Cake Pattern in RocketTiles.scala

The cake pattern involves two parallel chains of inheritance. It also involves "twins", where twins involve lazy modules. So, one twin establishes the presence of a circuit element, but doesn't have a concrete instance of it.. this is the outer twin. It's job is three fold -- it establishes the presence of the circuit element, and it establishes the software interface, and it takes place in Diplomacy calculations about the concrete details of the interface, such as how many bits wide, and how many copies, and most especially, which particular concrete implementation.

The other "twin" is an "inner" twin -- this is where you get instances of objects that are circuit elements. Not all inner twins instantiate an object, but if one is instantiated, it's inside an inner twin.

The idea is that the first twin establishes the presence of a circuit element (or a container that has many circuit elements or more containers, in a hierarchy). This is established statically. So when the program initializes, these first twins are present, and provide hooks by which the Diplomacy calculations can take place. Then, once the Diplomacy has completed, the second twins create the actual circuit elements and connects them together.

The way this is accomplished is via the Lazy Module mechanism.

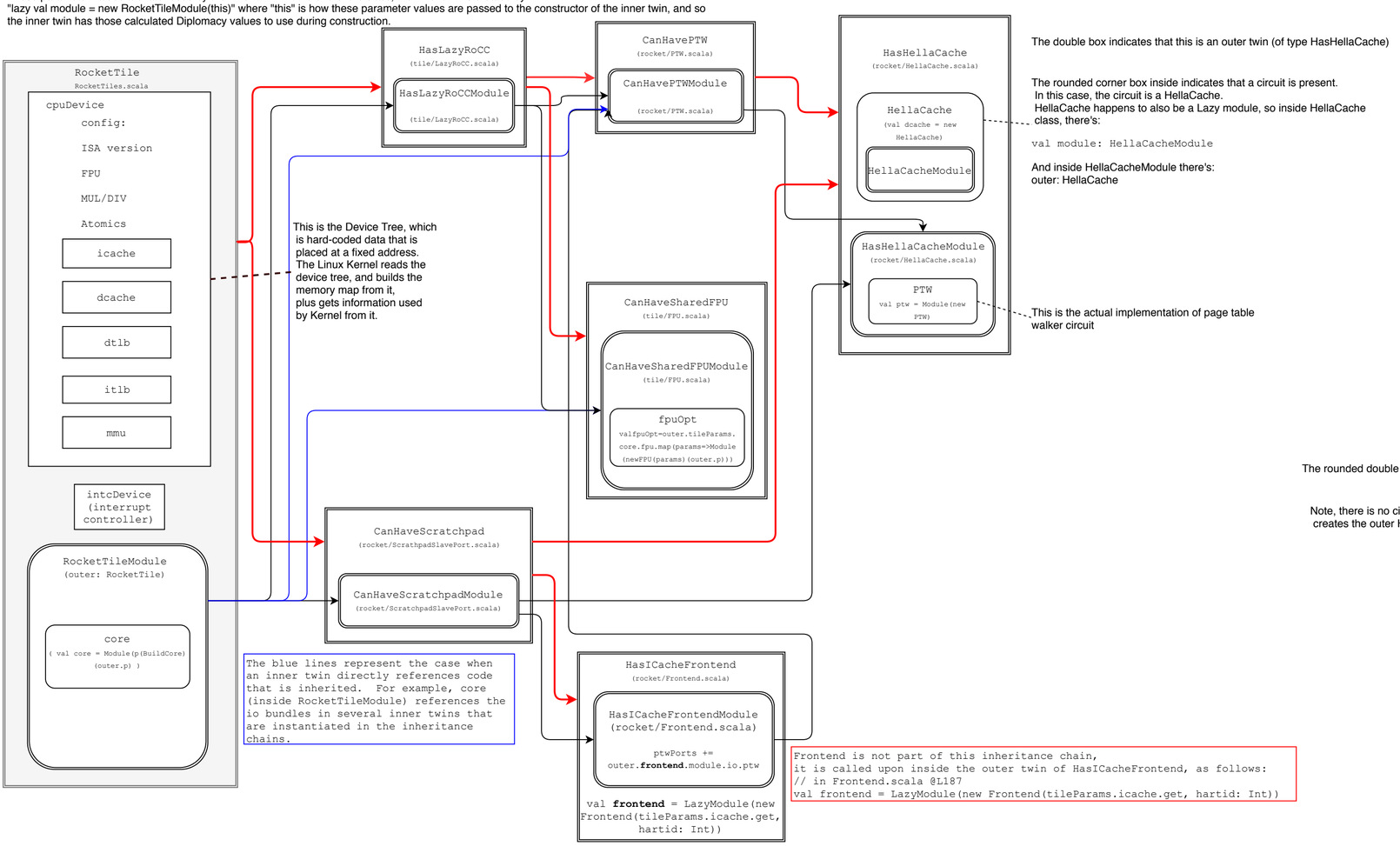
In the code, how it looks is that there is a rather empty outer twin class, and "trait" that defines the inner twin. (To make things complicated, the outer twin could also be a trait, but doesn't have to be, but the inner twin must be a trait). At the top of a particular hierarchy, the outer twin directly includes the inner twin, via:

```
lazy val module = new RocketTileModule(this)
```

Where this code is placed inside the outer twin's class definition (in this case, inside "class RocketTile" in the file RocketTiles.scala), and "new" creates an instance of the inner twin. Note that the name of the inner twin has a convention, in June 2017 it ends with "Module" but in newer it ends with "Impl" which is short for "Implementation".

The complication is that there isn't always an instantiation of the inner twin inside the class of the outer twin! The Cake Pattern allows long chains of inheritance. What's important is that, at the end of the chain, where the inner is, indeed, instantiated (as shown above) inside the outer, that each of the traits in the inner twin's chain do have a corresponding outer twin in the outer twin's chain.

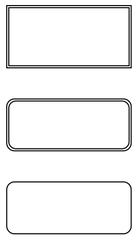
In the RocketTiles.scala code, the RocketTile class declares a cpuDevice ("val cpuDevice = new Device") -- this is not lazy and this is not an outer twin.. instead, this is part of the Diplomacy pattern -- this is where it calculates the set of values that define what should be included and their interfaces These parameter values are later used by the inner twin. At the end of the class, the inner twin is created by:
"lazy val module = new RocketTileModule(this)" where "this" is how these parameter values are passed to the constructor of the inner twin, and so the inner twin has those calculated Diplomacy values to use during construction.



Note that the Cake pattern, and the underlying Lazy Module, is mainly used in order to support Diplomacy. In the background, Diplomacy calculates parameter values. It only instantiates the lazy modules after parameter value negotiation among connected pieces of an SoC has completed. This calculation of parameter values is hidden from the implementation.

Legend

The shapes have specific meaning, which relates to the code



Means an "outer twin"
The text says the name of the class or trait so you can search the code for that string and find the definition

Means an "inner twin"
The text says the name of the class or trait so you can search the code for that string and find the definition

Means an object instantiated inside an inner twin
The text says the line of code that creates the object

The double box indicates that this is an outer twin (of type HasHellaCache)

The rounded corner box inside indicates that a circuit is present.
In this case, the circuit is a HellaCache.
HellaCache happens to also be a Lazy module, so inside HellaCache class, there's:

```
val module: HellaCacheModule
```

And inside HellaCacheModule there's:
outer: HellaCache

This is the actual implementation of page table walker circuit

The rounded double box indicates that this is an inner twin (of type `HasHellaCacheModule`)

Note, there is no circuit instantiated inside the inner twin -- the outer HasHellaCache twin creates the outer HellaCache, which also creates its own twin inside itself.

```
/** Mix-ins for constructing tiles that might have a PTW */
// rocket/PTW.scala @L203-217
trait CanHavePTW extends HasHellaCache {                    <-- outer twin
  ...
  val module: CanHavePTWModule
  ... }

trait CanHavePTWModule extends HasHellaCacheModule {        <-- inner twin
  val outer: CanHavePTW
  ...
  val ptw = Module(new PTW(outer.nPTWPorts)(outer.dcache.node.edgesOut(0), outer.p))
}
```

```
/** Mix-ins for constructing tiles that may have an FPU external to the core pipeline */
// in tile/FPU.scala @L807-811
trait CanHaveSharedFPU extends HasTileParameters      <-- outer twin

trait CanHaveSharedFPUModule {                        <-- inner twin
  val outer: CanHaveSharedFPU
  ... }
```

```
/** Mixins for including RoCC **/
// in tile/LazyRoCC.scala @L80-145

trait HasLazyRoCC extends CanHaveSharedFPU with CanHavePTW with HasTileLinkMasterPort {   <-- outer twin
  implicit val p: Parameters
  val module: HasLazyRoCCModule
  ... }

trait HasLazyRoCCModule extends CanHaveSharedFPUModule                                    <-- inner twin
  with CanHavePTWModule
  with HasCoreParameters
  with HasTileLinkMasterPortModule {
  val outer: HasLazyRoCC
  ... }
```

```
/** Mix-ins for constructing tiles that have optional scratchpads */
// in rocket/ScratchpadSlavePort.scala @L101-138
trait CanHaveScratchpad extends HasHellaCache with HasICacheFrontend with HasCoreParameters {   <-- outer twin
  val module: CanHaveScratchpadModule
  ... }

trait CanHaveScratchpadModule extends HasHellaCacheModule with HasICacheFrontendModule {        <-- inner twin
  val outer: CanHaveScratchpad
  ... }
```

```
/** Mix-ins for constructing tiles that have an ICache-based pipeline frontend */
// in rocket/Frontend.scala @L185-200
trait HasICacheFrontend extends CanHavePTW with HasTileLinkMasterPort {                    <-- outer twin
  val module: HasICacheFrontendModule
  ... }

trait HasICacheFrontendModule extends CanHavePTWModule with HasTileLinkMasterPortModule {  <-- inner twin
  val outer: HasICacheFrontend
  ... }
```