



Hardware Simulator Tutorial

This program is part of the software suite
that accompanies the book

The Elements of Computing Systems

by Noam Nisan and Shimon Schocken

MIT Press

www.idc.ac.il/tecs

This software was developed by students at the
Efi Arazi School of Computer Science at IDC

Chief Software Architect: Yaron Ukrainitz

Background

The hardware simulator is part of the software suite that accompanies the book "The Elements of Computing". The book evolves around the construction of a complete computer system, done in the framework of a 1- or 2-semester course.

In the first part of the book, we build all the logic gates and chips-set of a simple yet powerful computer, called Hack. This is done using HDL (Hardware Description Language) and the hardware simulator described in this tutorial.

In the second part of the book, we build the computer's software hierarchy, consisting of an assembler, a virtual machine, a simple Java-like language called Jack, a compiler for it, and a simple operating system, written in Jack.

The book is completely self-contained, requiring only programming as a pre-requisite.

The book's software suite includes some 200 test programs, test scripts, and all the software tools necessary for doing the projects.



The book's software suite

Simulators

(*HardwareSimulator, CPUEmulator, VMEulator*):

All the tools are dual-platform:

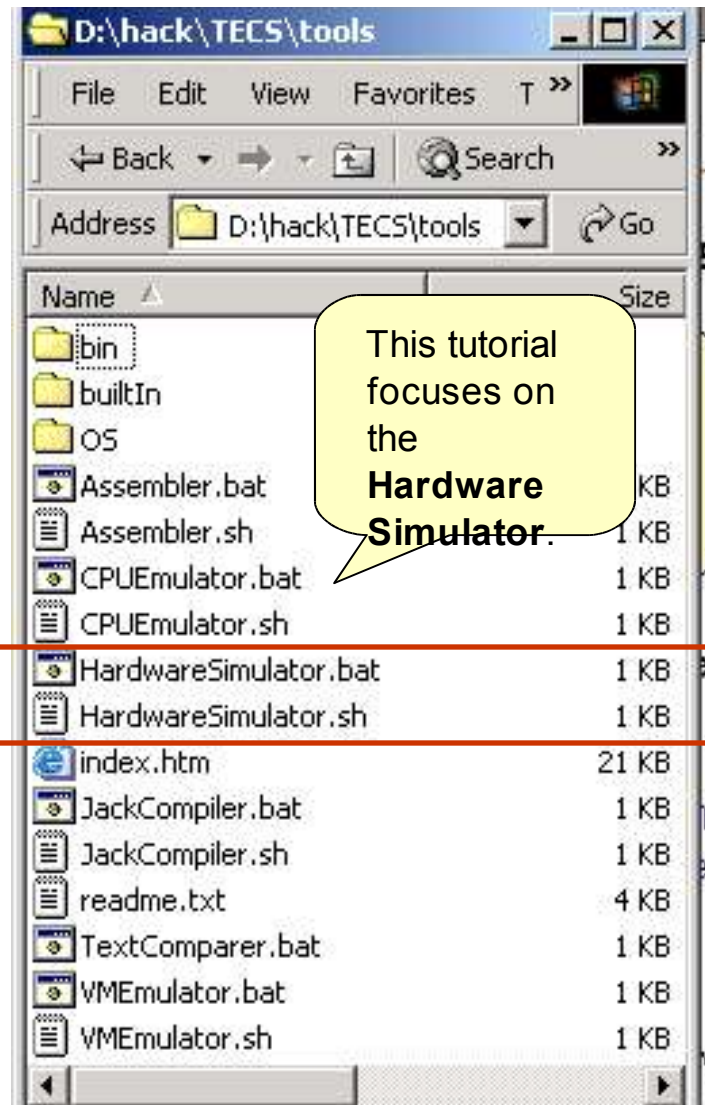
- **Xxx.bat**: starts Xxx in Windows;
- **Xxx.sh**: starts Xxx in Unix.

Translators (*Assembler, JackCompiler*):

- Used to translate from high-level to low-level;
- Developed by the students, using the book's specs; Executable solutions supplied by us.

Other

- **Bin**: simulators and translators software;
- **builtIn**: executable versions of all the logic gates and chips mentioned in the book;
- **OS**: executable version of the Jack OS;
- **TextComparer**: a text comparison utility.



The Hack computer

The **Hardware Simulator** described in this tutorial can be used to build and test many different hardware platforms. In this book, we focus on one particular computer, called Hack.

Hack -- a 16-bit computer equipped with a screen and a keyboard -- resembles hand-held computers like game machines, PDA's, and cellular telephones.

The first 5 chapters of the book specify the elementary gates, combinational chips, sequential chips, and hardware architecture of the Hack computer.

All these modules can be built and tested using the hardware simulator described in this tutorial.

Indeed, that's how hardware engineers build chips for real: first, the hardware is designed, tested, and optimized on a simulator. Only then, the resulting gate logic is committed to silicon.



Hardware Simulation Tutorial

- I. Getting started
- II. Test scripts
- III. Built-in chips
- IV. Clocked chips
- V. GUI-empowered chips
- VI. Debugging tools
- VII. The Hack Platform

Relevant reading (from “*The Elements of Computing Systems*”):

- Chapter 1: *Boolean Logic*
- Appendix A: *Hardware Description Language*
- Appendix B: *Test Scripting Language*



Chip definition (.hdl file)

Chip
interface

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

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

Chip definition (.hdl file)

Chip
interface

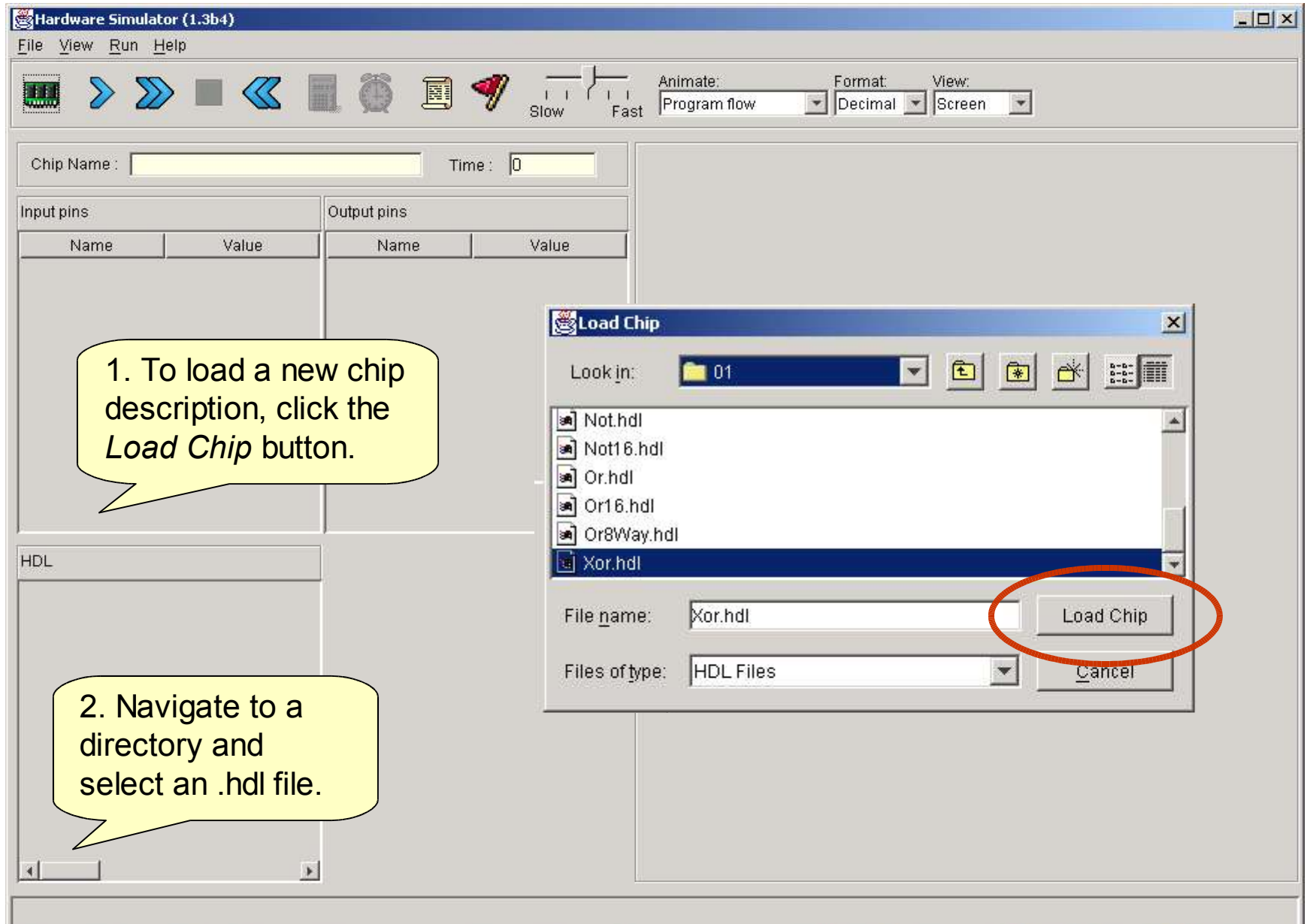
Chip
implementation

```
/** 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=w1);
        And(a=nota, b=b, out=w2);
        Or(a=w1, b=w2, out=out);
}
```

- Any given chip can be implemented in several different ways. This particular implementation is based on: $Xor(a,b) = Or(And(a,Not(b)), And(b,Not(a)))$
- **Not**, **And**, **Or**: *Internal parts* (previously built chips), invoked by the HDL programmer
- **nota**, **notb**, **w1**, **w2**: *internal pins*, created and named by the HDL programmer; used to connect internal parts.

Loading a chip



Loading a chip

Hardware Simulator (1.4b1) - G:\TECS\projects\01\Xor.hdl

File View Run Help

Chip Name: Xor Time: 0

Input pins

Name	Value
a	0
b	0

Output pins

Name	Value
out	0

HDL

```
/**
 * 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=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

- Names and current values of the chip's input pins;
- The pin values may be changed by the user.

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

- 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

The screenshot shows the Hardware Simulator (1.4b1) interface. The title bar indicates the file path is G:\TECS\projects\01\Xor.hdl. The menu bar includes File, View, Run, and Help. The toolbar contains various icons for simulation control, including a chip icon, step-through buttons, a clock icon, and a play button. The 'Animate' dropdown is set to 'Program flow', 'Format' is 'Decimal', and 'View' is 'Screen'. The 'Chip Name' field is 'Xor' and the 'Time' is '0'.

Below the toolbar, there are two tables: 'Input pins' and 'Output pins'.

Name	Value
a	0
b	0

Name	Value
out	0

Below these tables, there is a large empty area for the chip's internal logic diagram.

At the bottom left, the 'HDL' section shows the code for the XOR gate:

```
/**
 * 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=w1);
  And(a=nota, b=b, out=w2);
  Or(a=w1, b=w2, out=out);
}
```

At the bottom right, the 'Internal Parts' table shows the components used in the chip:

Chip Name	Type	Clocked
Not	Composite	<input type="checkbox"/>
Not	Composite	<input type="checkbox"/>
And	Composite	<input type="checkbox"/>
And	Composite	<input type="checkbox"/>
Or	Composite	<input type="checkbox"/>

Two yellow speech bubbles provide instructions:

1. Click the PARTS keyword

2. A table pops up, showing the chip's internal parts (lower-level chips) and whether they are:

- Primitive ("given") or composite (user-defined)
- Clocked (sequential) or unclocked (combinational)

Exploring the chip logic

Hardware Simulator (1.4b1) - G:\TECS\projects\01\Xor.hdl

File View Run Help

Chip Name: Xor Time: 0

Input pins

Name	Value
a	0
b	0

Output pins

Name	Value
out	0

HDL

```
/**
 * 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=w1);
  And(a=nota, b=b, out=w2);
  Or(a=w1, b=w2, out=out);
}
```

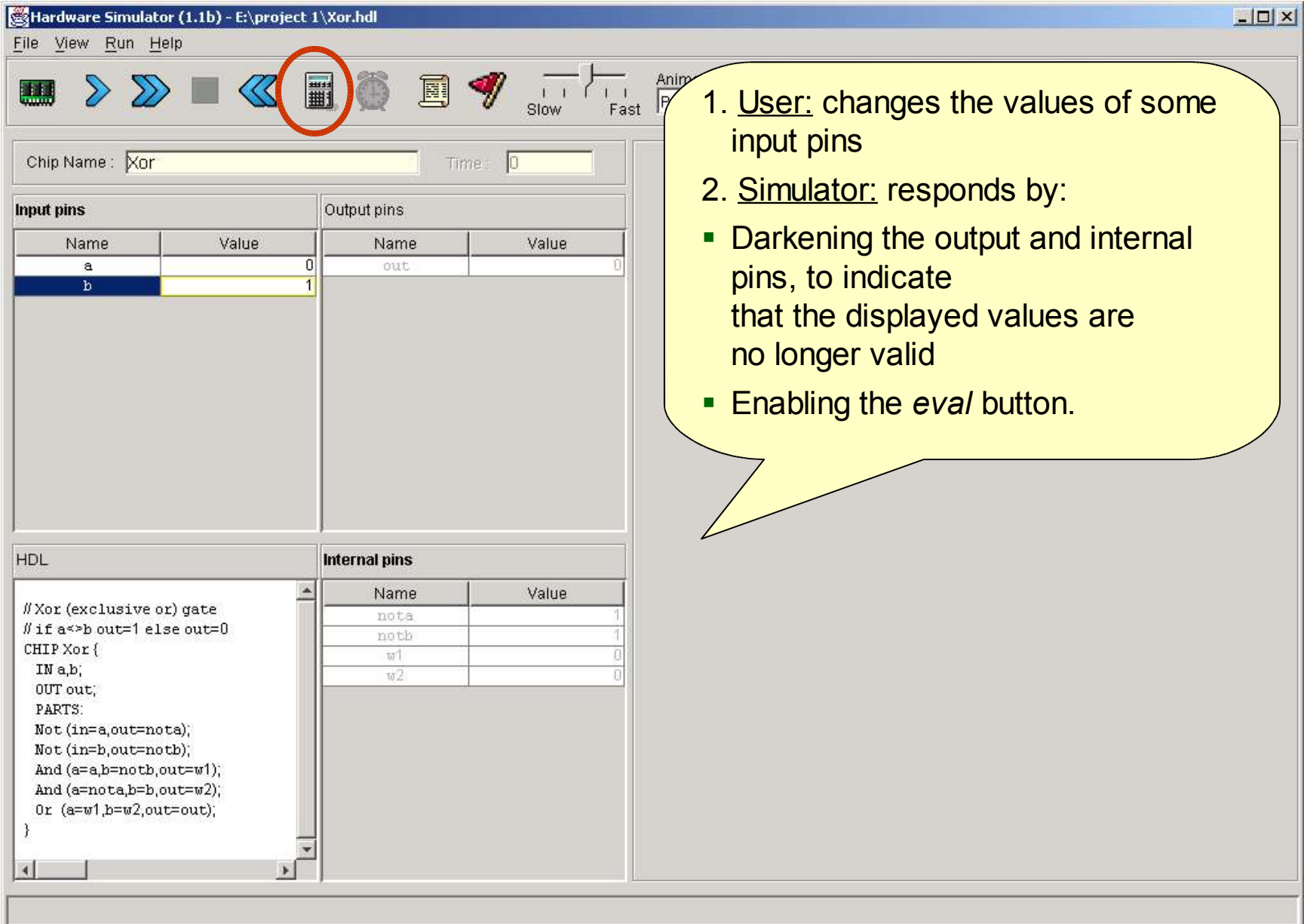
Part pins

Part pin	Gate pin	Value
in	a	0
out	nota	0

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



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

File View Run Help

Chip Name: Xor Time: 0

Input pins

Name	Value
a	0
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	1
notb	1
w1	0
w2	0

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* button.

Interactive chip testing

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

File View Run Help

Chip Name: Time:

Input pins		Output pins	
Name	Value	Name	Value
a	0	out	1
b	1		

Re-calc

Internal pins	
Name	Value
nota	1
notb	0
w1	0
w2	1

```
// 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);
}
```

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* button.
3. User: Clicked the *eval* button
4. Simulator: re-calculates the values of the chip's internal and output pins (done by applying 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.



Test scripts

```
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;  
Etc
```

Generated
output file
(Xor.out)

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

Test scripts:

- Are used for specifying, automating and replicating chip testing
- Are supplied for every chip mentioned in the book
- Can effect, batch-style, any operation that can be done interactively
- Are written in a simple language described in Appendix B of the book
- Can create an output file that records the results of the chip test
- If the script specifies a compare file, the simulator will compare the .out file to the .cmp file, line by line.

Loading a test script

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

File View Run Help

Chip Name: Xor Time: 0

Input pins

Name	Value
a	0
b	0

Output pins

Name	Value
out	0

Animate: Program flow Format: Decimal View: Screen

Slow Fast

To load a script (.tst file), click the *Load Script* button;

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

HDL

```
// Xor
// in
CHI
IN
OU
PA
Not
Not (in=b,ou
And (a=a,b
And (a=n
Or (a=
out=out);
}
```

Script controls

The screenshot shows the Hardware Simulator (1.1b) interface. The title bar reads "Hardware Simulator (1.1b) - E:\project 1\Xor.hdl". The menu bar includes File, View, Run, and Help. The toolbar contains icons for loading, running, pausing, and stepping through the script, along with speed controls (Slow, Fast) and dropdown menus for Animate (Program flow), Format (Decimal), and View (Script).

Annotations with callouts explain the following controls:

- Controls the script execution speed:** Points to the Speed slider in the toolbar.
- Resets the script:** Points to the "Reset" button (circular arrow icon) in the toolbar.
- Pauses the script execution:** Points to the "Pause" button (square icon) in the toolbar.
- Executes the entire script, until a pause:** Points to the "Run" button (play icon) in the toolbar.
- Executes the next simulation step:** Points to the "Step" button (right arrow icon) in the toolbar.

The interface also displays the following components:

- Chip Name:** A text field.
- Time:** A display showing "0".
- Input pins table:**

Name	Value
a	0
b	0
- Output pins table:**

Name	Value
out	0
- HDL Editor:** Contains the following code:

```
// 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 (in=nota,out=and1);
    And (in=notb,out=and2);
    Or (in=and1,out=out);
    Or (in=and2,out=out);
}
```
- Script Editor:** Contains the following 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;
```
- Bottom Status Bar:** Displays "New script loaded: E:\project 1\Xor.tst".

Running a script

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

File View Run Help

Chip Name: Time: 0

Input pins

Name	Value
a	0
b	0

Output pins

Name	Value
out	0

Typical script's "init" code:

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

```
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;
```

Script execution flow

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

Running a script

The screenshot shows the Hardware Simulator (1.1b) interface. The title bar indicates the file is `E:\project 1\Xor.hdl`. The menu bar includes `File`, `View`, `Run`, and `Help`. The toolbar contains various icons for simulation control, including a green chip, navigation arrows, a clock, and a red flag. The `Animate` dropdown is set to `Program flow`, `Format` is `Decimal`, and `View` is `Script`. The `Chip Name` is `Xor` and `Time` is `0`.

The `Input pins` table shows:

Name	Value
a	1
b	1

The `Output pins` table shows:

Name	Value
out	0

A yellow speech bubble contains the text: "When the script execution ends, the simulator reports the comparison results between the output file and the compare file."

The `HDL` section shows the following code:

```
// 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);
}
```

The `Internal pins` table shows:

Name	Value
nota	0
notb	0
w1	0
w2	0

The `Script` section shows the following code:

```
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;
```

The last line of the script, `output;`, is highlighted in yellow. An orange arrow points to this line with the text "Script execution ends".

The status bar at the bottom shows: "End of script - Comparison ended successfully".

Viewing output and compare files

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

File View Run Help

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

View: Script, Output, Compare, Screen

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;

To view the generated output (.out) file or the compare (.cmp) file, choose "output" or "compare" from the View button.

End of script - Comparison ended successfully

Viewing output and compare files

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

File View Run Help

Animate: Program flow Format: Decimal View: Output

Chip Name: Xor Time: 0

Input pins		Output pins	
Name	Value	Name	Value
a	1	out	0
b	1		

HDL			Internal pins	
			Name	Value
<pre>// 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); }</pre>			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 successfully



Built-In chips

General

- A built-in chip has an HDL interface and a Java implementation (e.g. `Mux16.class`)
- The name of the Java class is specified following the `BUILTIN` keyword
- Built-In versions of all the chips that appear in the book are supplied with the HW simulator, and stored in the `tools/builtIn` directory.

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

Built-in chips are used to:

- Implement primitive gates (in the computer built in the book: `Nand` and `DFF`)
- Implement chips that have peripheral side effects (like I/O devices)
- Implement chips that feature a GUI (for debugging)
- Replace (automatically) chips that the user didn't implement for some reason
- Improve simulation speed and save memory (when used as parts in complex chips)
- Facilitate behavioral simulation of a chip prior to actually building it in HDL
- Built-in chips can be used either explicitly or implicitly, as we now turn to show.

Explicit use of built-in chips

The screenshot shows the Hardware Simulator (1.4b1) interface. The title bar indicates the file path is G:\TECS\tools\builtin\Mux16.hdl. The menu bar includes File, View, Run, and Help. The toolbar contains various icons, with the first icon (a chip) circled in red. The main window displays the chip name 'Mux16' and a time of 0. Below this, there are tables for input and output pins. The input pins table has columns for Name and Value, with rows for a[16], b[16], and sel, all with a value of 0. The output pins table has columns for Name and Value, with a row for out[16] with a value of 0. A yellow speech bubble points to the input pins table, stating: 'Built-in chip logic. Can be used as-is or as a part in other chips, without having to implement it in HDL.' Another yellow speech bubble points to the output pins table, stating: 'Executable versions of all the chips mentioned in the book are stored in the **tools/builtIn** directory.' In the bottom left, the HDL code is visible, showing the chip definition for Mux16. A 'Load Chip' dialog box is open, showing the 'builtin' directory. The file list includes HalfAdder.hdl, Inc16.hdl, Keyboard.hdl, Mux.hdl, Mux16.hdl (selected), and Mux4Way16.hdl. The 'File name' field is set to 'Mux16.hdl' and the 'Files of type' is set to 'HDL Files'. The 'Load Chip' button is circled in red.

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

File View Run Help

Chip Name: Mux16 Time: 0

Input pins		Output pins	
Name	Value	Name	Value
a[16]	0	out[16]	0
b[16]	0		
sel	0		

Executable versions of all the chips mentioned in the book are stored in the **tools/builtIn** directory.

Built-in chip logic. Can be used as-is or as a part in other chips, without having to implement it in HDL.

HDL

```
// MIT Press 2004. File name: http://www
// 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;
}
```

Look in: builtin

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

File name: Mux16.hdl

Files of type: HDL Files

Load Chip

Cancel

Implicit use of built-in chips

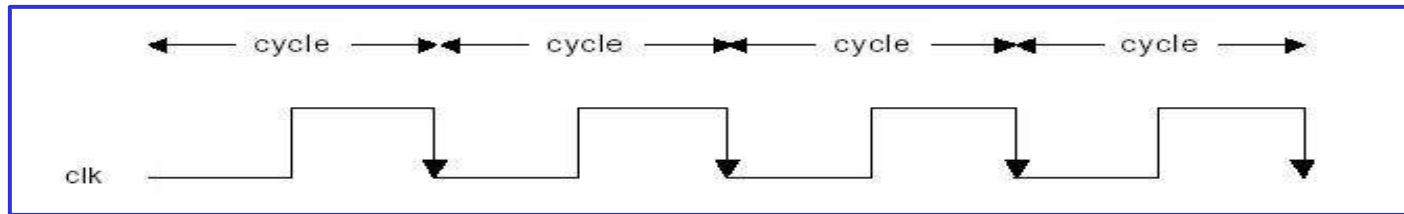
```
/** Exclusive-or gate. out = a xor b */  
CHIP Xor {  
    IN a, b;  
    OUT out;  
    PARTS:  
        Not(in=a,out=Nota) ;  
        Not(in=b,out=Notb) ;  
        And(a=a,b=Notb,out=aNotb) ;  
        And(a=Nota,b=b,out=bNota) ;  
        Or(a=aNotb,b=bNota,out=out) ;  
}
```

- When an HDL file is loaded (e.g. `Xor.hdl`), the simulator parses its definition. For each internal chip `Xxx(...)` mentioned in the PARTS section, the simulator looks for a `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 (presumably, 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 chips before first building their lower-level parts.



Clocked (sequential) chips

- The implementation of clocked chips is based on *sequential logic*
- The operation of clocked chips is regulated by the computer's internal clock:



- In our jargon, a clock cycle = *tick*-phase (down), followed by a *tock*-phase (up)
- During a *tick-tock*, the internal states of all the clocked chips are allowed to change, but their outputs are “latched”
- At the beginning of the next *tick*, the outputs of all the clocked chips in the architecture commit to the new values
- In a real computer, the clock is implemented by an oscillator; in simulators, clock cycles can be simulated either manually by the user, or repeatedly by a test script.

The D-Flip-Flop (DFF) gate

```
/** Data Flip-flop:
 * out(t)=in(t-1)
 * where t is the time unit.
 */
CHIP DFF {
    IN in;
    OUT out;

    BUILTIN DFF;
    CLOCKED in, out;
}
```

DFF:

- A primitive memory gate that can “remember” a state over clock cycles
- Can serve as the basic building block of all the clocked chips in a computer.

Clocked chips

- Clocked chips include registers, RAM devices, counters, and the CPU
- The simulator knows that the loaded chip is clocked when one or more of its pins is declared “clocked”, or one or more of its parts (or sub-parts, recursively) is a clocked chip
- In the hardware platform built in the book, all the clocked chips are based, directly or indirectly, on (many instances of) built-in DFF gates.

Simulating clocked chips

Hardware Simulator (1.4b1) - G:\TECS\tools\builtIn\RAM8.hdl

File View Run Help

Chip Name : RAM8 (Clocked)

Input pins

Name	Value
in[16]	
load	
address[3]	

HDL

```
* In words: the chip always outputs  
* location specified by address. If  
* into the memory location specifi  
* available through the out output  
*  
CHIP RAM8 {  
  
  IN in[16], load, address[3];  
  OUT out[16];  
  
  BUILTIN RAM8;  
  CLOCKED in, load;  
}
```

Clocked (sequential) chips are clock-regulated.

Therefore, the standard way to test a clocked chip is to set its input pins to some values (as with combinational chips), simulate the progression of the clock, and watch how the chip logic responds to the ticks and the tocks.

For example, let us consider the simulation of an 8-word random-access memory chip (RAM8).

A built-in, clocked chip (**RAM8**) is loaded

happens to be GUI-empowered, the simulator displays its GUI (More about GUI-empowered chips, soon)

Simulating clocked chips

Hardware Simulator (1.4b1) - G:\TECS\tools\builtIn\RAM8.hdl

File View Run Help

Chip Name: Time: 0

Input pins

Name	Value
in[16]	112
load	1
address[3]	5

Output pins

Name	Value
out[16]	0

RAM 8:

0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0

HDL

```
* In words: the chip always outputs  
* location specified by address. It  
* into the memory location specified  
* available through the out output  
*  
CHIP RAM8 {  
  
  IN in[16], load, address[3];  
  OUT out[16];  
  
  BUILTIN RAM8;  
  CLOCKED in, load;  
}
```

1. User: enters some input values and clicks the clock icon once (*tick*)

A built-in, clocked chip (**RAM8**) is loaded

Simulating clocked chips

Hardware Simulator (1.4b1) - G:\TECS\tools\builtIn\RAM8.hdl

File View Run Help

Chip Name: Time: 0+

Input pins

Name	Value
in[16]	112
load	1
address[3]	5

Output pins

Name	Value
out[16]	0

RAM 8:

Address	Value
0	0
1	0
2	0
3	0
4	0
5	112
6	0
7	0

HDL

```
* In words: the chip always outputs  
* location specified by address. It  
* into the memory location specified  
* available through the out output  
*  
CHIP RAM8 {  
  
  IN in[16], load, address[3];  
  OUT out[16];  
  
  BUILTIN RAM8;  
  CLOCKED in, load;  
}
```

1. User: enters some input values and clicks the clock icon once (*tick*)

2. Simulator: changes the internal state of the chip, but note that the chip's output pin is not yet effected.

A built-in, clocked chip (**RAM8**) is loaded

Simulating clocked chips

Hardware Simulator (1.3b1) - D:\hack\tools\BuiltIn\RAM8.hdl

File View Run Help

Chip Name: RAM8 (Clocked) Time: 0+

Input pins

Name	Value
in[16]	112
load	1
address[3]	5

Output pins

Name	Value
out[16]	0

RAM 8:

0	0
1	0
2	0
3	0
4	0
5	112
6	0
7	0

HDL

```
/**
 * Memory of 8 registers, each 16 bits
 * stored at the memory location specified by address
 * memory location specified by address
 * from the next time step.
 */

CHIP RAM8 {
    IN in[16], load, address[3];
    OUT out[16];

    BUILTIN RAM8;
```

Annotations:

- 1. User: enters some input values and clicks the clock icon once (*tick*)
- 2. Simulator: changes the internal state of the chip, but note that the chip's output pin is not yet effected.
- 3. User: clicks the clock icon again (*tock*)
- A built-in, clocked chip (**RAM8**) is loaded

Simulating clocked chips

Hardware Simulator (1.4b1) - G:\TECS\tools\builtIn\RAM8.hdl

File View Run Help

Chip Name: Time: 1

Input pins

Name	Value
in[16]	112
load	1
address[3]	5

Output pins

Name	Value
out[16]	112

RAM 8:

0	0
1	0
2	0
3	0
4	0
5	112
6	0
7	0

HDL

```
* In words: the chip always outputs  
* location specified by address. It  
* into the memory location specified  
* available through the out output  
*  
CHIP RAM8 {  
  
  IN in[16], load, address[3];  
  OUT out[16];  
  
  BUILTIN RAM8;  
  CLOCKED in, load;  
}
```

3. User: clicks the clock icon again (*tock*)

1. User: enters some input values and clicks the clock icon once (*tick*)

4. Simulator: commits the chip's output pin to the value of the chip's internal state.

2. Simulator: changes the internal state of the chip, but note that the chip's output pin is not yet effected.

A built-in, clocked chip (**RAM8**) is loaded

Simulating clocked chips using a test script

Hardware Simulator (1.1b) - D:\hack\tools\BuiltIn\RAM8.hdl

File View Run Help

Animate: Program flow Format: Decimal View: Script

Chip Name: RAM8 (Clocked) Time: 1

Input pins

Name	Value
in[16]	112
load	1
address[3]	5

Output pins

Controls the script speed, and thus the simulated clock speed, and thus the overall chip execution speed

Single-action tick-tock

Tick-tocks repeatedly and infinitely

Repeat{
Tick;
Tock;
}

Default script: always loaded when the simulator starts running;

The logic of the default script simply runs the clock repeatedly;

Hence, executing the default script has the effect of causing the clock to go through an infinite train of tics and tocks.

This, in turn, causes all the clocked chip parts of the loaded chip to respond to clock cycles, repeatedly.

HDL

```
// 8-registers memory  
CHIP RAM8 {  
    IN in[16], load, address[3];  
    OUT out[16];  
  
    BUILTIN RAM8;  
  
    CLOCKED in, load;  
}
```



Built-in chips with GUI effects

Hardware Simulator (1.1b) - E:\GUIDemo.hdl

File View Run Help

Animate: Program flow Format: Decimal View: Screen

Chip Name: GUIDemo (Clocked) Time: 0

Input pins		Output pins	
Name	Value	Name	Value
in[16]	0	out[16]	0
load	0		
address[15]	0		

HDL

```
// demo GUI-empowered chips
CHIP GUIDemo {
  IN in[16], load, address[15];
  OUT out[16];

  PARTS:
    RAM16K(in=in, load=load,
      address=address[0..13],
      out=a);
    Screen(in=in, load=load,
      address=address[0..12],
      out=b);
    Keyboard(out=c);
}
```

Internal pins

Name	Value
	0
	0
	0

1. A chip whose parts include **built-in chips** was loaded into the simulator (ignore the chip logic for now)

Note: the signature of the internal part does not reveal if the part is implemented by a built-in chip or by another chip built by the user. Thus in this example you have to believe us that all the parts of this loaded chip are built-in chips.

Built-in chips with GUI effects

The screenshot shows the Hardware Simulator (1.1b) interface. The title bar reads "Hardware Simulator (1.1b) - E:\GUIDemo.hdl". The menu bar includes File, View, Run, and Help. The toolbar contains icons for simulation control (play, pause, stop, reset, etc.) and a dropdown menu for "Animate:" set to "Program flow". The "Format:" dropdown is set to "Decimal" and the "View:" dropdown is set to "Screen".

The main window is divided into several sections:

- Chip Name:** GUIDemo (Clocked) | **Time:** 0
- Input pins:** A table with columns "Name" and "Value".

Name	Value
in[16]	0
load	
address[15]	
- HDL:** A text editor showing the chip definition:

```
// demo GUI-empowered chips
CHIP GUIDemo {
  IN in[16], load, address[15];
  OUT out[16];

  PARTS:
    RAM16K(in=in, load=load,
      address=address[0..13],
      out=a);
    Screen(in=in, load=load,
      address=address[0..12],
      out=b);
    Keyboard(out=c);
}
```
- Internal pins:** A table with columns "Name" and "Value".

Name	Value
a	0
b	0
c	0
- RAM 16K:** A table showing memory contents.

Address	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0

Annotations and callouts:

- Yellow callout (2):** "2. If the loaded chip or some of its parts have GUI side-effects, the simulator displays the GUI's here." (Points to the input pins section)
- Yellow callout (1):** "1. A chip whose parts include built-in chips was loaded into the simulator (ignore the chip logic for now)" (Points to the HDL section)
- Orange callout:** "For each GUI-empowered built-in chip that appears in the definition of the loaded chip, the simulator does its best to put the chip GUI in this area." (Points to the right side of the simulator window)
- Orange callout:** "The actual GUI's behaviors are then effected by the Java classes that implement the built-in chips." (Points to the right side of the simulator window)
- Orange callout:** "GUI of the built-in RAM16K.hdl chip" (Points to the RAM 16K table)
- Orange callout:** "GUI of the built-in Keyboard.hdl chip" (Points to the Keyboard section)

The logic of the GUIDemo chip

```
// Demo of built-in chips with GUI effects
CHIP GUIDemo {
    IN in[16], load, address[15];
    OUT out[16];
    PARTS:
        RAM16K(in=in, load=load, address=address[0..13], out=null);
        Screen(in=in, load=load, address=address[0..12], out=null);
        Keyboard(out=null);
}
```

RAM16K,
Screen, &
Keyboard
are built-in
chips with GUI
side-effects

- **Effect:** When the simulator evaluates this chip, it displays the GUI side-effects of all the built-in chip parts
- **Chip logic:** The only purpose of this demo chip is to force the simulator to show the GUI of some built-in chips. Other than that, the chip logic is meaningless: it simultaneously feeds the 16-bit data input (**in**) into the **RAM16K** and the **Screen** chips, and it does nothing with the keyboard.

Built-in chips with GUI effects

The screenshot shows the Hardware Simulator (1.1b) interface. The title bar reads "Hardware Simulator (1.1b) - E:\GUIDemo.hdl". The menu bar includes File, View, Run, and Help. The toolbar contains icons for loading, running, pausing, and stepping through code, with a clock icon circled in red. Below the toolbar, the "Chip Name" is set to "GUIDemo (Clocked)" and the "Time" is "5+". The "Input pins" table is circled in red and contains the following data:

Name	Value
in[16]	-1
load	1
address[15]	5012

The "Output pins" table shows "out[16]" with a value of 0. The HDL code in the bottom-left pane is as follows:

```
// demo GUI-empowered chips
CHIP GUIDemo {
  IN in[16], load, address[15];
  OUT out[16];

  PARTS:
    RAM16K(in=in, load=load,
      address=address[0..13],
      out=a);
    Screen(in=in, load=load,
      address=address[0..12],
      out=b);
    Keyboard(out=c);
}
```

Three callouts provide additional information:

- 1. User enters:**
 - in = -1 (=16 1's in binary)
 - address = 5012
 - load = 1
- 2. User:** runs the clock
- 3. 16 black pixels are drawn beginning row=156 col=320**

A red callout box states: **A simulated 256 rows by 512 columns screen**.

An explanation callout states: **Explanation:** According to the specification of the computer architecture described in the book, the pixels of the physical screen are continuously refreshed from an 8K RAM-resident memory map implemented by the `Screen.hdl` chip. The exact mapping between this memory chip and the actual pixels is specified in Chapter 5. The refresh process is carried out by the simulator.



System variables

The simulator recognizes and maintains the following variables:

- Time: the number of time-units (clock-cycles) that elapsed since the script started running is stored in the variable `time`
- Pins: the values of all the input, output, and internal pins of the simulated chip are accessible as variables, using the names of the pins in the HDL code
- GUI elements: the values stored in the states of GUI-empowered built-in chips can be accessed via variables. For example, the value of register 3 of the `RAM8` chip can be accessed via `RAM8 [3]`.

All these variables can be used in scripts and *breakpoints*, for debugging.

Breakpoints

The screenshot shows the Hardware Simulator (1.1b) interface. The title bar reads "Hardware Simulator (1.1b) - D:\hack\tools\BuiltIn\RAM8.hdl". The menu bar includes File, View, Run, and Help. The toolbar contains various icons, with a red circle highlighting the breakpoint icon (a red flag). The main window is divided into several panels:

- Input pins:** A table with columns Name and Value. It lists in[16], load, and address[3].
- Output pins:** A table with columns Name and Value. It lists out[16] with a value of 112.
- Breakpoint Panel:** A floating window with a red border. It contains a table with columns Variable Name and Value, listing Time (15), In (1024), and RAM8[3] (172). At the bottom of this panel, there are three buttons: a plus sign (+), a minus sign (-), and a checkmark (✓), which are also circled in red.
- RAM 8:** A memory view showing a list of values. The value 112 is highlighted in yellow.
- HDL:** A text editor showing the code for the 8-registers memory chip.

Annotations and instructions are provided in yellow speech bubbles:

1. Open the breakpoints panel
2. Previously-declared breakpoints
3. To update an existing breakpoint, double-click it
3. Add, delete, or update breakpoints

The breakpoints logic:

- Breakpoint = variable name and declared value
- When the specified variable in some breakpoint reaches its specified value, the script pauses and a message is displayed
- A powerful debugging tool.

Test scripts of complex chips

```
load Computer.hdl
ROM32K load Max.hack,
output-file ComputerMax.out,
compare-to ComputerMax.cmp,
output-list time%S1.4.1
    reset%B2.1.2
    ARegister[]%D1.7.1
    DRegister[]%D1.7.1
    PC[]%D0.4.0
    RAM16K[0]%D1.7.1
    RAM16K[1]%D1.7.1
    RAM16K[2]%D1.7.1;
breakpoint PC 10;
// First run: compute max(3,5)
set RAM16K[0] 3,
set RAM16K[1] 5,
output;
repeat 14 {
    tick, tock, output;
}
// Reset the PC (preparing for
// second run)
set reset 1,
tick, tock, output;
// Etc.
```

- Test scripts are normally written and supplied by the hardware platform architects
- All the chips that you have to develop in the course have supplied test scripts
- Scripts that test the CPU chip or the computer chip described in the book usually start by loading a machine-language program (.asm or .hack file) into the ROM chip
- The rest of the script typically uses various features like:
 - Output files
 - Loops
 - Breakpoints
 - Variables manipulation
 - tick, tock
 - Etc.
- All these features are described in Appendix B of the book (*Test Scripting Language*).

Visual options

The screenshot shows the Hardware Simulator (1.1b) window. The title bar indicates the file path is D:\hack\tools\BuiltIn\RAM8.hdl. The menu bar includes File, View, Run, and Help. The toolbar contains icons for simulation control (stop, single step, multi step, back, forward, clock, script, and a red arrow) and speed settings (Slow and Fast). The 'Animate' dropdown is set to 'Program flow', 'Format' is 'Decimal', and 'View' is 'Screen'. The 'Chip Name' field shows 'RAM8 (Clocked)' and the 'Time' field shows '1'. Below these are tables for 'Input pins' and 'Output pins'. The 'Input pins' table has columns 'Name' and 'Value', with rows for 'in[16]' (value 112), 'load', and 'address'. The 'Output pins' table has columns 'Name' and 'Value', with a row for 'out[16]' (value 112). To the right, a 'RAM 8:' table shows memory addresses 0 to 4, all with a value of 0. On the left, the 'HDL' section shows code for an 8-register chip.

Chip Name: RAM8 (Clocked) Time: 1

Input pins

Name	Value
in[16]	112
load	
address	

Output pins

Name	Value
out[16]	112

RAM 8:

0	0
1	0
2	0
3	0
4	0

HDL

```
// 8-regis  
CHIP RAM8  
IN in[16]  
OUT out[16]  
BUILT-IN  
CLOCKED  
{
```

- **Program flow:** animates the flow of the currently loaded program
- **Program & data flow:** animates the flow of the current program and the data flow throughout the GUI elements displayed on the screen
- **No animation (default):** program and data flow are not animated.
- **Tip:** When running programs, any animation effects slow down the simulation considerably.

Format of displayed pin values:

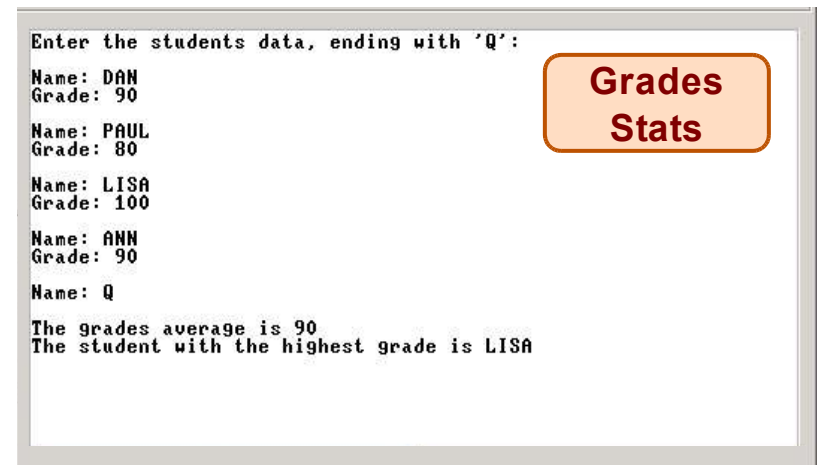
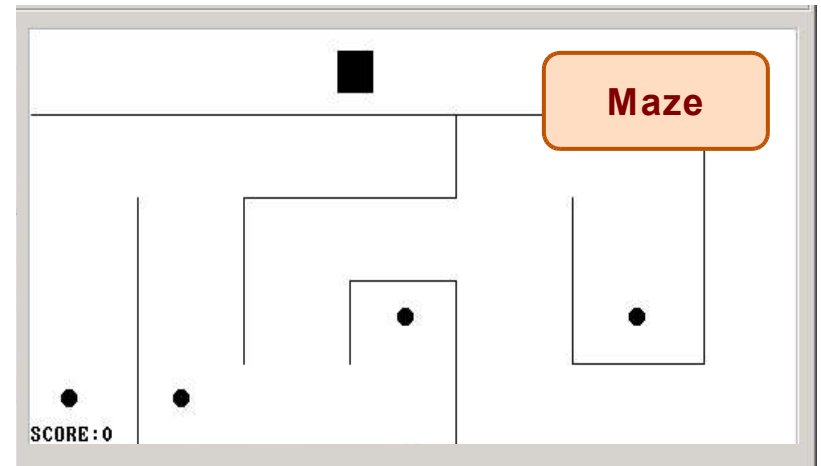
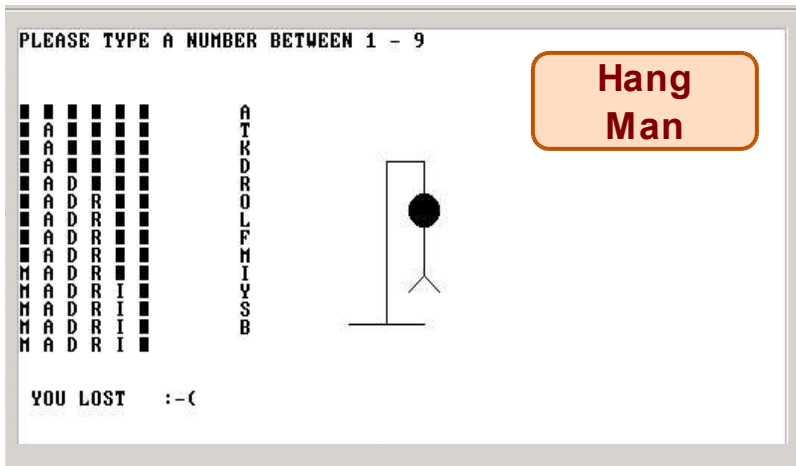
- **Decimal** (default)
- **Hexadecimal**
- **Binary**

- **Script:** displays the current test script
- **Output:** displays the generated output file
- **Compare:** displays the supplied comparison file
- **Screen:** displays the GUI effects of built-in chips, if any.



Hack is a general-purpose 16-bit computer

Sample applications running on the Hack computer:



These programs (and many more) were written in the Jack programming language, running in the Jack OS environment over the Hack hardware platform. The hardware platform is built in Chapters 1-5, and the software hierarchy in Chapters 6-12.

The Hack chip-set and hardware platform

Elementary logic gates

(Project 1):

- Nand (primitive)
- Not
- And
- Or
- Xor
- Mux
- Dmux
- Not16
- And16
- Or16
- Mux16
- Or8Way
- Mux4Way16
- Mux8Way16
- DMux4Way
- DMux8Way

Combinational chips

(Project 2):

- HalfAdder
- FullAdder
- Add16
- Inc16
- ALU

Sequential chips

(Project 3):

- DFF (primitive)
- Bit
- Register
- RAM8
- RAM64
- RAM512
- RAM4K
- RAM16K
- PC

Computer Architecture

(Project 5):

- Memory
- CPU
- Computer

Most of these chips are generic, meaning that you need them for the construction of any computer.

The Hack chip-set and hardware platform can be built using the HW simulator in recursive ascent, starting with primitive Nand.hdl and DFF.hdl gates and culminating in the Computer.hdl chip.

This construction is described in chapters 1,2,3,5 of the book, and carried out in the respective projects.

I was surprised to find that the chips were covered with such combatants, that it was not a duellum, but a bellum, a war between two races of ants, the red always pitted against the black, and frequently two red ones to one black. The legions of these Myrmidons covered all the hills and vales in my wood-yard, and the ground was already strewn with the dead and dying, both red and black.



It was the only battle which I have ever witnessed, the only battlefield I ever trod while the battle was raging; internecine war; the red republicans on the one hand, and the black imperialists on the other. On every side they were engaged in deadly combat, yet without any noise that I could hear, and human soldiers never fought so resolutely.... The more you think of it, the less the difference. And certainly there is not the fight recorded in Concord history, at least, if in the history of America, that will bear a moment's comparison with this, whether for the numbers engaged in it, or for the patriotism and heroism displayed.

From "Brute Neighbors," Walden (1854).