CSCE-312 | Summer 2020 | Project3

Sequential Chips

Due Date: Submit on eCampus by Thursday, July 16th, 11:59 PM

Grading

(A) Project Execution [100%]:
You will be graded for correctness of the chips you have designed and coded. Your work will be drawn from the codes downloaded from eCampus and exercised using Nand2tetris software (Hardware Simulator). So, make sure to test and verify your codes before finally submitting on eCampus.

Deliverables & Submission

You need to turn in completed HDL, TST, CMP files as applicable for all the designed chips. Put your full name and UIN in the introductory comment present in each HDL code. Use relevant code comments and indentation. Also, include the cover sheet with your signature. Zip all the required files and the signed cover sheet into a compressed file *FirstName-LastName-UIN.zip* . Submit this zip file on eCampus.

Late Submission Policy: Refer to the Syllabus

Full Name: **Section:** **UIN:**

Any assignment turned in without a fully completed cover page will NOT BE GRADED.
Please list all below all sources (people, books, web pages, etc) consulted regarding this assignment:

CSCE 312 Students	Other People	Printed Material	Web Material (URL)	Other
1.	1.	1.	1.	1.
2.	2.	2.	2.	2.
3.	3.	3.	3.	3.

Please consult the Aggie Honor System Office for additional information regarding academic misconduct – it is your responsibility to understand what constitutes academic misconduct and to ensure that you do not commit it.

I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received nor given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment.

eCampus Submission Date: ___________________

Printed Name (in lieu of a signature): _________________________

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 the memory register ("using the address") 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 to start with, and subsequently chips that you will build on top of them, and chips described in earlier projects. To build the RAM chips you need to have some understanding of how a bit and a register works. Details about these will be discussed in the labs to develop a working level understanding of these chips.**

Chips

Chips Name:	**Description**	**File Name**
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
Fibonacci	Fibonacci Sequence generator	Fibonacci.hdl

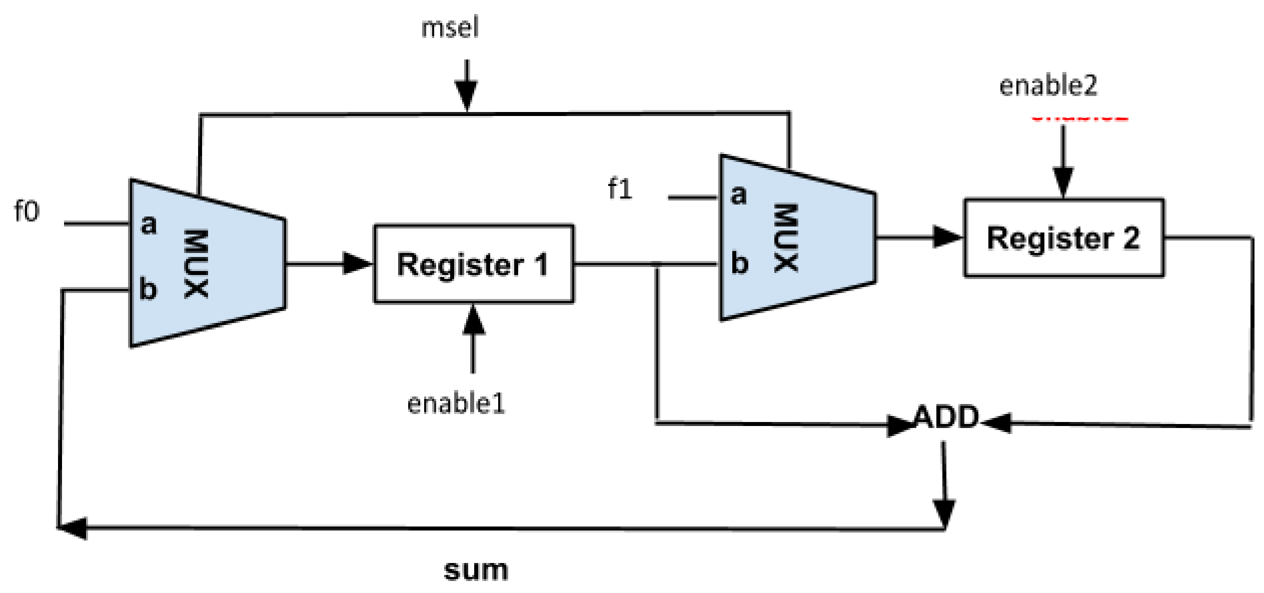
Comments:

- **Program Counter:** Design as discussed in class and as presented in the textbook

- **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 the 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. **You need to understand the working of this circuit along with its .tst and .cmp files. Explain the logic as comments in your HDL file.**

The ADD operation in the circuit diagram is the addition of the two previous Fibonacci numbers and you can use any 16 bit adder chip for this purpose.



To use this circuit, you have to control these signals, namely, *enable1, enable2* 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* activate respective registers by **loading** the corresponding input values to corresponding register outputs

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.

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 .cmp ("compare file") containing the correct output that this test should generate.

For **Fibonacci**, we provide **two** .tst and .cmp files. Your implementation must pass both the tests.

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.

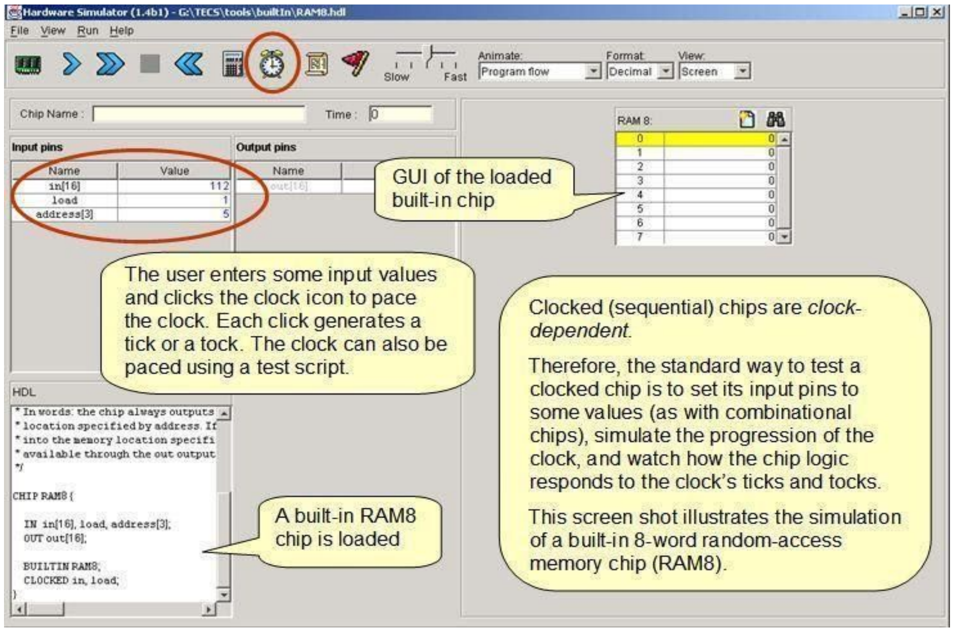
The resources that you need for this project are the supplied Hardware Simulator and the files listed above. Download your hdl files from ecampus and replace these files to those stored in your projects/03 directory.

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:



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

Chip Name :
Time : 0

Animate: Program flow
Format: Decimal
View: Screen

Input pins

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

Output pins

Name	Value
out[16]	

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;
}
```

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.

Clocked (sequential) chips are *clock-dependent*.

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 clock's ticks and tocks.

This screen shot illustrates the simulation of a built-in 8-word random-access memory chip (RAM8).

A built-in RAM8 chip is loaded