Project 3

The computer's main memory, also called Random Access Memory, or RAM, is an addressable sequence of registers, each designed to hold an *n*-bit value. In this project you will gradually build a RAM unit. This involves two main issues: (i) using gate logic to store bits persistently, over time, and (ii) using gate logic to locate ("address") the memory register on which we wish to operate.

Objective

Build the following chips:

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

The only building blocks that you can use are some of the chips listed in Projects 1 and 2 and the chips that you will gradually build on top of them in this project. The DFF chip is considered primitive and thus there is no need to build it.

Files (Same guidelines as in Projects 1 and 2)

For each chip Xxx, (we use the terms *chip* and *gate* interchangeably) we provide a skeletal Xxx.hdl program (sometimes called a stub file) with a missing implementation part. In addition, for each chip we provide an Xxx.tst script that tells the hardware simulator how to test it, along with an Xxx.cmp compare file that lists the correct output that the supplied test is expected to generate. All these files are available in your nand2tetris/projects/03 folder. Your job is to complete and test all the Xxx.hdl files in this folder. These files can be written and edited using any plain text editor.

Contract: When loaded into the hardware simulator, your chip design (modified .hdl program), tested on the supplied .tst file, should produce the outputs listed in the supplied .cmp file. If the actual outputs generated by the simulator disagree with the desired outputs, the simulator will stop the simulation and produce an error message.

Structure of the Project Folder

When constructing RAM chips from lower-level RAM chip-parts, we recommend using built-in versions of the latter. Otherwise, the simulator will recursively generate many memory-resident software objects, one for each of the many chip parts that make up a typical RAM unit, all the way down to the individual register and bit levels. This may cause the simulator to run slowly, or, worse, run out of the memory of the host computer on which the simulator is executing.

To avert this problem, we've partitioned the RAM chips built in this project into two sub-folders. The RAM8.hdl and RAM64.hdl files are stored in projects/03/a, and the files of the other, higher-level RAM chips are stored in projects/03/b. This partition is done for one purpose only: when evaluating the RAM chips stored in the b folder, the simulator will resort to using builtin implementations of the RAM64 chip-parts, simply because RAM64.hdl is not to be found in the b folder.

Correction of a supplied HDL file: The correct documentation of PC.hdl is as follows:

```
/**
 * A 16-bit counter.
 * if      reset[t]: out[t+1] = 0
 * else if load[t]:  out[t+1] = in[t]
 * else if inc[t]:   out[t+1] = out[t] + 1
 * else              out[t+1] = out[t]
 */
```

So, use the above documentation, and implement your Program Counter (PC) operation accordingly.

References

Clock Demo

Register Demo

RAM Demo

Program Counter Demo

The remaining references and implementation tips for this project are identical to those of project 1. Read them before you start working on project 3. In particular, remember that good HDL programs use as few chip-parts as possible, and that there is no need to invent and implement any “helper chips”; your HDL programs should use only chips that were specified in projects 1 and 2, and in this project.

Depending on the course that you are taking, you may find more implementation tips in your lecture notes.