Blackboxing With Chisel

Supplemental Slides

Christopher Yarp

CS250 – Spring 2016

When to use blackboxing

- Blackboxing is used when …
  - You have some IP written in Verilog that you would like to include in your Chisel design
  - You cannot express some module because of Chisel's semantics (passgates, tristate switches, …)

How does blackboxing work?

- In Chisel, you define new modules by creating a class that extends `Module`
- You can create instances of that module using
  val myModule = Module(new ModuleClass)
- We need some way to instantiate a module that has no description in chisel.
- We do this by creating a dummy class that extends `BlackBox` and replicates the interface of the Verilog module.
- We can instantiate this dummy class and interact with it like any chisel module.
- When the Verilog files are generated, instances of blackboxes are left as simple Verilog instantiations.
- The name of the blackboxed module needs to match that of the Verilog module so that the Verilog compiler can properly resolve the instantiation.

Files when blackboxing

- Verilog file containing module
  - Contains the actual design of the module you are blackboxing
- Scala file containing the dummy class that extends `BlackBox`
  - Replicates the interface of the Verilog module
- Scala files that represent the rest of your chisel design
  - Use the blackboxed module by instantiating the dummy class like any other Module

Example Verilog Module

```
module vec_sum (
 ap_clk,
 ap_rst,
 ap_start,
 ap_done,
 ap_idle,
 ap_ready,
 vect_req_din,
 vect_req_full_n,
 vect_req_write,
 vect_rsp_empty_n,
 vect_rsp_read,
 vect_address,
 vect_datain,
 vect_dataout,
 vect_size,
 len,
 ap_return );

//Module Functionality Here...

endmodule
```

Example Chisel Blackbox

```
import Chisel._
class VecSumBlackbox() extends BlackBox() {
  val io = new Bundle {
    val ap_clk     = Bool(INPUT )
    val ap_rst     = Bool(INPUT )
    val ap_start   = Bool(INPUT )
    val ap_done    = Bool(OUTPUT)
    val ap_idle    = Bool(OUTPUT)
    val ap_ready   = Bool(OUTPUT)
    val ap_return  = Bits(OUTPUT, width = 64)
    val vect_req_din     = Bool(OUTPUT)
    val vect_req_full_n  = Bool(INPUT )
    val vect_req_write   = Bool(OUTPUT)
    val vect_rsp_empty_n=  Bool(INPUT )
    val vect_rsp_read    = Bool(OUTPUT)
    val vect_address     = Bits(OUTPUT,width = 32)
    val vect_datain      = Bits(INPUT ,width = 64)
    val vect_dataout     = Bits(OUTPUT,width = 64)
    val vect_size        = Bits(OUTPUT,width = 32)
    val scalar_io  = Bits(INPUT, width = 64)
}

//Continued to the right ...
```

```
  //set names of ports (note that the val name
  //does not need to match the wire name
  io.ap.start.setName("ap_start")
  io.ap.done.setName("ap_done")
  io.ap.idle.setName("ap_idle")
  io.ap.ready.setName("ap_ready")
  io.ap.rtn.setName("ap_return")
  io.vect_req_din.setName("vect_req_din")
  io.vect_req_full_n.setName("vect_req_full_n")
  io.vect_req_write.setName("vect_req_write")
  io.vect_rsp_empty_n.setName("vect_rsp_empty_n")
  io.vect_rsp_read.setName("vect_rsp_read")
  io.vect_address.setName("vect_address")
  io.vect_datain.setName("vect_datain")
  io.vect_dataout.setName("vect_dataout")
  io.vect_size.setName("vect_size")
  io.scalar_io.setName("len")\

  //Add explicit clock
  addClock(Driver.implicitClock)

  //Rename the clock and reset lines to match
  //verilog
  renameClock("clk", "ap_clk")
  renameReset("ap_rst")

  //set module name to match verilog
  moduleName = "vec_sum"
}
```

Compiling your design with blackboxes

- The Chisel C++ emulator will no longer work because it does not know the function of the blackboxes
- The tools will not know where to find your Verilog modules the way the Makefiles are currently set up.
- A good place to put your Verilog modules is in
  `src/main/verilog/`
- You will need to modify the Makefiles to add that directory to add your Verilog files to the source lists used by VCS and DC
- SRAMs actually use a similar approach
  - Look through the Makefrags for srams_v and srams_dir
  - Note that the srams-v Verilog files are passed to VCS in the vcs-sim-rtl Makefrag
  - Note that the srams_dir is included as a search path for DC and ICC. This can be found in their corresponding Makefrags
- A good tactic to include your own blackboxes is to mirror what the SRAMs do
- It may take a couple tries to modify the Makefiles