

From Nand to Tetris

Building a Modern Computer From First Principles

[Home](#)[Projects](#)[Book](#)[Software](#)[License](#)[Papers](#)[Cool Stuff](#)[About](#)[Team](#)[Stay in Touch](#)[Q&A](#)

Project 3: Sequential Chips

Background

The computer's main memory, also called Random Access Memory, registers, each designed to hold an n-bit value. In this project you have two main issues: (i) how to use gate logic to store bits persistently and (ii) how to locate ("address") the memory register on which we wish to operate.

Objective

Build all the chips described in Chapter 3 (see list below), leading to a computer that can execute the Tetris program. The only building blocks that you can use are primitive DFF gates, chips described in previous chapters.

Chips

Chip Name	Description	
DFF	Data Flip-Flop (primitive)	-
Bit	1-bit register	1
Register	16-bit register	1
RAM8	16-bit / 8-register memory	1
RAM64	16-bit / 64-register memory	1
RAM512	16-bit / 512-register memory	1
RAM4K	16-bit / 4096-register memory	1
RAM16K	16-bit / 16384-register memory	1

Contract

When loaded into the supplied Hardware Simulator, your chip design supplied .tst script, should produce the outputs listed in the supplied simulator will let you know. This contract must be satisfied for each which is considered primitive, and thus there is no need to implement

Resources

The relevant reading for this project is [Chapter 3](#) and [Appendix A](#). Chapter 3 should be implemented in the Hardware Description Language

For each chip, we supply a skeletal .hdl file with a missing implementation supply a .tst script that instructs the hardware simulator how to test the correct output that this test should generate. Your job is to complete the files.

The resources that you need for this project are the supplied Hardware you've downloaded the Nand2Tetris Software Suite, these files are in a directory is further partitioned into two sub-directories, for reasons

Tips

The Data Flip-Flop (DFF) gate is considered primitive and thus the simulator encounters a DFF chip part in an HDL program, it automatically invokes nand2tetris/tools/builtInChips/DFF.hdl implementation.

Built-in chips: When constructing RAM chips from lower-level RAM versions of the latter. Otherwise, the simulator will recursively generate objects, one for each one of the many chip parts that make up a test the simulator to run slowly, or, worse, out of memory. i.e. out of the memory the simulator is running.

To avert this problem, we've partitioned the RAM chips that you have into two directories, named projects/03/a and projects/03/b. This partitioning is only: when building the chips stored in b, the simulator is forced to use low-level chip parts whose .hdl programs are stored in a but not in b.

Tools

All the chips mentioned in projects 0-5 can be implemented and tested. Here is a screen shot of testing a built-in RAM8.hdl chip implementation

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]	

GUI of the loaded built-in chip

The user enters some input values and clicks the clock icon to pace the clock. Each click generates a tick or a tock. The clock can also be paced using a test script.

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;  
}
```

A built-in RAM8 chip is loaded

Clock dependency
The clock
some chip
clock
response
This
of a
memory