PASS - Computer Systems

Week 2

1. Gate Logic

Hardware –a set of chips. Chips are made of simpler chips –down to the simplest structure. That structure is an elementary logic gate(Nand). Logic gates are simply implementations (in hardware form) of Boolean functions (AND, OR). This allows us to represent these logical statements in computer form.

a. What are the below Boolean functions?

$x \cdot y$	x AND Y
$x + y$	x OR Y
$\overline{x}$	NOT x
$x \cdot \overline{y} + \overline{x} \cdot y$	x and NOT Y OR not x and not y - XOR
$\overline{x \cdot y}$	NOT x AND Y -NAND

2. Every Boolean function can be expressed using at least one Boolean expression called the canonical representation.

a. What is the ***canonical representation*** of the below Boolean function?

x	**y**	**Z**	**f(x,y)**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Answer	$\overline{x} \cdot y \cdot \overline{z} + x \cdot \overline{y} \cdot \overline{z} + x \cdot y \cdot \overline{z}$

b. What is the ***canonical representation*** of the below Boolean function?

x	**y**	**f(x,y)**
1	1	1
0	1	0
0	0	1
1	0	1

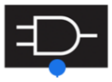
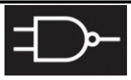
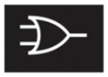
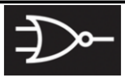
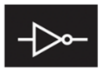
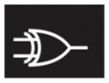
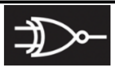
Answer	$x \cdot y + \overline{x} \cdot \overline{y} + x \cdot \overline{y}$

c. What is the ***canonical representation*** of the below Boolean function?

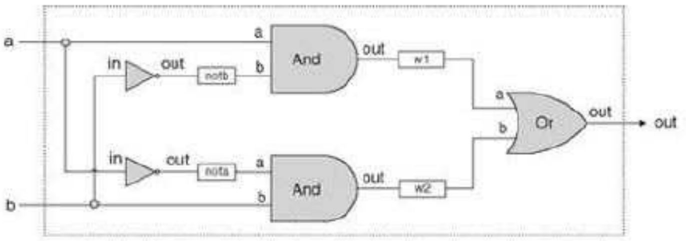
x	**y**	**f(x,y)**
0	0	1
1	0	0
0	1	0
1	1	1

Answer	$\bar{x} \cdot \bar{y} + x \cdot y$

3. What are the logic gates below?

						
AND	NAND	OR	NOR	NOT	XOR	XNOR

4. Which part is the interface? Which part is the implementation? Which part is HDL program?



HDL program (Xor.hdl)

```
/* Xor (exclusive or) gate:
   If a<>b out=1 else out=0. */
CHIP Xor {
   IN a, b;
   OUT out;
   PARTS:
   Not(in=a, out=nota);
   Not(in=b, out=notb);
   And(a=a, b=notb, out=w1);
   And(a=nota, b=b, out=w2);
   Or(a=w1, b=w2, out=out);
}
```

HDL Program

interface

implementation

Test script (Xor.tst)

```
load Xor.hdl,
output-list a, b, out;
set a 0, set b 0,
eval, output;
set a 0, set b 1,
eval, output;
set a 1, set b 0,
eval, output;
set a 1, set b 1,
eval, output;
```

Output file (Xor.out)

```
a | b | out
- - - - - - -
0 | 0 | 0
0 | 1 | 1
1 | 0 | 1
1 | 1 | 0
```

Figure 1.6 HDL implementation of a Xor gate.

What is the difference between an interface, and an implementation?	interface describes input/output is unique implementation is how gate is wired
What is HDL?	High Description Language
Why use HDL?	describe and implement hardware using software

5. Work in groups of two. One person writes the HDL code and the other person draws the implementation
    a. AND gate

Using only Nand gates, draw the logic circuit and write HDL code for the And gate. Clearly label the internal wires with the names used in your HDL.

HDL

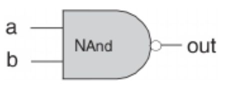
```
Chip And                                   // The available chips:
{                                          // Nand(a=?,b=?,out=?)
    IN a, b ;
    OUT out ;
```

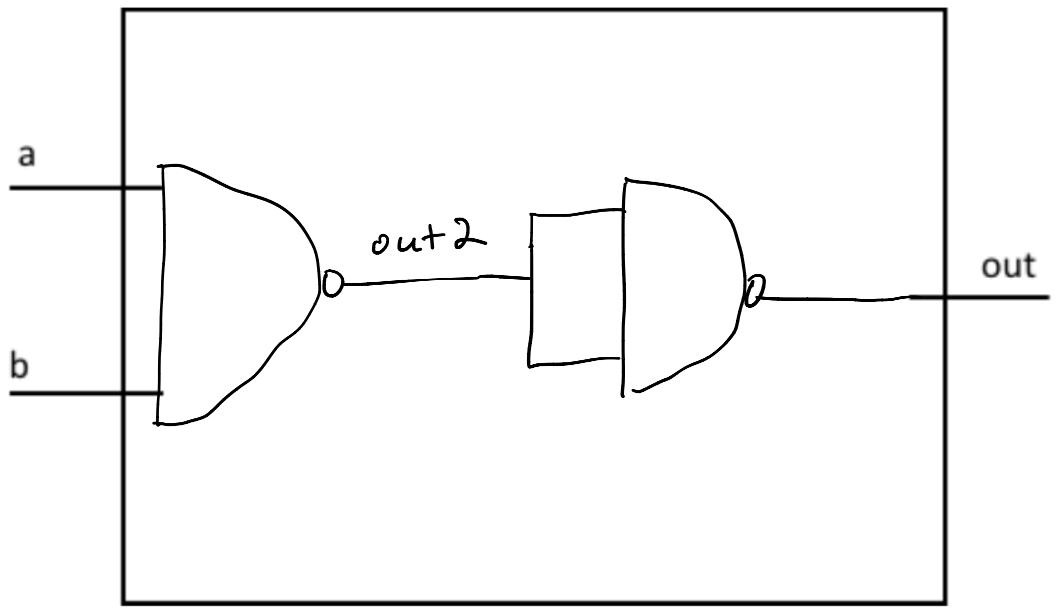
PARTS:

// write your HDL code in this box

```
Nand (a = a, b = b, out = out2)
Nand ( a = out2, b = out2, out = out)
```

Circuit Diagram **Available chips:**





6. OR gate

Using only Nand gates, draw the logic circuit and write HDL code for the Or gate. Clearly label the internal wires with the names used in your HDL.

HDL

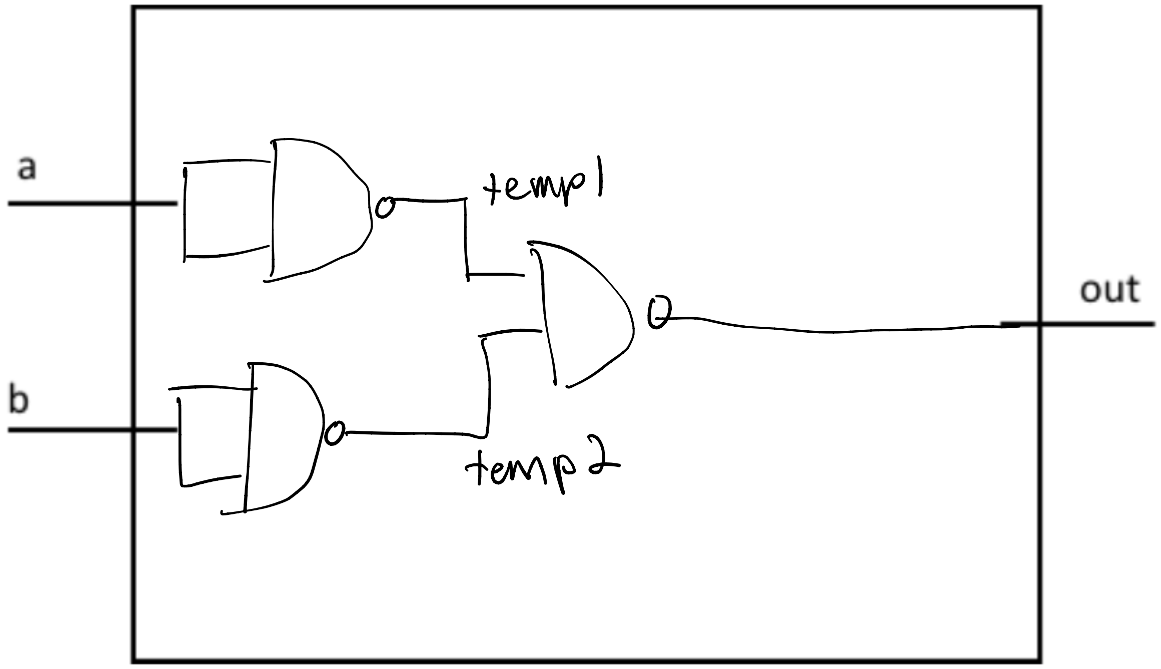
```
Chip Or                                    // The available chips:
{                                          // Nand(a=?,b=?,out=?)
    IN a, b ;
    OUT out ;
```

PARTS:

// write your HDL code in this box

```
Nand (a = a , b = a ,out = temp1 )
Nand ( a=b, b = b, out= temp2)
Nand (a = temp1 , b= temp 2, out =out)
```

Circuit Diagram **Available chips:**



7. Multiplexor

Using only And, Or and Not gates, draw the logic circuit and write HDL code for Multiplexor. The Multiplexor for two values a and b, and selection bit sel must be implemented as: $a \cdot \overline{sel} + b \cdot sel$. Clearly label the internal wires with the names used in your HDL.

HDL

CHIP Mux {

// The available chips:

// And(a=?, b=?, out =?) // Or(a=?, b=?, out=?) // Not(in=?, out=?)

IN a, b, sel;

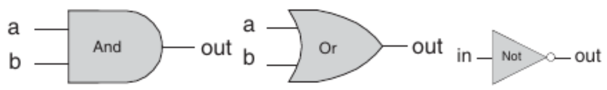
OUT out;

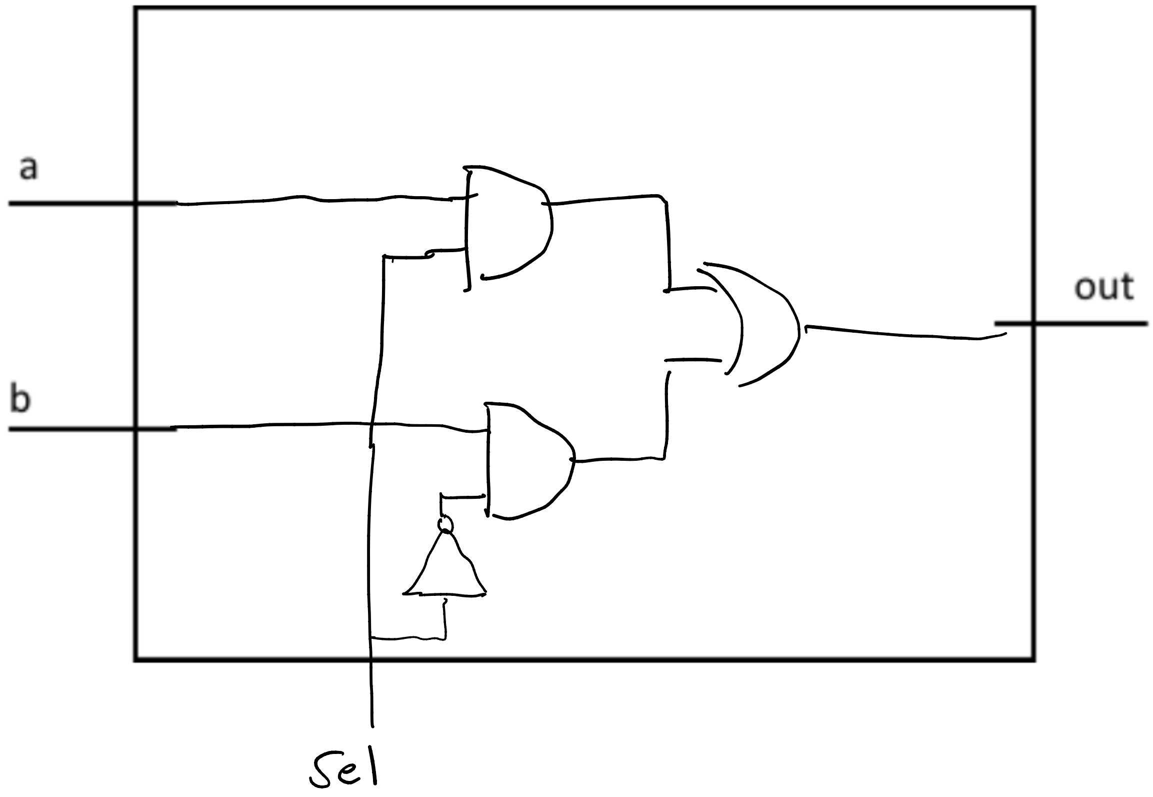
PARTS:

// write your HDL code in this box

```
Not( in = sel, out = notsel)
And (a = a, b = sel, out = outa)
And (a = b, b = notsel, out = outb)
or( a = outa, b = outb, out = out)
```

Circuit Diagram **Available chips:**





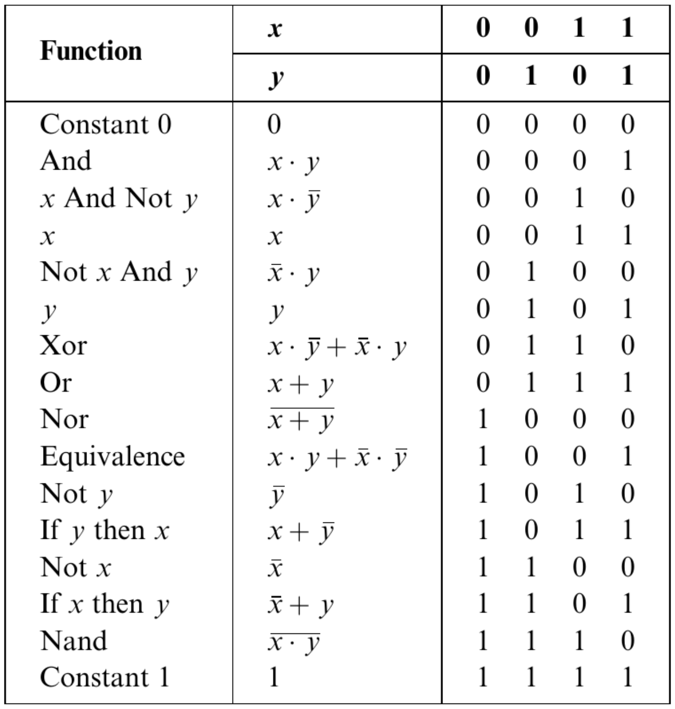
8. Basing on the answers to question 7, answer questions below

What is the minimum number of NAND gates needed to construct a **NOT** gate?	1
What is the minimum number of NAND gates needed to construct an **AND** gate?	2
What is the minimum number of NAND gates needed to construct an **OR** gate?	3
Is there a limit on the number of ways to express a Nand chip in terms of other chips?	no !

Useful Resources

1. It's handy to remember Demorgan's Law:
    - NOT (A OR B) = (NOT A) AND (NOT B)
    - NOT (A AND B) = (NOT A) OR (NOT B)

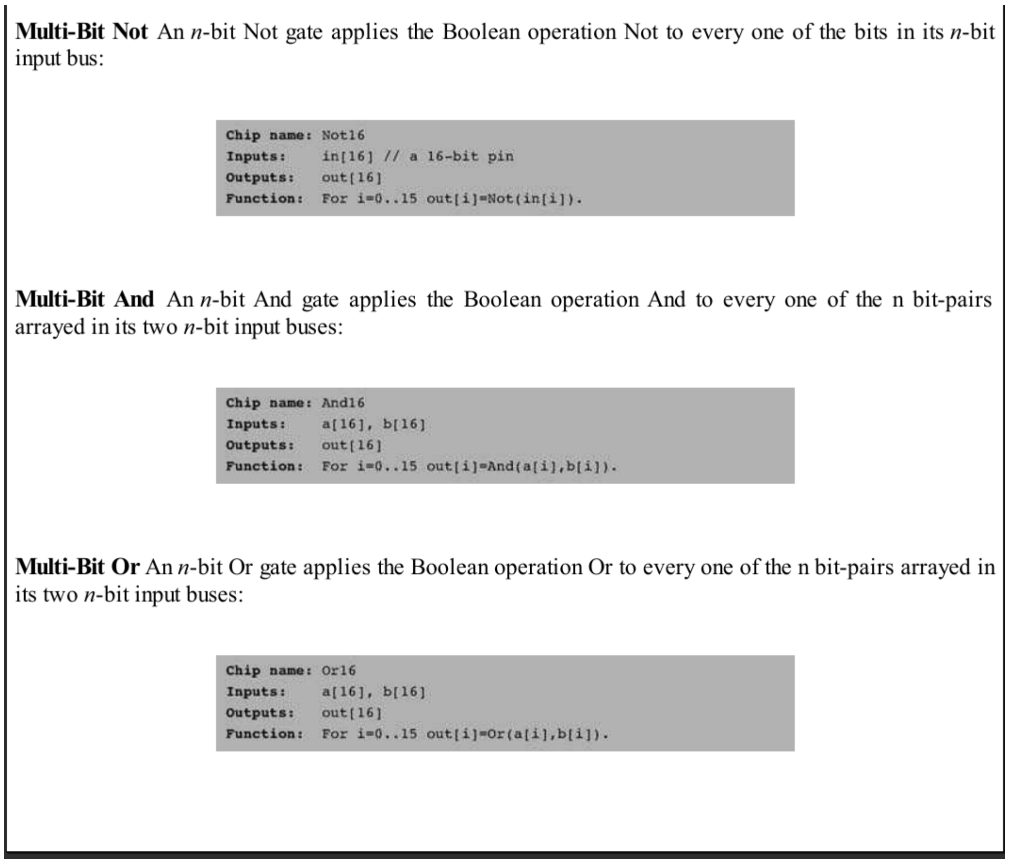
2. Boolean functions



Function	x	**0**	**0**	**1**	**1**
	y	**0**	**1**	**0**	**1**
Constant 0	0	0	0	0	0
And	$x \cdot y$	0	0	0	1
x And Not y	$x \cdot \bar{y}$	0	0	1	0
x	x	0	0	1	1
Not x And y	$\bar{x} \cdot y$	0	1	0	0
y	y	0	1	0	1
Xor	$x \cdot \bar{y} + \bar{x} \cdot y$	0	1	1	0
Or	$x + y$	0	1	1	1
Nor	$\overline{x + y}$	1	0	0	0
Equivalence	$x \cdot y + \bar{x} \cdot \bar{y}$	1	0	0	1
Not y	$\bar{y}$	1	0	1	0
If y then x	$x + \bar{y}$	1	0	1	1
Not x	$\bar{x}$	1	1	0	0
If x then y	$\bar{x} + y$	1	1	0	1
Nand	$\overline{x \cdot y}$	1	1	1	0
Constant 1	1	1	1	1	1

Figure 1.2 All the Boolean functions of two variables.

3. Multi-bit basic gates



Multi-Bit Not An *n*-bit Not gate applies the Boolean operation Not to every one of the bits in its *n*-bit input bus:

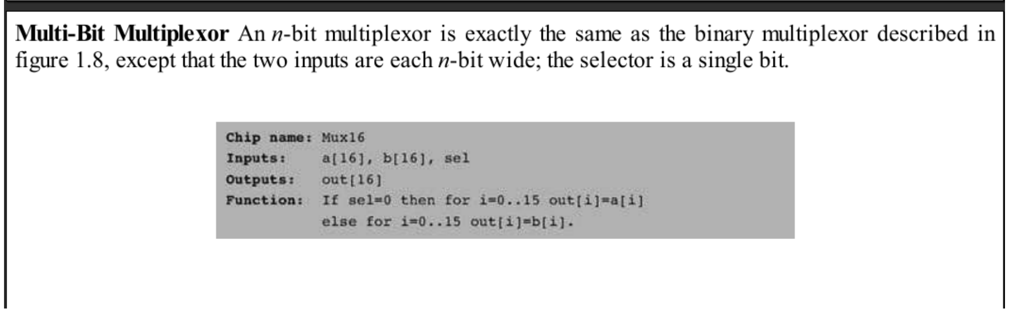
```
Chip name: Not16
Inputs:    in[16] // a 16-bit pin
Outputs:   out[16]
Function:  For i=0..15 out[i]=Not(in[i]).
```

Multi-Bit And An *n*-bit And gate applies the Boolean operation And to every one of the n bit-pairs arrayed in its two *n*-bit input buses:

```
Chip name: And16
Inputs:    a[16], b[16]
Outputs:   out[16]
Function:  For i=0..15 out[i]=And(a[i],b[i]).
```

Multi-Bit Or An *n*-bit Or gate applies the Boolean operation Or to every one of the n bit-pairs arrayed in its two *n*-bit input buses:

```
Chip name: Or16
Inputs:    a[16], b[16]
Outputs:   out[16]
Function:  For i=0..15 out[i]=Or(a[i],b[i]).
```



Multi-Bit Multiplexor An *n*-bit multiplexor is exactly the same as the binary multiplexor described in figure 1.8, except that the two inputs are each *n*-bit wide; the selector is a single bit.

```
Chip name: Mux16
Inputs:    a[16], b[16], sel
Outputs:   out[16]
Function:  If sel=0 then for i=0..15 out[i]=a[i]
           else for i=0..15 out[i]=b[i].
```