CS250 Discussion 2
RISC-V, Rocket, and RoCC

Spring 2016

Christopher Yarp

What’s new in Lab 2:

- In lab 1, you built a SHA3 unit that operates in isolation
- We would like Sha3Accel to act as an accelerator for a processor
- Lab 2 introduces the interface we will use to connect Sha3Accel to a processor

RISC-V

- RISC-V is a new Instruction Set Architecture (ISA) developed at the Aspire Lab
- It is designed to be a simple and open
- Is intended for education and research (although there is commercial interest as well)
- It is not architected for any particular microarchitecture (out-of-order, microcoded …)
- Has 32 bit, 64 bit, and 128 bit options for address space
- Supports the inclusion of accelerators by defining "custom" instruction in the ISA spec

More info at http://riscv.org/

Rocket

- Rocket is one implementation of the RISC–V ISA
- Rocket is a 64 bit implementation that has an integrated L1 and L2 data cache
- A special interface, known as the RoCC interface, was defined to help attach accelerators to Rocket
- We will be integrating Sha3Accel with Rocket

More info at https://github.com/ucb-bar/rocket-chip

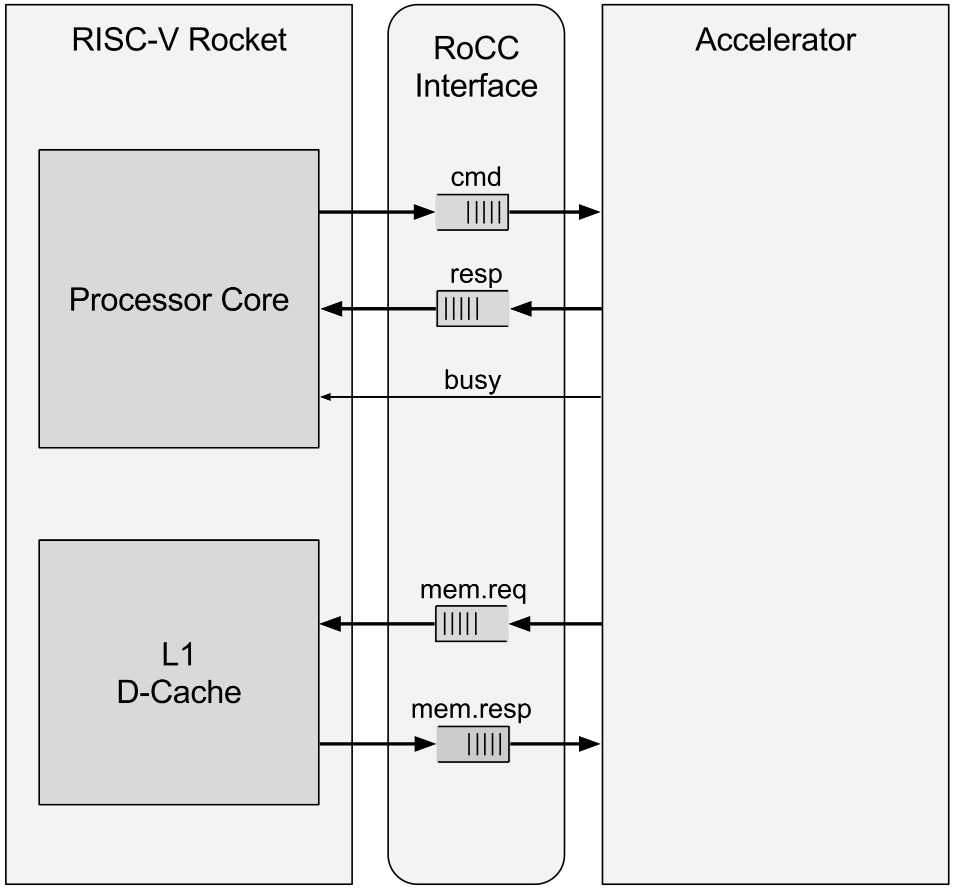
Custom Instruction Format

- The RISC-V specification is rather general on creating custom instructions
- The RoCC accelerators follow a standard instruction format
  - 2 register values can optionally be passed to the accelerator
  - An optional destination register can also be passed to the accelerator
  - A function code is passed to the accelerator and can be used to trigger specific behavior in the accelerator

31 25	24 20	19 15	14	13	12	11 7	6 0
funct7	rs2	rs1	xd	xs1	xs2	rd	opcode
7	5	5	1	1	1	5	7
roccinst[6:0]	src2	src1				dest	*custom-0/1/2/3*

The RoCC Interface

- The RoCC interface is split into several wires and bundles
  - cmd is a decoupled interface that carries the 2 register values along with the entire instruction
  - resp is a decoupled interface that carries the value to be written into the destination reg
  - busy signals to the processor that the accelerator is busy
  - mem.req is a decoupled interface that carries memory requests
  - mem.resp is a decoupled interface that carries a response to a mem request



Simplified View of RoCC

The Memory Sub-System

- The memory system operates in a *request-response* manner
- Load and store requests are passed to the memory system
- Later, a corresponding memory response will be passed to the accelerator
- Multiple memory transactions can be "in flight" at the same time
  - The number of "in flight" requests supported is specified when rocket is instantiated
- Transactions are **not** guaranteed to occur in order
- A *tag* field is used to differentiate responses

```
class RoCCInterface(implicit p: Parameters)
extends CoreBundle()(p)  {
        val cmd = Decoupled(new
        RoCCCommand).flip
        val resp = Decoupled(new  RoCCResponse)
        val mem = new
        HellaCacheIO()(p.alterPartial({ case
        CacheName => "L1D" }))
        val busy = Bool(OUTPUT)

        //many lines used for advanced features
        override  def cloneType = new
        RoCCInterface().asInstanceOf[this.type]
}
```

Bundles
Wires

The source for RoCC can be found in rocc.scala
https://github.com/ucb-bar/rocket/blob/master/src/main/scala/rocc.scala

```
class RoCCCommand(implicit  p:
Parameters) extends CoreBundle()(p)  {
        val inst = new  RoCCInstruction
        val rs1 = Bits(width  = xLen)
        val rs2 = Bits(width  = xLen)
}
```

```
class RoCCInstruction extends Bundle  {
        val funct  = Bits(width  = 7)
        val rs2 = Bits(width  = 5)
        val rs1 = Bits(width  = 5)
        val xd = Bool()
        val xs1 = Bool()
        val xs2 = Bool()
        val rd = Bits(width = 5)
        val opcode  = Bits(width = 7)
}
```

```
Class RoCCResponse(implicit p:
Parameters) extends CoreBundle()(p)  {
        val rd = Bits(width = 5)
        val data = Bits(width  = xLen)
}
```

```
class RoCCInterface(implicit p: Parameters)
extends CoreBundle()(p) {
        val cmd = Decoupled(new
        RoCCCommand).flip
        val resp = Decoupled(new RoCCResponse)
        val mem = new
        HellaCacheIO()(p.alterPartial({ case
        CacheName => "L1D" }))
        val busy = Bool(OUTPUT)
        //many lines used for advanced features …

        override def cloneType = new
        RoCCInterface().asInstanceOf[this.type]
}
```

```
class HellaCacheIO(implicit p: Parameters)
extends CoreBundle()(p) {
        val req = Decoupled(new HellaCacheReq)
        val resp = Valid(new HellaCacheResp).flip
        //more lines we don't use
}
```

Bundles
Wires

The source for the cache can be found in nbdcache.scala
https://github.com/ucb-bar/rocket/blob/master/src/main/scala/nbdcache.scala

```
//Class is comprised of many inherited traits
//Effective interface is:
class HellaCacheReq(implicit p: Parameters){
        val addr = UInt(width = coreMaxAddrBits)
        val tag = Bits(width =
        coreDCacheReqTagBits)
        val cmd = Bits(width = M_SZ)
        val typ = Bits(width = MT_SZ)
        val data = Bits(width = coreDataBits)
}
```

```
//Class is comprised of many inherited traits
//Effective interface is:
class HellaCacheResp(implicit p: Parameters){
        val addr = UInt(width = coreMaxAddrBits)
        val tag = Bits(width =
        coreDCacheReqTagBits)
        val cmd = Bits(width = M_SZ)
        val typ = Bits(width = MT_SZ)
        val data = Bits(width = coreDataBits)
        //we don't typically use the greyed out
        wires above
        //more lines we don't use
}
```

Chisel Parameters -> CDE

- A decision was made to partition advanced chisel parameters into a separate package: Context Dependent Environments (CDE)
  - These parameters take the form of a key-value store
  - They are *different* from function parameters
- It has a similar syntax to advanced chisel parameters but a couple changes are required
  - import cde.{Parameters, Field, Ex, World, ViewSym, Knob, Dump, Config}
    import cde.Implicits._
  - class Sha3Accel()(implicit p: Parameters) extends SimpleRoCC()(p)

Scala Implicits

- Scala implicit parameters are just like regular parameters
  - You can pass a compatible argument to them just like you normally would in a function call
- However, if you do not pass an argument to the function when you call it, one will be filled in for you
  - The compiler will look into the current scope and attempt to identify a candidate to pass automatically

Information from http://docs.scala-lang.org/tutorials/tour/implicit-parameters.html and http://docs.scala-lang.org/tutorials/FAQ/finding-implicits.html

CDE Use of Implicits

- Instead of defining a global key-value store, modules using CDE receive a cde.Parameters object and pass a cde.Parameters object to each sub-module
  - The CDE module passed to the sub-modules can be the same as the parent or different
- Why do this?
  - Sometimes, you want parameterizations to changed based on the context within the design.
    - Ex. You may want one submodule to use a different width than another

Example of CDE in Lab 2

```
import cde.{Parameters, Field, Ex, World, ViewSym, Knob, Dump, Config}
import cde.Implicits._

case object WidthP extends Field[Int]
case object Stages extends Field[Int]

class Sha3Accel()(implicit p: Parameters) extends SimpleRoCC()(p) {
        //parameters
        val W = p(WidthP)
        val S = p(Stages)

        //more wires
}
```

CDE Parameters for Design Space Exploration

- If you parameterize your design, it is easy to try different configurations and observe tradeoffs
- Wouldn’t it be great if the process was automated?
- If you use CDE, there is an automated flow!
- The tools are called Jackhammer and bar-crawl
  - Jackhammer produces the different configurations
  - bar-crawl partitions and distributes the jobs across a cluster
- More on this later!

A Quick Example of a Configuration and Knobs

```
class DefaultConfig() extends Config {
  override val topDefinitions:World.TopDefs = {
    (pname,site,here) => pname match {
      case WidthP => 64
      case Stages => Knob("stages")
    }
  }
  override val topConstraints:List[ViewSym=>Ex[Boolean]] = List(
    ex => ex(WidthP) === 64,
    ex => ex(Stages) >= 1 && ex(Stages) <= 4 && (ex(Stages)%2 === 0 ||
ex(Stages) === 1)
  )
  override val knobValues:Any=>Any = {
    case "stages" => 1
  }
}
```