Generating SRAMs in Chisel

CS250 Tutorial 10
November 19, 2011
Brian Zimmer

Overview

To create a memory in Chisel, you use the `Mem()` component. This will produce a memory in the emulator, and a simple flip-flop based memory in Verilog. However, for large memories, we will be using SRAMs for VLSI implementations. In this tutorial, you will learn how to use Chisel and class-specific memory compilers to create the necessary models needed for RTL simulation, synthesis, and place-and-route, and then how to include these new files in the existing flow Makefiles so that they are included in the design.

Prerequisite

You will need to be using the latest version of Chisel:

```
% cd sbt/work/lib
% rm hcl_2.8.1-1.0.jar
% ln -s ~cs250/fa11/install/chisel/hcl_mem4b.jar hcl_2.8.1-1.0.jar
```

Step 1: Instantiating an SRAM in Chisel

First, you will need to change the memory instantion from `Mem` to `Mem4` Second, you will need to add one line to tell Chisel the target for this memory. For example:

```
val data = Mem4(4096,...);
data.setTarget('inst);
```

Now, generate your VLSI again

```
% make vlsi
% cd vlsi
% cd generated-src
% ls
```

Inside this directory you will see `Top.conf`. Inside is a command that runs a program that creates a shim between what Chisel thinks a memory looks like and what the memory generator interface looks like, and also generates a configuration file that the sram generator uses. Run it with:

```
% bash Top.conf
```

Now generate the SRAMs by doing (replace the name with your specific name):

```
% ucbsc Top_mem_data_gen.conf
```

Note, you only need to do these steps when you change the memory inside of Chisel. Look inside `Top_mem_data_xx/`, you will see

- `.v` Plain text description of function properties for simulation
- `.lib` Plain text description of electrical properties for synthesis
- `.lef` Plain text description of physical properties for P&R
- `.db` Compiled .lib information for synthesis
- `.mw` Compiled .lef information for P&R

Now, you will need to include these files in all of your Makefiles. The following line will hold all of these files.

```
memories = $(wildcard $(srcdir)/*/*.v) $(wildcard $(srcdir)/*.v)
```

Now, include it by replacing:

```
$(srcdir)/Top.v \
```

With

```
$(memories) \
```

At this point you have now successfully implemented SRAMs in your design.

If you would like to load them, you will need to do so from your Verilog testharness. Use a command like:

```
$readmemh('file.hex',mem.data.mem.memory );
```

Where the 2nd argument points to the memory inside the generated .v file for the SRAM.

Note, if you would like to load riscv binaries into the memory, you can use the following command to convert from ELF to Hex.

```
elf2hex 16 1024 elffilename > hexfilename
```

For example, this will load the program for vcs-sim-rtl

```
$(global_asm_tests_out): %.out: $(global_tstdir)/% $(vcs_sim)
  elf2hex 16 1024 $< > $(patsubst %.out, %.hex, $@)
  ./$(vcs_sim) +exe=$(strip $(patsubst %.out, %.hex, $@)) > $@
  mv vcdplus.vpd $(patsubst %.out, %.vpd, $@)
  @echo; perl -ne 'print "  [$$1] $$ARGV \t$$2\n" if /\*{3}(.{8})\*{3}(.*)/' \
         $@; echo;
```