Adding SRAMs to Your Accelerator

CS250 Laboratory 3 (Version 100913)
Written by Colin Schmidt
Adpated from Ben Keller

Overview

In this lab, you will use the CAD tools and jackhammer to explore tradeoffs in the different implementation of your Sha3 Accelerator from Lab 2. For the first part of this assignment, you will use registers (flip-flops) to implement the storage for these buffers. Later, you will replace the flip-flop arrays with SRAM macros to produce a smaller and more energy efficient implementation.

Deliverables

This lab is due **Thursday, October 2 at 9:30AM**. The deliverables for this lab are:

- (a) your working Chisel RTL checked into your private git repository at Github
- (b) Reports (only!) generated by DC Compiler, IC Compiler, and PrimeTime PX checked into your git repo for the three implementations of your design
- (c) written answers to the questions given at the end of this document checked into your git repository as `writeup/lab3-report.pdf` or `writeup/lab3-report.txt`

You are encouraged to discuss your design with others in the class, but you must write your own code and turn in your own work.

The VLSI Flow

In this lab you will be experimenting with SRAMs in your Sha3 design and seeing the results when pushed through all of the CAD tools. The full toolflow can be seen in figure **??**. In addition to using SRAMs we will be adding two new libraries of standard cells for the tools to use. The new tool we will be using is ICC. You can find a good reference on how these tool work and what steps the makefiles go through, below.

ICC

This section will be updated later to explain the manual steps ICC goes through, for your reference

Multi-V_t Flow

Thus far in CS250, you have used standard cells from a single library to build your design. Now you will have an opportunity to take advantage a modern VLSI power-saving techinque: multi-V_t design. The Synopsys educational libraries provide different "flavors" of standard cells. In this lab, you'll be using both regular-V_t, high-V_t, and low-V_t standard cells. Recall that transistors with a higher threshold voltage are slower but leak less. If provided with multiple standard-cell libraries, the tools will use the faster low-V_t devices only on the critical path, and will swap in slower devices elsewhere to save power.

Take a look at the top-level `Makefrag` in the `build/vlsi`. By linking to each library and providing a few key commands, Design Compiler can take advantage of the different types of standard cells to better optimize the design. The `cells_*` variables determine which libraries the CAD tools are able to use. The given Makefrag shows how to use all three libraries at once but the evaluation requests that you limit the tools for one set of runs to see the advantanges of a mult-vt setup.

SRAM

This section of the lab describes the general flow for having Chisel include SRAMs in your design. In this lab you are responsible for converting the absorb buffer from flip-flops to an SRAM macro.

Implementing your buffer as flip-flops introduces considerable overhead in area and energy. With some simple modifications, you can instantiate an SRAM instead.

Chisel's `Mem` class can be used to implement memories with arbitrary numbers of read and write ports. Writes always happen on the positive clock edge, but the timing of reads can be either combinational (result available during the same cycle as address is provided) or sequential (address is registered on positive clock edge, result available during the next cycle). However, the ASIC SRAMs we will be using in this course all use sequential reads. Therefore, to enable the Chisel compiler to map a `Mem`-generated memory to an ASIC SRAM macro (as opposed to an array of flip-flops), you must specify sequential read timing and put a register at the output of the SRAM. Below is an example of how to do this for a one read, one write memory (the kind you should use in your design). You should instantiate the memory for your absorb buffer in a similar fashion. For more about the `Mem` class, see Chapter 9 of the Chisel manual.

```
val ram1r1w = Mem(UInt(width = 64), 17, seqRead = true)
val reg_raddr = Reg(UInt())
```

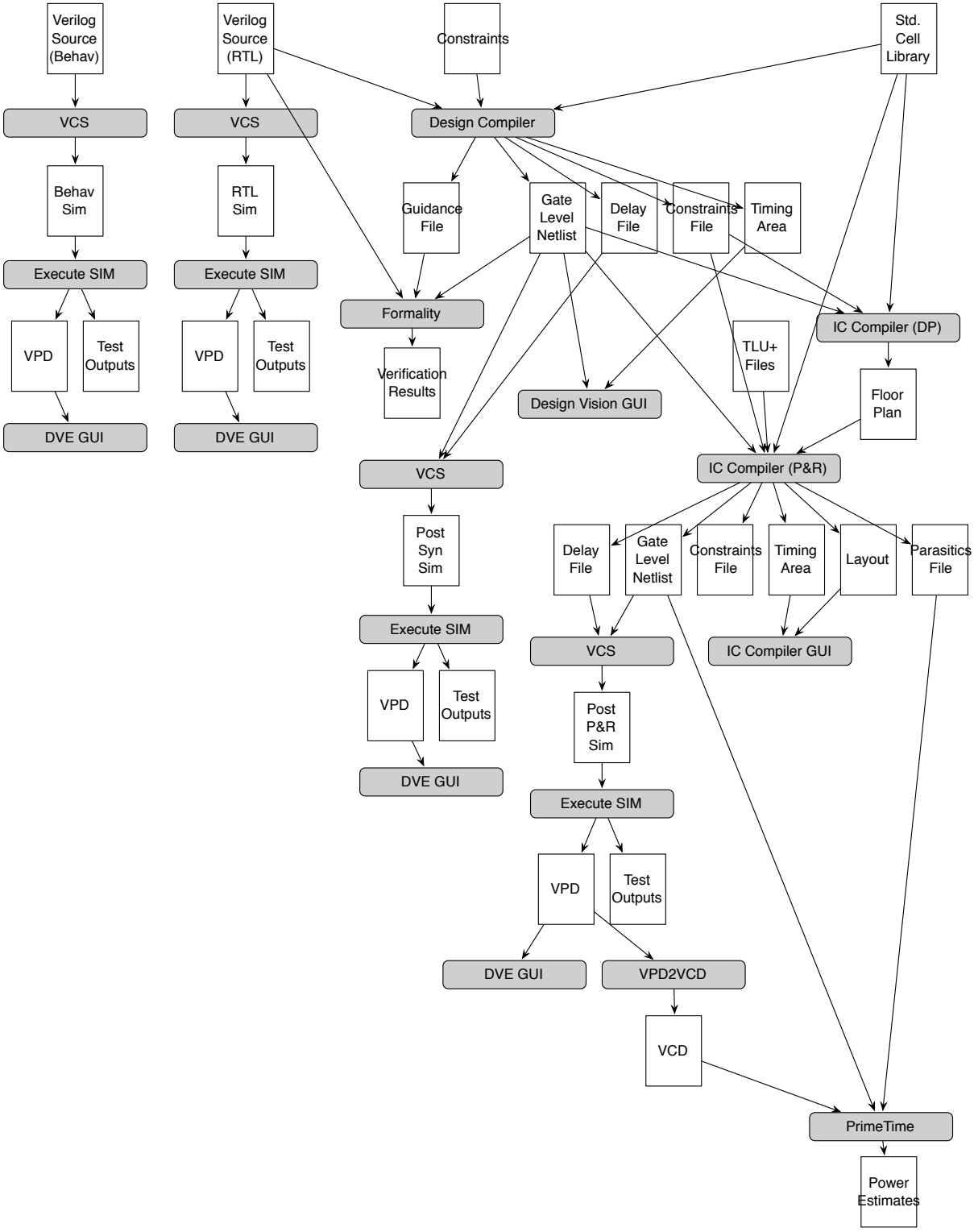

Figure 1: CS250 Toolflow for Lab 2

```
when (wen) { ram1r1w(waddr) := wdata }
when (ren) { reg_raddr := raddr }
val rdata = ram1r1w(reg_raddr)
```

You must instantiate 1088-bit SRAM as sized for a `width` of 64, as we have only given you one SRAM macro to use in this design. The macro is called `sram8t17x64`; it is dual-ported, 64 bits wide, and 17 entries deep. Its library files and Verilog model (used for simulation) are located in the `generated-rams` directory. This macro was generated using Cacti, an SRAM modeling suite. Make sure you the memory you instantiate using `Mem` has the same depth and width at the SRAM macro.

If you use the `Mem` class in a way that does not imply a sequential memory, Chisel will just instantiate an array of flip-flops instead. Take care to define and register the ports in the order shown above. Be sure to check the generated Verilog files to see that the SRAM macro is being instantiated.

Once you have successfully instantiated an SRAM, build your design with the provided flow. You can look at the `Makefrag` to see how the build setup has been modified to include the additional macro.

Submission and Writeup

Write a script to collect the data necessary to fill in the following tables. Use the reports generated by ICC for area, power, and timing information.

You should use Jackhammer to run the experiments to generate the data. The given Jackhammer configuration will run the design through four qors with all design points, however this may not be the best configuration to start with and you should feel free to modify IclusterSettings.scala to better suit your needs.

You should create both tables for all 6 points in our design space, and include the bits hashed per second for the final test with each set of tables.

Area		RVT	Multi-VT	Multi-VT + SRAM
Sha3 Dpath	um^2			
Total	um^2			
Cell Count	LVT			
Cell Count	RVT			
Cell Count	HVT			

Power		RVT	Multi-VT	Multi-VT + SRAM
Entire Design				
Leakage	uW			
Total	uW			
Sum Cache				
Leakage	uW			
Total	uW			

Questions: Multiple VT Flow

1. What impact does switching from a single VT to a multi-VT flow have on area? Does this match your expectations?

2. What effect does switching to a multi-VT have on the power consumption of your design? How much does it reduce leakage power?

3. Can you think of a reason why you wouldn't want to use multiple VT cells to implement a design?

4. What portion of the cells used in the multi-VT version of your design are regular VT? Does this match your expectations?

5. Where in your design do regular VT cells appear? Why do you think this is the case?

Questions: SRAM

1. How much area do you save by using an SRAM instead of registers to implement storage for your window buffer?

2. What is the area of the SRAM macro?

3. Assuming that the individual bitcells have an area of 0.415 um^2, what is the efficiency (area used for bitcells over total area) of this SRAM macro?

4. Would you expect this SRAM to use 6T or 8T bitcells, and why?

5. What are the other components of an SRAM, besides the bitcells?

Submission

To complete this lab, you should commit the following files to your private Github repository:

- Your working Chisel code.
- The `reports` directories from DC, ICC, and Primetime.
- Your script.
- Your answers to the questions above, in a file called `writeup.txt` or `writeup.pdf`.

Some general reminders about lab submission:

- Please note in your writeup if you discussed or received help with the lab from others in the course. This will not affect your grade, but is useful in the interest of full disclosure.
- Please note in your writeup (roughly) how many hours you spent on this lab in total.

Acknowledgements

Parts of this lab, particularly the section on SRAMs, were originally written by Rimas Avizienis for CS250 Fall 2012 Lab 2. The `AdvTester` Chisel class and much of the test infrastructure for this lab were written by Stephen Twigg.