



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.nand2tetris.org

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

Chief Software Architect: Yaron Ukrainitz

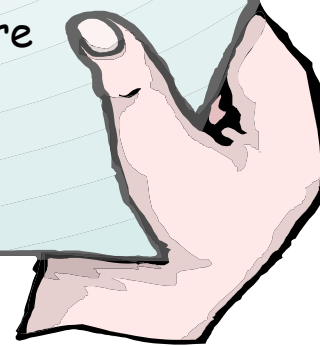
Background

The Elements of Computing Systems evolves around the construction of a complete computer system, done in the framework of a 1- or 2-semester course.

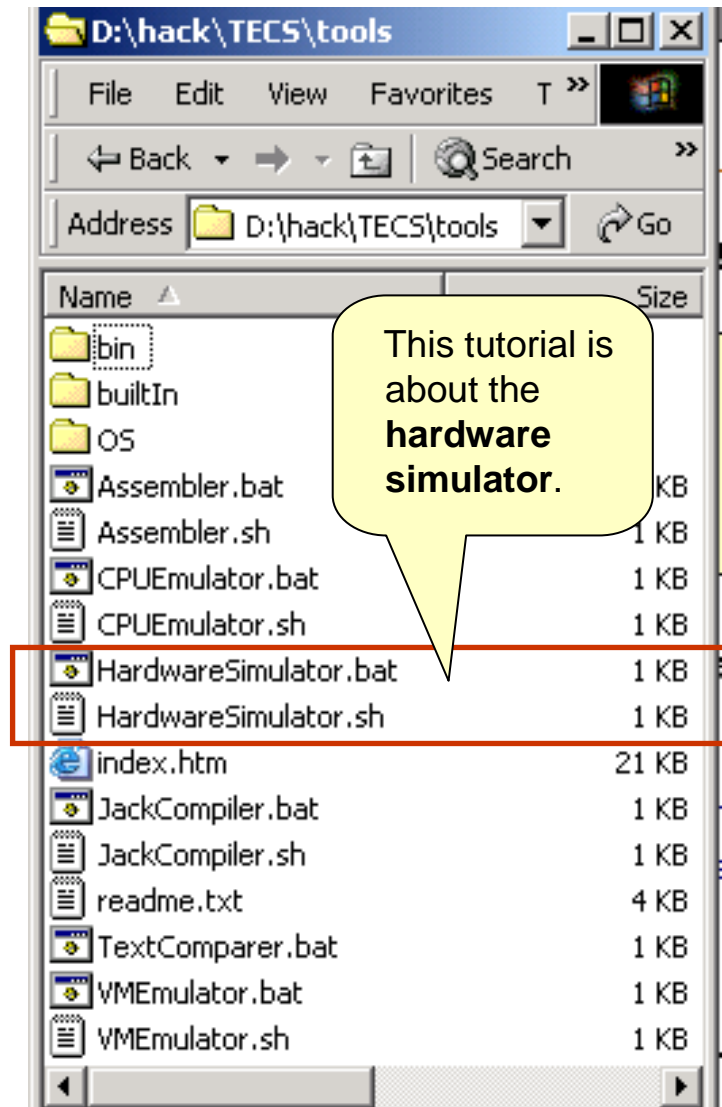
In the first part of the book/course, we build the hardware platform of a simple yet powerful computer, called Hack. In the second part, we build the computer's software hierarchy, consisting of an assembler, a virtual machine, a simple Java-like language called Jack, a compiler for it, and a mini operating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

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

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



The book's software suite



(All the supplied tools are dual-platform: **xxx.bat** starts **xxx** in Windows, and **xxx.sh** starts it in Unix)

Simulators

(**HardwareSimulator**, CPUEmulator, VMEulator):

- Used to build hardware platforms and execute programs;
- Supplied by us.

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.

That is how hardware engineers build chips for real: first, the hardware is designed, tested, and optimized on a software 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*

Hardware Simulation Tutorial



Chip Definition (.hdl file)

chip
interface

ovdje se kuca logika,
npr ako opisujemo AND kolo
koje je dio šeme, ide:
And(a=a, b=b, out=and_out);
i dalje možemo koristiti
and_out varijablu... :)

```
/** 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

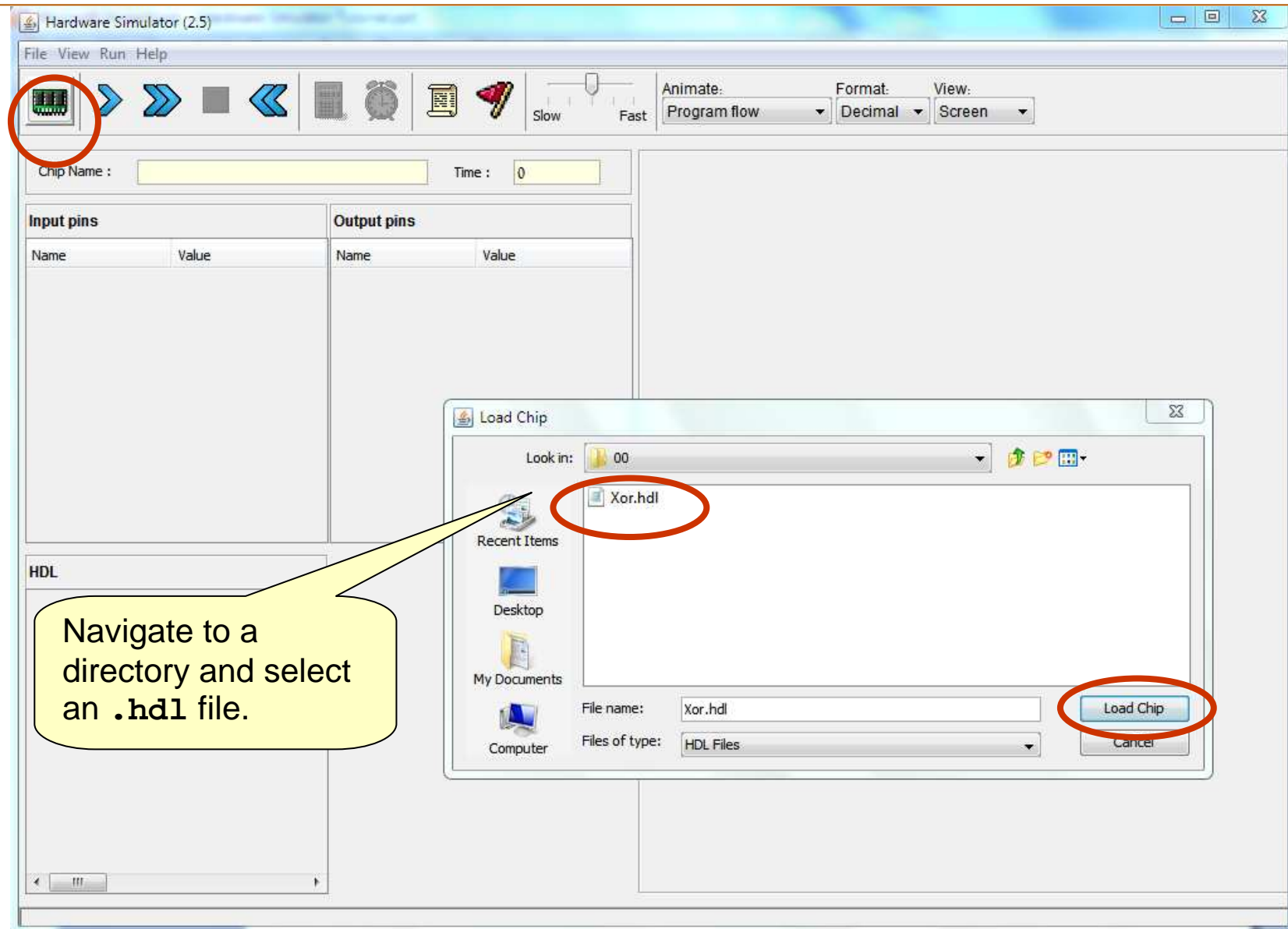
chip
implementation

```
/** 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

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

Output pins

Name	Value
out	0

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

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);
}
```

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

Internal pins

Name	Value
nota	0
notb	0
w1	0
w2	0

- 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.

Exploring the Chip Logic

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

File View Run Help

Animate: Program flow Format: Decimal View: Screen

Chip Name: Xor Time: 0

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

1. Click the **PARTS** keyword

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

Internal Parts		
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"/>

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

Animate: Program flow Format: Decimal View: Screen

Chip Name: Xor Time: 0

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

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

```
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=noth);
  And(a=a, b=noth, 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

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.

izmijeniš varijablu i klikneš kalkulator
da izbaci rezultat na out pinovima

Interactive Chip Testing

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, with the calculator icon circled in red. Below the toolbar, the 'Chip Name' is set to 'Xor' and 'Time' is 0. The 'Input pins' table shows 'a' with value 0 and 'b' with value 1. The 'Output pins' table shows 'out' with value 0. The 'HDL' section 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 (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 the following values:

Name	Value
nota	1
notb	1
w1	0
w2	0

A yellow callout box contains the following text:

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

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

File View Run Help

Chip Name: Xor Time: 0

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* (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.

Hardware Simulation Tutorial



kad implementiraš kolo, učitáš ga u simulator
onda učitáš i skriptu za testiranje T0G kola
i pustiš fast-forward da istestira..

rezultat bude spremljen i u kolo.out fajl

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** (so you don't have to write them)
- 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 Script

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

File View Run Help

Chip Name: Xor Time: 0

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

HDL

```
// Xor
// if
CHI
IN
OUT
PA
Not
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);
}
```

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

The screenshot shows the Hardware Simulator (1.1b) interface. The top menu bar includes File, View, Run, and Help. Below the menu is a toolbar with icons for simulation control: a single step (blue arrow), multi-step (blue double arrow), pause (grey square), reset (blue double arrow pointing left), and a clock icon. To the right of the clock is a slider for execution speed, labeled 'Slow' and 'Fast'. Further right are dropdown menus for 'Animate:' (set to 'Program flow'), 'Format:' (set to 'Decimal'), and 'View:' (set to 'Script').

Below the toolbar, there is a 'Chip Name:' field and a 'Time:' field. The main window is divided into several panes. On the left, there are 'Input pins' and 'Output pins' tables. Below these is the 'HDL' pane showing Verilog code for an XOR gate. On the right, there is a 'Script' pane containing a list of simulation steps, each ending with a semicolon. The script steps are:

- 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;

Yellow callout boxes with arrows point to specific controls:

- 'Controls the script execution speed' points to the speed slider.
- 'Resets the script' points to the reset button (blue double arrow pointing left).
- 'Pauses the script execution' points to the pause button (grey square).
- 'Multi-step execution, until a pause' points to the multi-step button (blue double arrow).
- 'Executes the next simulation step' points to the single step button (blue arrow).

A yellow box on the right side of the script pane contains the text: 'Script = series of simulation steps, each ending with a semicolon.'

At the bottom of the window, a status bar displays the message: 'New script loaded: E:\project 1\Xor.tst'.

Running a Script

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 various icons, with the "Run" icon (a blue double arrow) circled in red. Below the toolbar, there are fields for "Chip Name:" and "Time: 0". To the left, there are tables for "Input pins" and "Output pins". The "Input pins" table has columns "Name" and "Value", with rows for "a" and "b", both with a value of 0. The "Output pins" table has columns "Name" and "Value", with a row for "out" with a value of 0. The main area on the right displays a script. The first line, "load Xor.hdl,", is highlighted in yellow and circled in red. Below it, the script continues with "output-file Xor.out,", "compare-to Xor.cmp,", "output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;", and several "set", "eval", and "output" commands. A large orange arrow pointing downwards is labeled "Script execution flow". A yellow callout box on the left contains the text "Typical 'init' code:" followed by a numbered list: 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. The status bar at the bottom reads "New script loaded: E:\project 1\Xor.tst".

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 "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.

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

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

File View Run Help

Chip Name: Xor Time: 0

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

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

```
// 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);
}
```

```
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 ends

End of script - Comparison ended successfully

Viewing Output and Compare Files

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 simulation control (play, pause, stop, reset, etc.) and a speed slider. The "Chip Name" field is set to "Xor" and the "Time" field is 0. The "Input pins" table shows 'a' and 'b' both with a value of 1. The "Output pins" table shows 'out' with a value of 0. The "HDL" section contains the Verilog code for an XOR gate. The "Internal pins" table shows 'nota', 'notb', 'w1', and 'w2' all with a value of 0. The "Script" window on the right shows the simulation script, with the line "output;" highlighted in yellow. A red circle highlights the "View" menu, which is open and shows the "Output" option selected.

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

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;
```

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

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. here: [Mux16.class](#))
- The name of the Java class is specified following the **BUILTIN** keyword
- Built-In implementations of all the chips that appear in the book are supplied in the [tools/builtIn](#) directory.

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

ako NE ZNAMO ili NEĆEMO da implementiramo neki čip,
već postoje BUILTIN ugrađene implementacije

Built-in chips are used to:

- Implement primitive gates (in the computer built in this book: [Nand](#) and [DFF](#))
- Implement chips that have peripheral side effects (like I/O devices)
- Implement chips that feature a GUI (for debugging)
- 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)
- Facilitate behavioral simulation of a chip before actually building it in HDL
- Built-in chips can be used either *explicitly*, or *implicitly*.

Explicit Use of Built-in Chips

The screenshot shows the Hardware Simulator (1.4b1) interface. The title bar indicates the file path: 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. Below the toolbar, the 'Chip Name' is set to 'Mux16' and 'Time' is 0. The 'Input pins' table shows 'a[16]', 'b[16]', and 'sel' all with a value of 0. The 'Output pins' table shows 'out[16]' with a value of 0. The 'HDL' window displays the following code:

```
// MIT Press 2004. Book site: 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;
}
```

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 'Files of type' is set to 'HDL Files'. The 'Load Chip' button is circled in red.

Standard interface.

Built-in implementation.

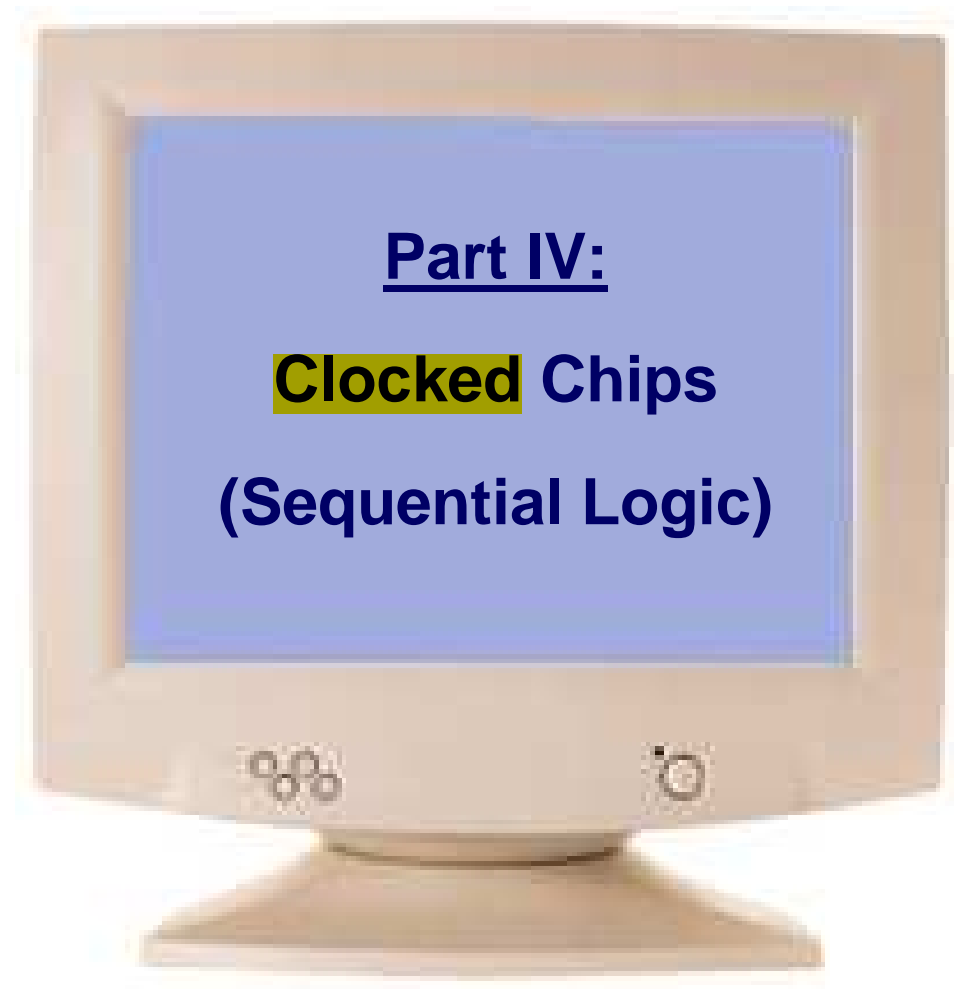
The chip is loaded from the **tools/builtIn** directory (includes executable versions of all the chips mentioned in the book).

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,out=Nota,b=Notb,out=aNotb);
        And(a=Nota,b=b,out=bNota);
        Or(a=aNotb,b=bNota,out=out);
}
```

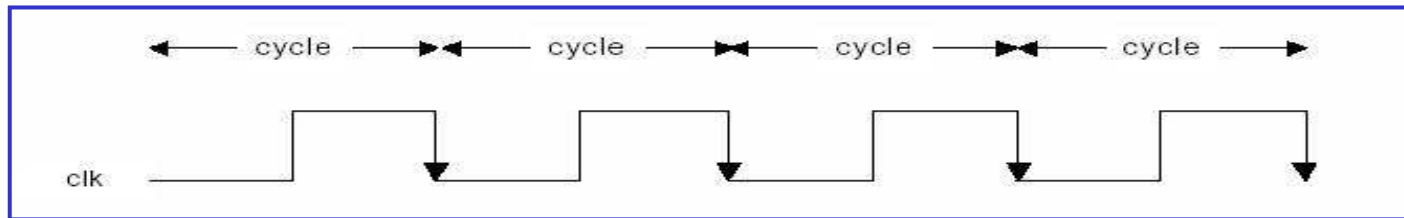
simulator
prvo traži
NAŠU
implementaciju
pa ako ne nađe
koristi
BUILTIN

- 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.



Clocked (Sequential) Chips

- The implementation of clocked chips is based on **sequential logic**
- The operation of clocked chips is regulated by a **master clock signal**:



- In our jargon, a clock cycle = *tick*-phase (low), followed by a *tock*-phase (high)
- 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, 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. 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;
}
```

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

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

Simulating Clocked Chips

The screenshot shows the Hardware Simulator (1.4b1) window. The title bar indicates the file path: G:\TECS\tools\builtIn\RAM8.hdl. The menu bar includes File, View, Run, and Help. The toolbar contains icons for loading a file, running, pausing, and stepping through the simulation, along with speed controls (Slow, Fast) and dropdown menus for Animate (Program flow), Format (Decimal), and View (Screen).

The main workspace is divided into three sections:

- Input pins:** A table with columns Name and Value. The values are: in[16] = 112, load = 1, and address[3] = 5. This section is circled in red.
- Output pins:** A table with columns Name and Value. The value is: out[16] = 0.
- HDL:** A text area showing the HDL code for the RAM8 chip. The code is as follows:

```
CHIP RAM8 {  
    IN in[16], load, address[3];  
    OUT out[16];  
  
    BUILTIN RAM8;  
    CLOCKED in, load;  
}
```

The RAM8 memory table is located on the right side of the workspace. It has columns for address (0-7) and value (0-0). The value at address 5 is highlighted in yellow and circled in red.

Annotations explain the simulation process:

- 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 bit
 * stored at the memory location spe
 * memory location specified by add
 * 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. 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;  
}
```

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*)

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

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

Simulating Clocked Chips Using a Test Script

The screenshot shows the Hardware Simulator (1.1b) interface. The title bar indicates the file path: D:\hack\tools\BuiltIn\RAM8.hdl. The menu bar includes File, View, Run, and Help. The toolbar contains icons for single-action tick-tock, repeated tick-tocks, and a clock icon. The 'Chip Name' field is set to 'RAM8 (Clocked)' and the 'Time' field is set to '1'. The 'Input pins' table shows values for 'in[16]', 'load', and 'address'. The 'Output pins' table shows values for 'out[16]'. The 'HDL' window displays the chip definition code. The 'Script' window shows the default script: 'Repeat { Tick, Tock; }'. A dropdown menu is open over the 'Script' button, showing options: Script, Output, Compare, and Screen. Three yellow callout boxes provide explanations: 'Single-action tick-tock' points to the single-action tick-tock icon; 'Controls the script speed, and thus the simulated clock speed, and thus the overall chip execution speed' points to the clock icon; 'Tick-tocks repeatedly and infinitely' points to the repeated tick-tocks icon.

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

File View Run Help

Chip Name: RAM8 (Clocked) Time: 1

Input pins

Name	Value
in[16]	112
load	1
address	5

Output pins

Name	Value
out[16]	112

HDL

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

    BUILTIN RAM8;

    CLOCKED in, load;
}
```

Script

```
Repeat {
    Tick,
    Tock;
}
```

Single-action tick-tock

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

Tick-tocks repeatedly and infinitely

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 react to clock cycles, repeatedly.



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 running the simulation (play, fast play, stop, reset, etc.) and a slider for "Animate" (Slow to Fast). 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	0
address[15]	0
- HDL:** A code 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 with columns "Address" and "Value".

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 table)
- 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 code)
- 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 **Keyboard.hdl** chip" (Points to the Keyboard icon in the internal pins table)
- Orange callout:** "GUI of the built-in **RAM16K.hdl** chip" (Points to the RAM 16K table)

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 its 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.

GUIDemo Chip in Action

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

File View Run Help

Chip Name: GUIDemo (Clocked) Time: 5+

Input pins

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

Output pins

Name	Value
out[16]	0

2. User: runs the clock

3. 16 black pixels are drawn beginning in row = 156 col = 320

1. User enters:

- in = -1 (=16 1's in binary)
- address = 5012
- load = 1

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);
}
```

RAM 16K

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

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

File View Run Help

Chip Name: RAM8 (Clocked) Time: 1

Input pins

Name	Value
in[16]	
load	
address[3]	

Output pins

Name	Value
out[16]	112

Breakpoint Panel

Variable Name	Value
Time	15
In	1024
RAM8[3]	172

RAM 8:

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

1. Open the breakpoints panel

2. Previously-declared breakpoints

3. To update an existing breakpoint, double-click it

3. Add, delete, or update breakpoints

HDL

```
// 8-registers memory
CHIP RAM8 {
```

The breakpoints logic:

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

Scripts for Testing the Topmost **Computer** chip

```
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.
clear-breakpoints;
```

- 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 **ROM32K** 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 reads "Hardware Simulator (1.1b) - D:\hack\tools\BuiltIn\RAM8.hdl". The menu bar includes "File", "View", "Run", and "Help". The toolbar contains icons for loading a program, running, pausing, and stepping through code, along with a clock icon and a "Slow/Fast" toggle. The "Animate:" dropdown is set to "Program flow", "Format:" is set to "Decimal", and "View:" is set to "Screen". Below the toolbar, the "Chip Name:" field is "RAM8 (Clocked)" and the "Time:" field is "1". The "Input pins" table shows "in[16]" with value "112" and "load address" with value "112". The "Output pins" table is empty. The "HDL" window shows code for "CHIP RAM8". The "RAM 8:" window shows a table with values 0, 1, 2, 3, 4. Three callouts provide details on the visual options.

Chip Name : RAM8 (Clocked) Time : 1

Input pins

Name	Value
in[16]	112
load address	112

Output pins

Name	Value

HDL

```
// 8-regis
CHIP RAM
IN in[16]
OUT out[16]
BUILT-IN
CLOCKED
}
```

RAM 8:

0	0
1	0
2	0
3	0
4	0

- **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* on the **CPU** or **Computer** chip, any animation effects slow down the simulation considerably.

Format of displayed pin values:

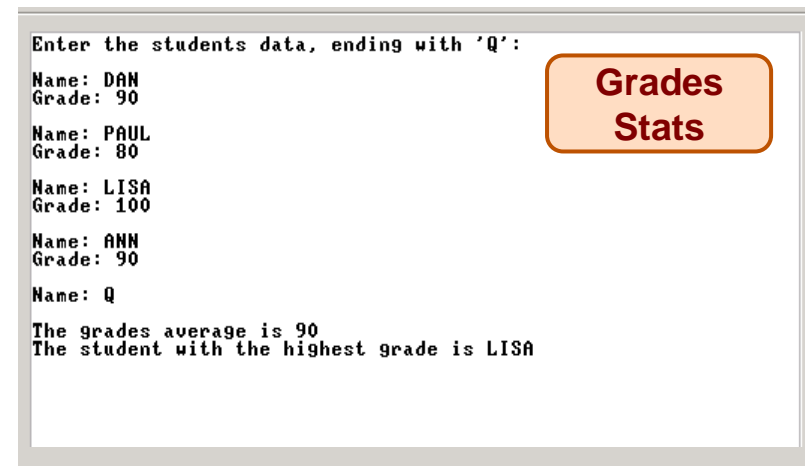
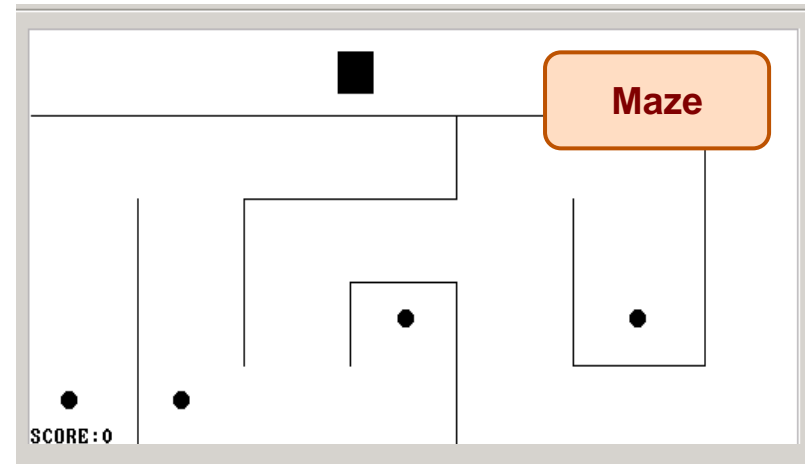
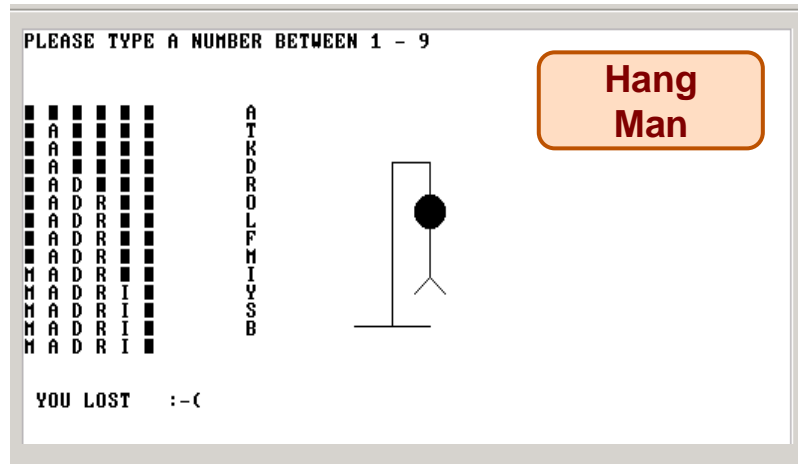
- **Decimal** (default)
- **Hexadecimal**
- **Binary**

- **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: 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 they can be used in the construction of many different computers.

The Hack chip-set and hardware platform can be built using the hardware simulator, 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.

Aside: H.D. Thoreau about chips, bugs, and close observation:

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).