CSCE-312 | Spring 2019

Project 3: Sequential Chips

Due Date: Submit on eCampus by , **Friday, Feb 22nd, 11:59 PM**

Grading

(A) Project Demo [60%]: TBA

You will be graded for correctness of the chips (hdl) you have designed and coded. You will be running **live** test of all your HDL codes using Nand2tetris software (Hardware Simulator) with TA/PT. The same simulator you would have used to check your chips in the project. So, make sure to test and verify your codes before finally submitting on eCampus.

***Rubric**:* See the grading sheet. Each circuit needs to pass all its test cases to get the points, else you will receive 0 on that circuit.

(B) Code Review/Q&A [40%]: To be held with the LIVE Demo

Code review of randomly selected chips. The questions can involve drawing circuit diagram of randomly selected chips. Should not be difficult for you if you have understood the core inner workings of your project.

Deliverables & Submission

You need to turn in the HDL files for all the chips.

Put your full name in the introductory comment present in each HDL code.

Use relevant code comments and indentation in your code.

Zip all the required HDL files into a compressed file *FirstName-LastName-UIN.zip*

Submit this zip file on eCampus.

Late Submission Policy: Refer to the Syllabus

Background

The computer's main memory, also known as Random Access Memory (RAM), is an addressable sequence of n-bit registers, each designed to hold an n-bit value. In this project you will gradually build a RAM module. This involves two main issues: (i) how to use gate logic to store bits persistently, over time, and (ii) how to use gate logic to locate ("address") the memory register on which we wish to operate. In addition, you will build functions that are constructed with combinational and sequential logic design elements.

Objective

Build all the chips described in the list below. **The only building blocks that you can use are primitive DFF gates, chips that you will build on top of them, and chips described in previous chapters.**

Chips

Chips Name:	**Description**	**File Name**
Bit	1-bit register (use DFF)	Bit.hdl
Register	16-bit register	Register.hdl
RAM8	8 16-bit register memory	RAM8.hdl
RAM64	64 16-bit register memory	RAM64.hdl
RAM512	512 16-bit register memory	RAM512.hdl
PC	16-bit program counter	PC.hdl
Aggie Cipher	4-bit counter using D flip flop	AggieCipher.hdl
Fibonacci	Fibonacci Sequence generator	Fibonacci.hdl

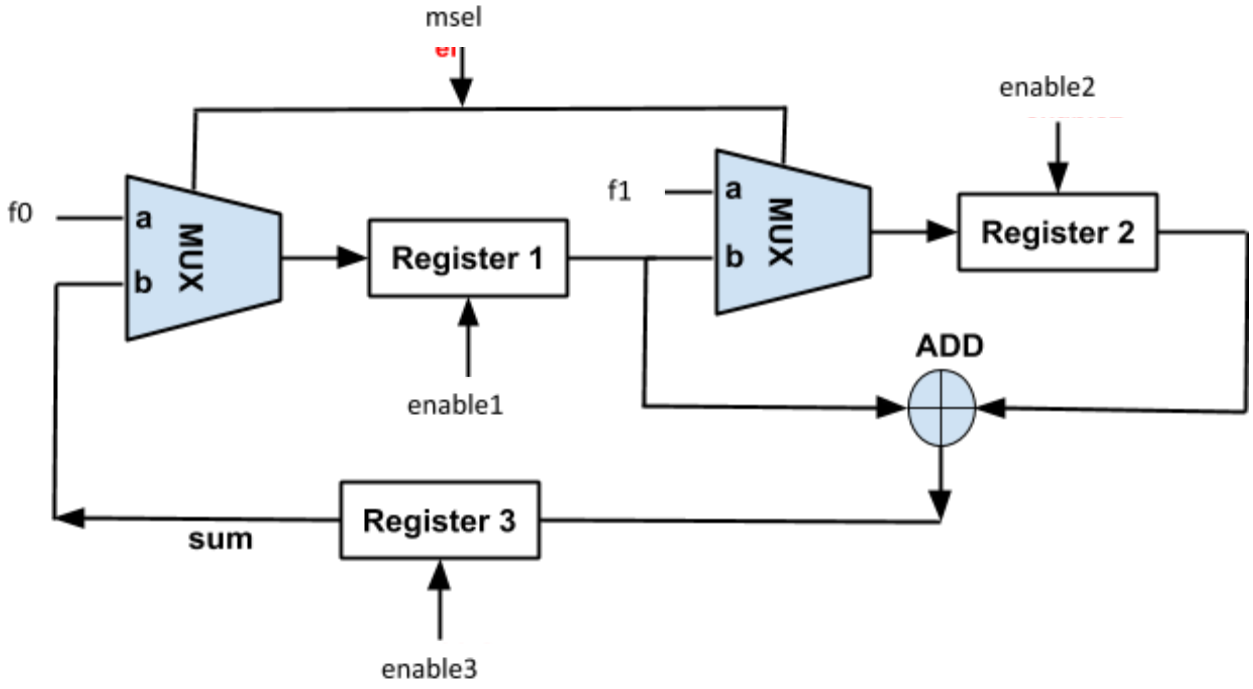
Aggie Cipher

Design a simple cipher logic which generates a code which is equal to a user-provided 4-bit *input + the value generated from a counter*, where counter value starts from 0000, and increments by 1 every clock cycle. The counter wraps to 0000 when it reaches a count of 15. You may use the program counter (PC) designed in prior exercise to implement the Aggie Cipher logic.

***out=in+counter**, where counter=<0,1,2,3,4,5,6,....,15,0,1,2,...>*

Fibonacci Sequence generator:

The general Fibonacci sequence is a sequence that starts with *f0=0* and *f1=1* . The next number in the sequence is the *sum* of previous two numbers. So the Fibonacci number sequence generated in our circuit will be:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89…
Here is the circuit you may use. **Please make an effort to understand the working of this circuit and explain it as comments in your HDL file.**



To use this circuit, you have to control these signals, namely, *enable1, enable2, enable3* and *msel*.

- *msel=0* will select the starting values *f0* and *f1* of the Fibonacci Sequence
- *msel=1* will keep running the Fibonacci sequence with $sum(t+1) \leftarrow sum(t) + sum(t-1)$ for clock cycle *t*
- *enable1=1 or enable2=1* or *enable3=1* activate respective registers by loading the corresponding input values to corresponding register outputs
- *enable1=0 or enable2=0* or *enable3=0* retain the register outputs from the previous cycle

The test file Fibonacci.tst assigns the values to these control signals.
See how output in the Fibonacci.out file changes while changing those signals.

Contract
When loaded into the supplied Hardware Simulator, your chip design (modified .hdl program), tested on the supplied .tst script, must produce the outputs listed in the supplied .cmp file. If that is not the case, the simulator will let you know.

Resources
The relevant reading for this project is
Chapter 3 https://docs.wixstatic.com/ugd/44046b_1801b5682e4d4a67bd05e14235665d8b.pdf
Appendix A https://docs.wixstatic.com/ugd/44046b_d715f80dca2a43af926131a52e3d3d90.pdf

Specifically, all the chips described in Chapter 3 should be implemented in the Hardware Description Language (HDL) specified in Appendix A.

For each chip, we supply a skeletal .hdl file with a missing implementation part. In addition, for each chip we supply a .tst script that instructs the hardware simulator how to test it, and a .cmp ("compare file") containing the correct output that this test should generate. Your job is to complete and test the supplied skeletal .hdl files.

The resources that you need for this project are the supplied Hardware Simulator and the files listed above. Download your hdl files from ecampus and replace these files to those stored in your projects/03 directory.

Tips

The Data Flip-Flop (DFF) gate is considered primitive and thus there is no need to build it: when the simulator encounters a DFF chip part in an HDL program, it automatically invokes the built-in tools/builtInChips/DFF.hdl implementation.

Tools

Following is a screenshot of testing a built-in RAM8.hdl chip implementation on the Hardware Simulator: