SiFive IC Agile Design Methodology: Principle and Implementation

Kevin Liu
Ph.D
Design Engineer, SiFive China

Nov 2019

Agenda

- Chisel
- Diplomacy
- Config
- TileLink
- Rocket-Chip

Chisel

- A domain-specific language (DSL) for generating RTLs,
  - Not High Level Synthesis (HLS).

- Embedded in Scala

https://github.com/freechipsproject/chisel3

https://github.com/ucb-bar/chisel-tutorial

https://github.com/freechipsproject/chisel-bootcamp

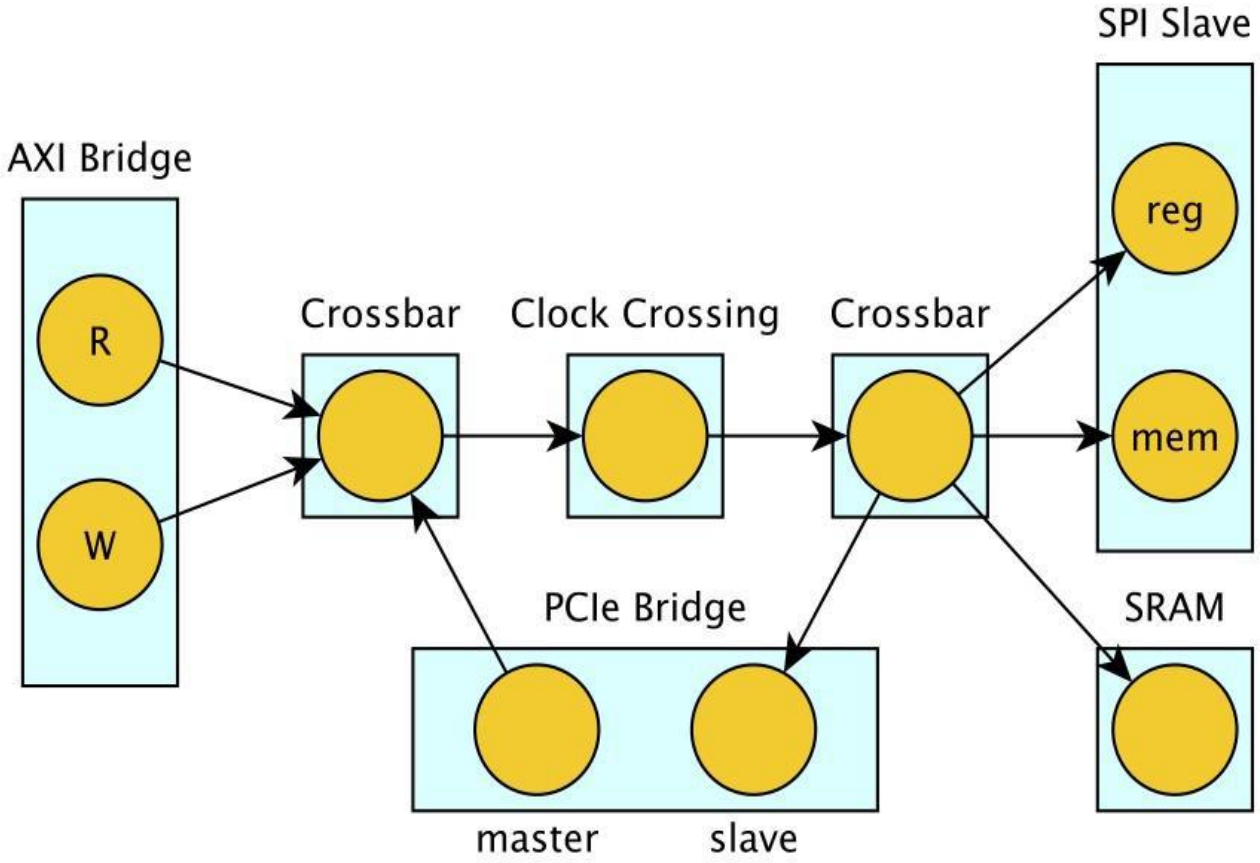
Chisel

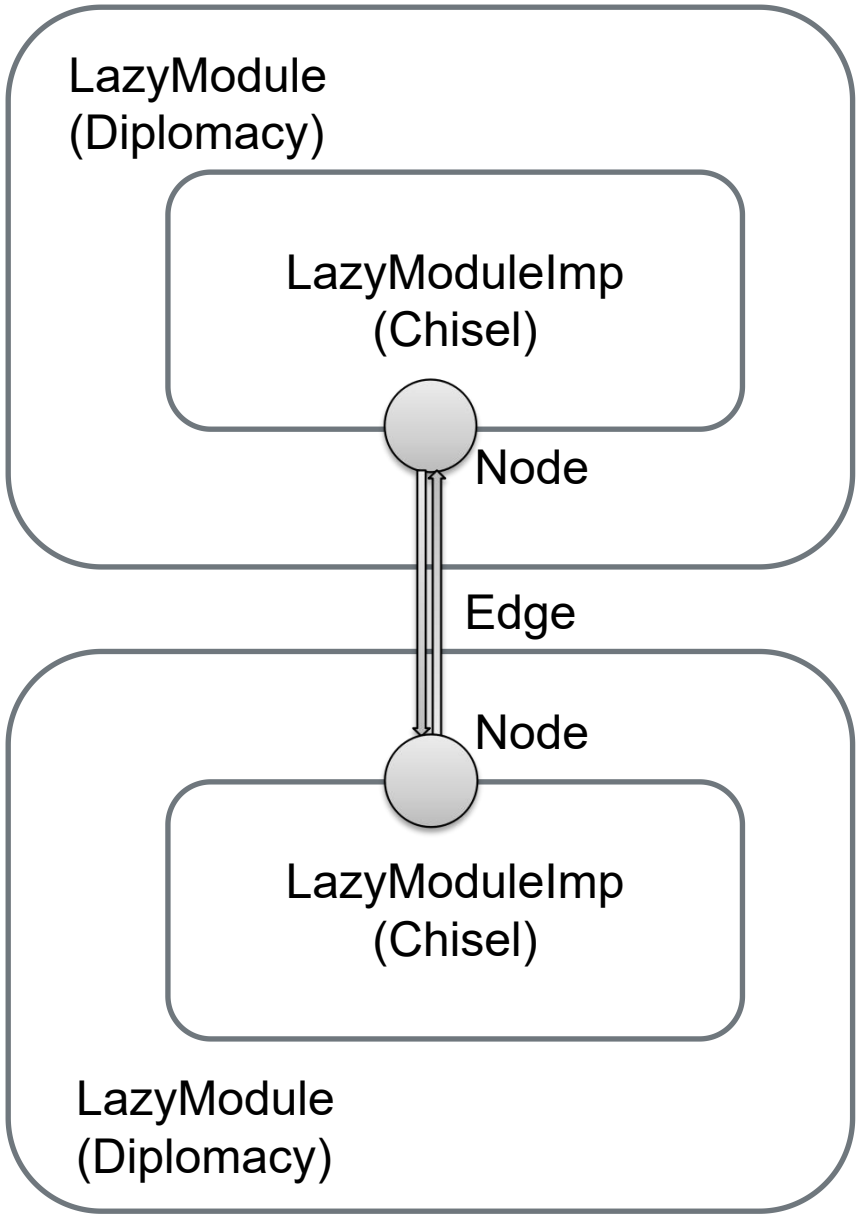
- However…
- Chisel isn't good enough on its own.
  - Need to be able to resolve interconnect misconfiguration among Chisel modules before generating any hardware for complex design.
  - Parameterization is a source of complexity.

Diplomacy

- Design:
  - A framework of interconnected components that can negotiate their mutual requirements for correct interconnection and operation.
- Abstraction: Graph of logical interconnectivity
  - Directed Acyclic Graph (DAG) of nodes and edges.
  - Node: A point where parameterized hardware is going to be generated.
  - Edge: A directed connection between one master and slave node.
  - LazyModule: Diplomatic Chisel modules that may have many nodes.
- Parameters are transmitted and derived among LazyModules through edges.

Diplomacy





Diplomacy

- What parameters can be diplomatic?
  - Presence or width of certain fields within data or control wire bundles
  - Type and size of operations issued by each master and slave
  - Operation capability of particular regions.

    (e.g. modifiability, executability, cacheability)
  - …

Diplomacy

- Phases of Diplomacy
  - Negotiation:
    - Graph is created by declaring edages between LazyModules' nodes.
    - A node can have multiple egdes

```
xbar.node := master0.node
xbar.node := master1.node
```

  - Elaboration:
    - Chisel hardware is elaborated in LazyModuleImps using the nodes' edges' parameters.

```
val my_wire = Wire(UInt(node.edges.in.bundle.width.W))
```

Config

- Design:
  - A globally parameter management and distribution system for specific designs.
  - Parameters can be fetched automatically and implicitlly from Config for circuit elaboration according to name keys.
- Component:
  - Field: Single 'key -> parameter' mapping.
  - Config: A data structure that stores Fields.
  - View: Functions that help find a certain Field in Config.
  - Parameters: Subclass of View, helps constuct Config.

Config

- Usage:
  - Define parameter and key:

```
case class myParam(width: Int)
case object myParamKey extends Field[myParam]
```

  - Set parameter value and construct Config for the design:

```
class myParamConfig extends Config( (View, View, View) => {
  case myParamKey => 8 })
class myDesignConfig extends Config(
  new myParamConfig ++ new BasicConfig)
```

  - Fetch actual parameter value via implicit Parameters:

```
class myModule(implicit p: Parameters) extends LazyModule{
  ...
  val myWidth =  p(myParamKey)
  val myWire = Wire(UInt(myWidth.W))
  ...
}
```

TileLink

	AHB	Wishbone	AXI4	ACE	CHI	TileLink
Open standard	✔	✔	✔	✔	✗	✔
Easy to implement	✗	✔		✗	?	✔
Cache block motion	✗	✗	✗	✔	✔	✔
Multiple cache layers	-	-	-	✗	?	✔
Reusable off-chip	✗				?	✔
High performance	✗	✔	✔		?	✔

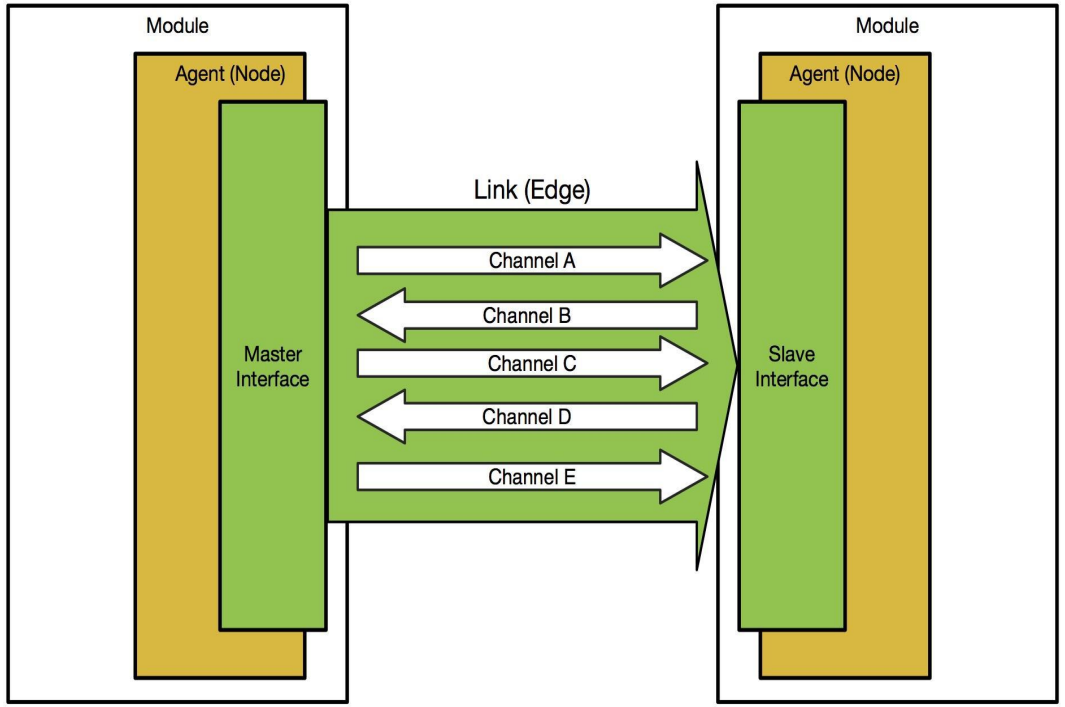
https://www.youtube.com/watch?v=EVITxp-SEp4&t=1242s

TileLink

- Open Source and In Production
  - Rocket-chip open source SOC is entirely built using TileLink
    - >30 public modules, include cores, caches, Xbars
    - broadcast-hub coherency manager
    - bridges to AXI/AHB/APB, clock crossing, fuzzer, monitor
- SiFive-blocks: open I2C, SPI, UART, GPIO, PWM TileLink slaves
- Chiplink for communication between chips

TileLink

- Directed Acyclic Graph (DAG) (again)
- Agent (Node) (again)
  - point where message are created
- Link (Edge) (again)
  - a directed pairing between two ager master and slave interface
- Modules may have many agents and agents may have many links



TileLink

	TL-UL	TL-UH	TL-C
Read/Write Operations	✔	✔	✔
Multibeat Messages		✔	✔
Atomic Operations		✔	✔
Hint (Prefetch) Operations		✔	✔
Cache Block Transfers			✔
Priorities B+C+E			✔

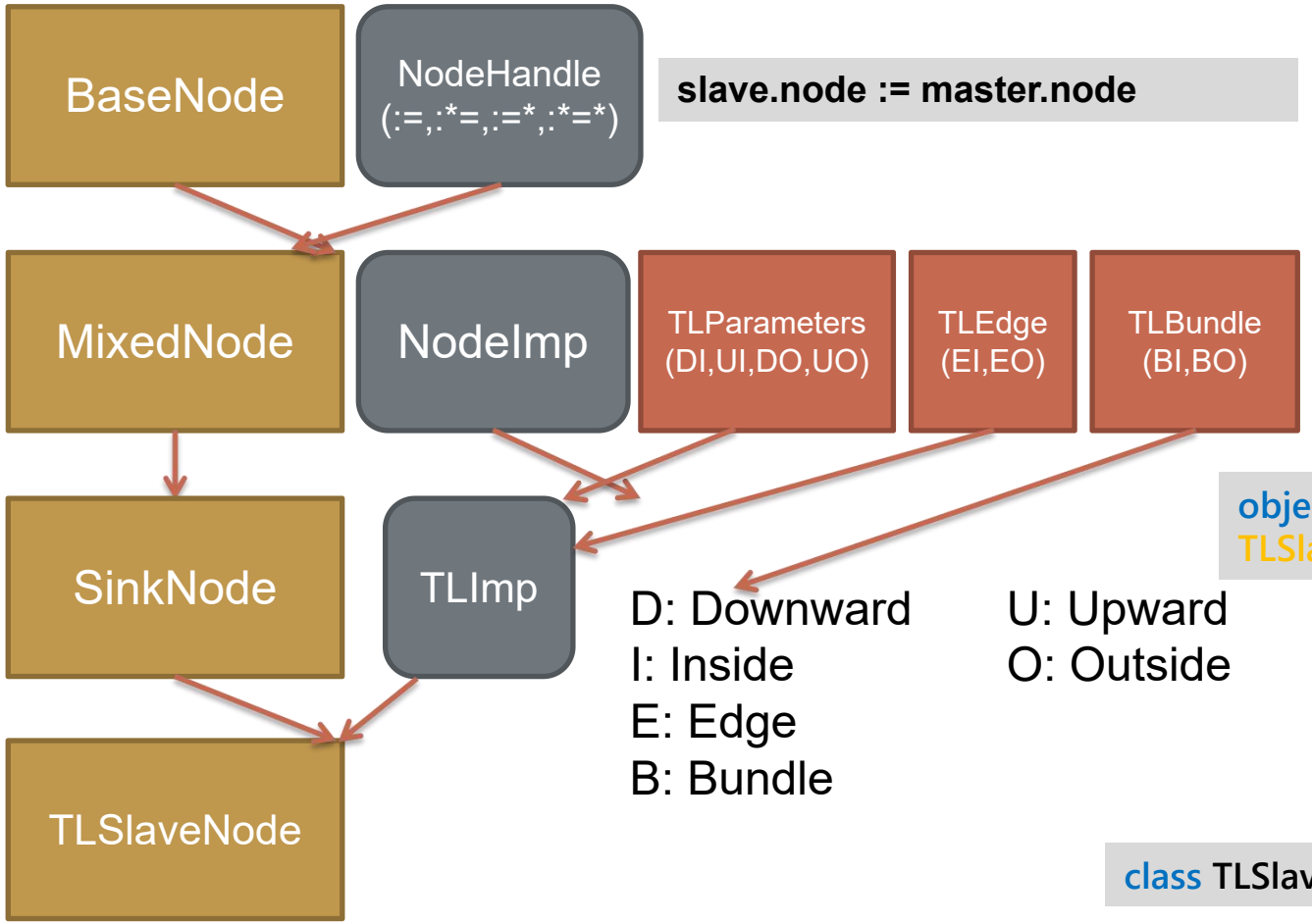
TileLink

- Implementation with Diplomacy

```
class TLSlaveNode extends SinkNode(TLImp) {...}

object TLImp extends NodeImp[TLMasterParameters, TLSlaveParameters,
TLEdgeOut, TLEdgeIn, TLBundle] {...}

class SinkNode[D, U, EO, EI, B <: Data](imp: NodeImp[D, U, EO, SI, B])
    extends MixedNode(imp, imp) {...}

abstract class MixedNode[DI, UI, EI, BI <: Data, DO, UO, EO, BO <: Data](
    val inner: InwardNodeImp[DI, UI, EI, BI],
    val outer: OutwardNodeImp[DO, UO, EI, BI])
    extends BaseNode with NodeHandle[DI,UI,EI,BI,DO,UO,EO,BO<:Data]
 {...}
```

TileLink

```
abstract class MixedNode[DI, UI, EI, BI <: Data, DO, UO, EO, BO <: Data]
extends BaseNode with NodeHandle[DI,UI,EI,BI,DO,UO,EO,BO<:Data]
```



slave.node := master.node

```
class SinkNode[D, U, EO, EI, B <: Data]
(imp: NodeImp[D, U, EO, SI, B])
extends MixedNode(imp, imp) {...}
```

```
object TLImp extends NodeImp[TLMasterParameters,
TLSlaveParameters, TLEdgeOut, TLEdgeIn, TLBundle] {...}
```

D: Downward
I: Inside
E: Edge
B: Bundle

U: Upward
O: Outside

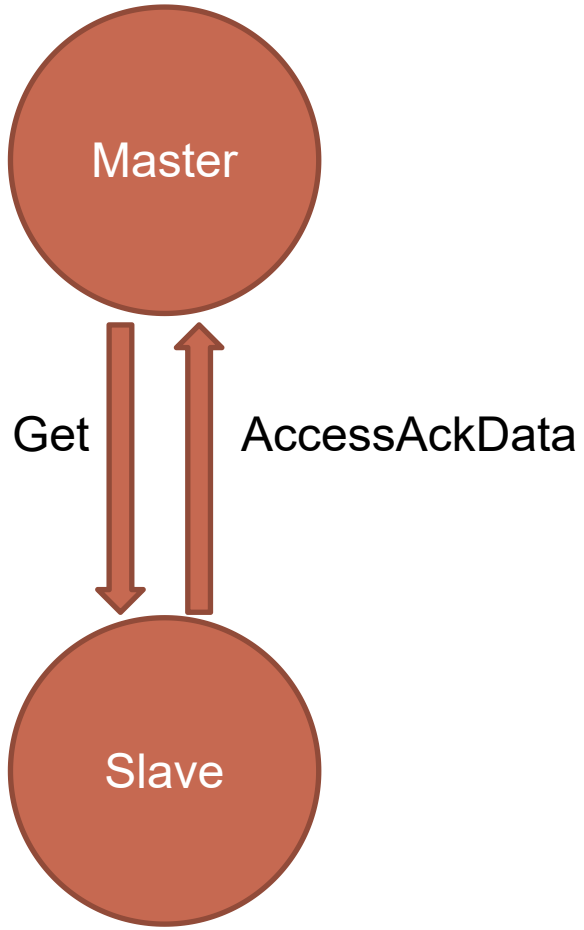
```
class TLSlaveNode extends SinkNode(TLImp) {...}
```

Cake Pattern

TileLink

- TLEdge:
  - Parameterized functions that issue tilelink operations

```
class TLEdgeOut extends TLEdge  {
  …
  def Get(fromSource: UInt, toAddress: UInt) = {
    val a = Wire(new TLBundleA)
    a.opcode := TLMessages.Get
    a.source := fromSource
    a.address := toAddress
    a }
  …
}
class TLEdgeIn extends TLEdge {
  …
  def  AccessAck(toSource: UInt, data: UInt) = {
    val d = Wire(new TLBundleD)
    d.opcode := TLMessages.AccessAckData
    d.source := toSource
    d.data  := data
    d }
  …
}
```



Rocket-Chip

- An open sourced SoC generator:
  - Rocket-tile: 5 stages in-order pipeline, with L1 I$, D$.
  - Tilelink Xbars
  - AMBA adapters
  - Peripheral Tilelink slave devices (SiFive-Blocks)
  - FPGA shells

https://github.com/chipsalliance/rocket-chip

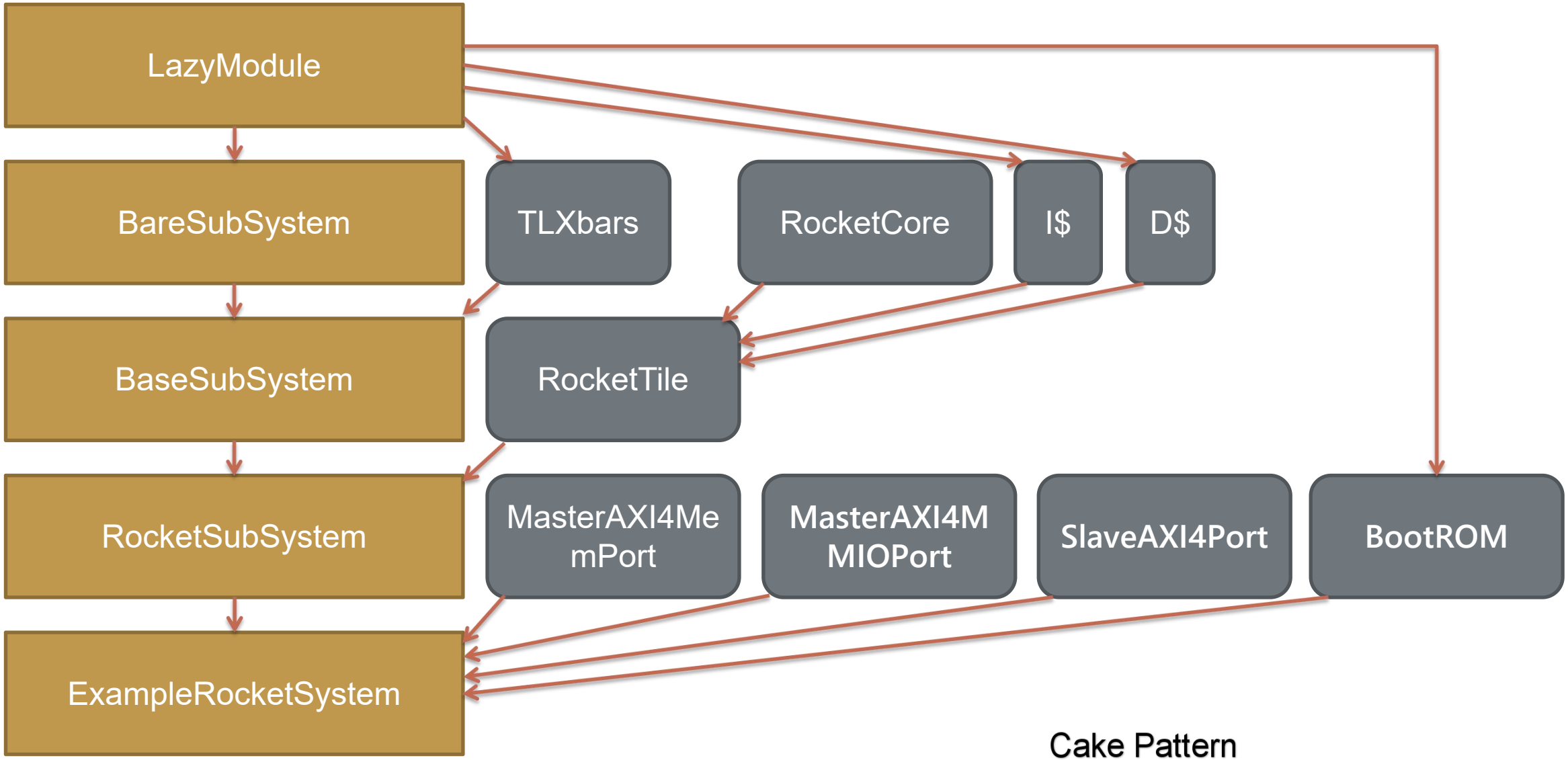
https://github.com/sifive/freedom

https://github.com/sifive/sifive-blocks/

Rocket-Chip

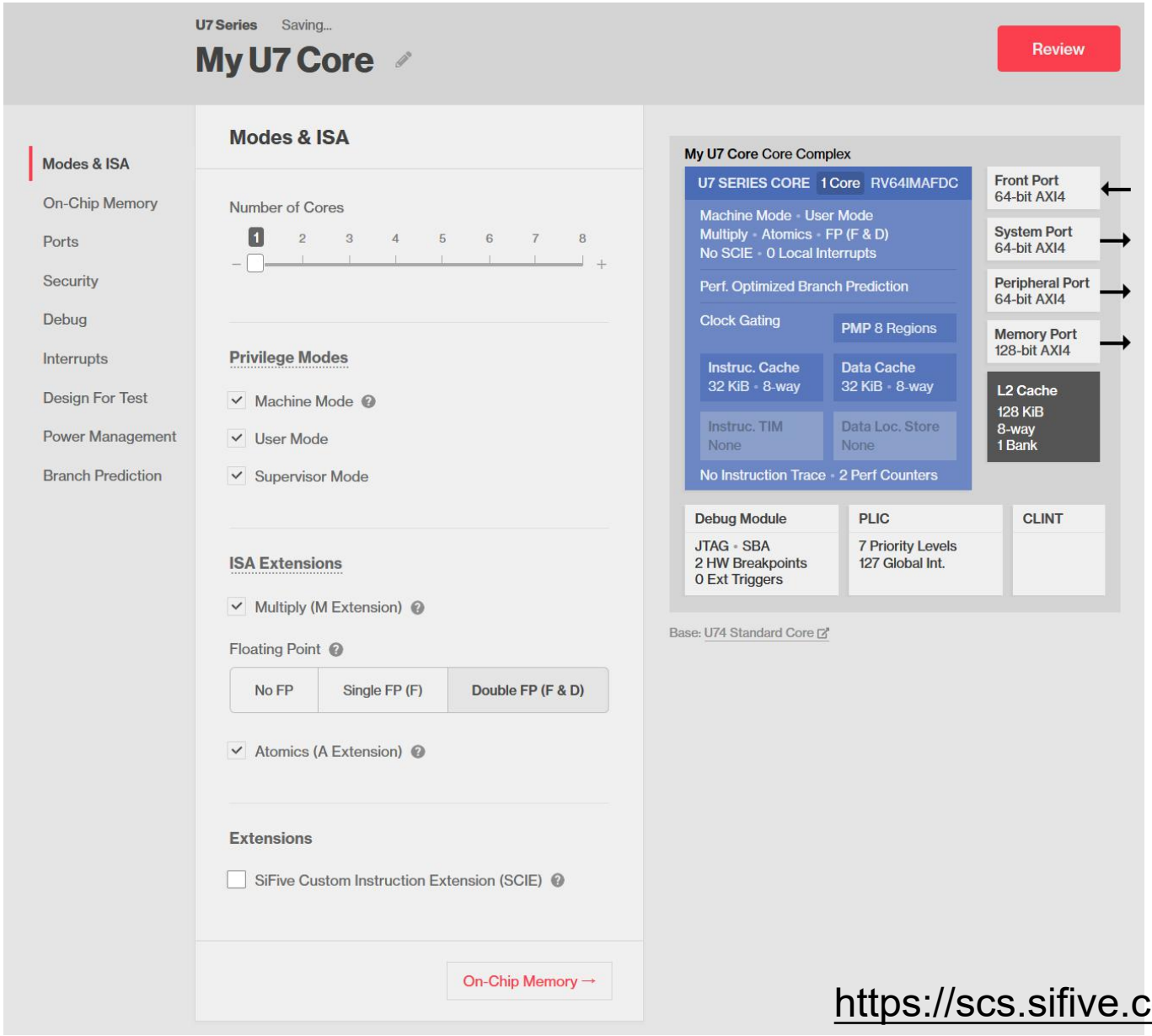
```
class ExampleRocketSystem (implicit p: Parameters) extends
RocketSubsystem
        with CanHaveMasterAXI4MemPort
        with CanHaveMasterAXI4MMIOPort
        with CanHaveSlaveAXI4Port
        with HasPeripheryBootROM{
  override lazy val module = new ExampleRocketSystemModuleImp(this)
}

class ExampleRocketSystemModuleImp extends
RocketSubsystemModuleImp
        with CanHaveMasterAXI4MemPortImp
        with CanHaveMasterAXI4MMIOPortImp
        with CanHaveSlaveAXI4PortImp
        with HasPeripheryBootROMImp
```

Rocket-Chip



Cake Pattern

SiFive Cloud Services



https://scs.sifive.com/core-designer/

We are hiring!

- CPU pipeline
- Chisel
- Cache
- Bus
- SoC
- Verification
- AI Algorithm
- Software

recruitment@sifive-china.com

End