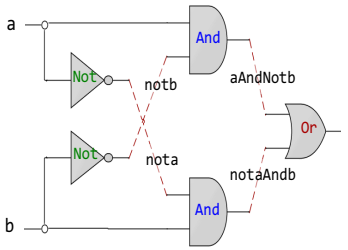
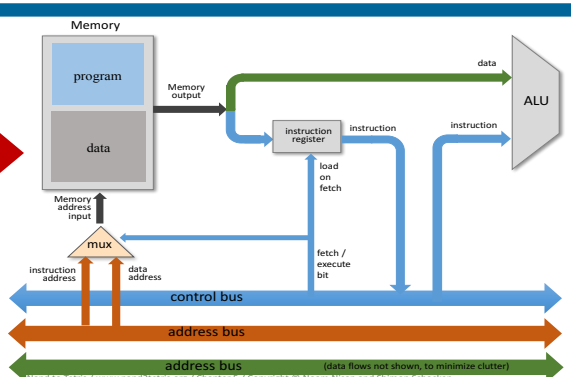
Digital Logic Design




```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
  Not(in=a, out=nota);
  Not(in=b, out=notb);
  And(a=nota, b=b, out=w1);
  And(a=a, b=notb, out=w2);
  Or(a=w1, b=w2, out=out);
}
```




```
@R1
D=M
@temp
M=D
```

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```

Lecture # 01-04

Overview of Course and HDL

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
   msg: db "Learning is fun with Arif", 0Ah, 0h
   len_msg: equ $ - msg
SECTION .text
   main:
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,len_msg
      syscall
      mov rax,60
      mov rdi,0
      syscall
```

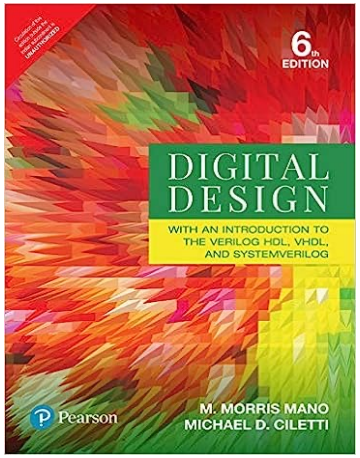
```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

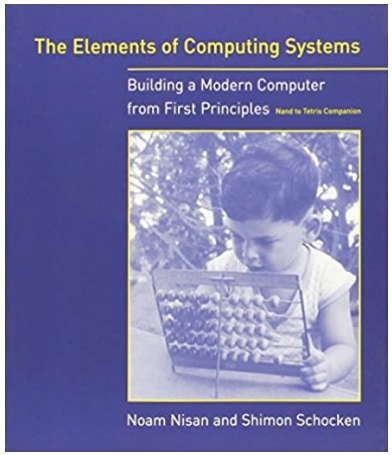
Slides of first half of the course are adapted from:
https://www.nand2tetris.org
Download s/w tools required for first half of the course from the following link:
https://drive.google.com/file/d/0B9c0BdDJz6XpZUh3X2dPR1o0MUE/view

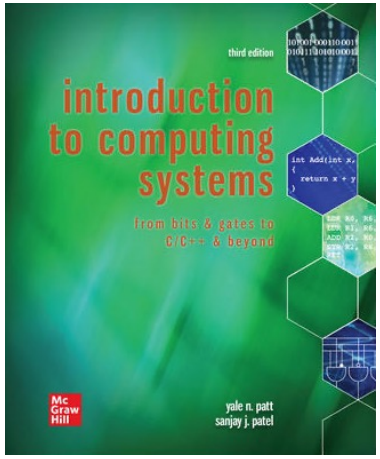
Instructor: Muhammad Arif Butt, Ph.D.

Course Information

- Required Textbooks:
  - Digital Design, by M. Morris Mano, Michael D. Ciletti, 6th Edition, ISBN: 978-93-530-6201-9
  - The Elements of Computing Systems, Building a modern computer, by Noam Nisan and Shimon Schocken, 2nd Ed, Published in 2020, ISBN-13: 978-0262640688
  - Introduction to Computing Systems: from bits and gates to C and beyond, by Yale Patt and Sanjay Patel, 3rd Ed, Published in 2020, ISBN13: 9781260150537







Course Information

- Grades Website: http://online.pucit.edu.pk
- Resources Website: http://arifbutt.me
- Course Prerequisites : Nil
- Students Counseling hours:
  - Mentioned on http://arifbutt.me
- Teaching Assistant info:
  - Mentioned on http://arifbutt.me
- 24 hour turnaround for email: arif@pucit.edu.pk

Where to find Stuff?

Where to find stuff?

http://www.arifbutt.me

- Lecture Slides
- Quizzes + Assignments + Labs
- Announcements
- Teaching Assistants
- SOPs and Course related Policies
- Download s/w tools, codes and other resources required for the course from the following link:

https://github.com/arifpucit/

https://bitbucket.org/arifpucit/

Lecture Format

Lab Format

- Please come to Labs (in time)
- Quizzes might be taken in class or in Lab, so don't miss
- Contents covered in the Lab will come in the Quizzes as well as in the Mid and Final exams

How Will You Be Evaluated?

- Final exam: 40%
- Mid-exam: 35%
- Sessional: 25%
  - Surprise Quizzes: 15%
  - Assignments / Home Tasks: 10%

Surprise Quizzes

- There will be surprise quizzes, given at the start of a lecture, during <u>any</u> lecture. The total number of quizzes could be anywhere between 4 and 40
- NO LATE or MAKEUP SURPRISE QUIZZES, under any circumstances whatsoever
- Surprise quizzes are completely individual efforts
- Your best strategy is to play it safe – attend every lecture and do the reading/programming assignments

Cheating Policy

- Academic integrity
- Both the cheater and the student who aided the cheater will be held responsible for the cheating
- The instructor may take actions such as:
  - require repetition of the subject work,
  - assign 'zero' or may be ‘negative’ marks for the subject work,
  - for serious offenses, assign an F grade for the course

Late Policy for Home Works and PA

- Late policy for Assignment, Quizzes, and other deliverables
  - No late Assignment submissions!
  - No late quizzes or exams!
- Sticking to dates is your responsibility!
  - Check announcements on lecture notes regularly
- Your best strategy is to play it safe – submit everything on time

Playing Safe in CS-223

If you follow these 4 simple rules during the CS223 class, you'll make sure that you do well in the course:

1. Attend every lecture + Lab
2. Study/Understand the course material (textbook sections assigned + slides + Reading assignments), and practice the concepts on the provided tools (H/W simulator, CPU emulator, Assembler,…)
3. Submit everything (PAs, HWs, quizzes, exams) on time - don't be late
4. Don't cheat

Overview of the Course

Problem Solving on Computer

- Human Thought
- Algorithms
- Applications Software
- Systems Software / Compiler
- OS/Runtimes
- **Assembly Language**
- **Machine Language (Instruction Set Architecture)**
- **Microarchitecture (core + memory hierarchy)**
- **Logic Design**
- Device Level
- Physical Design
- Semiconductors/Silicon used to build transistors
- Properties of atoms, electrons, resistors, capacitors

Two Recurring Themes

Abstraction:

- Use of abstraction is all around us
- Take me to the air port
- Go straight 1.2 km, then make a right turn, go down 500 m, then take a left, then go straight for another 750 m, then take a right and so on
- Abstraction is a technique for establishing a simpler way for a person to interact with a system, removing the details that are unnecessary for the person to interact effectively with that system
- It is a productivity enhancer – don't need to worry about details, until some thing goes wrong! And then, it becomes important to understand the components and how they works together

Two Recurring Themes (cont…)

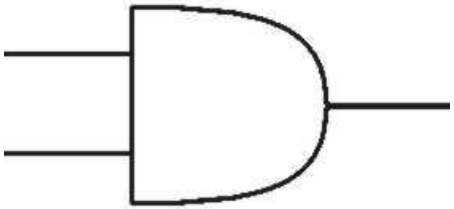
Hardware vs. Software:

- Both are components of a computer system. Even if you specialize in one, you should understand capabilities and limitation of both
- Data types vs finite word length of a computer
- Functions vs function calling convention
- Recursion vs memory layout
- Pointers vs memory layout
- Data structures vs memory layout

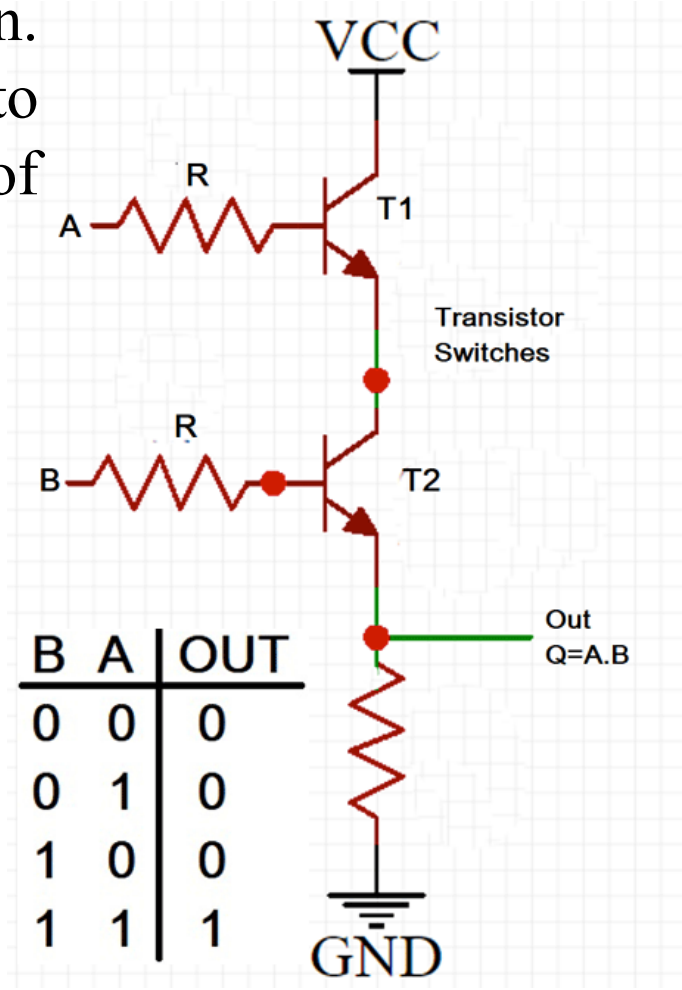
A computer scientist can design much better solutions, when he/she has a mastery of both the worlds!

The Notion of Abstraction: (And Gate)

- Every layer in CS is an abstraction. Depending on which layer you want to live at, you will have different views of the computer

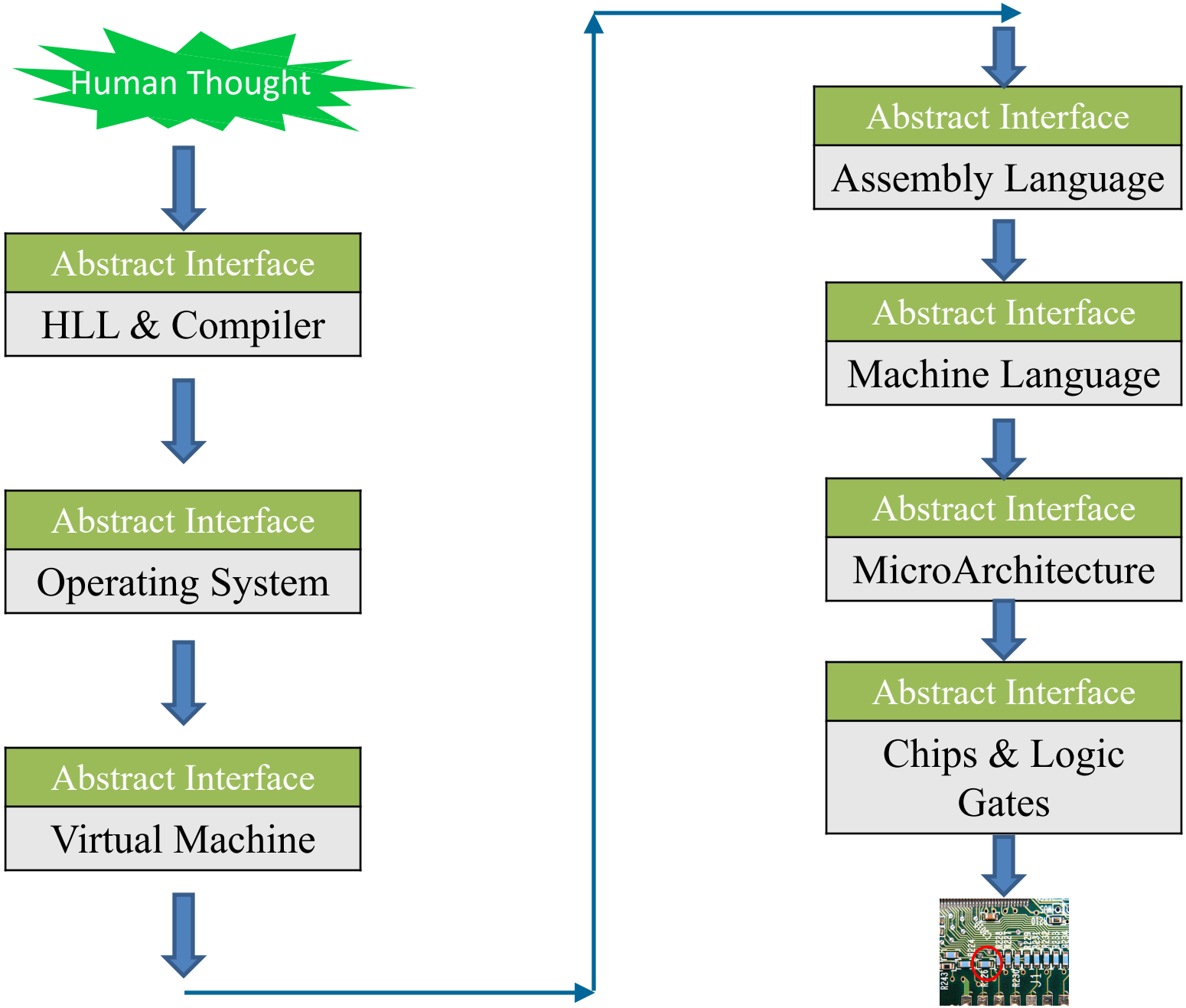


- A transistor is an electronic device that has three ends: a source, a sink, and a gate
- An Intel processor measuring less than a square inch has well over 1.5 billion transistors on it



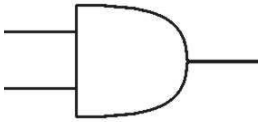
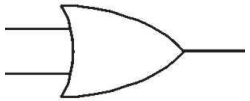
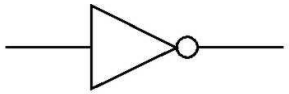
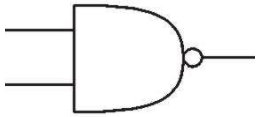
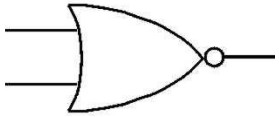
B	A	OUT
0	0	0
0	1	0
1	0	0
1	1	1

AND gate using NPN transistor


Human Thought
Abstract Interface
HLL & Compiler
Abstract Interface
Operating System
Abstract Interface
Virtual Machine
Abstract Interface
Assembly Language
Abstract Interface
Machine Language
Abstract Interface
MicroArchitecture
Abstract Interface
Chips & Logic Gates

Review
Boolean Logic

Elementary Boolean Operations

Gate	Symbol	Operator
And		$A.B$
Or		$A + B$
Not		A'
Nand		$(A.B)'$
Nor		$(A + B)'$
Xor		$A \oplus B$

Boolean Algebra

- Boolean Algebra (George Boole) is the branch of Algebra that deals with logical operations and Binary variables. (Elementary Algebra deals with numerical variables and arithmetic variables)
- Boolean Function is an expression formed by binary variables, logical operators, parenthesis and an equal to sign. The value of a Boolean function can either be zero or one. (Boolean Term can be a product term or sum term)

$$f(x, y) = x'y + xy'$$

$$f(x, y, z) = xy'z + xyz$$

$$f(w, x, y, z) = w'x'y'z + wxyz$$

Boolean Function → Truth Table

- A Truth Table is a listing of all possible combinations of logical variables with the corresponding outputs.
- The number of rows in a Truth Table is 2^n, where n is the number of variables.

$$f(x, y, z) = xy + x'z$$

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Truth Table → Boolean Function

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$f(x,y,z) = \sum(1,3,6,7)$$

$$f(x,y,z) = x'y'z + x'yz + xyz' + xyz$$

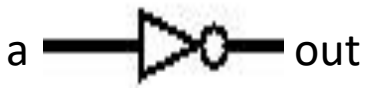
$$f(x,y,z) = xy + x'z$$

You can simplify a Boolean function using Boolean Identities or Karnough Map methods

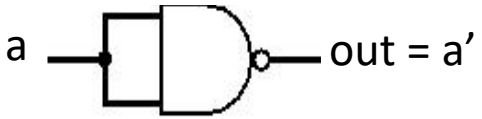
Representation of Boolean Functions using Logic Gates

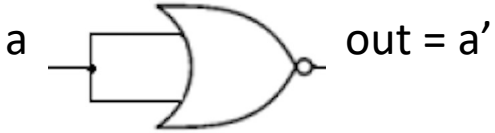
- A logic gate is a device that implements basic logic functions
- Basic Gates (Not, And, Or)
- Special Gates (Xor, Xnor)
- Universal Gates (Nand, Nor)

Implement NOT gate, using NAND/NOR gates:



a	out
0	1
1	0

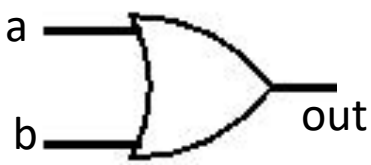




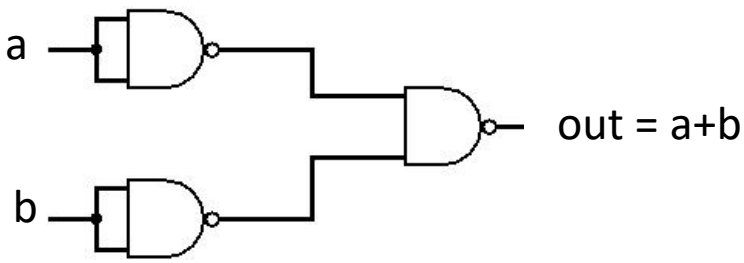
Representation of Boolean Functions using Logic Gates

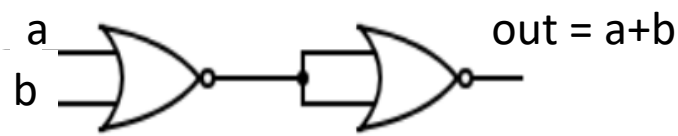
- A logic gate is a device that implements basic logic functions
- Basic Gates (Not, And, Or)
- Special Gates (Xor, Xnor)
- Universal Gates (Nand, Nor)

Implement OR gate, using NAND/NOR gates:



a	b	out
0	0	0
0	1	1
1	0	1
1	1	1

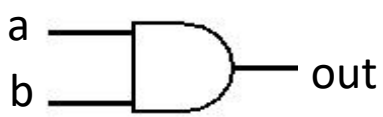




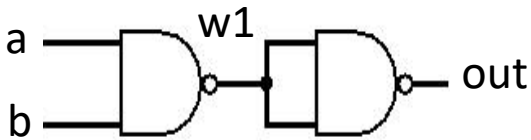
Representation of Boolean Functions using Logic Gates

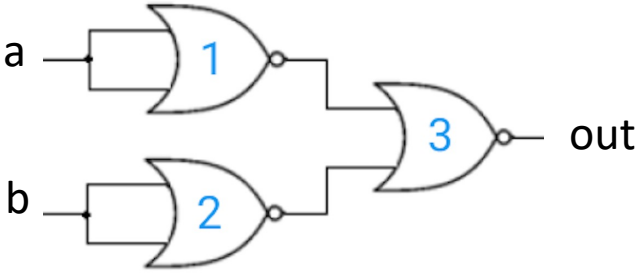
- A logic gate is a device that implements basic logic functions
- Basic Gates (Not, And, Or)
- Special Gates (Xor, Xnor)
- Universal Gates (Nand, Nor)

Implement AND gate, using NAND/NOR gates:



a	b	out
0	0	0
0	1	0
1	0	0
1	1	1





Boolean Identities

$$(xy) = (yx)$$
$$(x + y) = (y + x)$$

Commutative law: Changing the order/sequence of variables does not have any effect on output

$$x(yz) = (xy)z$$
$$(x + (y + z) = (x + y) + z)$$

Associative law: Changing the order in which the logic operations are performed does not have any effect on output

$$x(y + z) = xy + xz$$
$$x + (yz) = (x + y) + (x + z)$$

Distributive law

$$(xy)' = (x' + y')$$
$$(x + y)' = (x'y')$$

De Morgan law

Hardware Description Language

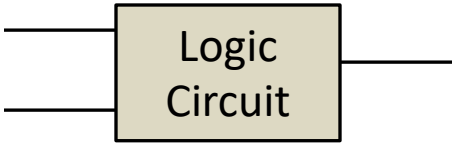
Hardware Description Language

- **Hardware Description Language** is a language that describes the hardware of digital system in textual form
- There are two applications of HDL processing
  - **Hardware Simulation:** We let our HDL programs run inside a h/w simulator to simulate and debug the design. The h/w simulator interprets the HDL and produce readable o/p, that predicts how the h/w will behave before it is actually fabricated
  - **Hardware Synthesis:** The HDL programs can be compiled into h/w implementation using synthesizer and h/w compilation tools. The output of h/w synthesizer is gate level netlist, which is later used to fabricate an IC or to layout a Printed Circuit Board (PCB)
- There are a variety of HDLs available in the market. The most common are SystemVerilog (based on C) and VHDL (Very high speed integrated circuit Hardware Description Language) (based on Ada)
- In this course we will be using a simple/minimal HDL designed and developed by Noam and Shimon (Designers of the course nand2tetris)

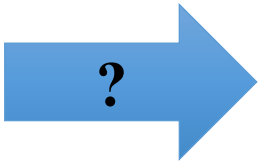
Hardware Simulator

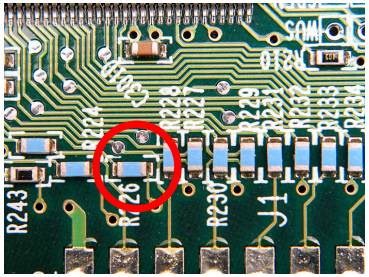
- HDL simulators are software packages that simulate expressions written in one of the hardware description languages, like VHDL, Verilog, SystemVerilog, and so on
- Hardware Simulator that we will be using is designed and developed by students of Interdisciplinary Center Herzliya Efi Arazi School of Computer Science
- It can be used to build and test many different hardware platforms. In this course, we will use it to design a complete computer, called Hack -- a 16-bit computer equipped with a screen and a keyboard
- To design and build this Hack computer we need to write hdl programs for elementary gates, combinational circuits, sequential circuits, registers, RAM, ALU, control unit and its data path. Every time, we write these hdl programs, we will test and debug them on this hardware simulator
- This is how h/w engineers build chips for real:
  - First the h/w is designed tested and optimized on a software simulator
  - Later the resulting gate logic is committed to silicon

Design Process



Behavior

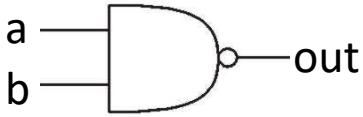




Design Process:

- Design your circuit using the universal NAND gate only
- Write down the HDL program file specifying your logic circuit, using the built-in Nand gate chip having interface *`Nand(a=,b=,out=)`*
- Test the chip in a hardware simulator
- Optimize the design
- Realize the optimized design in silicon

Nand.hdl



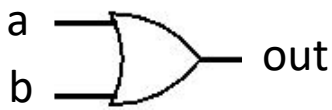

```
/* Nand gate: out = a Nand b */

CHIP Nand {
    IN  a, b;
    OUT out;

    BUILTIN Nand;
}
```

Chip Interface

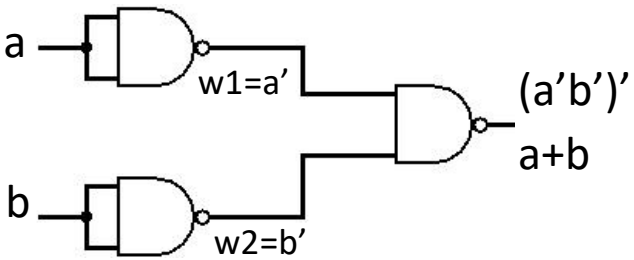
Chip Implementation

You can also write down the HDL for the And, Or and Not gates using Nand gate and then use these And, Or and Not gates to build the logic circuit as usual

Design of Or Gate Chip



a	b	out
0	0	0
0	1	1
1	0	1
1	1	1



Or.hdl

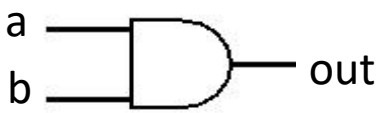
chip interface

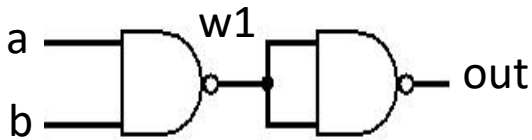
chip implementation

```
/** Or gate: out = a or b */
CHIP Or {
  IN a, b;
  OUT out;

  PARTS:
    Nand(a=a, b=a, out=w1); //(a)'
    Nand(a=b, b=b, out=w2); //(b)'
    Nand(a=w1, b=w2, out=out); //(a'b')'
}
```

```
CHIP Nand {
    IN  a, b;
    OUT out;

    BUILTIN Nand;
}
```

Design of And Gate Chip





a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

And.hdl

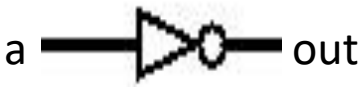
```
/** And gate: out = a And b */
CHIP And {
  IN a, b;
  OUT out;

  PARTS:
    Nand(a=a, b=b, out=w1); //(ab)'
    Nand(a=w1, b=w1, out=out); //ab
}
```

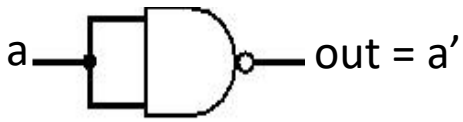
chip interface

chip implementation

```
CHIP Nand {
    IN  a, b;
    OUT out;

    BUILTIN Nand;
}
```

Design of Not Gate Chip





a	out
0	1
1	0

Not.hdl

```
/** Not gate: out = a'*/
CHIP Not {
  IN in;
  OUT out;

  PARTS:
    Nand(a=in, b=in, out=out); //a'

}
```

chip interface

chip implementation

```
CHIP Nand {
    IN  a, b;
    OUT out;

    BUILTIN Nand;
}
```

Interactive Chip Testing on Hardware Simulator

How to Download the H/W Simulator?

- Type the following URL in your browser:

https://github.com/arifpucit/

https://bitbucket.org/arifpucit/

- In the public repositories pain, click the ***coal-repo*** repository, containing all the source codes as well as the software tools used in this course
- In the left pane, click ***Downloads*** to download the entire repository on your system. Now on your system just check the contents of ***tools*** directory that you have just downloaded

```
Arif-MacBook:arifpucit-coal-repo/tools$ ls
HardwareSimulator.sh            HardwareSimulator.bat
CPUEmulator.sh                  CPUEmulator.bat
Assembler.sh                    Assembler.bat
VMEmulator.sh                   VMEmulator.bat
JackCompiler.sh                 JackCompiler.bat
TextComparer.sh                 TextComparer.bat
builtInChips    builtInVMCode    bin    OS
```

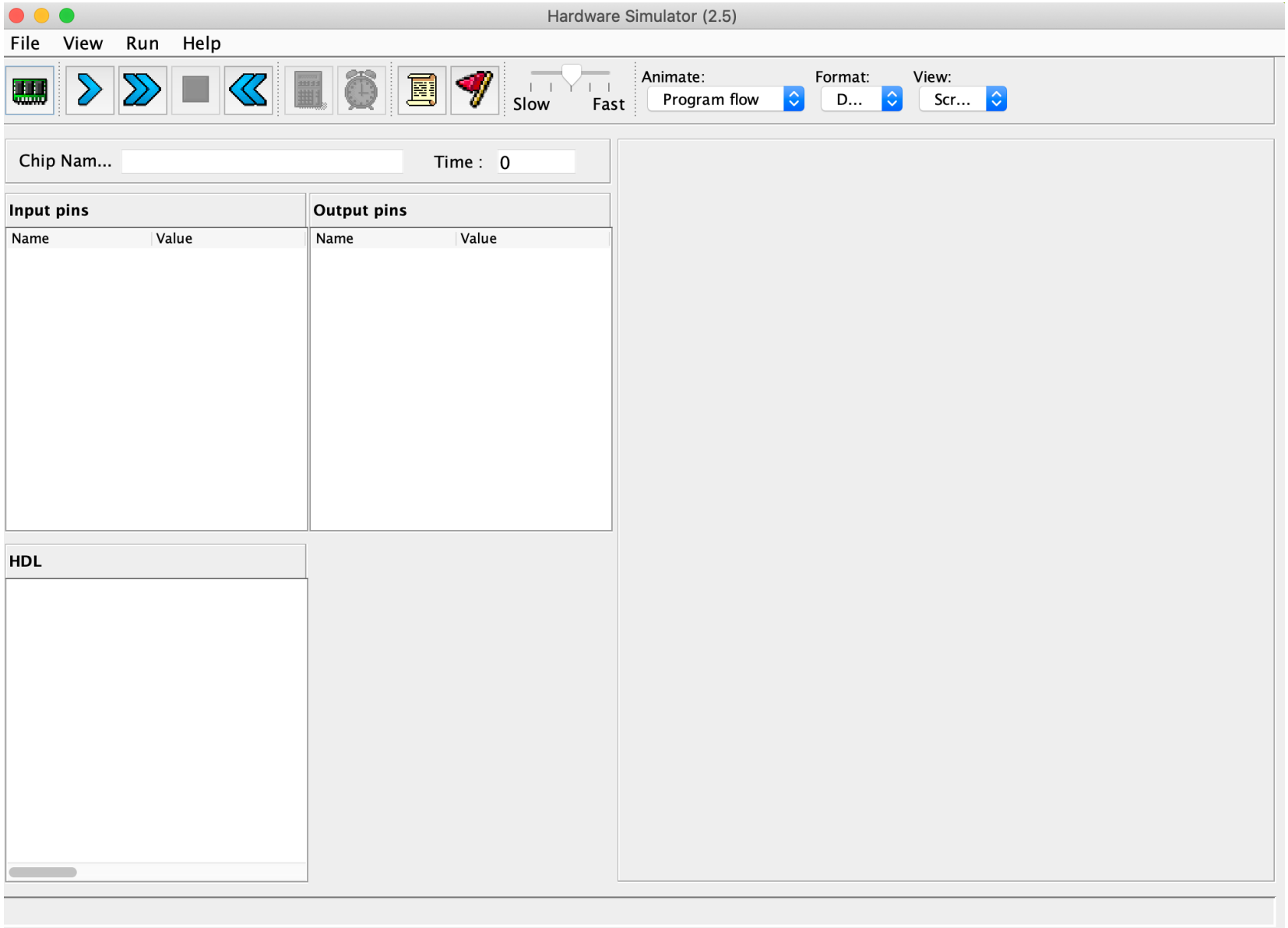
Starting the H/W Simulator

- Follow the following steps to start the h/w simulator on UNIX/Mac OS:
  - Open the terminal
  - Go to tools directory
  - Set execute permissions of the file `HardwareSimulator.sh`
  - Execute it

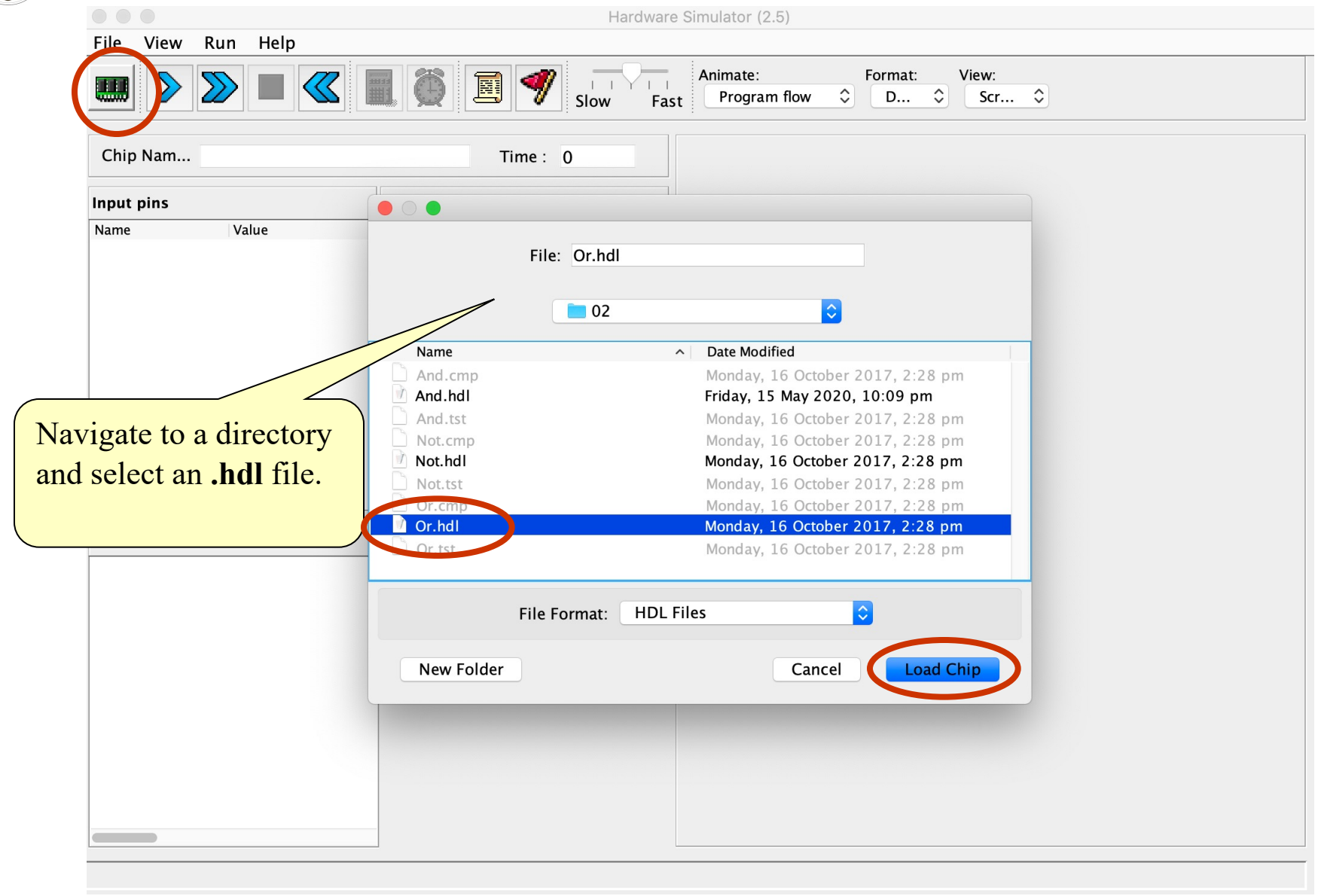
```
tools — -bash — 77×18
(base) Arifs-MacBook-Pro:tools arif$ ls
Assembler.bat           JackCompiler.bat        VMEmulator.sh
Assembler.sh            JackCompiler.sh         bin
CPUEmulator.bat         OS                      builtInChips
CPUEmulator.sh          TextComparer.bat        builtInVMCode
HardwareSimulator.bat   TextComparer.sh
HardwareSimulator.sh    VMEmulator.bat
(base) Arifs-MacBook-Pro:tools arif$ chmod +x HardwareSimulator.sh
(base) Arifs-MacBook-Pro:tools arif$ ./HardwareSimulator.sh
```

Interactive Chip Testing Demo

Java Based H/W Simulator



Loading a Chip in the H/W Simulator



Navigate to a directory and select an **.hdl** file.

Exploring the GUI of the H/W Simulator

Hardware Simulator (2.5) - /Users/arif/Documents/01 Arif-CS223-COAL/Lecture Slides (Video Sessions)/0 Lecture Codes/02/Or.hdl

File View Run Help

Slow Fast Animate: Program flow Format: D... View: Scr...

Chip Nam... Or Time : 0

Input pins

Name	Value
a	0
b	0

Output pins

Name	Value
out	0

- Names and current values of the chip's output pins;
- Calculated by the simulator; read-only.

- Names and current values of the chip's input pins;
- To change their values, enter the new values here.

HDL

```
// File name: 02/Or.hdl
 /**
 * Or gate:
 * out = 1 if (a == 1 or b == 1)
 *       0 otherwise
 */
CHIP Or {
  IN a, b;
  OUT out;
  PARTS:
    Nand(a=a, b=true, out=w1);
    Nand(a=b, b=b, out=w2);
    Nand(a=w1, b=w2, out=out);
}
```

Internal pins

Name	Value
w1	1
w2	1

- Names and current values of the chip's internal pins (used to connect the chip's parts, forming the chip's logic);
- Calculated by the simulator; read-only.

- Read-only view of the loaded **.hdl** file;
- Defines the chip logic;
- To edit it, use an external text editor.

Exploring The Chip Logic

Hardware Simulator (2.5) - /Users/arif/Documents/01 Arif-CS223-COAL/Lecture Slides (Video Sessions)/0 Lecture Codes/02/Or.hdl

File View Run Help

Slow Fast Animate: Program flow Format: D... View: Scr...

Chip Nam... Or Time : 0

Input pins

Name	Value
a	0
b	1

Output pins

Name	Value
out	1

HDL

```
// File name: 02      l
 /**
 * Or gate:
 * out = 1 if (a      or b == 1)
 *       0 otherwi
 */
CHIP Or {
  IN a, b;
  OUT out;
  PARTS:
    Nand(a=a, b=true, out=w1);
    Nand(a=b, b=b, out=w2);
    Nand(a=w1, b=w2, out=out);
}
```

Part pins Nand

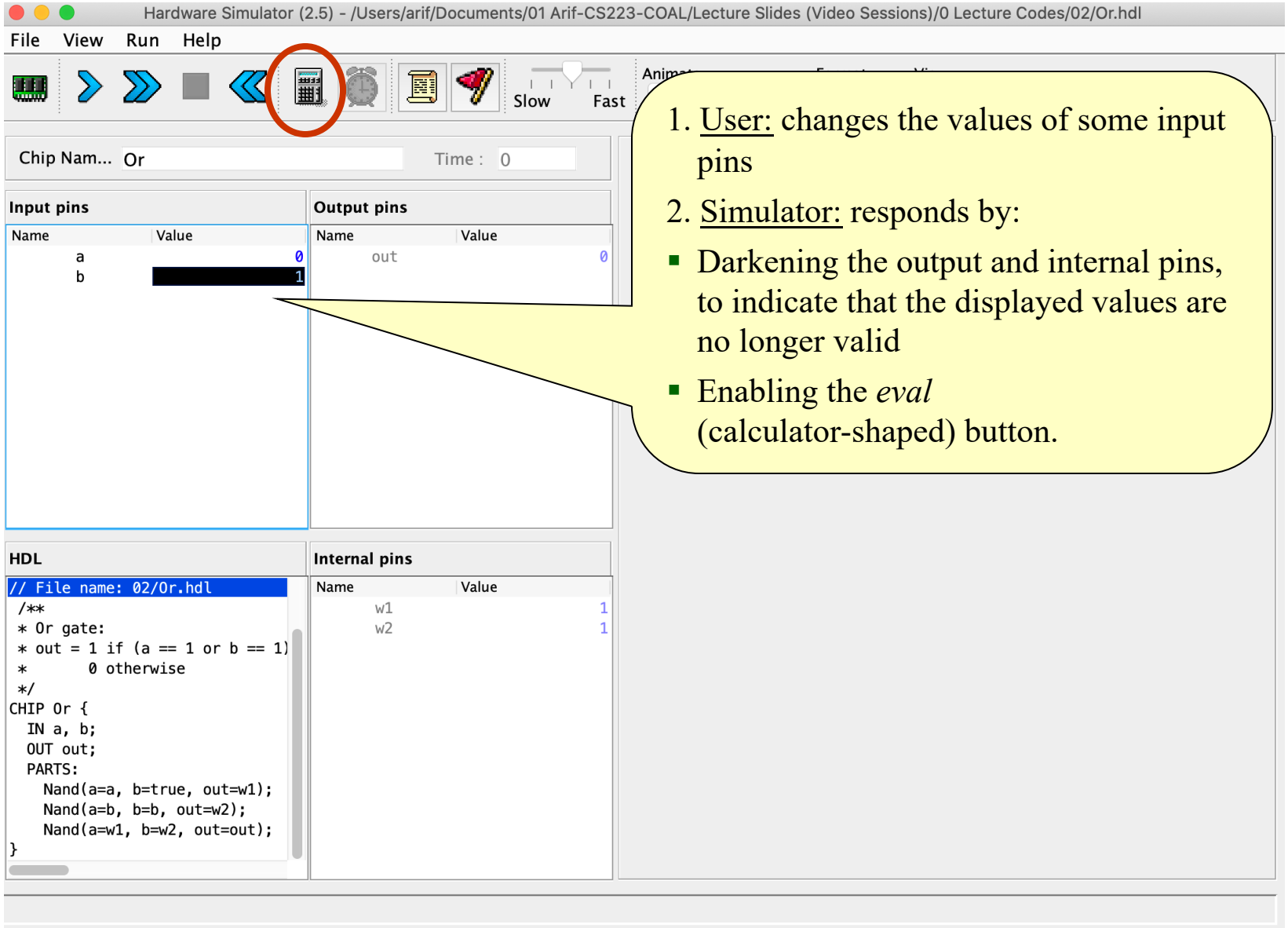
Part pin	Gate pin	Value
a	a	0
b	true	1
out	w1	1

1. Click any one of the chip **PARTS**

2. A table pops up, showing the input/output pins of the selected part (actually, its API), and their current values;

A convenient debugging tool.

Interactive Chip Testing



1. User: changes the values of some input pins
2. Simulator: responds by:
   - Darkening the output and internal pins, to indicate that the displayed values are no longer valid
   - Enabling the *eval* (calculator-shaped) button.

Interactive Chip Testing (cont…)

Hardware Simulator (2.5) - /Users/arif/Documents/01 Arif-CS223-COAL/Lecture Slides (Video Sessions)/0 Lecture Codes/02/Or.hdl

File View Run Help

Slow Fast Animate

Chip Nam... Or Time : 0

Input pins

Name	Value
a	1
b	1

Output pins

Name	Value
out	1

HDL

```
// File name: 02/Or.hdl
 /**
 * Or gate:
 * out = 1 if (a == 1 or b == 1)
 *       0 otherwise
 */
CHIP Or {
  IN a, b;
  OUT out;
  PARTS:
    Nand(a=a, b=true, out=w1);
    Nand(a=b, b=b, out=w2);
    Nand(a=w1, b=w2, out=out);
}
```

Internal pins

Name	Value
w1	0
w2	0

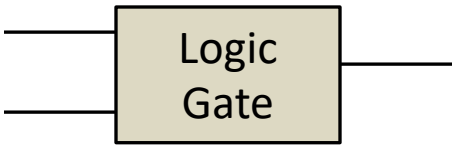
Re-calc

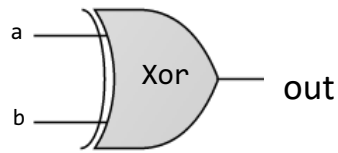
1. User: changes the values of some input pins
2. Simulator: responds by:
   - Dimming the output and internal pins, to indicate that the displayed values are no longer valid
   - Enabling the *eval* (calculator-shaped) button.

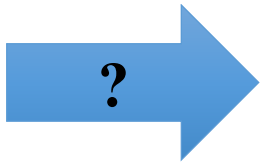
3. User: Clicked the *eval* button
4. Simulator: re-calculates the values of the chip's internal and output pins (i.e. applies the chip logic to the new input values)
5. To continue interactive testing, enter new values into the input pins and click the *eval* button.

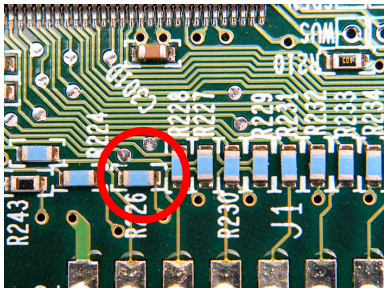
Designing Xor Chip

Designing and Building Xor Chip





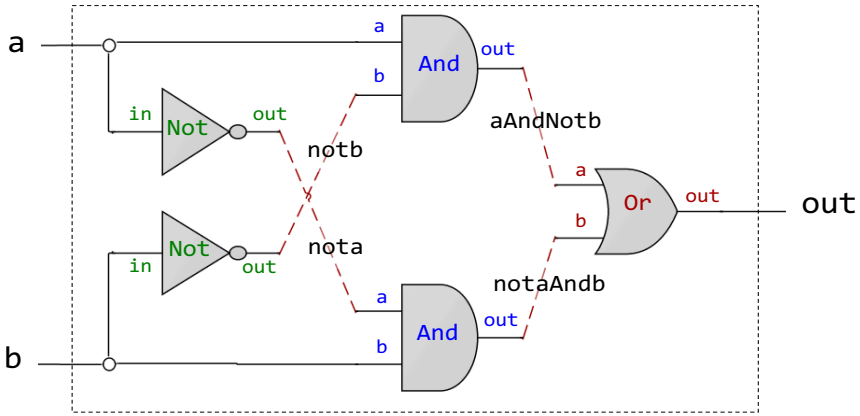




Output is 1 if one, and only one, of its inputs, is 1

a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

$$out(a,b) = a'b + ab'$$

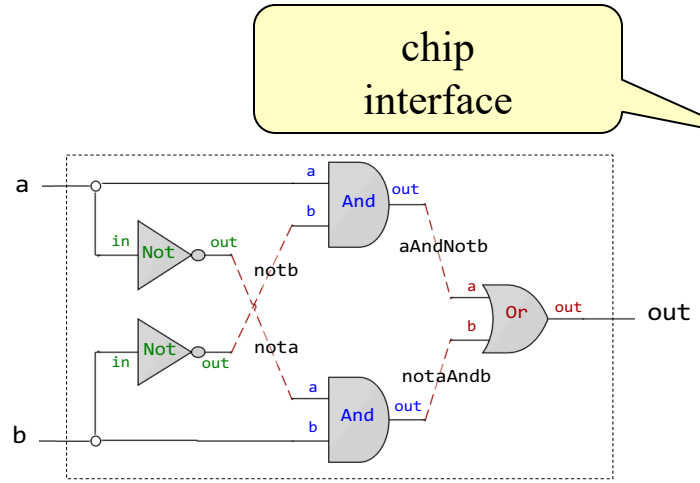


Design Process:

- From truth table derive the simplifie Boolean Function
- Design the gate architecture
- Specify the architecture in HDL
- Test the chip in a hardware simulator
- Optimize the design
- Realize the optimized design in silicon

```
CHIP Xor {
    IN a, b;
    OUT out;
    PARTS:
// Chip Implementation
}
```

Chip Interface

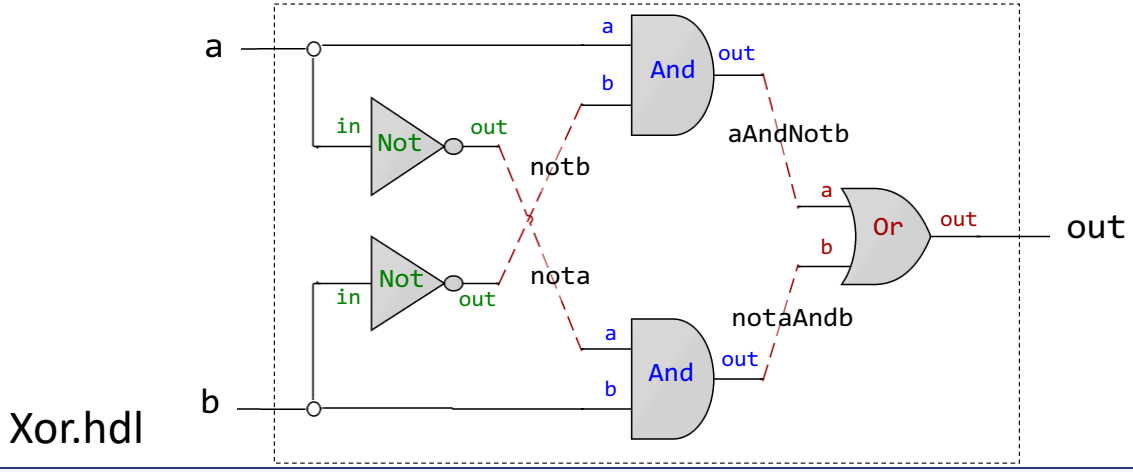



```
/** Exclusive-or gate. out = a xor b */
CHIP Xor {
    IN a, b;
    OUT out;

    // Implementation missing.
}
```

Chip Interface:

- Chip interface is typically supplied by the chip architect; similar to an API, or a contract, which contains:
  - Name of the chip
  - Names of its input and output pins
  - Documentation of the intended chip operation

HDL for Xor Chip




```
/** Xor gate: out = (a And Not(b)) Or (Not(a) And b)) */
 CHIP Xor {
    IN a, b;
    OUT out;
    PARTS:
       Not(in=a, out=nota);
       Not(in=b, out=notb);
       And(a=a, b=notb, out=aAndNotb);
       And(a=nota, b=b, out=notaAndb);
       Or(a=aAndNotb, b=notaAndb, out=out);
```

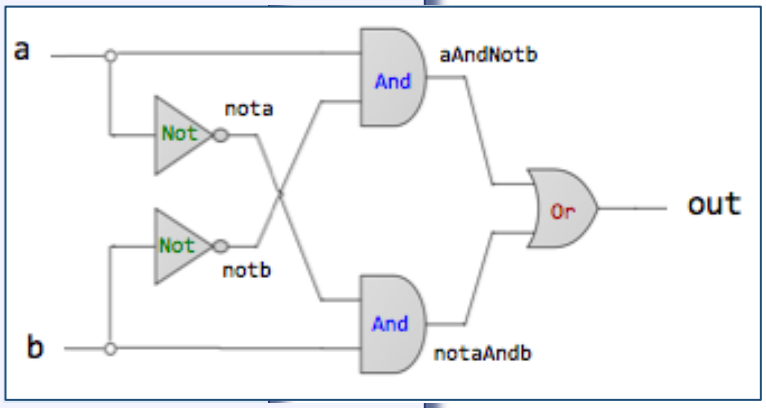
Chip Interface

Chip Implementation

Other Xor implementations are possible!

HDL Some Comments

```
/** Xor gate: out = (a And Not(b)) Or (Not(a) And b)) */

CHIP Xor {
 IN a, b;
 OUT out;
 PARTS:
  Not (in=a, out=nota);
  Not (in=b, out=notb);
  And (a=a, b=notb, out=aAndNotb);
  And (a=nota, b=b, out=notaAndb);
  Or (a=aAndNotb, b=notaAndb, out=out);
```



- HDL is a functional/declarative language
- The order of HDL statements is insignificant
- Before using a chip part, you must know its interface. For example: `Not(in= ,out=)`, `And(a= ,b= ,out= )`, `Or(a= ,b= ,out= )`

Interactive Chip Testing Demo

Class Quiz

Class Quiz (Part:1)

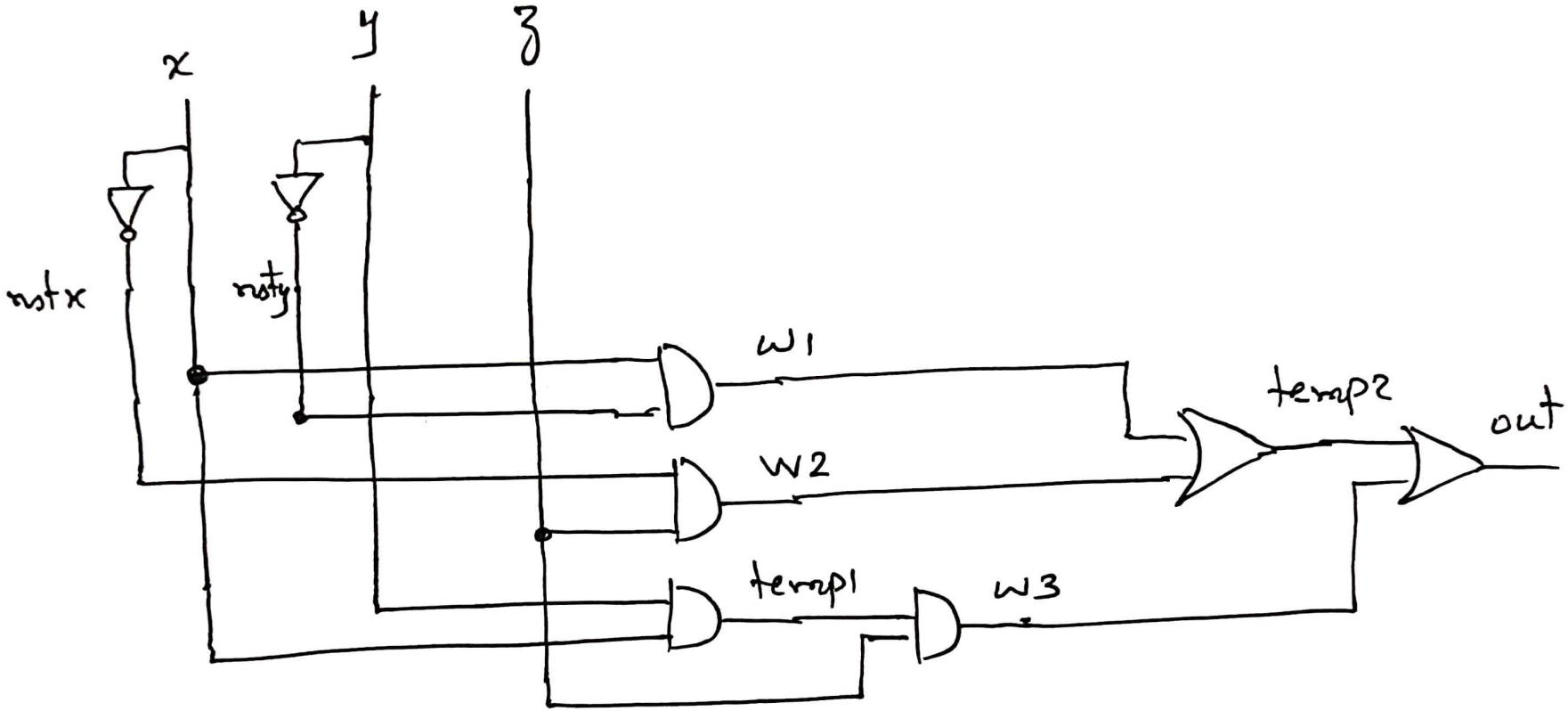
Write down the Truth Table of following Boolean Function that describes it's behavior. Also draw its logic diagram/circuit, which is the graphical representation of a Boolean Function that shows the wiring and connection of each logic gate is called a logic circuit

Note: For more than two inputs AND/OR gate cascade them as both operations are commutative as well as associative

$$\boldsymbol{f(x, y, z) = xy' + x'z + xyz}$$

Class Quiz (Solution Part:1)

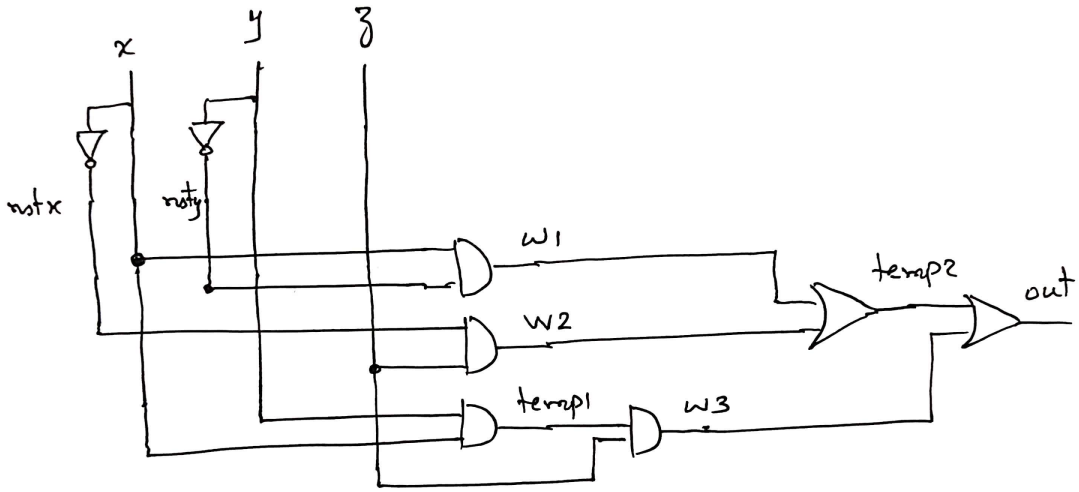
$$f(x, y, z) = xy' + x'z + xyz$$



Class Quiz (Part:2)

Write down the HDL code of this circuit in file named Quiz.hdl

$$f(x, y, z) = xy' + x'z + xyz$$



Assume that you have unlimited quantities of two inputs And, Or gate chips and single input Not gate chips

```
Not(in= ,out=)
And(a= ,b= ,out= )
Or(a= ,b= ,out= )
```

Class Quiz (Solution Part:2)

$$f(x, y, z) = xy' + x'z + xyz$$

```
Not(in= ,out=)
And(a= ,b= ,out= )
Or(a= ,b= ,out= )
```

```
/** Class Quiz */
CHIP Quiz {
  IN x, y, z;
  OUT out;
  PARTS:
  Not(in=x, out=notx);
  Not(in=y, out=noty);
  And(a=x, b=noty, out=w1);
  And(a=notx, b=z, out=w2);
  And(a=x, b=y, out=tmp1);
  And(a=tmp1, b=z, out=w3);
  Or(a=w1, b=w2, out=tmp2);
  Or(a=tmp2, b=w3, out=out);
}
```

Load this chip in the Hardware Simulator and verify the behavior of the function as described in the Truth Table by giving all possible inputs to the chip inside the Hardware Simulator

Representation of Boolean Functions

Representation of Boolean Functions

- A **minterm** is a product term obtained by ANDing the 'n' variables, with each variable being primed if the corresponding bit of the binary number is zero
- A **maxterm** is a sum term obtained by ORing the 'n' variables, with each variable being primed if the corresponding bit of the binary number is one
- Minterms and Maxterms are complement of each other
- A Boolean Function can be represented in any of the following forms:
  - Canonical Form

AND-OR ▪ Sum of Minterms $f(x,y,z) = \sum(1,3,6,7) = x'y'z + x'yz + xyz' + xyz$

OR-AND ▪ Product of Maxterms $f(x,y,z) = \prod(2,5) = (x + y' + z)(x' + y + z')$

  - Standard Form

AND-OR ▪ Sum of Products $f(x,y,z) = y' + xy + x'yz'$

OR-AND ▪ Product of Sums $f(x,y,z) = x(y' + z)(x' + y' + z')$

  - Non-Standard Form $f(x,y,z) = (xy)(x + y) + yz$

Note: Boolean functions in standard and non-standard form can be converted to canonical form by plugging in the missing values

Practice Questions

Given the following Boolean Functions, write down the Truth Table, draw logic circuit, and write the HDL. Count number of gates and number of levels. (Assume you have Not, And, Or chips in the current working directory)

$$f(x,y,z) = \sum(1,3,6,7) = x'y'z + x'yz + xyz' + xyz$$

$$f(x,y,z) = \prod(0,2,5) = (x+y+z)(x+y'+z)(x'+y+z')$$

$$f(x,y,z) = x'z' + xy'z + yz$$

$$f(x,y,z) = x(y'+z)(x'+y'+z')$$

$$f(x,y,z) = (xy).(x+y) + yz$$

$$f(x,y,z) = (xy)'.(x+y)'.z$$

AND-OR to NAND

Given the following Boolean Functions in SOP, draw its logic circuit using AND-OR configuration. Can you implement it using NAND gates only? Write the corresponding Boolean function. Compare the truth table of both functions. Draw the logic circuit using NAND gates only. Write the HDL

$$f(x, y, z) = xy + xz + y'z'$$

$$f(x, y, z) = xz + x'z' + x'y$$

$$f(x, y, z) = xy + x'y' + y'z$$

OR-AND to NOR

Given the following Boolean Function in POS, draw its logic circuit using OR-AND configuration. Can you implement it using NOR gates only? Write the corresponding Boolean function. Compare the truth table of both functions. Draw the logic circuit using NOR gates only. Write the HDL

$$f(x, y, z) = (x + y)(x + z)(y' + z')$$

$$f(x, y, z) = (y' + z)(x' + y)(x' + z)$$

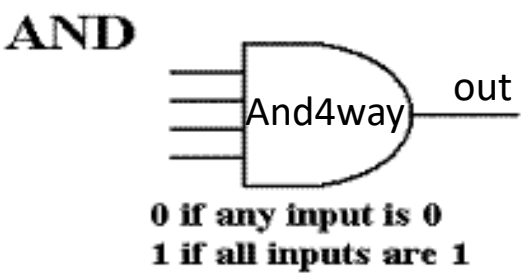
Summary of Concepts related to Boolean Functions

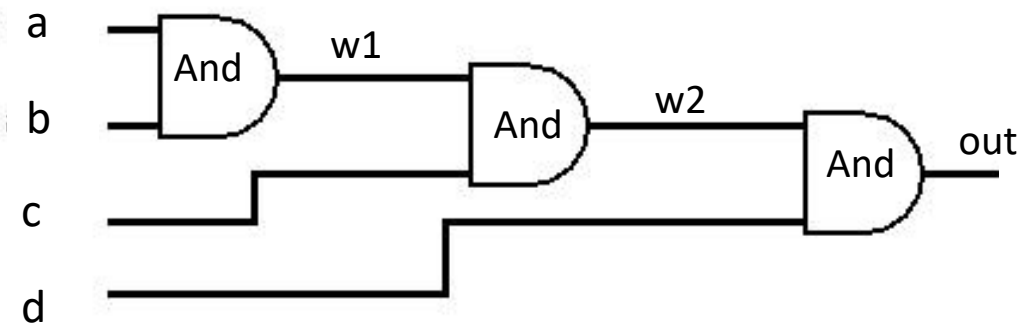
- Know how to draw a logic circuit from Boolean Function and vice-versa
- Know how to express a Boolean function as a Truth Table and vice-versa
- Know how to convert a Boolean Function from Sum of Minterm to Product of Maxterm and vice-versa
- Know how to form a two-level gate structure from a Boolean function in sum of products form
- Know how to form a two-level gate structure from a Boolean function in product of sums form
- Know how to convert a AND-OR circuit to NAND
- Know how to convert a OR-AND circuit to NOR

Gates Having More Than Two Inputs

And4way: Gate that ANDs 4 bits

- Suppose we want to design an AND gate chip with four inputs
- Although we can design it using the built-in NAND gate, but why to reinvent the wheel.
- Let us design it using the already designed AND gate chips with two inputs





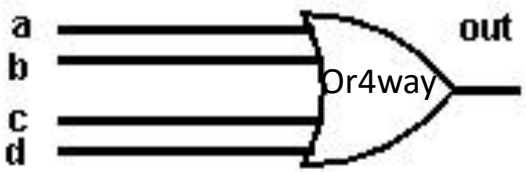
And4way.hdl

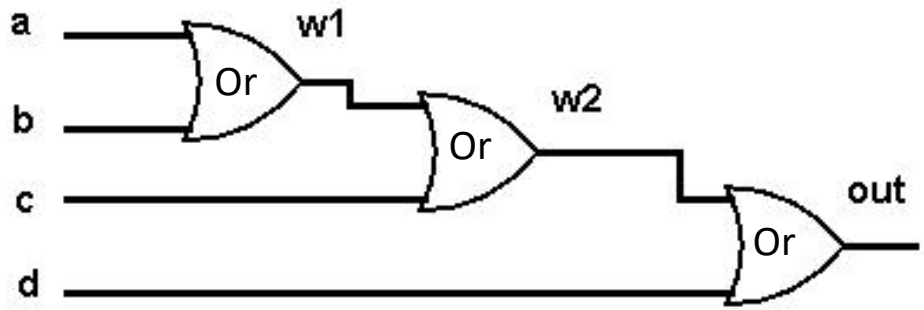
```
CHIP And4way{
  IN a,b,c,d;
  OUT out;
  PARTS:
    And(a=a, b=b, out=w1);
    And(a=w1, b=c, out=w2);
    And(a=w2, b=d, out=out);
}
```

To Do: Design AND8way chip using two AND4way chips and a simple AND chip

Or4way: Gate that ORs 4 bits

- In a similar fashion, we can design an OR gate chip with four inputs using the already designed OR gate chips with two inputs





Or4way.hdl

```
CHIP Or4way{
  IN a,b,c,d;
  OUT out;
  PARTS:
  Or(a=a, b=b, out=w1);
  Or(a=w1, b=c, out=w2);
  Or(a=w2, b=d, out=out);
}
```

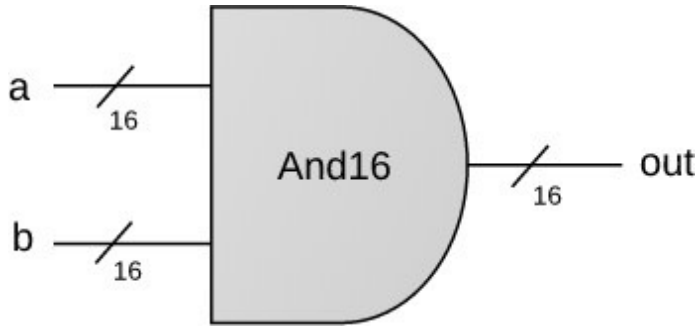
To Do: Design OR8way chip using two OR4way chips and a simple OR chip

Multi Bit Gates Demo

Gates Having Two Inputs Each of 16 Bits

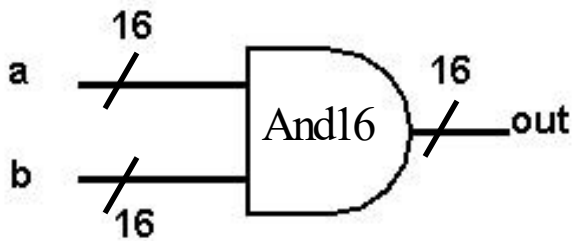
Array Of Bits

- While designing hardware, a lot of times we need to manipulate a bunch of bits together and it is conceptually convenient to think about the bunch of bits that are manipulated together as one entity called busses
- Example: A chip that performs bit-wise AND of two 16 bit numbers. So the chip has two inputs each of 16 bits. The chip also has an output of 16 bits. So in reality, the chip has 32 wires feeding into it, and 16 wires going out of it, but still it's convenient to think about it as two numbers feeding in and one number feeding out




```
CHIP And16 {
    IN a[16], b[16];
    OUT out[16];

    PARTS:
    //  Put your code here:
}
```

And16: Gate that AND two 16-bit Numbers

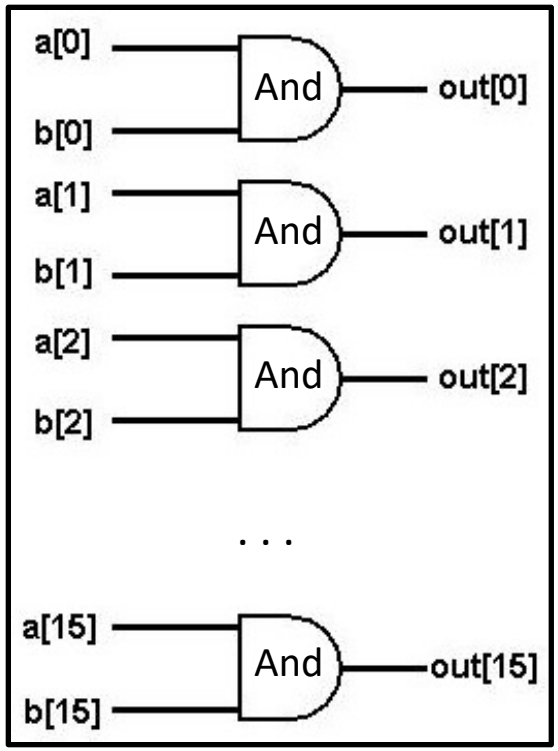


a = 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0 0

b = 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0

out = 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0

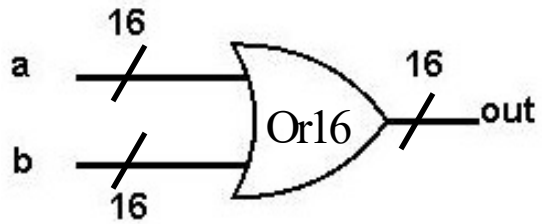
To Do: Design And8 chip using two And4 chips and a simple And chip



And16.hdl

```
CHIP And16{
  IN a[16], b[16];
  OUT out[16];

  PARTS:
     And(a=a[0], b = b[0], out=out[0]);
     And(a=a[1], b = b[2], out=out[1]);
     And(a=a[2], b = b[3], out=out[2]);
   . . . .
     And(a=a[15], b = b[15], out=out[15]);
}
```

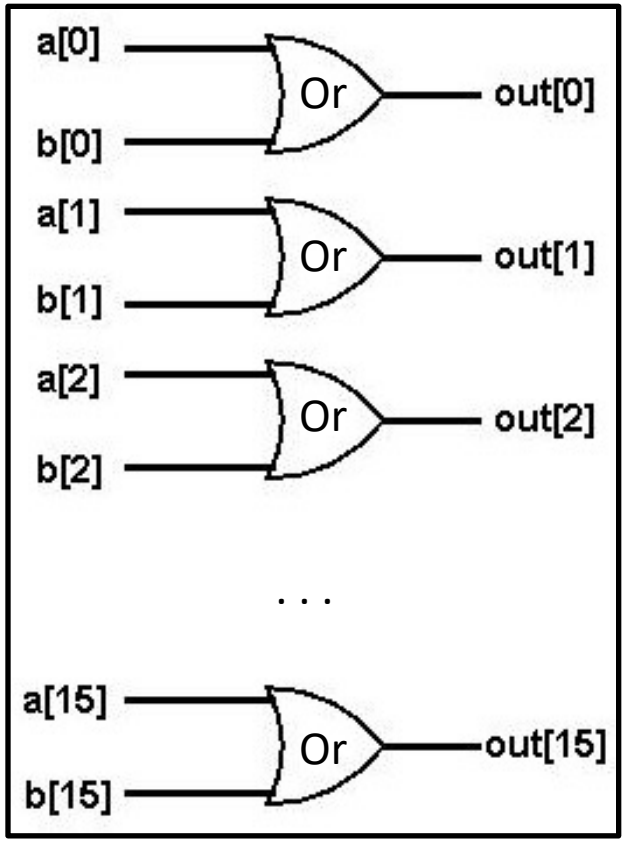
Or16: Gate that OR two 16-bit Numbers



a = 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0 0

b = 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0

out = 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 0

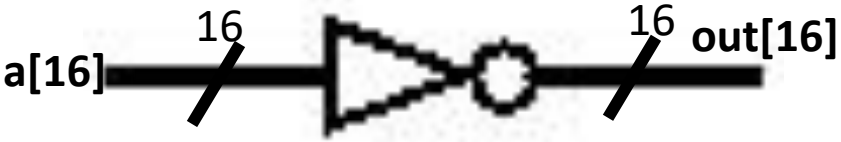


To Do: Design Or8 and Or4 chip, and then use four Or8 chips and one Or4 chip to build Or32 chip

Or16.hdl

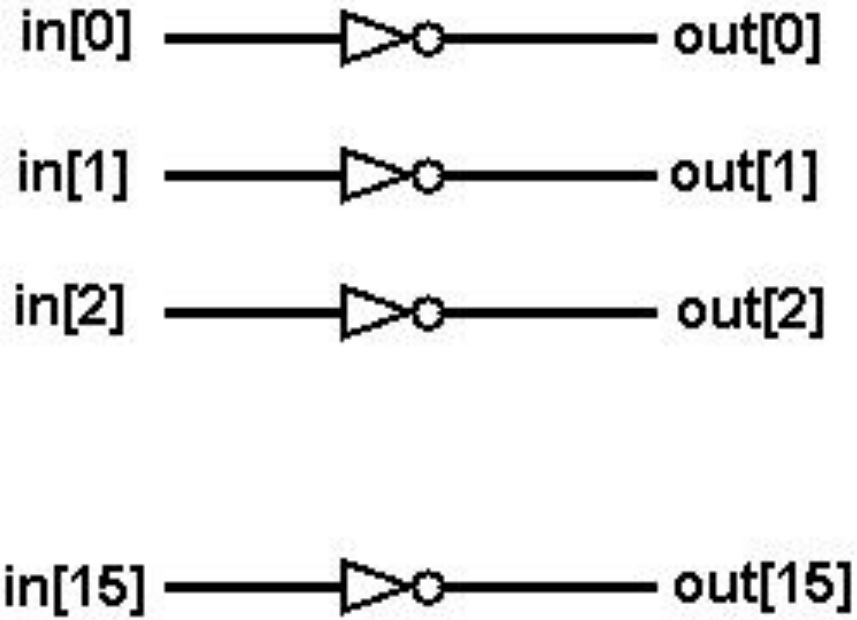
```
CHIP Or16{
  IN a[16], b[16];
  OUT out[16];

  PARTS:
     Or(a=a[0], b = b[0], out=out[0]);
     Or(a=a[1], b = b[2], out=out[1]);
     Or(a=a[2], b = b[3], out=out[2]);
    . . . .
     Or(a=a[15], b = b[15], out=out[15]);
}
```

Not16: Gate that Perform Not of 16-bit Number



a = 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0

out = 1 1 0 1 0 0 1 0 1 1 0 1 0 1 0 1



Not16.hdl

```
CHIP Not16{
  IN a[16];
  OUT out[16];

  PARTS:
    Not(in=a[0], out=out[0]);
    Not(in=a[1], out=out[1]);
    Not(in=a[2], out=out[2]);
   . . . .
    Not(in=a[15], out=out[15]);
}
```

Concept of Sub-Buses

- Buses are indexed right to left: if foo is a 16-bit bus, Then foo[0] is the right-most bit (LSb), and foo[15] is the left-most bit (MSb)
- Buses can be composed from sub-buses, i.e., we can compose a 16 bit bus from two 8 bit buses
- Example: In the code snippet below, we have two 8 bit buses namely lsb and msb. In the first 16 bit value to And16 chip we plug in the 8 bits of lsb and the 8 bits of msb. Note the dotdot notation using which we can mention the sub range of a bus
- Lastly if you want to initialize an entire bus with zeros or ones, you can do so in one command by assigning “true” or “false” to the bus

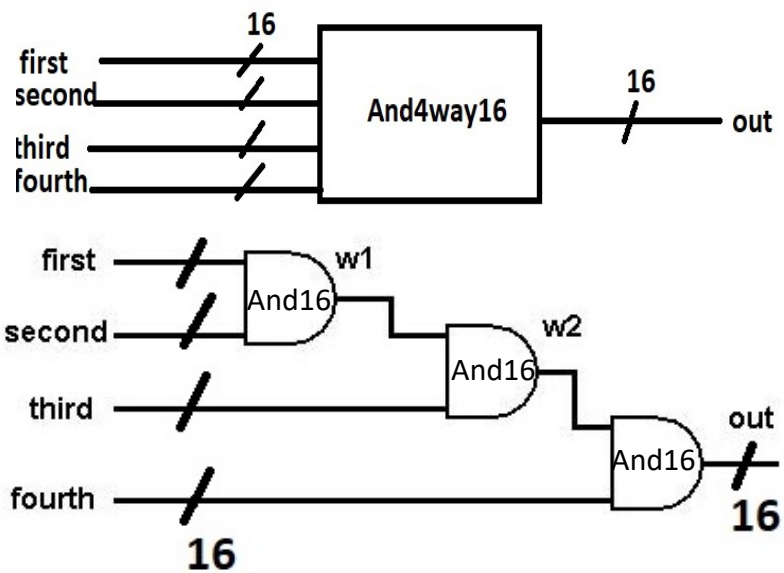
```
...
IN lsb[8], msb[8], …
...
And16(a[0..7]=lsb, a[8..15]=msb, b=…, out=…);
```

Gates with Buses: Demo

Gates Having More than 2 Input Variables each of 16 bits

And4way16

- Suppose now we need to build a chip that bit-wise And four 16 bit numbers. We can design this chip using three And16 chips each capable of Anding two 16 bit numbers
  - The first And16 chip will bit-wise And two 16 bit numbers and place the result in a variable, w1
  - The second And16 chip will bit-wise And the third 16 bit number with w1 and place the result in w2
  - The third Add16 chip will bit-wise And the fourth 16 bit number with w2 and generate the final output



And4way16.hdl

```
CHIP And4way16 {
  IN first[16], second[16],
third[16], fourth[16];
  OUT out[16];

  PARTS:
  And16(a=first, b = second, out=w1);
  And16(a=w1, b = third, out=w2);
  And16(a=w2, b = fourth, out=out);
}
```

To Do: Design Or4way16 chip

Multi-Bit Gates with Buses: Demo

What is a Built-in Chip?

Built-in Chips

General

- A built-in chip has an HDL interface and a Java implementation (e.g. here: Mux16.class)
- The name of the Java class is specified following the BUILTIN keyword
- Built-In implementations of all the chips that are supplied in the tools/buitInChips directory

```
// Mux16 gate (example)
CHIP Mux16 {
   IN a[16],b[16],sel;
   OUT out[16];
   BUILTIN Mux16;
}
```

Built-in chips are used to:

- Implement basic primitive gates to build other gates (Nand and DFF)
- Provide the functionality of chips that the user did not implement for some reason
- Improve simulation speed and save memory (when used as parts in complex chips)
- Implement chips that have peripheral side effects (like I/O devices)
- Implement chips that feature a GUI (for debugging)
- Built-in chips can be used either explicitly, or implicitly

Note: The supplied simulator software features built-in chip implementations of all the chips in the Hack chip set. If you don't implement some chips from the Hack chipset, you can still use them as chip-parts of other chips: Just rename their given stub files to, say, Mux16.hdl1. This will cause the simulator to use the built-in chip implementation

Explicit Use Of Built-in Chips

Hardware Simulator (1.4b1) - G:\TECS\tools\builtIn\Mux16.hdl

File View Run Help

Slow Fast

Animate: Program flow | Format: Decimal | View: Screen

Chip Name : Mux16 Time : 0

Input pins

Name	Value
a[16]	0
b[16]	0
sel	0

Output pins

Name	Value
out[16]	0

HDL

```
// MIT Press 2004. Book site: http://ww
// File name: tools/builtIn/Mux16.hdl

/**
 * 16-bit multiplexor. If sel=0 then
 */

CHIP Mux16 {

  IN a[16], b[16], sel;
  OUT out[16];

  BUILTIN Mux;
}
```

Standard interface.

Built-in implementation.

The chip is loaded from the **tools/buitIn** directory (includes executable versions of all the chips mentioned in the book).

Load Chip

Look in: builtIn

HalfAdder.hdl
Inc16.hdl
Keyboard.hdl
Mux.hdl
Mux16.hdl
Mux4Way16.hdl

File name: Mux16.hdl Load Chip

Files of type: HDL Files Cancel

Implicit Use Of Built-in Chips

```
/** Exclusive-or gate. out = a xor b */
CHIP Xor {
   IN a, b;
   OUT out;
   PARTS:
   Not(in=a,out=Nota);
   Not(in=b,out=Notb);
   And(a=a,b=Notb,out=aNotb);
   And(a=Nota,b=b,out=bNota);
   Or(a=aNotb,b=bNota,out=out);
}
```

- When any HDL file is loaded, the simulator parses its definition. For each internal chip Xxx(...) mentioned in the PARTS section, the simulator looks for an Xxx.hdl file in the same directory (e.g. Not.hdl, And.hdl, and Or.hdl in this example).
- If Xxx.hdl is found in the current directory (e.g. if it was also written by the user), the simulator uses its HDL logic in the evaluation of the overall chip.
- If Xxx.hdl is not found in the current directory, the simulator attempts to invoke the file tools/builtIn/Xxx.hdl instead.
- And since tools/builtIn includes executable versions of all the chips mentioned in the book, it is possible to build and test any of these chips before first building their lower-level parts.

Summary of Built-in Chips

- If you don't implement some chips, you can still use them as chip-parts in other chips (the built-in implementations will kick in)
- Remember a chip cannot be used in its own implementation

What is Script Based Chip Testing

Script-based Simulation

Simulation Options:

- Interactive
- Script Based: A test script is a series of commands to the simulator
  - With/without output file
  - With/without compare file

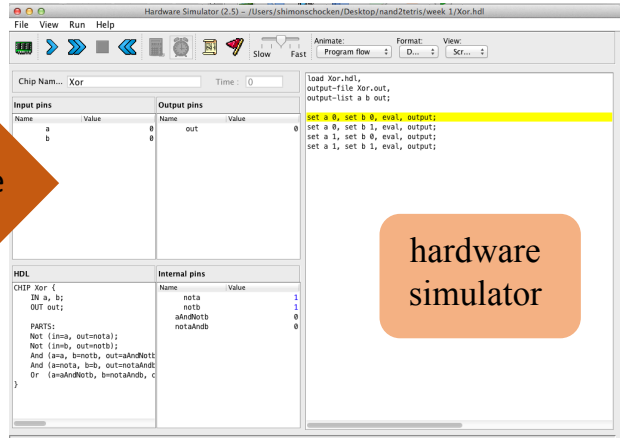
Xor.tst

```
load Xor.hdl;
set a 0, set b 0, eval;
set a 0, set b 1, eval;
set a 1, set b 0, eval;
set a 1, set b 1, eval;
```

Xor.hdl

```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
    Not(in=a, out=nota);
    Not(in=b, out=notb);
    And(a=a, b=notb, out=aAndNotb);
    And(a=nota, b=b, out=notaAndb);
    Or(a=aAndNotb, b=notaAndb, out=out);
```

simulate



Script-base Simulation with an Output File

Xor.hdl

```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
    Not(in=a, out=nota);
    Not(in=b, out=notb);
    And(a=a, b=notb, out=aAndNotb);
    And(a=nota, b=b, out=notaAndb);
    Or(a=aAndNotb, b=notaAndb, out=out);
```

Tested chip

Xor.tst

```
Load Xor.hdl,
output-file Xor.out,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;
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;
```

test script

Xor.out

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

output File, created by the test script as a side-effect of the simulation process

The logic of a typical test script

- Initialize by loading an HDL file
- Can create an empty output file
- List the names of the pins whose values will be written to the output file
- **Set-eval-output** and repeat

Script-base Simulation with Compare File

Xor.hdl

Tested chip

```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
    Not(in=a, out=nota);
    Not(in=b, out=notb);
    And(a=a, b=notb, out=aAndNotb);
    And(a=nota, b=b, out=notaAndb);
    Or(a=aAndNotb, b=notaAndb, out=out);
```

Xor.tst

test script

```
Load Xor.hdl,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;
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;
```

<u>Simulation-with-compare-file logic</u>

- If the script specifies a compare file, when each output command is executed, the outputted line is compared to the corresponding line in the compare file
- If the two lines are not the same, the simulator throws a comparison error

Xor.out

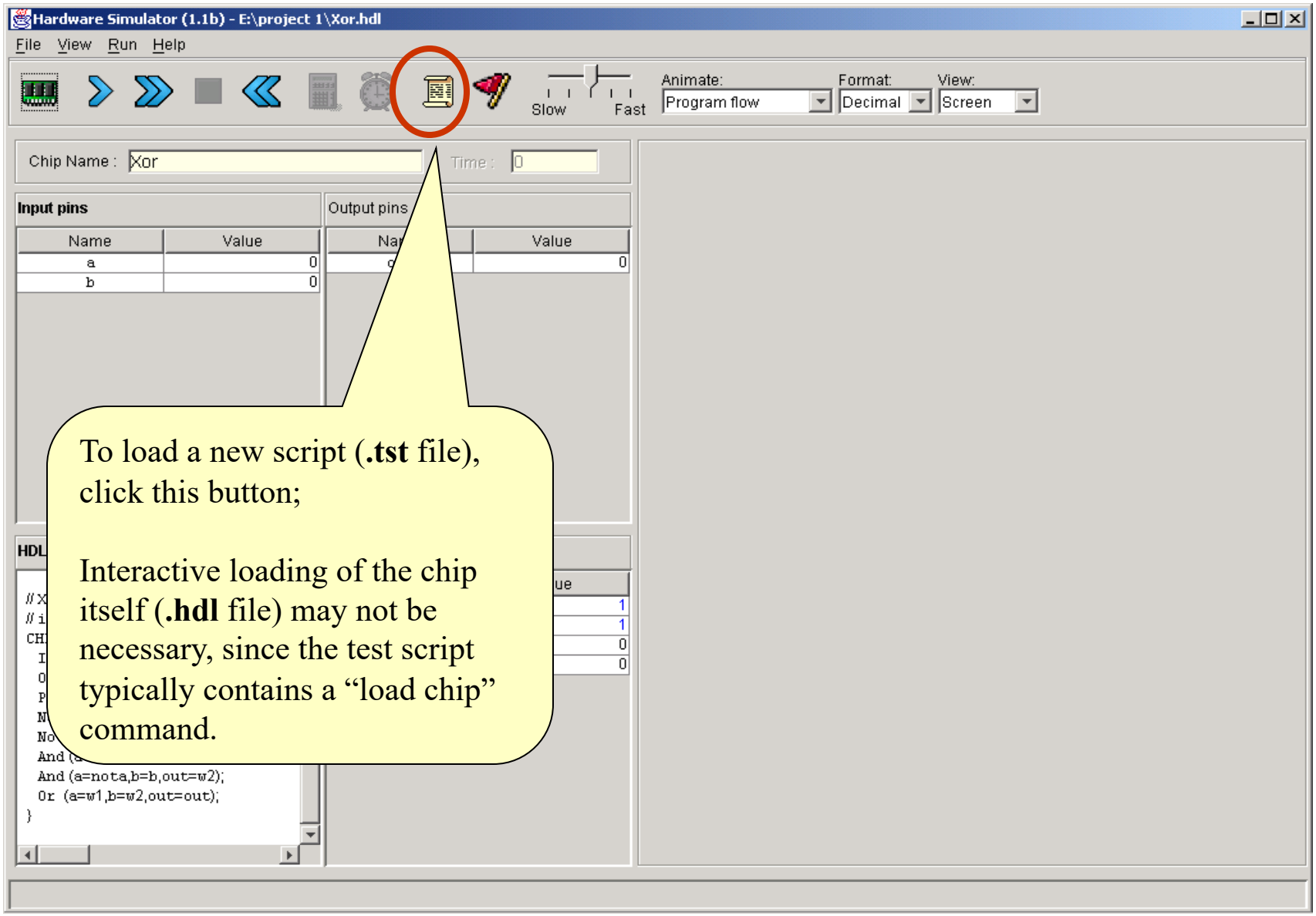
a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Xor.cmp

a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Script Based Chip Testing Demo

Loading A Script



To load a new script (**.tst** file), click this button;

Interactive loading of the chip itself (**.hdl** file) may not be necessary, since the test script typically contains a "load chip" command.

Script Controls

Hardware Simulator (1.1b) - E:\project 1\Xor.hdl

File View Run Help

Slow Fast

Animate: Program flow | Format: Decimal | View: Script

Chip Name : Time : 0

Input pins

Name	Value
a	0
b	0

Output pins

Name	
out	0
w1	0
	0

HDL

```
// Xor (ex  or) gate
// if a<>b  lse out=0
CHIP Xor
  IN a,b;
  OUT out
  PARTS:
  Not (in  ta)
  Not (in  tb);
  And (a=  ut=w1);
}
```

```
load Xor.hdl,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

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;
```

New script loaded: E:\project 1\Xor.tst

Controls the script execution speed

Resets the script

Pauses the script execution

Multi-step execution, until a pause

Executes the next simulation step

Script = series of simulation steps, each ending with a semicolon.

Running A Script

Hardware Simulator (1.1b) - E:\project 1\Xor.hdl

File View Run Help

Slow Fast Animate: Program flow Format: Decimal View: Script

Chip Name : Time : 0

Input pins

Name	Value
a	0
b	0

Output pins

Name	Value
out	0

```
load Xor.hdl,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

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;
```

```
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);
}
```

Typical "init" code:

1. Loads a chip definition (**.hdl**) file
2. Initializes an output (**.out**) file
3. Specifies a compare (**.cmp**) file
4. Declares an output line format.

Script exec-ution flow

New script loaded: E:\project 1\Xor.tst

Running A Script

Hardware Simulator (1.1b) - E:\project 1\Xor.hdl

File View Run Help

Animate: Program flow | Format: Decimal | View: Script

Slow Fast

Chip Name : Xor Time : 0

Input pins

Name	Value
a	1
b	1

Output pins

Name	Value
out	0

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);
}
```

Inter

Value
0
0
0
0

```
load Xor.hdl,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

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;
```

Comparison of the output lines to the lines of the **.cmp** file are reported.

Script exec-ution ends

End of script - Comparison ended successfuly

Viewing Output And Comparing Files

Hardware Simulator (1.1b) - E:\project 1\Xor.hdl

File View Run Help

Slow Fast Animate: Program flow Format: Decima View: Script

Script
Output
Compare
Screen

Chip Name : Xor Time : 0

Input pins

Name	Value
a	1
b	1

Output pins

Name	Value
out	0

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);
}
```

Internal pins

Name	Value
nota	0
notb	0
w1	0
w2	0

```
load Xor.hdl,
output-file Xor.out,
compare-to Xor.cmp,
output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;

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;
```

End of script - Comparison ended successfuly

Viewing Output And Compare Files

Hardware Simulator (1.1b) - E:\project 1\Xor.hdl

File View Run Help

Slow Fast Animate: Program flow Format: Decimal View: Output

Chip Name : Xor Time : 0

Input pins

Name	Value
a	1
b	1

Output pins

Name	Value
out	0

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);
}
```

Internal pins

Name	Value
nota	0
notb	0
w1	0
w2	0

a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Observation:
This output file looks like a **Xor** truth table

Conclusion: the chip logic (**Xor.hdl**) is apparently correct (but not necessarily efficient).

End of script - Comparison ended successfuly

Players Involved in a H/W Construction Project

System Architect:

- Decides which chips are needed, and for each chip the architect creates:
  - A chip API
  - A test script
  - A compare file

Developer:

- The above three files given to the developer provide a convenient specification of
  - The chip interface (.hdl file)
  - What the chip is supposed to do (.cmp file)
  - How to test the chip (.tst file)
- Developer tasks is to implement the chip using these resources