CPE166 Advanced Logic Design

Introduction

Professor Jing Pang

Advantage of Digital Devices

– Reproducibility of information

– Flexibility and functionality: easier to store, transmit and manipulate information

– Economy: cheaper device and easier to design

– Moore's law

- Transistor geometry
- Chips double its density (number of transistor) in every 18 months
- Devices become smaller, faster and cheaper

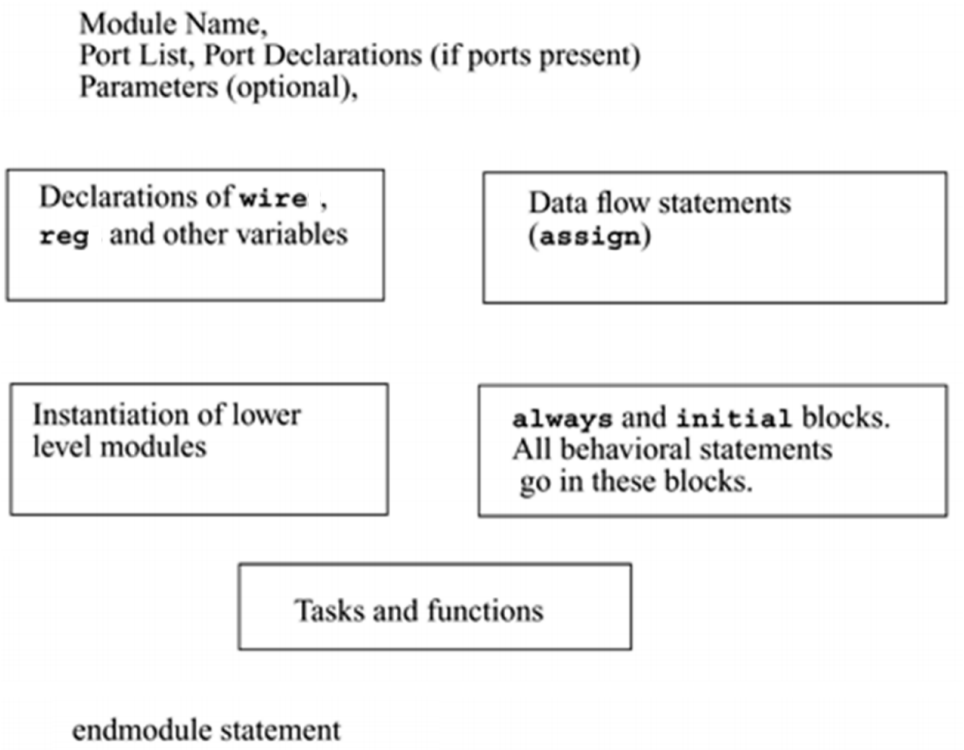
Two HDLs (Hardware Description Languages) Used Today

– VHDL and Verilog

– Syntax and ''appearance'' of the two languages are very different

– Both are IEEE and industrial standards

Components of a Verilog Module



Nesting of Modules

In Verilog nesting of modules is not permitted i.e., one module definition cannot contain another module definition within the module and endmodule statements.

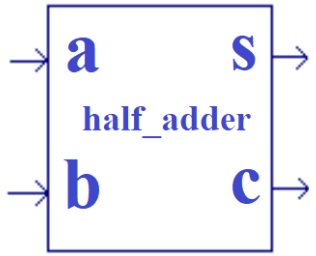
Example:

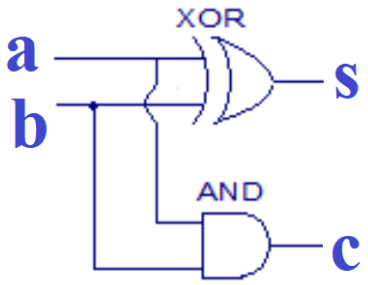
```
module counter(q, clk, reset);
output [3:0]q;
input clk, reset;

    module T_FF(q, clock, reset)   // Illegal
    .
        endmodule
endmodule
```

Structural Design

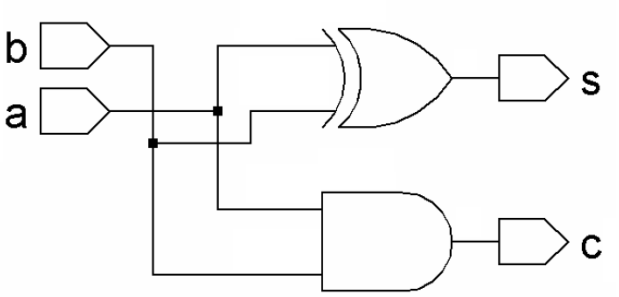
```
module half_adder (a,b,s,c);
output s,c;
input a, b;
wire  s, c;
xor  g1 (s,a,b);
and  g2(c,a,b);
endmodule
```

Inputs		Outputs	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1





```
module ha(a,b,s,c);
output s,c;
input a, b;
wire  s, c;
xor   g1(s,a,b);
and  g2(c,a,b);
endmodule
```




```
module ha_tb;
reg a, b;
wire s, c;
ha hh(a,b,s,c);
initial
begin
  a=0; b=0;
end
always
begin
   #2 a=1; b=0;
   #2 a=0; b=1;
   #2 a=1; b=1;
   #2 a=0; b=0;
end
initial $monitor($time, ".  a=%b,  b=%b,  carry=%b, sum=%b", a, b, c, s);
initial #24 $stop;
endmodule
```

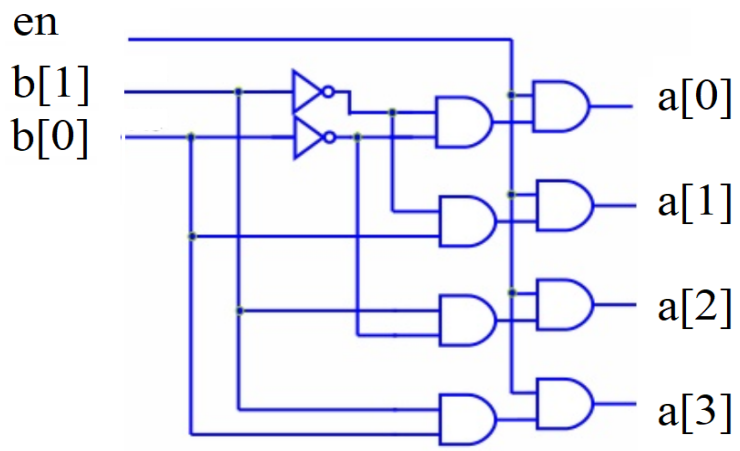
```
 0.   a=0,   b=0,   carry=0, sum=0
 2.   a=1,   b=0,   carry=0, sum=1
 4.   a=0,   b=1,   carry=0, sum=1
 6.   a=1,   b=1,   carry=1, sum=0
 8.   a=0,   b=0,   carry=0, sum=0
10.   a=1,   b=0,   carry=0, sum=1
12.   a=0,   b=1,   carry=0, sum=1
14.   a=1,   b=1,   carry=1, sum=0
16.   a=0,   b=0,   carry=0, sum=0
18.   a=1,   b=0,   carry=0, sum=1
20.   a=0,   b=1,   carry=0, sum=1
22.   a=1,   b=1,   carry=1, sum=0
```

2-to-4 Decoder

```
module dec2_4(a, b,en);
output [3:0] a;
input [1:0]  b;
input        en;
wire [1:0]    bb;

not  g1(bb[1], b[1]), (bb[0], b[0]);
and g2 (a[0], en, bb[1], bb[0]);
and g3 (a[1], en, bb[1], b[0]);
and g4 (a[2], en, b[1],   bb[0]);
and g5 (a[3], en, b[1], b[0]);
endmodule
```

```
module dec2_4_tb;
wire [3:0] a;
reg [1:0]   b;
reg         en;
dec2_4  uut(a, b, en);
initial
begin
      { b, en } = 3'b000;
 #2  { b, en } = 3'b001;
 #2   {b, en } = 3'b011;
 #2   {b, en } = 3'b101;
 #2   {b, en } = 3'b111;
end
initial
 $monitor ($time, " output a=%b, input b=%b, input en=%b", a, b, en);
endmodule
```

```
0 output a=0000, input b=00, input en=0
2 output a=0001, input b=00, input en=1
4 output a=0010, input b=01, input en=1
6 output a=0100, input b=10, input en=1
8 output a=1000, input b=11, input en=1
```



Data Flow Design

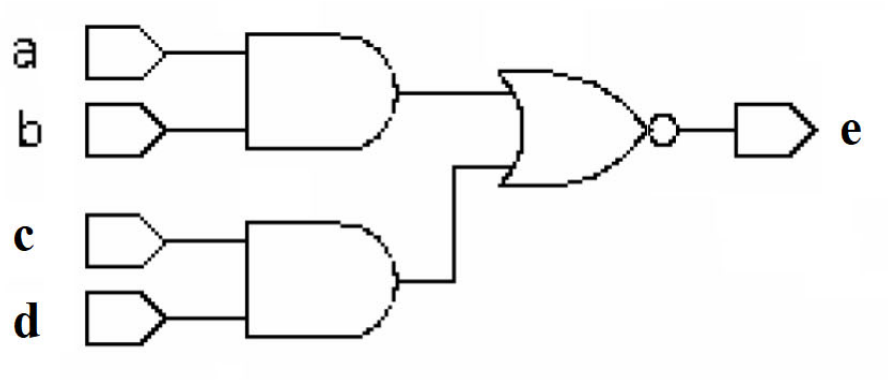
```
module half_adder (a,b,s,c);
output s,c;
input a,b;
wire s,c;
assign s=a ^b;
assign c= a & b;
endmodule
```

Behavioral Design

```
module ex (a,b,s,c);
output s,c;
input a,b;
reg s,c;
always @(a or b)
begin
  if (a==0 && b==0)
  begin
     s=1'b0; c=1'b0;
  end
else if ((a==1 && b==0) || (a==0 && b==1))
  begin
     s=1'b1; c=1'b0;
  end
  else
  begin
     s=1'b0; c=1'b1;
  end
end

endmodule
```

Concurrency

In an electronic circuit all the units are to be active and functioning concurrently.



If (a, b, c or d changes)

update e as

$$e = \overline{a.b + c.d}$$

```
module test_and;
reg  a1, a2;
wire b;
initial
begin
   a1=0;  a2=0;
 #3 a1=1;
 #3 a1=0;
     a2=1;
 #3 a1 =1;
 #10 $stop;
end
and g1(b, a1, a2);
initial $monitor($time, "a1=%b, a2=%b, b=%b", a1, a2, b);
initial  #100  $finish;
endmodule
```

```
0   a1=0, a2=0, b=0
3   a1=1, a2=0, b=0
6   a1=0, a2=1, b=0
9   a1=1, a2=1, b=1
```

Verilog Constructs and Conventions

- Operators

– unary operators: operates on a single operand.

  assign out = ~ a;

– binary operators: operates on two operands.

  assign out = a & b;

– ternary operators: operates on three operands.

  assign out = s ? a : b;

Comments

– single / one line comment example:

```
module d_ff (Q, dp, clk);
//This is the design description of a D flip-flop.
```

– multiple line / block comment Example:

```
/* this logic performs
   even parity design of
   multiple binary bits */
```

Identifiers

- Identifiers are used to define language constructs.
- Identifiers refer objects to be referenced in the design.
- Identifiers are made of alphabets (both cases), numbers, the underscore ‘_’ and the dollar sign ‘$’.
- They start with an alphabetic character or underscore.
- They cannot start with a number or with ‘$’ which is reserved for system tasks.
- Identifiers are case sensitive i.e., identifiers differing in their case are distinct.
- An identifier say count is different from COUNT, count and cOuNT.

Identifiers

- **name, _name. Name, name1, name_$, . .** . . all these are allowed as identifiers
- **name aa** not allowed as an identifier because of the blank (“name” and “aa” are interpreted as two different identifiers)
- **$name** not allowed as an identifier because of the presence of “$” as the first character.
- **1_name** not allowed as an identifier, since the numeral “1” is the first character
- **@name** not allowed as an identifier because of the presence of the character “@”.
- **A+b** not allowed as an identifier because of the presence of the character “+”.

String

A string is a sequence of characters enclosed within double quotes.
A string must be contained on a single line.
Special characters are specified by proceding them with the
"\" character.

"This is a string"
"This string is one \t with a gap in between"
"This is called a \"string\""

Examples for Identifiers

Count

COUNT

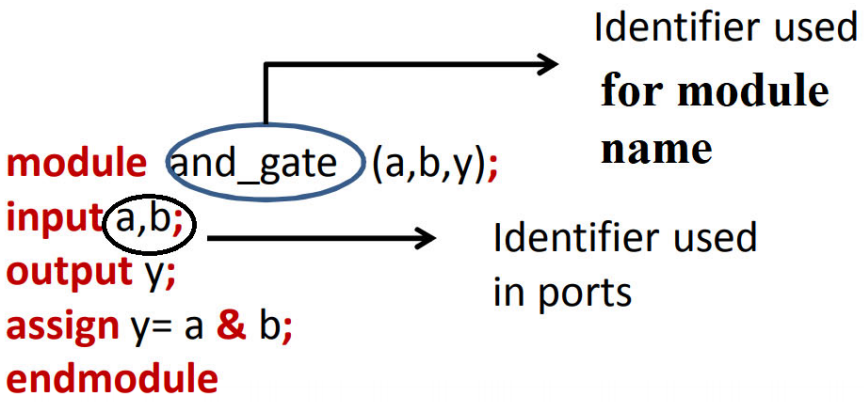
_R2_D2

R56_68

FIVE$

$count Illegal

12six_b Illegal



Keywords

Verilog HDL is case-sensitive.
All the keywords in Verilog must be in lower case.

module → signifies the beginning of a module definition.

endmodule → signifies the end of a module definition.

begin → signifies the beginning of a block of statements.

end → signifies the end of a block of statements.

if → signifies a conditional activity to be checked

Number Specification

Sized numbers.

<size> ‘<base format> <number>

’Unsized numbers.

‘<base format> <number>

<size> in decimal

<base format> can be b or B, d or D, o or O and h or H.

Numbers without <base format> are decimal by default.

Logic Values

1: logic high

0: logic low

x: unknown or uninitialized

z: high impedance or left floating

Numbers

- Sized numbers :

4'b1111 // This is a 4-bit binary number
12'habc // This is a 12-bit hexadecimal number
16'd255 // This is a 16-bit decimal number.
32 ’B z // this is a 32-bit high impedance number
6 ’h x // this is a 6-bit hex number

- Unsized numbers :

23456 // This is a 32-bit decimal number by default
'hc3 // This is a 32-bit hexadecimal number
'o21 // This is a 32-bit octal number

Numbers

5’037	5-bit octal
4’D2	4-bit decimal
9’b11011x01	x signifies the concerned bit to be of unknown value.
9’o12z	equivalent to 001 010 **zzz**
7’Hx	7-bit x (x extended), i.e…. xxxxxxx
4’hz	4-bit z (z extended), i.e… zzzz
4’d-4	Not legal
-4’d7	Its value in 2’s complement form is 7.
8 ‘h 2A	Spaces allowed between size & ‘ character & between *base* and *value*
3’ b001	Not legal: no space allowed between ’ and base b
10’b10	Padded with 0 to the left, 0000000010
11’hb0	equivalent value is 000 1011 0000.
5’h**z**a	A 5-bit hex number. Its value is taken as **z** 1010.
3’b1001_0011	is same as 3’b011

Scalars, Vectors & Parameters

```
wire   m;            // scalar wire type data m
reg    n;            // scalar reg type data n
wire [3:0]  a;       /* a is a 4-bit vector of wire type;  the bits are designated as
                        a[3], a[2], a[1], and a[0]  */

reg [2:0]  b;        /* b is a 3-bit vector of reg type;   the bits are designated as
                        b[2], b[1], and b[0]  */

wire signed [4:0] num;   // num is a vector in the range -16 to +15
reg   signed [3:0] num2; // num2 is a vector in the range -8 to 7

parameter  m = 3,  n=5;
```

Strings & White Space

Strings

- A string is a sequence of characters enclosed by double quotes.
- Spaces are not ignored in strings.
- Strings cannot be on multiple lines.

White Space

- Blank spaces ---> \b
- Tabs ---> \t
- New lines ---> \n
- White space is not ignored in strings.
- Example: $display(“The value of a=%b, b=%b, y=%b \n”, a,b,y);