic Operators | +, -, *, /, % |
| Relational Operators | <, <=, >, >= |
| Logical Equality Operators | ==, != |
| Case Equality Operators | ===, !== |
| Logical Operators | !, &&, |
| Bit-Wise Operators | ~, &, , ^(xor), ~^(xnor) |
| Unary Reduction Operators | &, ~&, , ~ , ^, ~^ |
| Shift Operators | >>, << |
| Conditional Operators | ? : |
| Concatenation Operator | { } |
| Replication Operator | { { } } |

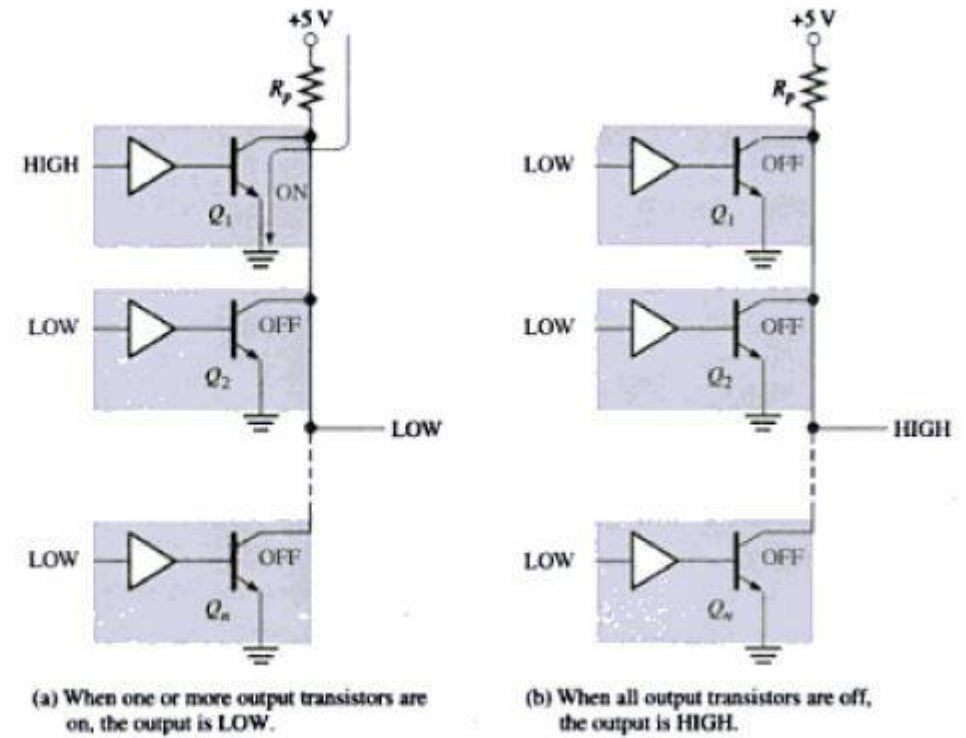
► FIGURE 11-44

A wired-AND configuration of four inverters



► FIGURE 11-45

Open-collector wired negative-AND operation with inverters



- **wired logic** A form of digital logic in which some logic functions are implemented by directly connecting together the outputs of one or more logic gates. The success of this technique depends on the electronic characteristics of the gates involved. The technique is commonly used in bus communication systems with tri-state output or with open-collector devices.
- **open-collector device** A particular implementation of an electronic logic device in which the output of the device is formed by the open-circuit collector termination of the output transistor (see diagram). The device's output is thus active-low and a pull-up resistor is required to establish the active-high state. These devices are used to drive loads with high supply voltages or to implement wired-logic buses.
- **pull-up resistor** A resistor that is connected between the power-supply line and a logic line and ensures that the line is normally pulled up to the supply potential. Open-collector logic devices may be connected to the logic line and each device is then capable of pulling the line low, i.e. to ground.
- **tri-state output (three-state output)** An electronic output stage consisting of a logic gate, commonly an inverter or buffer, that exhibits three possible logic states, namely logic 1, logic 0, and an inactive (high-impedance or open-circuit) state. The inactive state allows the device outputs to be combined with other similar outputs in a busing structure such that only one device is active on the bus at any one time.