CSCE-312 | Lab Exercise 3

Elementary Memory Abstractions

Submission

Grading

The grade for this assignment is 100% based on the correctness of the chips you submit. The grader will use the Hardware Simulator tool, along with test scripts based on those given to you, to verify the correctness of your chips. **No partial credit will be given for chips that are not *fully* functional:** We want you to create fully functional chips to realize complete understanding of the implemented functions. If a chip passes only part of its test cases, you will receive 0 points for that chip.

Please refer to the Syllabus for the late submission policy.

Deliverables

You need to turn in:

- Completed HDL files for all implemented chips.
    - Put your full name and UIN in the header comment present in each HDL file.
    - You do not need to submit the .tst or .cmp files

Zip all the required HDL files into a compressed file FirstName-LastName-UIN.zip and submit this zip file on CANVAS. If you do not follow these steps properly, you may receive a 0.

Background

This exercise serves as an introduction to sequential logic and Project 3, where you will build the Random Access Memory (RAM) module of the Hack hardware platform along with a few more sequential logic chips. We first approach the problem of building up a simple memory device to store a single bit. Once you can build a single memory cell, building the 8 and 16-bit registers is simply a scaling problem. Additionally, dealing with sequential logic means time and synchronization, thus this exercise serves as a gentle introduction to clocking mechanism as well.

Objective

Build all the chips in the table below. Since we've moved away from building basic gates, there is no restriction on using built-in gates (except the ones asked to be built for this assignment). As usual, you may add your own custom chips to simplify programming; if you do so, please be sure to include them in your submission.

Note that once you've completed these chips, feel free to copy them into your Project 3 directory to use as building blocks for the RAM modules.

Chip [points]	**Description**
Bit [40]	A single bit memory cell that retains its output even when the input data changes
Register8Bit [30]	8-bit memory cell, which you must build using Bit
Register16Bit [30]	16-bit Register, which you must build using Register8Bit

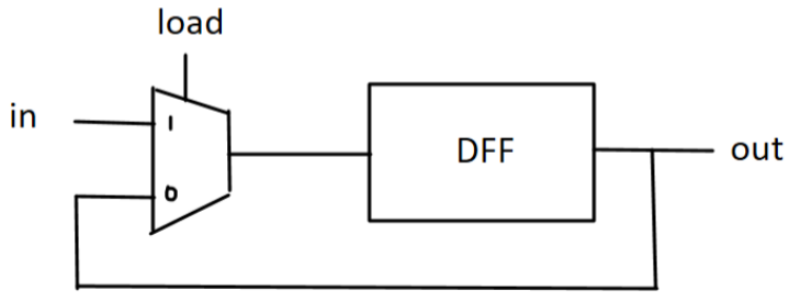
Chip Details

D flip-flop and Register

When you declare a variable in Python or C++ to store a value, the computer must have a storage device to put the value into. A flip-flop is a sequential circuit that can store a single bit of information. It works by taking an input signal and "latching" it, which means that it holds onto the input signal until it receives a new input signal. The output represents the stored bit. There are several different types of flip-flops, but the one we will be focusing on this lab is a Data flip-flop (DFF).

The DFF has a clock input (invisible in the built-in implementation) that continually changes according to the master clock's signal. Taken together, the data and the clock inputs enable the DFF to implement the time-based behavior *out(t) = in(t-1)*, where in and out are the gate's *input*

and *output* values and *t* is the current clock cycle. In other words, the DFF simply outputs the input value from the previous time unit.
A single bit register is simply a DFF with a loopback input from a MUX. This way, we can control when to load the next value. How does the DFF know when to update its latched value? Using a clock signal.



Single-bit memory cell's behavioral graph

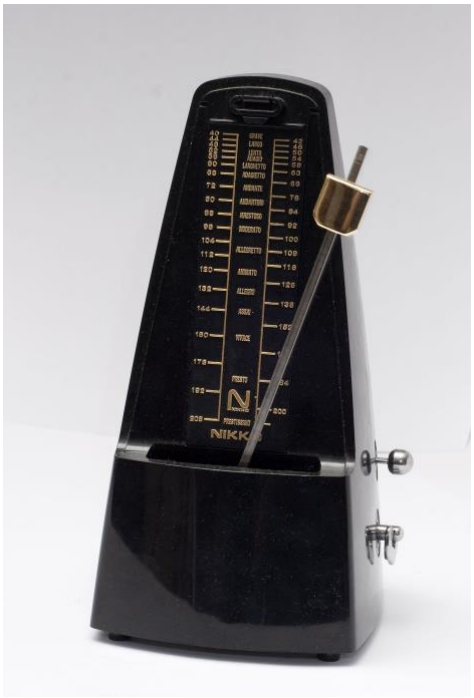
Wait a second, what is the clock?

A clock is simply a signal the goes low to high, then high to low, then repeats this. In sequential circuits, the output of one circuit is the input of another, and the clock is a way to synchronize these processes, so they do not operate randomly on their own. This is similar to a real-life clock that goes from tick to tock, then tock to tick, then tick to tock again. Let's say you throw a weekend party with friends. The first thing you do is ensuring an agreeable time where everyone can gather, 7:00pm for example. People do their own tasks and track the time using a common clock so that at exactly 7:00pm, everyone will be there ready to party, and no more person is allowed in. Without a common clock, there is no guarantee way to know when to gather.

On a different perspective, the computer clock is like a metronome, a simple device that produces a regular, steady beat to help musicians keep time when playing music. In similar fashion, a small computer crystal produces a steady beat, or pulse, that synchronizes the actions of all processing units along with memory devices. This regular pulse ensures that all components of the computer are working in a coordinated manner and that instructions are executed at appropriate times.

Remember that in high school physics, you probably learned the difference between period and frequency. A period is the time it takes to complete a single cycle and a frequency is the number of cycles that can be done in a specified amount of time. Frequency is the inverse of period (f = 1/p). This means that the higher the frequency, the more cycles can be completed in a single period, usually a single second. The clock speed is the frequency of the computer.

Clock speed can be adjusted to optimize its performance for various tasks, like how a musician adjusts the tempo of their playing by changing the metronome's speed. The faster the clock speed, the more instructions a computer can execute per given period (measured in time).



A metronome. Source: https://en.wikipedia.org/wiki/Metronome

How to represent clock in .tst and .cmp files

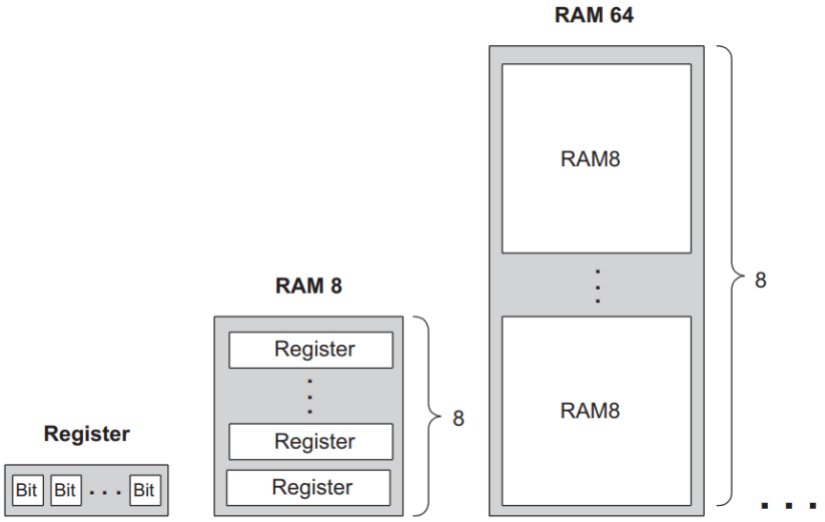
If you look at the Bit.cmp file in the assignment, the time column represents the clock as an invisible input for the device. For each time value (1, 2, 3,…), there is a plus version (1+, 2+, 3+,..., with 0+ being the only exception). Looking further into the tst file, you also see two new keywords: tick and tock. Just like a pendulum in a real-life clock, tick represents the first half of a full swing (also 1, 2, 3) and tock represents the second half (also 1+, 2+, 3+). By default, the DFF input and output correspond to tock clock. In the picture below, you can see that the Bit chip's output switches from 0 to 255 when load = 1 when time=5. This is the case every time DFF updates its input. Review both the .cmp and .tst files for additional information.

	time	in	load	out
2	0+	0	0	0
3	1	0	0	0
4	1+	0	1	0
5	2	0	1	0
6	2+	127	0	0
7	3	127	0	0
8	3+	127	0	0
9	4	127	0	0
10	4+	255	1	0
11	5	255	1	255
12	5+	121	1	255
13	6	121	1	121
14	6+	0	0	121
15	7	0	0	121
16	7+	0	1	121
17	8	0	1	0

A metronome

Register8Bit and Register16Bit

Once you can build a single register, an 8-bit register can be built by putting 8 registers together just like you did with Not16 and Mux16. From there, you now have the building blocks needed to build the RAM modules and many other cool sequential logic circuits like the program counter.



Building blocks up to the RAM module

Helpful HDL Note

It may be helpful to note that the [begin..end] notation for partial pin connection can be used on either side of a connection, e.g.: `MyChip(a[0..3]=p, …)` or `MyChip(a=p[2..5], …)`[1]. You can also use `true` and `false` to hardcode pin values, e.g.: `And(a=p, b=false, …)`. Sub-bus notation allows you to specify the values for a pin using multiple expressions, e.g. `MyChip(a[0..2]=x, a[3]=false, a[4..7]=y, …)`.

[1] Note that you cannot *sub-bus* an internal pin in this way, only input/output pins (or those on the left side, referring to the component's pin names).