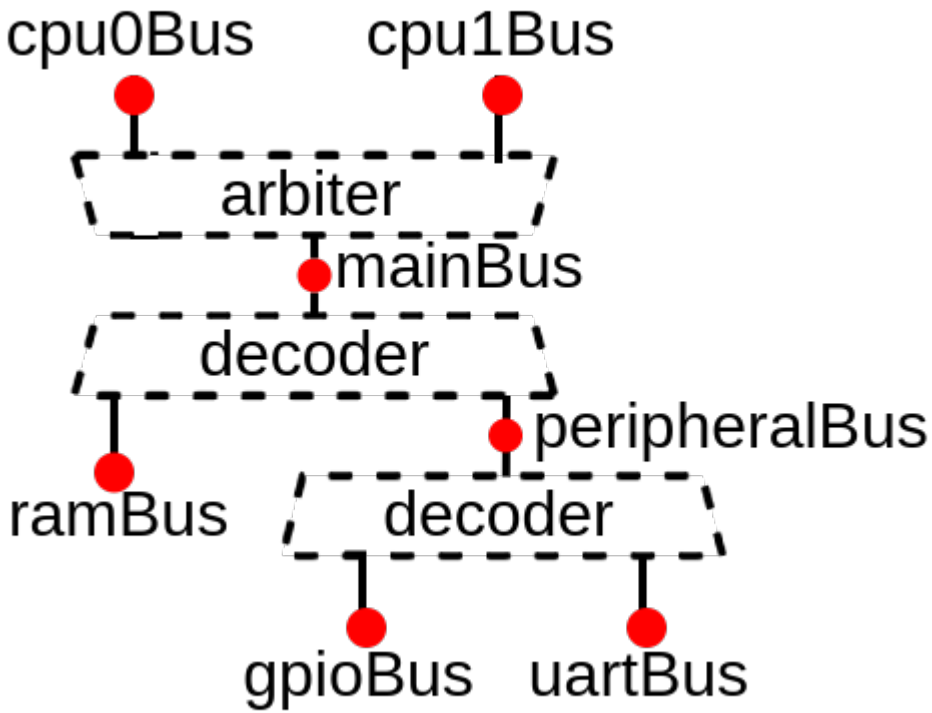
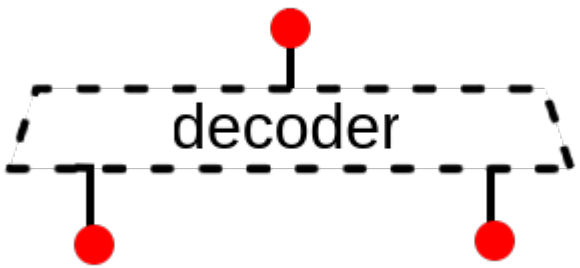
A progressive introduction to memory bus interconnect API in Software-Defined Hardware



Charles Papon

Software API

```
val commonBus = Apb3(20,32)
```

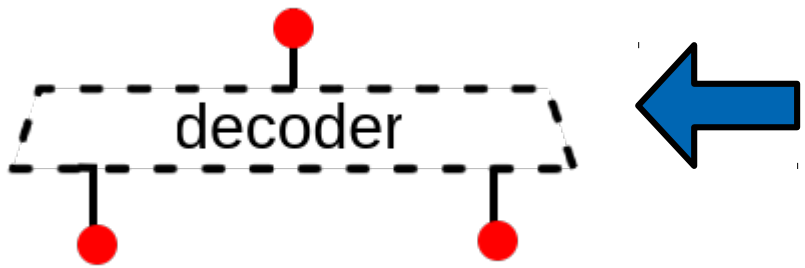



```
val uartBus = Apb3(12, 32)
```

```
val gpioBus = Apb3(12, 32)
```

Software API

```
val commonBus = Apb3(20,32)
```



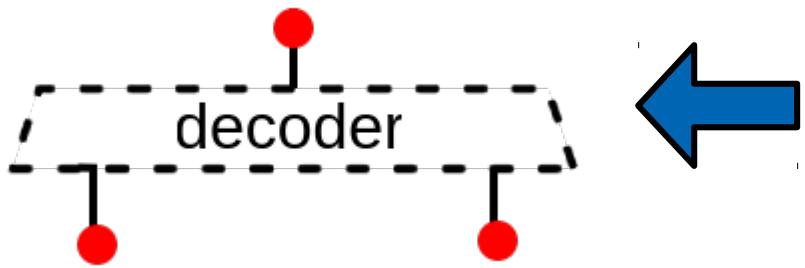

```
val apbDecoder = Apb3Decoder(
  master = commonBus,
  slaves = List(
    gpioBus -> (0x2000, 4 kB),
    uartBus -> (0x5000, 4 kB)
  )
)
```

```
val uartBus = Apb3(12, 32)
```

```
val gpioBus = Apb3(12, 32)
```

Software API

```
val commonBus = Apb3(20,32)
```



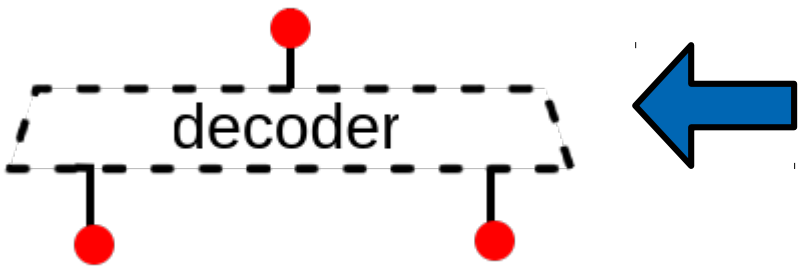

```
val apbDecoder = Apb3Decoder(
  master = commonBus,
  slaves = List(
    gpioBus -> (0x2000, 4 kB),
    uartBus -> (0x5000, 4 kB)
  )
)
```

```
val uartBus = Apb3(12, 32)
```

```
val gpioBus = Apb3(12, 32)
```

Software API

```
val commonBus = Apb3(20,32)
```

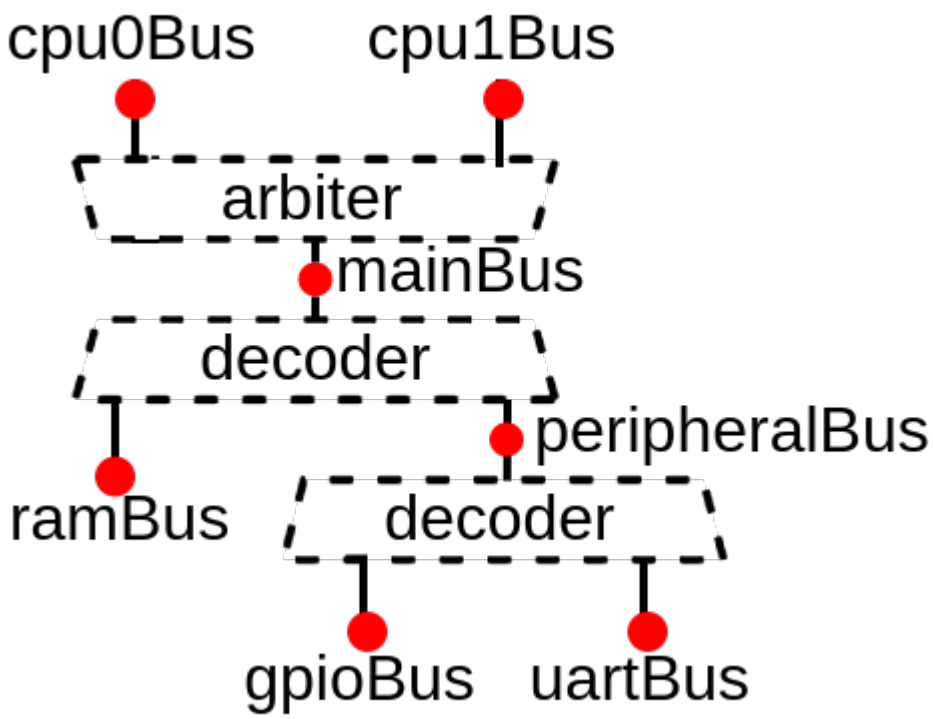



```
val apbDecoder = Apb3Decoder(
  master = commonBus,
  slaves = List(
    gpioBus -> (0x2000, 4 kB),
    uartBus -> (0x5000, 4 kB)
  )
)
```

```
val uartBus = Apb3(12, 32)
```

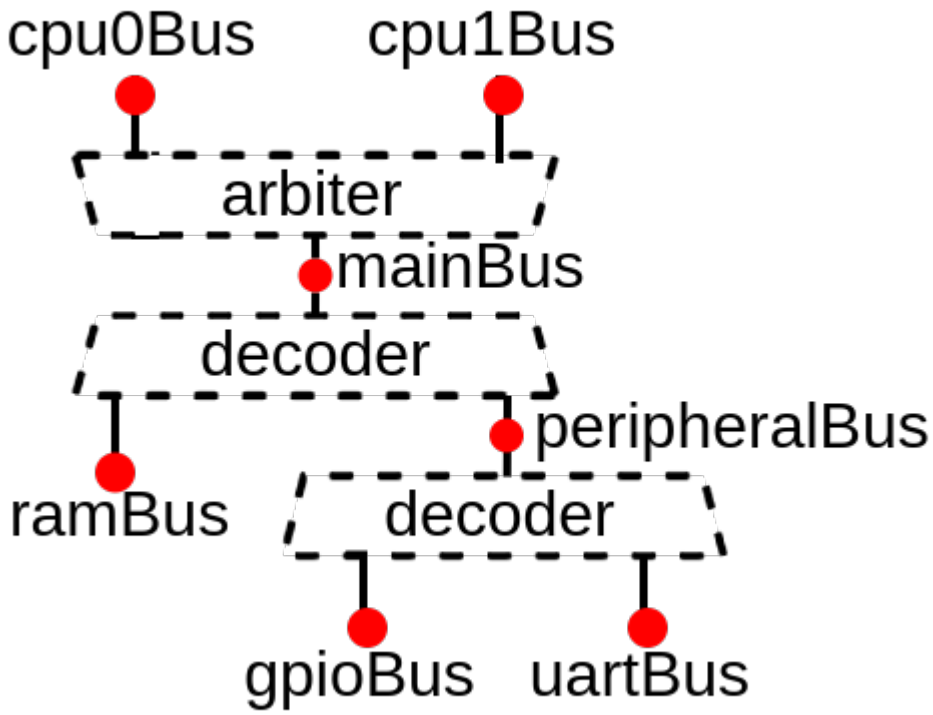
```
val gpioBus = Apb3(12, 32)
```

Automation



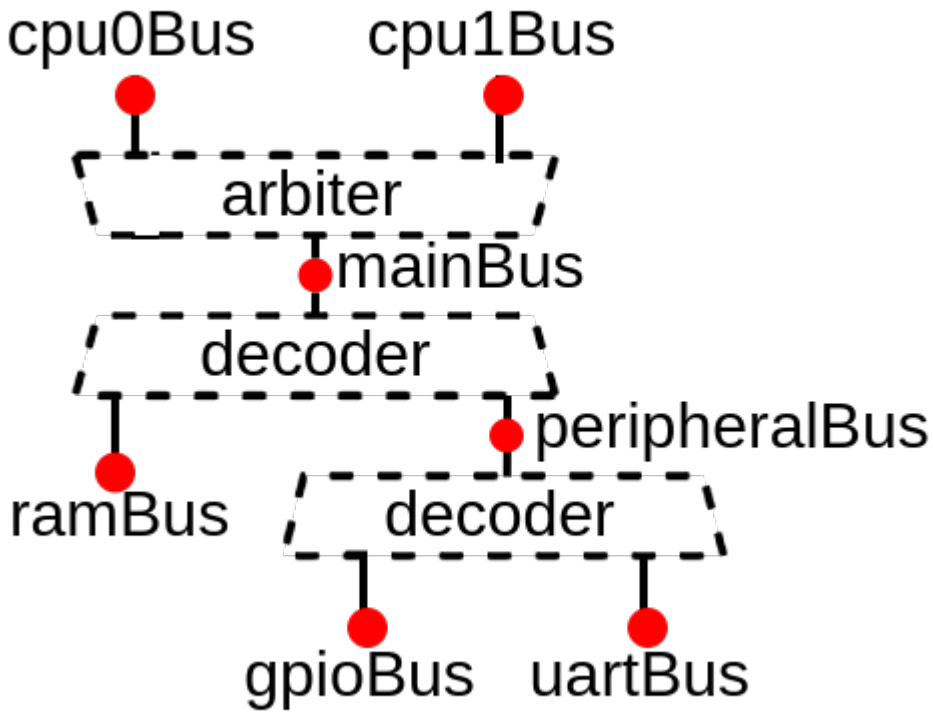

```
val cpu0Bus, cpu1Bus = Axi4(32, 32, 2)
val mainBus          = Axi4(32, 32, 4)
val ramBus           = Axi4(16, 32, 6)
val peripheralBus    = Axi4(20, 32, 6)
val gpioBus, uartBus = Axi4(12, 32, 8)

val axiCrossbar = Axi4CrossbarFactory()

axiCrossbar.addSlaves(
  mainBus        -> (0x00000000,  4 GB),
  ramBus         -> (0x80000000, 64 kB),
  peripheralBus -> (0x10000000,  1 MB),
  gpioBus        -> (    0x2000,  4 kB),
  uartBus        -> (    0x5000,  4 kB)
)

axiCrossbar.addConnections(
  cpu0Bus        -> List(mainBus),
  cpu1Bus        -> List(mainBus),
  mainBus        -> List(ramBus, peripheralBus),
  peripheralBus -> List(gpioBus, uartBus)
)

axiCrossbar.build()
```

Automation



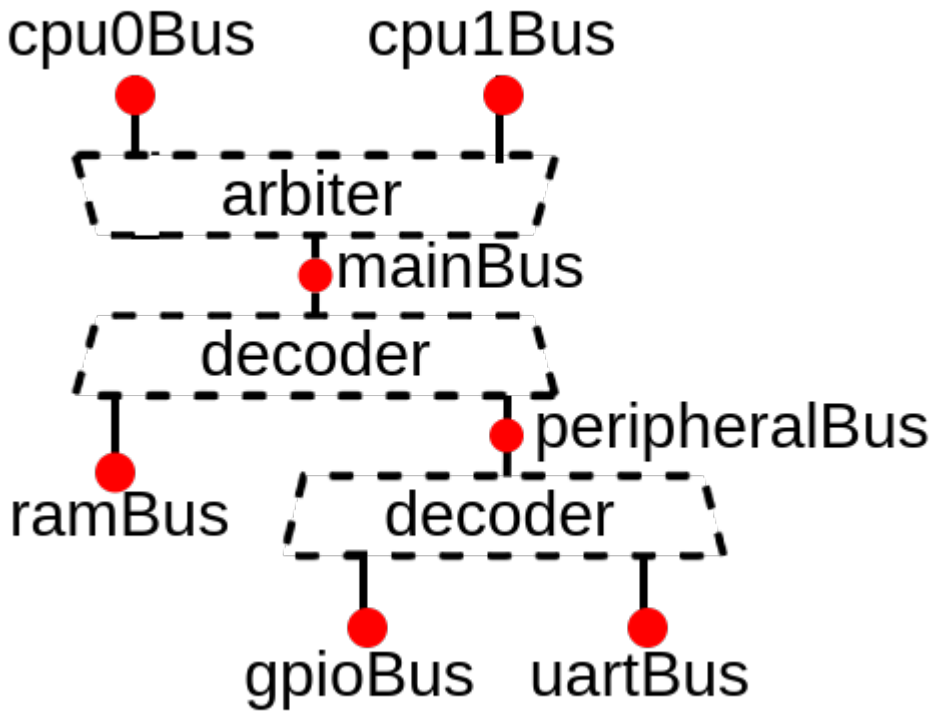

```
val cpu0Bus, cpu1Bus = Axi4(32, 32, 2)
val mainBus          = Axi4(32, 32, 4)
val ramBus           = Axi4(16, 32, 6)
val peripheralBus    = Axi4(20, 32, 6)
val gpioBus, uartBus = Axi4(12, 32, 8)

val axiCrossbar = Axi4CrossbarFactory()

axiCrossbar.addSlaves(
  mainBus        -> (0x00000000,  4 GB),
  ramBus         -> (0x80000000, 64 kB),
  peripheralBus -> (0x10000000,  1 MB),
  gpioBus        -> (    0x2000,  4 kB),
  uartBus        -> (    0x5000,  4 kB)
)

axiCrossbar.addConnections(
  cpu0Bus        -> List(mainBus),
  cpu1Bus        -> List(mainBus),
  mainBus        -> List(ramBus, peripheralBus),
  peripheralBus -> List(gpioBus, uartBus)
)

axiCrossbar.build()
```

Automation



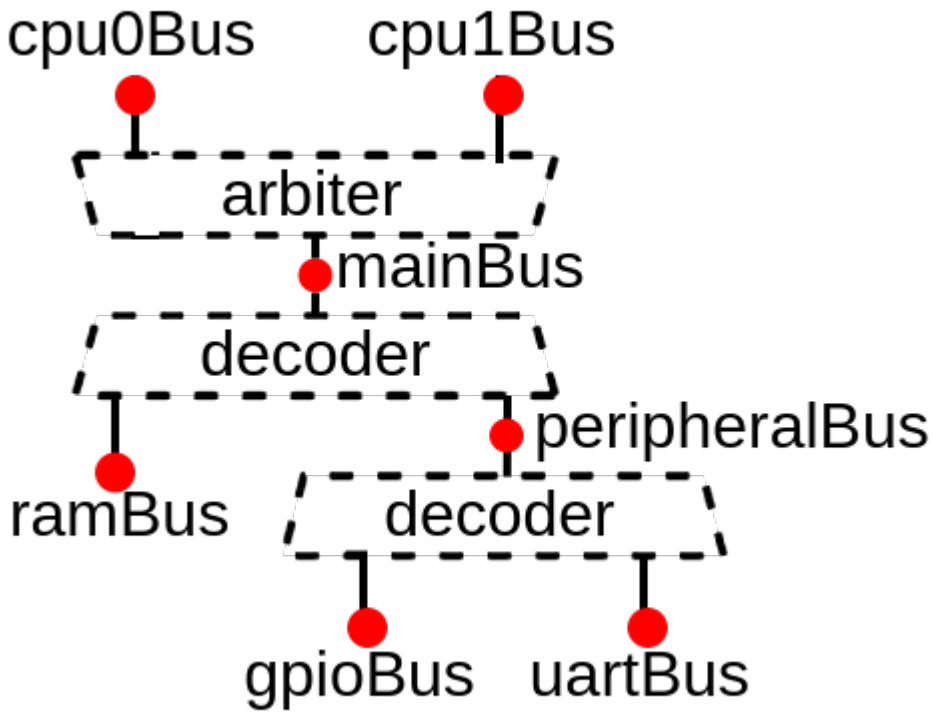

```
val cpu0Bus, cpu1Bus = Axi4(32, 32, 2)
val mainBus          = Axi4(32, 32, 4)
val ramBus           = Axi4(16, 32, 6)
val peripheralBus    = Axi4(20, 32, 6)
val gpioBus, uartBus = Axi4(12, 32, 8)

val axiCrossbar = Axi4CrossbarFactory()

axiCrossbar.addSlaves(
  mainBus        -> (0x00000000,  4 GB),
  ramBus         -> (0x80000000, 64 kB),
  peripheralBus -> (0x10000000,  1 MB),
  gpioBus        -> (    0x2000,  4 kB),
  uartBus        -> (    0x5000,  4 kB)
)

axiCrossbar.addConnections(
  cpu0Bus        -> List(mainBus),
  cpu1Bus        -> List(mainBus),
  mainBus        -> List(ramBus, peripheralBus),
  peripheralBus -> List(gpioBus, uartBus)
)

axiCrossbar.build()
```

Automation



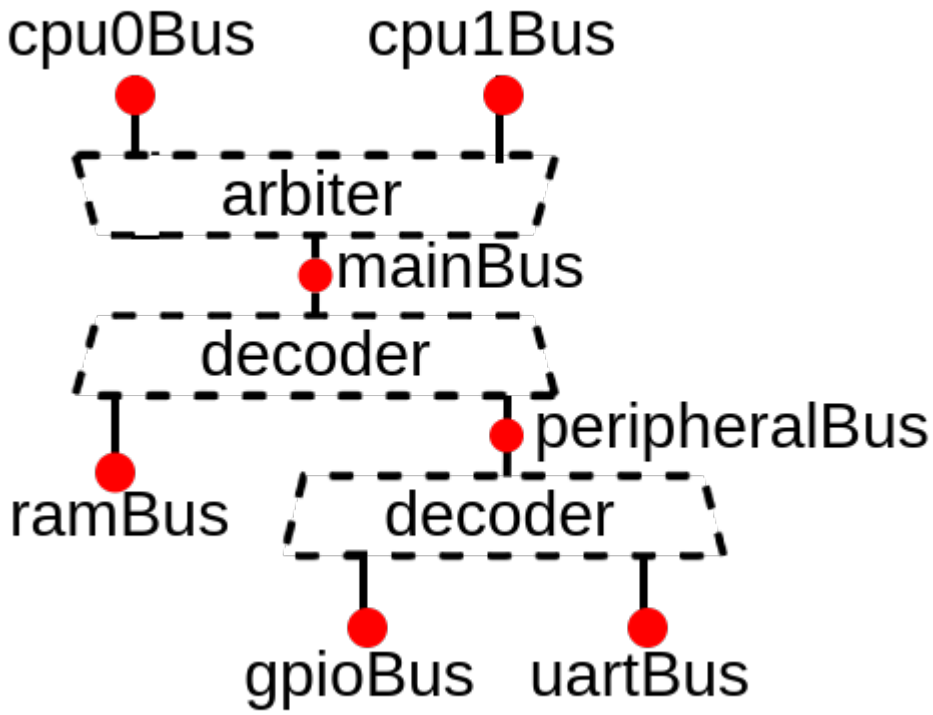

```
val cpu0Bus, cpu1Bus = Axi4(32, 32, 2)
val mainBus          = Axi4(32, 32, 4)
val ramBus           = Axi4(16, 32, 6)
val peripheralBus    = Axi4(20, 32, 6)
val gpioBus, uartBus = Axi4(12, 32, 8)

val axiCrossbar = Axi4CrossbarFactory()

axiCrossbar.addSlaves(
  mainBus        -> (0x00000000,  4 GB),
  ramBus         -> (0x80000000, 64 kB),
  peripheralBus -> (0x10000000,  1 MB),
  gpioBus        -> (    0x2000,  4 kB),
  uartBus        -> (    0x5000,  4 kB)
)

axiCrossbar.addConnections(
  cpu0Bus        -> List(mainBus),
  cpu1Bus        -> List(mainBus),
  mainBus        -> List(ramBus, peripheralBus),
  peripheralBus -> List(gpioBus, uartBus)
)

axiCrossbar.build()
```

Automation




```
val cpu0Bus, cpu1Bus = Axi4(32, 32, 2)
val mainBus          = Axi4(32, 32, 4)
val ramBus           = Axi4(16, 32, 6)
val peripheralBus    = Axi4(20, 32, 6)
val gpioBus, uartBus = Axi4(12, 32, 8)

val axiCrossbar = Axi4CrossbarFactory()

axiCrossbar.addSlaves(
  mainBus        -> (0x00000000,  4 GB),
  ramBus         -> (0x80000000, 64 kB),
  peripheralBus  -> (0x10000000,  1 MB),
  gpioBus        -> (    0x2000,  4 kB),
  uartBus        -> (    0x5000,  4 kB)
)

axiCrossbar.addConnections(
  cpu0Bus        -> List(mainBus),
  cpu1Bus        -> List(mainBus),
  mainBus        -> List(ramBus, peripheralBus),
  peripheralBus  -> List(gpioBus, uartBus)
)

axiCrossbar.build()
```

Automation




```
val cpu0Bus, cpu1Bus = Axi4(32, 32, 2)
val mainBus          = Axi4(32, 32, 4)
val ramBus           = Axi4(16, 32, 6)
val peripheralBus    = Axi4(20, 32, 6)
val gpioBus, uartBus = Axi4(12, 32, 8)

val axiCrossbar = Axi4CrossbarFactory()

axiCrossbar.addSlaves(
  mainBus        -> (0x00000000,  4 GB),
  ramBus         -> (0x80000000, 64 kB),
  peripheralBus  -> (0x10000000,  1 MB),
  gpioBus        -> (    0x2000,  4 kB),
  uartBus        -> (    0x5000,  4 kB)
)

axiCrossbar.addConnections(
  cpu0Bus        -> List(mainBus),
  cpu1Bus        -> List(mainBus),
  mainBus        -> List(ramBus, peripheralBus),
  peripheralBus  -> List(gpioBus, uartBus)
)

axiCrossbar.build()
```

Issues

- Centralized monolith
- No parameter negotiation / propagation
  - Data width
  - Address width
  - Type of accesses (read / write / atomic / ...)
  - ID width
  - ID to master mapping (Tilelink)
- Lack of awareness
  - Memory mapping from a given master perspective
  - Physical memory attributes

Issues

- Centralized monolith
- No parameter negotiation / propagation
  - Data width
  - Address width
  - Type of accesses (read / write / atomic / ...)
  - ID width
  - ID to master mapping (Tilelink)
- Lack of awareness
  - Memory mapping from a given master perspective
  - Physical memory attributes

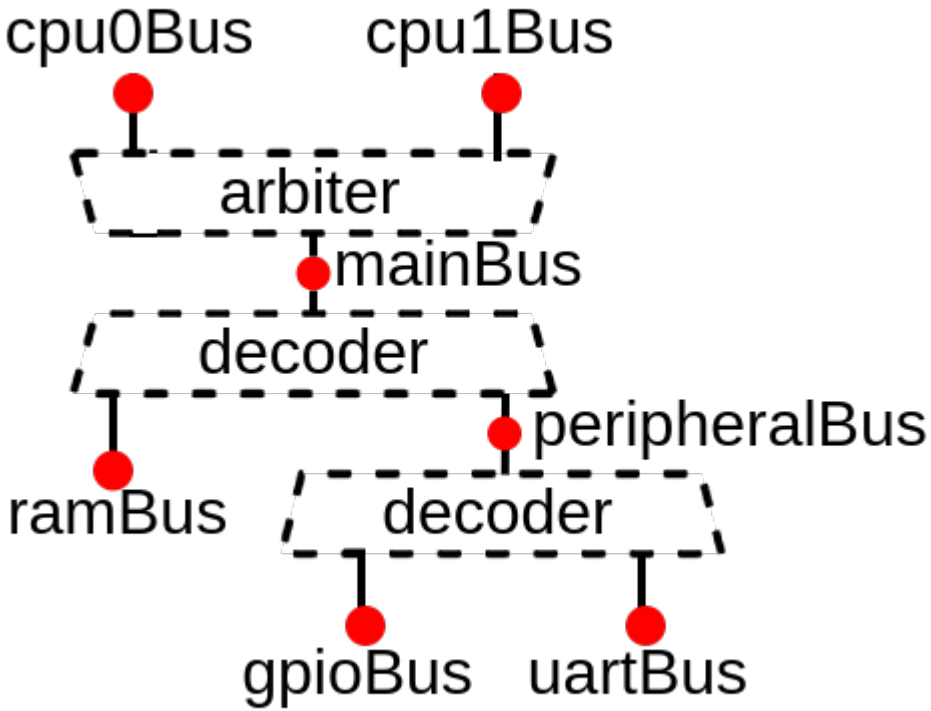
Issues

- Centralized monolith
- No parameter negotiation / propagation
  - Data width
  - Address width
  - Type of accesses (read / write / atomic / ...)
  - ID width
  - ID to master mapping (Tilelink)
- Lack of awareness
  - Memory mapping from a given master perspective
  - Physical memory attributes

Issues

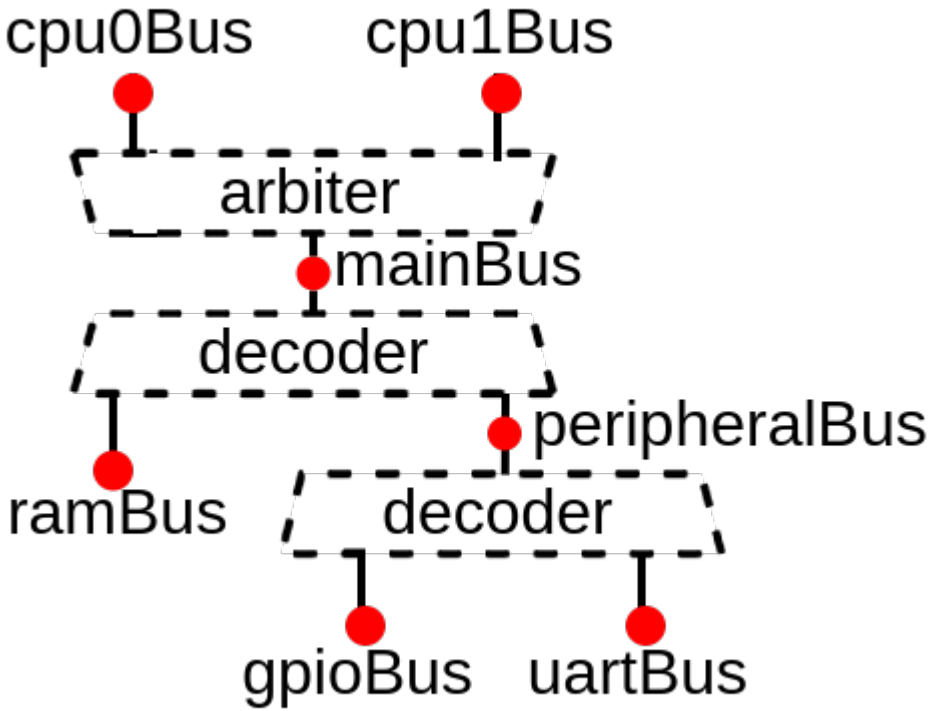
- Centralized monolith
- No parameter negotiation / propagation
  - Data width
  - Address width
  - Type of accesses (read / write / atomic / ...)
  - ID width
  - ID to master mapping (Tilelink)
- Lack of awareness
  - Memory mapping from a given master perspective
  - Physical memory attributes

Decentralization



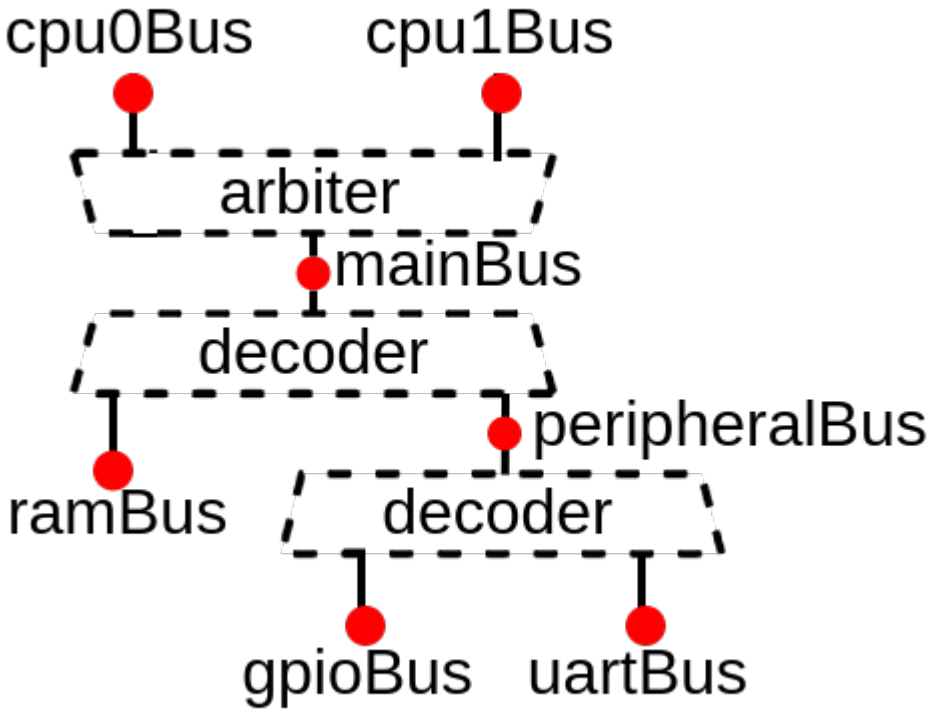

```
val cpu0Bus, cpu1Bus = tilelink.Node()
val mainBus          = tilelink.Node()
val ramBus           = tilelink.Node()
val peripheralBus    = tilelink.Node()
val gpioBus, uartBus = tilelink.Node()

mainBus       at 0x00000000 of(cpu0Bus, cpu1Bus)
ramBus        at 0x80000000 of mainBus
peripheralBus at 0x10000000 of ramBus
gpioBus       at     0x2000 of peripheralBus
uartBus       at     0x5000 of peripheralBus
```

Decentralization




```
val cpu0Bus, cpu1Bus = tilelink.Node()
val mainBus          = tilelink.Node()
val ramBus           = tilelink.Node()
val peripheralBus    = tilelink.Node()
val gpioBus, uartBus = tilelink.Node()

mainBus       at 0x00000000 of(cpu0Bus, cpu1Bus)
ramBus        at 0x80000000 of mainBus
peripheralBus at 0x10000000 of ramBus
gpioBus       at     0x2000 of peripheralBus
uartBus       at     0x5000 of peripheralBus
```

Decentralization




```
val cpu0Bus, cpu1Bus = tilelink.Node()
val mainBus          = tilelink.Node()
val ramBus           = tilelink.Node()
val peripheralBus    = tilelink.Node()
val gpioBus, uartBus = tilelink.Node()

mainBus       at 0x00000000 of(cpu0Bus, cpu1Bus)
ramBus        at 0x80000000 of mainBus
peripheralBus at 0x10000000 of ramBus
gpioBus       at     0x2000 of peripheralBus
uartBus       at     0x5000 of peripheralBus
```

Decentralization

```
class Node extends Area{
  // Node data model
  val bus   = Handle[tilelink.Bus]()
  val ups   = ArrayBuffer[Connection]()
  val downs = ArrayBuffer[Connection]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Generate the required arbiter / decoder logic.
  }
}
```

Decentralization

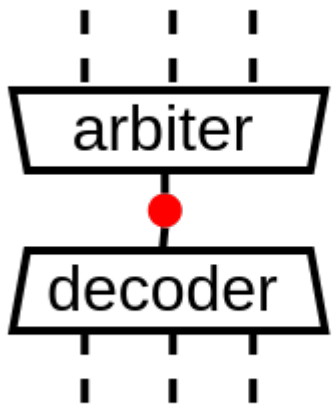
```
class Node extends Area{
  // Node data model
  val bus   = Handle[tilelink.Bus]()
  val ups   = ArrayBuffer[Connection]()
  val downs = ArrayBuffer[Connection]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Generate the required arbiter / decoder logic.
  }
}
```

Decentralization

```
class Node extends Area{
  // Node data model
  val bus   = Handle[tilelink.Bus]()
  val ups   = ArrayBuffer[Connection]()
  val downs = ArrayBuffer[Connection]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Generate the required arbiter / decoder logic.
  }
}
```

Decentralization

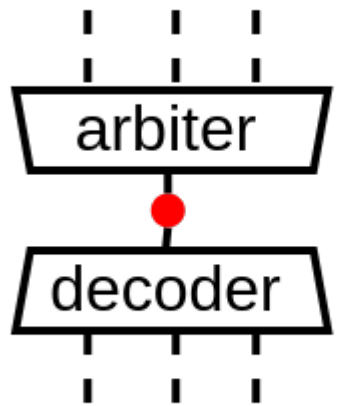
```
class Node extends Area{
  // Node data model
  val bus   = Handle[tilelink.Bus]()
  val ups   = ArrayBuffer[Connection]()
  val downs = ArrayBuffer[Connection]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Generate the required arbiter / decoder logic.
  }
}
```



Decentralization

```scala
class Node extends Area{
  // Node data model
  val bus   = Handle[tilelink.Bus]()
  val ups   = ArrayBuffer[Connection]()
  val downs = ArrayBuffer[Connection]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Generate the required arbiter / decoder logic.
  }
}
```

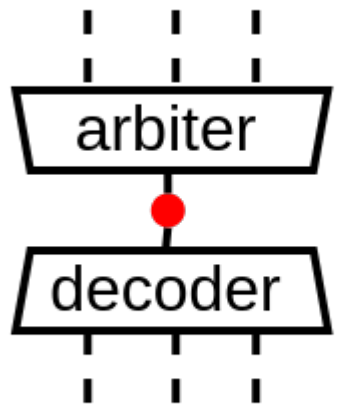
```scala
class Connection(val m : Node, val s : Node){
  val thread = Fiber build new Area{
    // Connect the m.decoder to the s.arbiter
  }
}
```



Decentralization

```scala
class Node extends Area{
  // Node data model
  val bus   = Handle[tilelink.Bus]()
  val ups   = ArrayBuffer[Connection]()
  val downs = ArrayBuffer[Connection]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Generate the required arbiter / decoder logic.
  }
}
```

```scala
class Connection(val m : Node, val s : Node){
  val thread = Fiber build new Area{
    // Connect the m.decoder to the s.arbiter
  }
}
```



Future, Promise, Async, Fibers...

```
val bus = Handle[tilelink.Bus]

val thread1 = Fiber build new Area{
  //Will wait on bus.load (from thread 2)
  bus.a.valid   := False
  bus.a.address := 42
}

val thread2 = Fiber build new Area{
  //Will allow thread 1 to continue
  bus load tilelink.Bus(config)
}
```

Future, Promise, Async, Fibers...

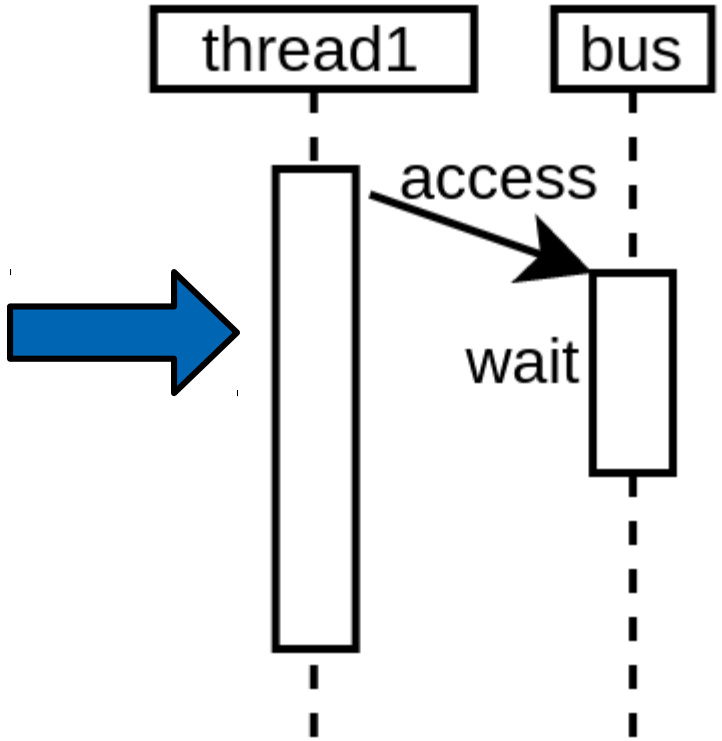
```
val bus = Handle[tilelink.Bus]

val thread1 = Fiber build new Area{
  //Will wait on bus.load (from thread 2)
  bus.a.valid   := False
  bus.a.address := 42
}

val thread2 = Fiber build new Area{
  //Will allow thread 1 to continue
  bus load tilelink.Bus(config)
}
```

Future, Promise, Async, Fibers...

```
val bus = Handle[tilelink.Bus]

val thread1 = Fiber build new Area{
  //Will wait on bus.load (from thread 2)
  bus.a.valid   := False
  bus.a.address := 42
}

val thread2 = Fiber build new Area{
  //Will allow thread 1 to continue
  bus load tilelink.Bus(config)
}
```

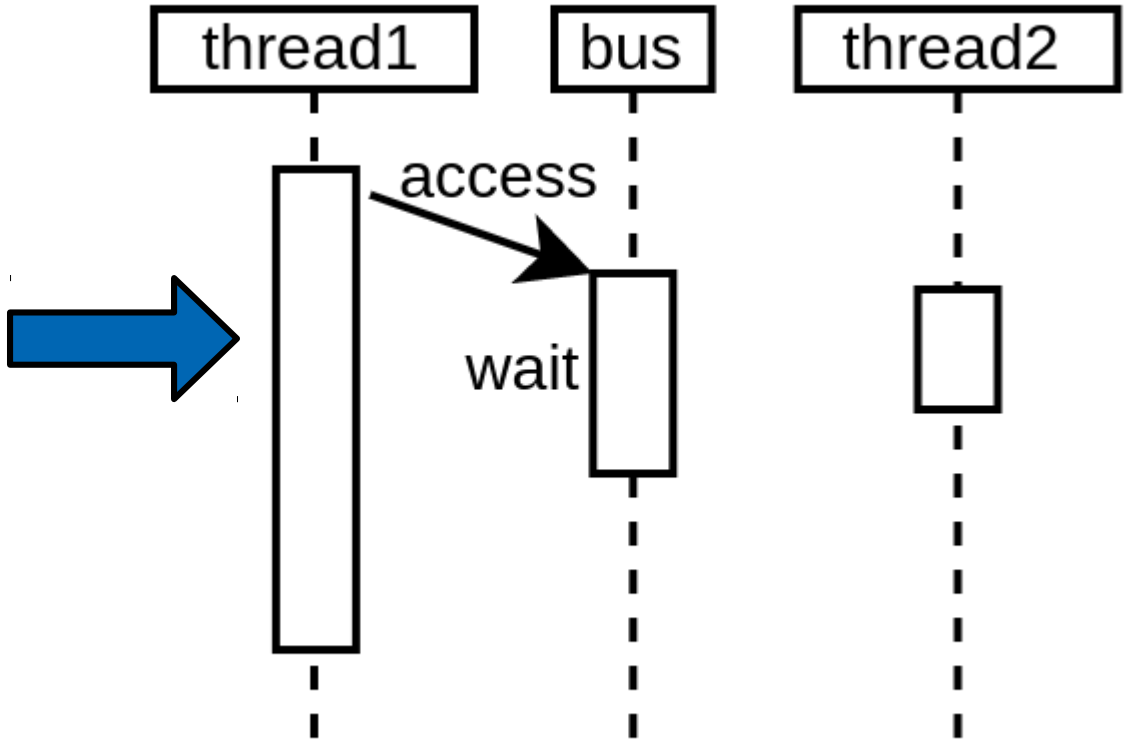
Future, Promise, Async, Fibers...

```
val bus = Handle[tilelink.Bus]

val thread1 = Fiber build new Area{
  //Will wait on bus.load (from thread 2)
  bus.a.valid   := False
  bus.a.address := 42
}

val thread2 = Fiber build new Area{
  //Will allow thread 1 to continue
  bus load tilelink.Bus(config)
}
```



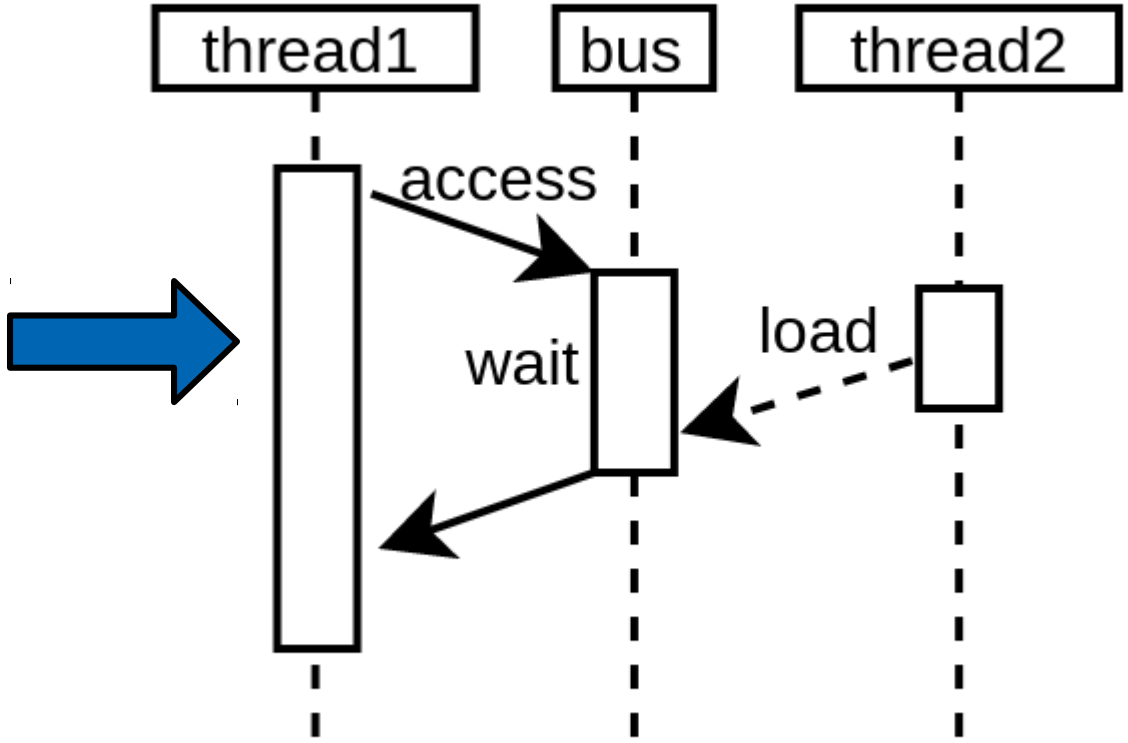
Future, Promise, Async, Fibers...

```
val bus = Handle[tilelink.Bus]

val thread1 = Fiber build new Area{
  //Will wait on bus.load (from thread 2)
  bus.a.valid   := False
  bus.a.address := 42
}

val thread2 = Fiber build new Area{
  //Will allow thread 1 to continue
  bus load tilelink.Bus(config)
}
```



Future, Promise, Async, Fibers...

```
val bus = Handle[tilelink.Bus]

val thread1 = Fiber build new Area{
  //Will wait on bus.load (from thread 2)
  bus.a.valid   := False
  bus.a.address := 42
}

val thread2 = Fiber build new Area{
  //Will allow thread 1 to continue
  bus load tilelink.Bus(...)
}
```



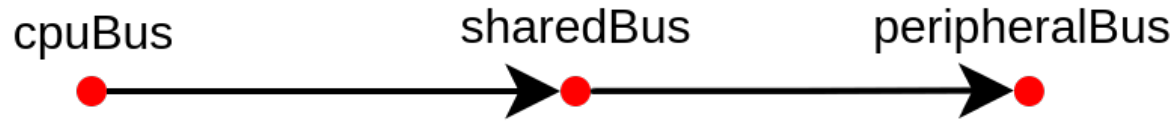
Negotiation

```
class Node extends Area{
  // Node data model
  val proposed = Handle[M2sSupport]()
  val supported = Handle[M2sSupport]()
  val parameters = Handle[M2sParameters]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Do the Negotiation
    // Generate the required arbiter / decoder logic.
  }
}
```

Negotiation

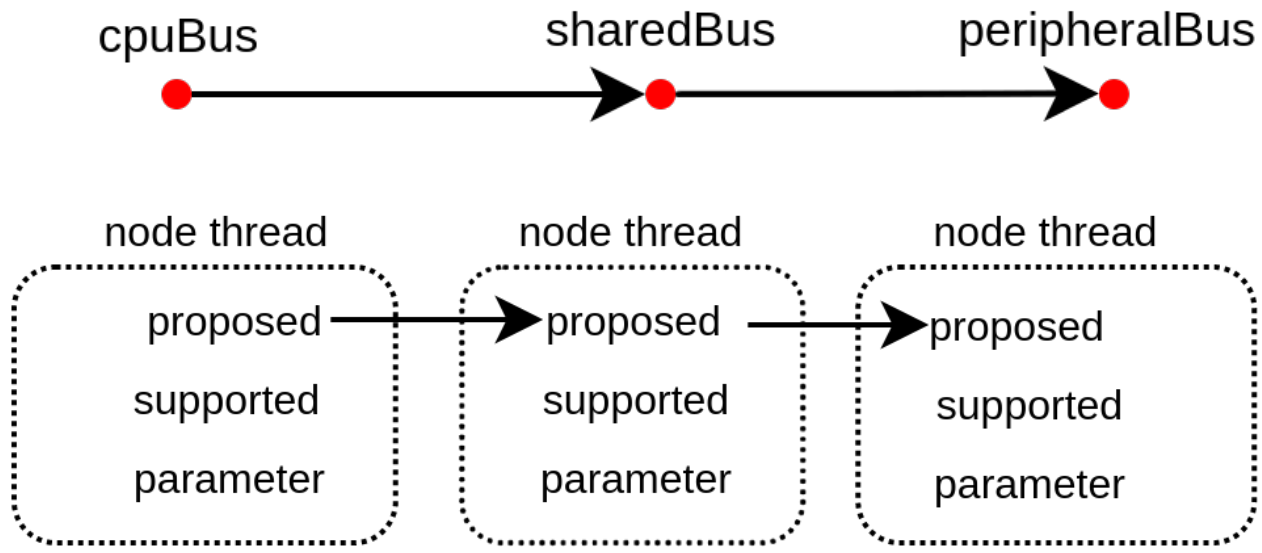
```
class Node extends Area{
  // Node data model
  val proposed = Handle[M2sSupport]()
  val supported = Handle[M2sSupport]()
  val parameters = Handle[M2sParameters]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Do the Negotiation
    // Generate the required arbiter / decoder logic.
  }
}
```

Negotiation



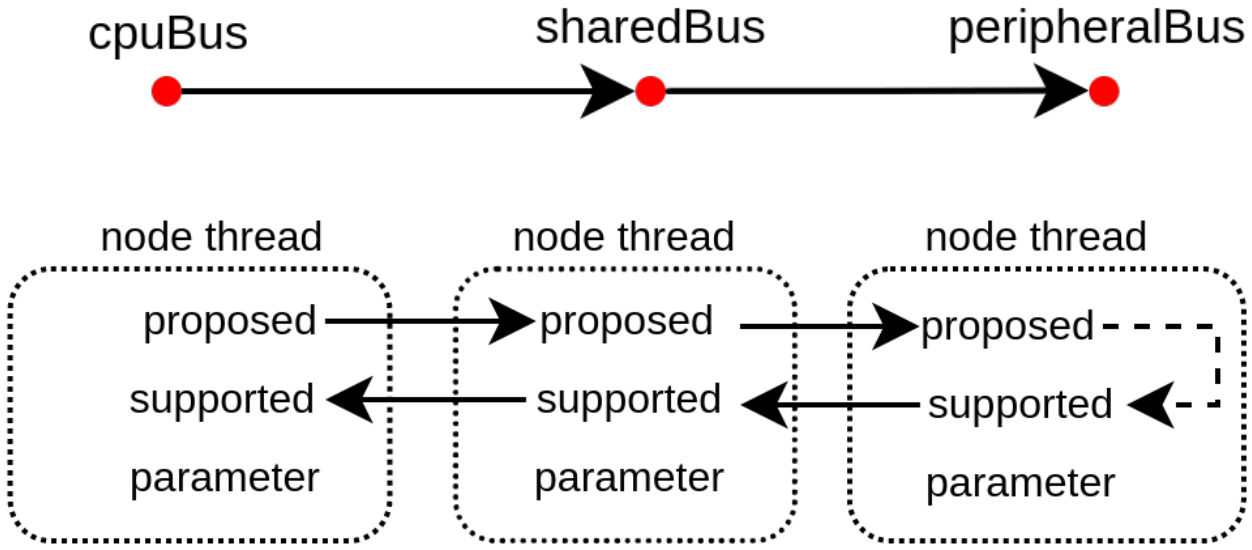

```
class Node extends Area{
  // Node data model
  val proposed = Handle[M2sSupport]()
  val supported = Handle[M2sSupport]()
  val parameters = Handle[M2sParameters]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Do the Negotiation
    // Generate the required arbiter / decoder logic.
  }
}
```

Negotiation



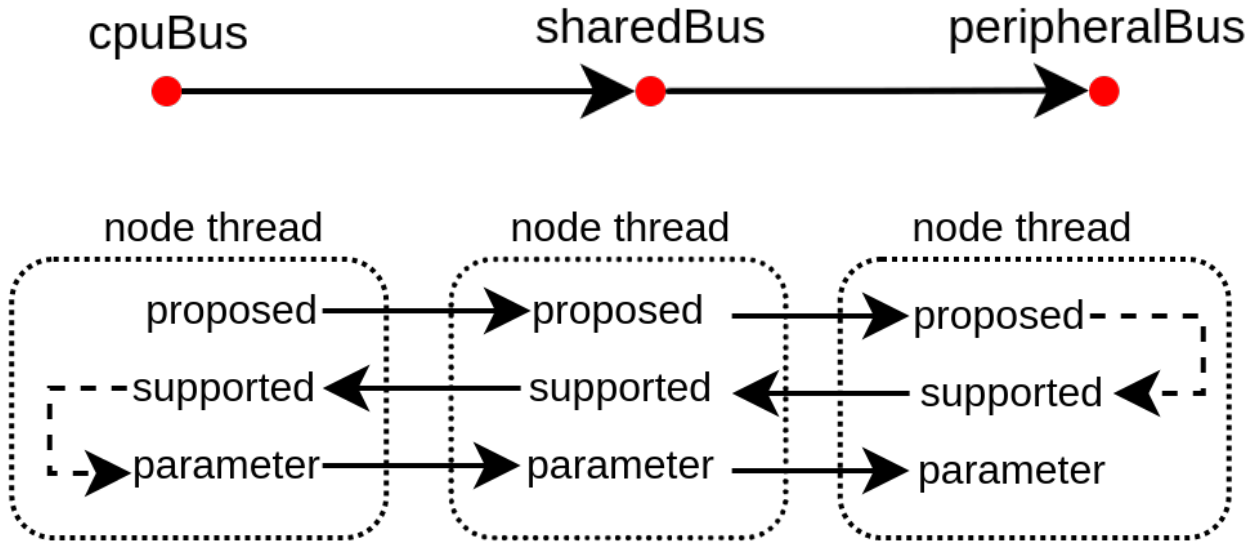

```
class Node extends Area{
  // Node data model
  val proposed = Handle[M2sSupport]()
  val supported = Handle[M2sSupport]()
  val parameters = Handle[M2sParameters]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Do the Negotiation
    // Generate the required arbiter / decoder logic.
  }
}
```

Negotiation




```
class Node extends Area{
  // Node data model
  val proposed = Handle[M2sSupport]()
  val supported = Handle[M2sSupport]()
  val parameters = Handle[M2sParameters]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Do the Negotiation
    // Generate the required arbiter / decoder logic.
  }
}
```

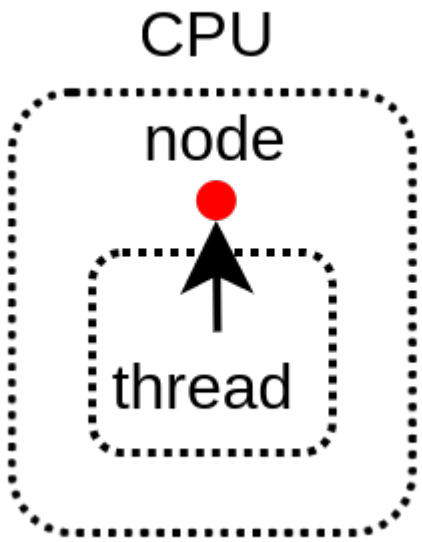
Negotiation




```
class Node extends Area{
  // Node data model
  val proposed = Handle[M2sSupport]()
  val supported = Handle[M2sSupport]()
  val parameters = Handle[M2sParameters]()

  //Fork an elaboration thread
  val thread = Fiber build new Area{
    // Do the Negotiation
    // Generate the required arbiter / decoder logic.
  }
}
```

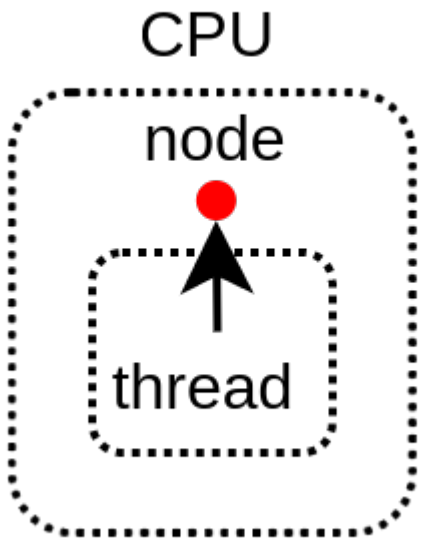
Master example

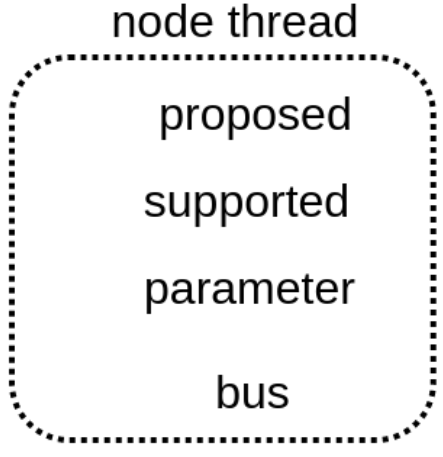
```
class Cpu() extends Area {
  val node = Node.master()

  val thread = Fiber build new Area {
    // Negotiate things
    node.proposed load M2sSupport(
      addressWidth = 32,
      dataWidth = 64,
      transfers = ...
    )
    node.parameters load M2sParameters(
      support = node.supported,
      sourceCount = 4
    )

    // Implement the actual CPU hardware
    node.bus.a.valid := False
    node.bus.a.address := 0x42
  }
}
```



Master example

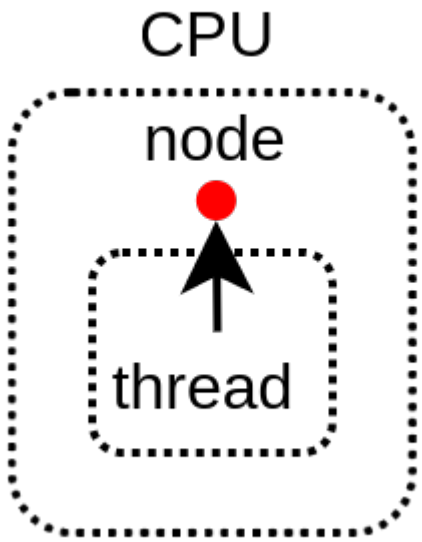
```
class Cpu() extends Area {
  val node = Node.master()

  val thread = Fiber build new Area {
    // Negotiate things
    node.proposed load M2sSupport(
      addressWidth = 32,
      dataWidth = 64,
      transfers = ...
    )
    node.parameters load M2sParameters(
      support = node.supported,
      sourceCount = 4
    )

    // Implement the actual CPU hardware
    node.bus.a.valid := False
    node.bus.a.address := 0x42
  }
}
```

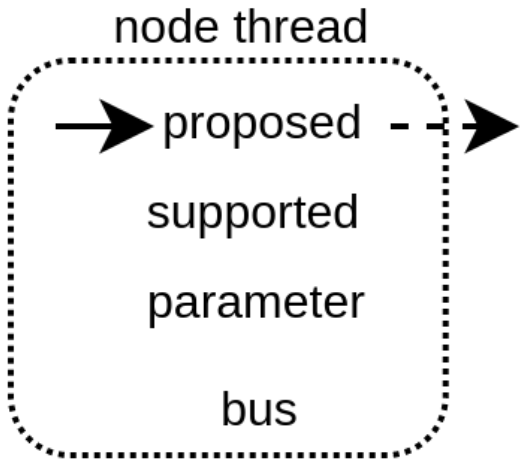




Master example

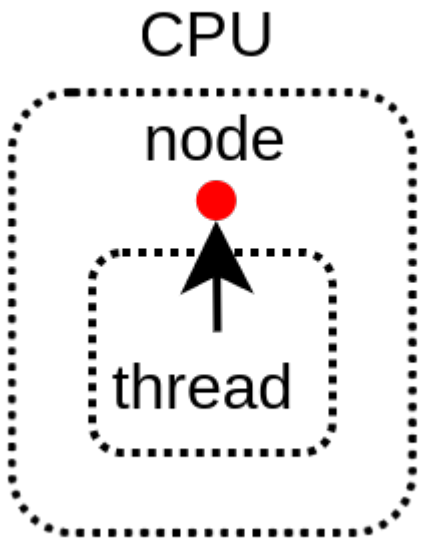
```
class Cpu() extends Area {
  val node = Node.master()

  val thread = Fiber build new Area {
    // Negotiate things
    node.proposed load M2sSupport(
      addressWidth = 32,
      dataWidth = 64,
      transfers = ...
    )
    node.parameters load M2sParameters(
      support = node.supported,
      sourceCount = 4
    )

    // Implement the actual CPU hardware
    node.bus.a.valid := False
    node.bus.a.address := 0x42
  }
}
```

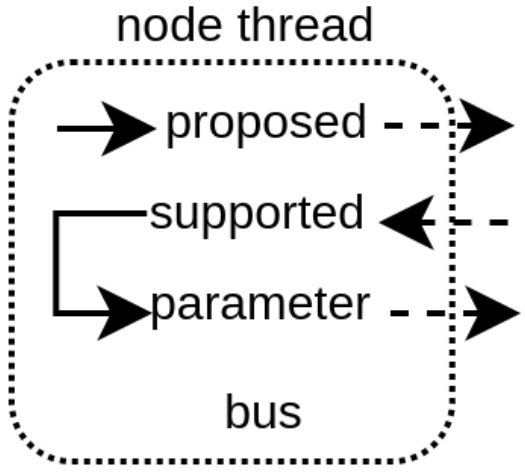




Master example

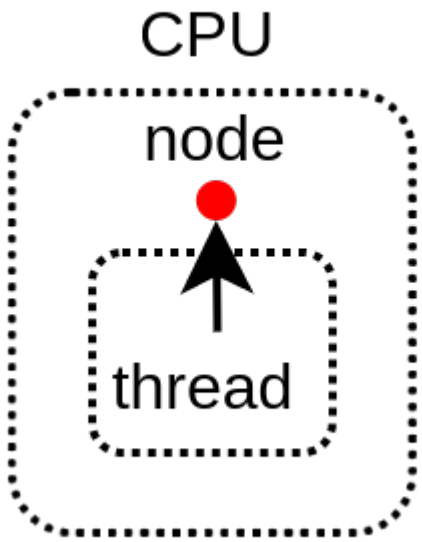
```
class Cpu() extends Area {
  val node = Node.master()

  val thread = Fiber build new Area {
    // Negotiate things
    node.proposed load M2sSupport(
      addressWidth = 32,
      dataWidth = 64,
      transfers = ...
    )
    node.parameters load M2sParameters(
      support = node.supported,
      sourceCount = 4
    )

    // Implement the actual CPU hardware
    node.bus.a.valid := False
    node.bus.a.address := 0x42
  }
}
```

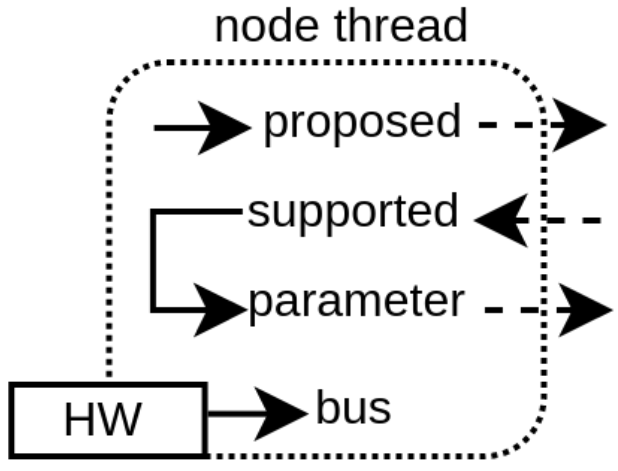




Master example

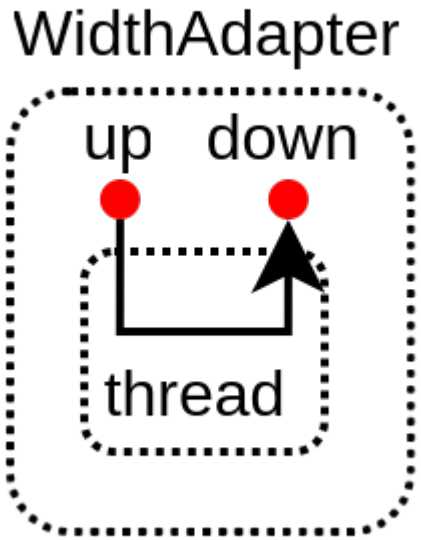
```
class Cpu() extends Area {
  val node = Node.master()

  val thread = Fiber build new Area {
    // Negotiate things
    node.proposed load M2sSupport(
      addressWidth = 32,
      dataWidth = 64,
      transfers = ...
    )
    node.parameters load M2sParameters(
      support = node.supported,
      sourceCount = 4
    )

    // Implement the actual CPU hardware
    node.bus.a.valid := False
    node.bus.a.address := 0x42
  }
}
```

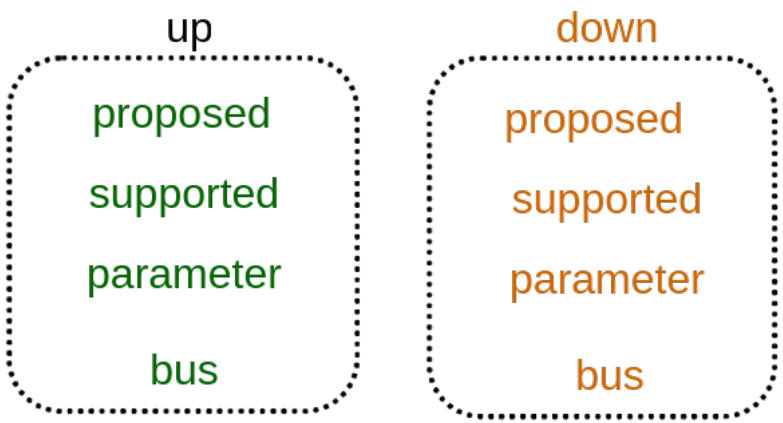




Adapter example

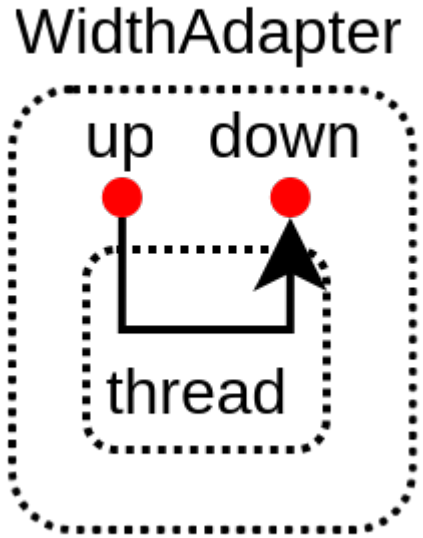
```
class WidthAdapter() extends Area{
  val up = Node.slave()
  val down = Node.master()

  val thread = Fiber build new Area{
    down.proposed load up.proposed
    up.supported load down.supported.copy(
      dataWidth = up.proposed.dataWidth
    )
    down.parameters load up.parameters.copy(
      dataWidth = down.supported.dataWidth
    )

    val bridge = new WidthAdapterHw(up.bus.p, down.bus.p)
    bridge.io.up << up.bus
    bridge.io.down >> down.bus
  }
}
```

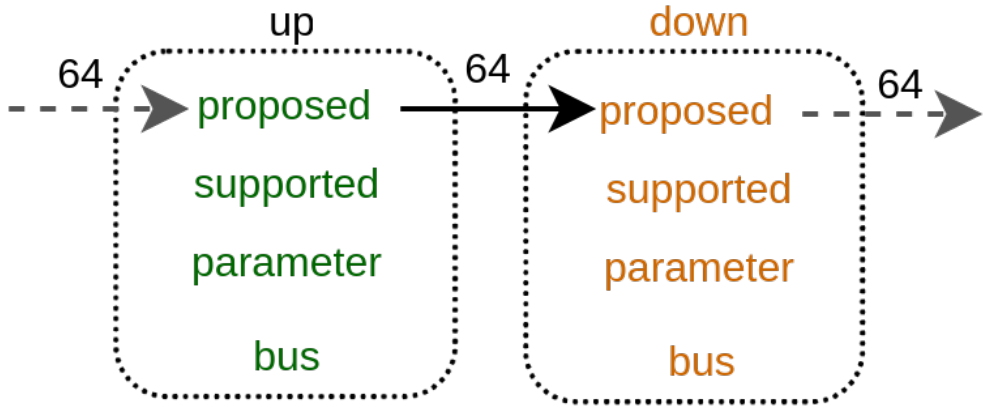




Adapter example

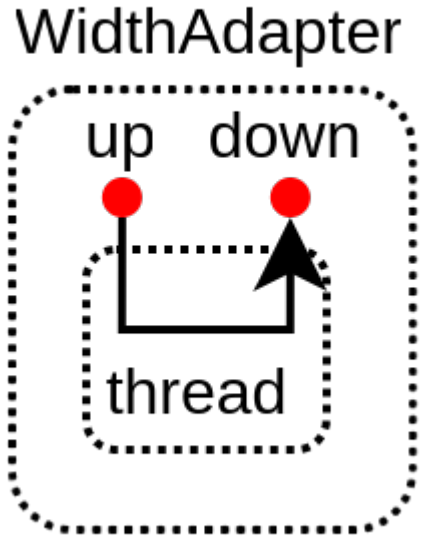
```
class WidthAdapter() extends Area{
  val up = Node.slave()
  val down = Node.master()

  val thread = Fiber build new Area{
    down.proposed load up.proposed
    up.supported load down.supported.copy(
      dataWidth = up.proposed.dataWidth
    )
    down.parameters load up.parameters.copy(
      dataWidth = down.supported.dataWidth
    )

    val bridge = new WidthAdapterHw(up.bus.p, down.bus.p)
    bridge.io.up << up.bus
    bridge.io.down >> down.bus
  }
}
```

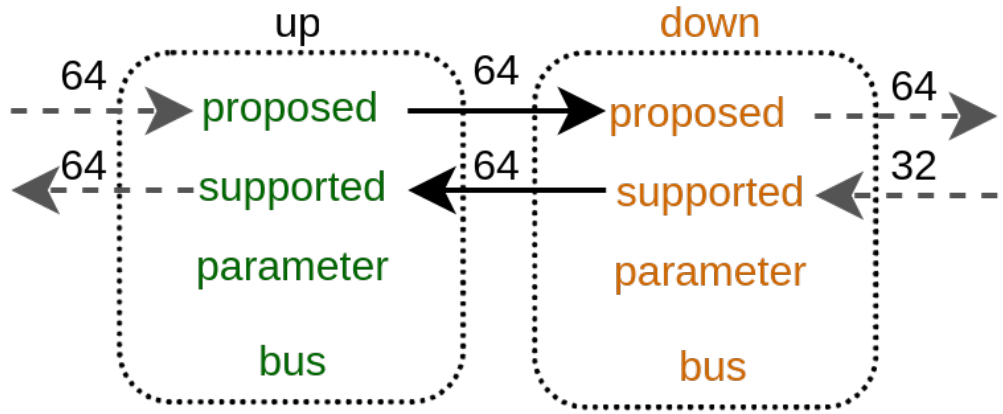




Adapter example

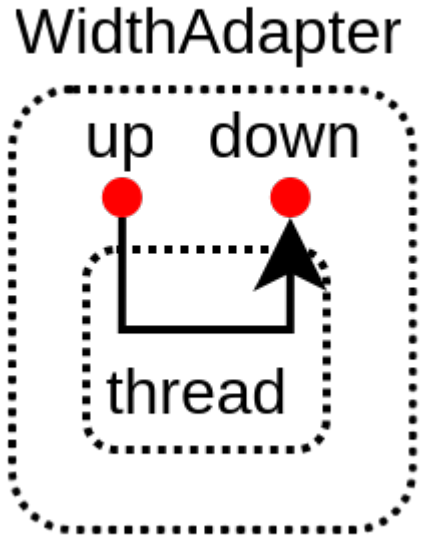
```
class WidthAdapter() extends Area{
  val up = Node.slave()
  val down = Node.master()

  val thread = Fiber build new Area{
    down.proposed load up.proposed
    up.supported load down.supported.copy(
      dataWidth = up.proposed.dataWidth
    )
    down.parameters load up.parameters.copy(
      dataWidth = down.supported.dataWidth
    )

    val bridge = new WidthAdapterHw(up.bus.p, down.bus.p)
    bridge.io.up << up.bus
    bridge.io.down >> down.bus
  }
}
```

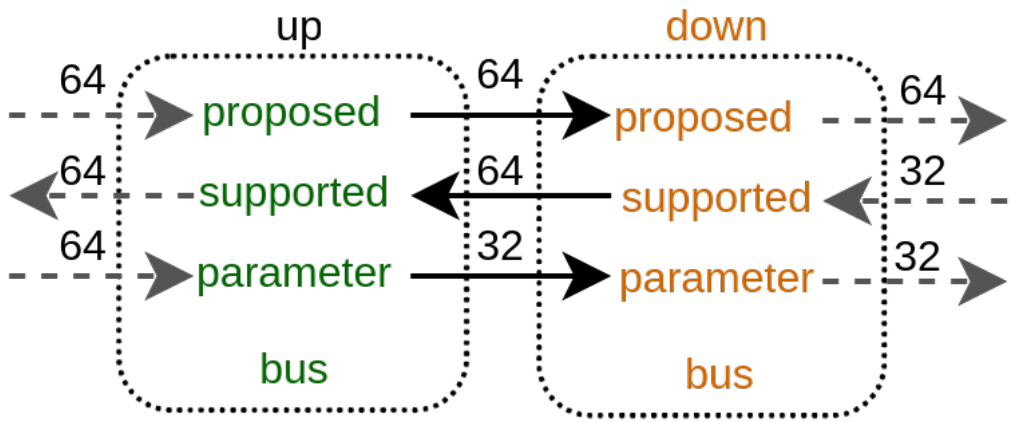




Adapter example

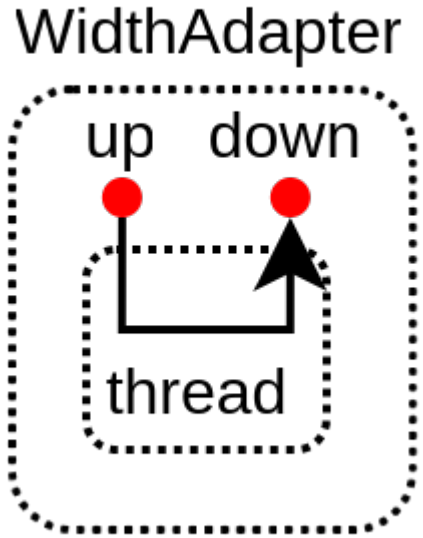
```
class WidthAdapter() extends Area{
  val up = Node.slave()
  val down = Node.master()

  val thread = Fiber build new Area{
    down.proposed load up.proposed
    up.supported load down.supported.copy(
      dataWidth = up.proposed.dataWidth
    )
    down.parameters load up.parameters.copy(
      dataWidth = down.supported.dataWidth
    )

    val bridge = new WidthAdapterHw(up.bus.p, down.bus.p)
    bridge.io.up << up.bus
    bridge.io.down >> down.bus
  }
}
```

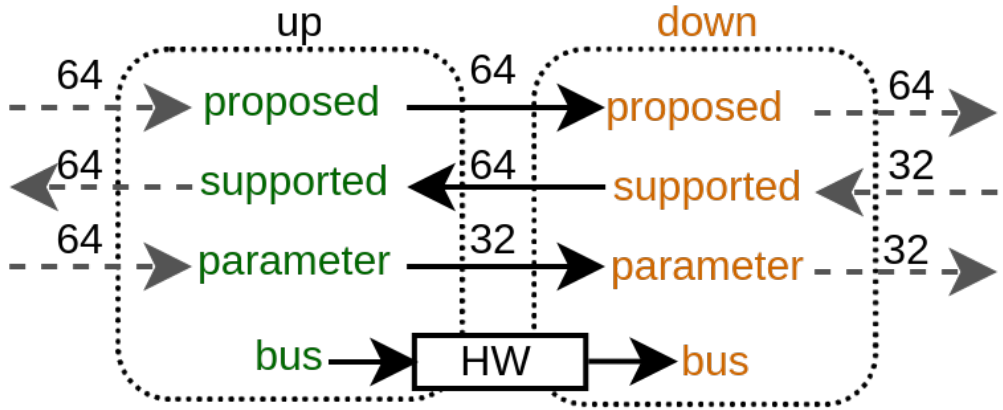




Adapter example

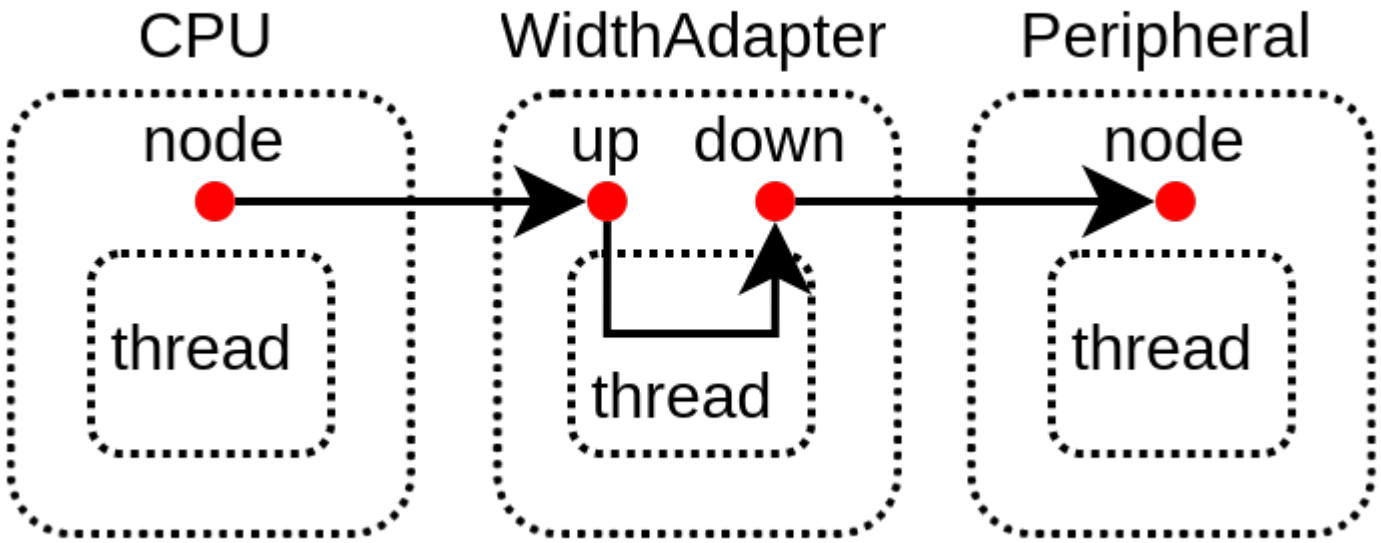
```
class WidthAdapter() extends Area{
  val up = Node.slave()
  val down = Node.master()

  val thread = Fiber build new Area{
    down.proposed load up.proposed
    up.supported load down.supported.copy(
      dataWidth = up.proposed.dataWidth
    )
    down.parameters load up.parameters.copy(
      dataWidth = down.supported.dataWidth
    )

    val bridge = new WidthAdapterHw(up.bus.p, down.bus.p)
    bridge.io.up << up.bus
    bridge.io.down >> down.bus
  }
}
```





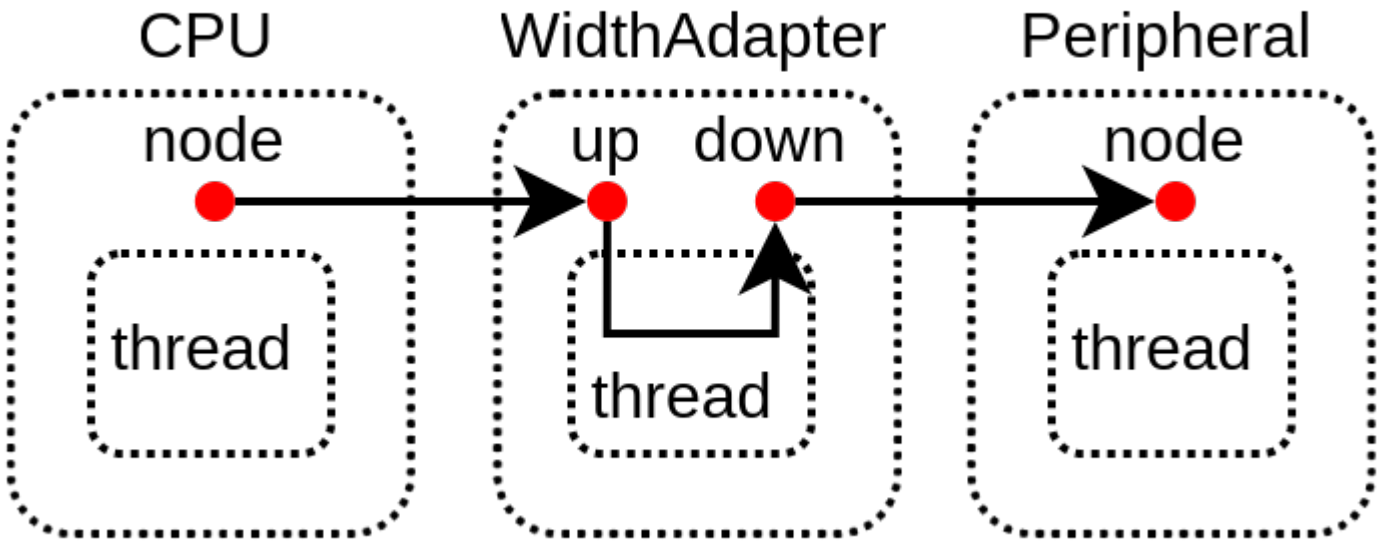
Deployment

```
val cpu        = new Cpu()
val adapter    = new WidthAdapter()
val peripheral = new Peripheral()

adapter.up      at 0x00000000 of cpu.node
peripheral.node at     0x2000 of adapter.down
```



Deployment

```
val cpu        = new Cpu()
val adapter    = new WidthAdapter()
val peripheral = new Peripheral()

adapter.up      at 0x00000000 of cpu.node
peripheral.node at     0x2000 of adapter.down
```



Introspection

```
val counter = Reg(UInt(8 bits))
counter := counter + 1

class CustomTag(val str : String) extends SpinalTag

counter.addTag(new CustomTag("hello"))
counter.addTag(new CustomTag("miaou"))

counter.foreachTag{
  case ct : CustomTag => println(ct.str)
  case _ =>
}
```

Introspection

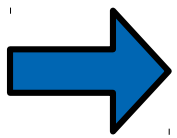
```
val counter = Reg(UInt(8 bits))
counter := counter + 1

class CustomTag(val str : String) extends SpinalTag

counter.addTag(new CustomTag("hello"))
counter.addTag(new CustomTag("miaou"))

counter.foreachTag{
  case ct : CustomTag => println(ct.str)
  case _ =>
}
```

Introspection

```
val counter = Reg(UInt(8 bits))
counter := counter + 1

class CustomTag(val str : String) extends SpinalTag

counter.addTag(new CustomTag("hello"))
counter.addTag(new CustomTag("miaou"))

counter.foreachTag{
  case ct : CustomTag => println(ct.str)
  case _ =>
}
```

Introspection

```
val counter = Reg(UInt(8 bits))
counter := counter + 1

class CustomTag(val str : String) extends SpinalTag

counter.addTag(new CustomTag("hello"))
counter.addTag(new CustomTag("miaou"))

counter.foreachTag{
  case ct : CustomTag => println(ct.str)
  case _ =>
}
```



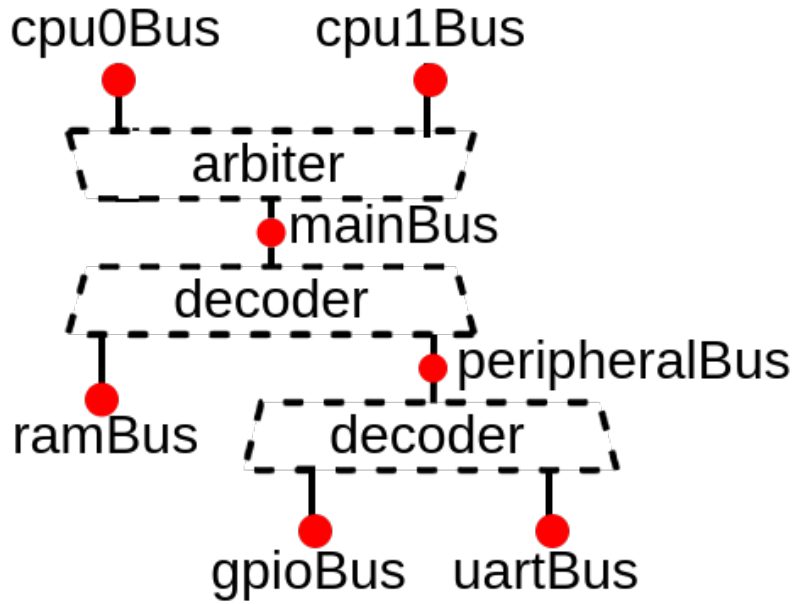
hello
miaou

Introspection

```
trait MemoryConnection extends SpinalTag {
  def m : SpinalTagReady
  def s : SpinalTagReady
  def mapping : SizeMapping
}
```

Introspection

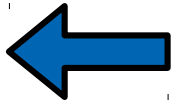
```
trait MemoryConnection extends SpinalTag {
  def m : SpinalTagReady
  def s : SpinalTagReady
  def mapping : SizeMapping
  override def toString = s"$m $s $mapping"
}
```

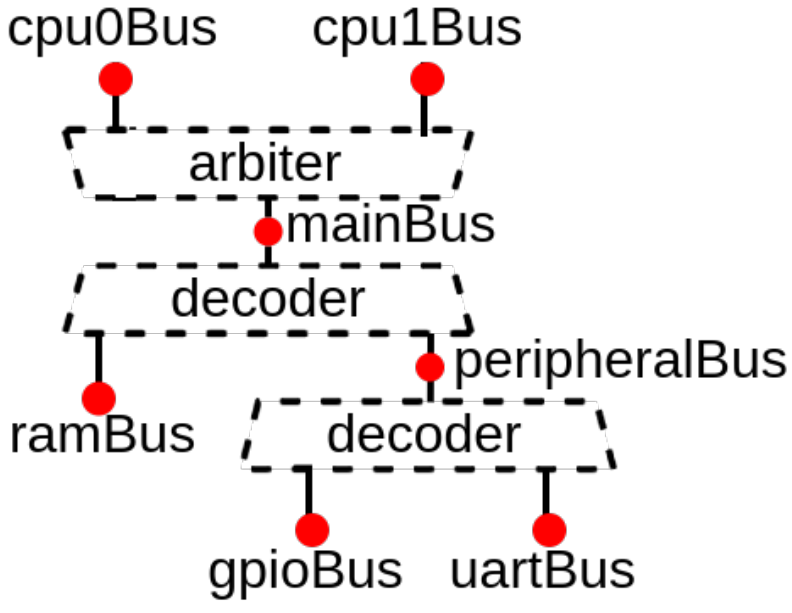


Introspection

```
trait MemoryConnection extends SpinalTag {
  def m : SpinalTagReady
  def s : SpinalTagReady
  def mapping : SizeMapping
  override def toString = s"$m $s $mapping"
}
```

```
val tag = new MemoryConnection{
  def m = peripheralBus
  def s = gpioBus
  def mapping = SizeMapping(0x5000, 0x1000)
}
peripheralBus.addTag(tag)
gpioBus.addTag(tag)
```

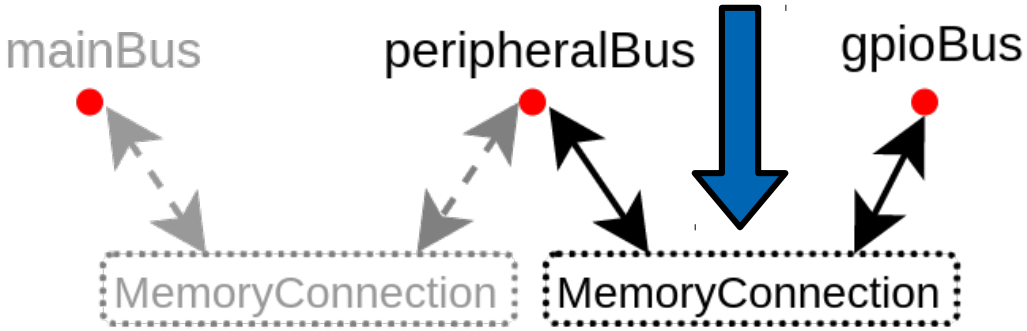


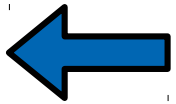


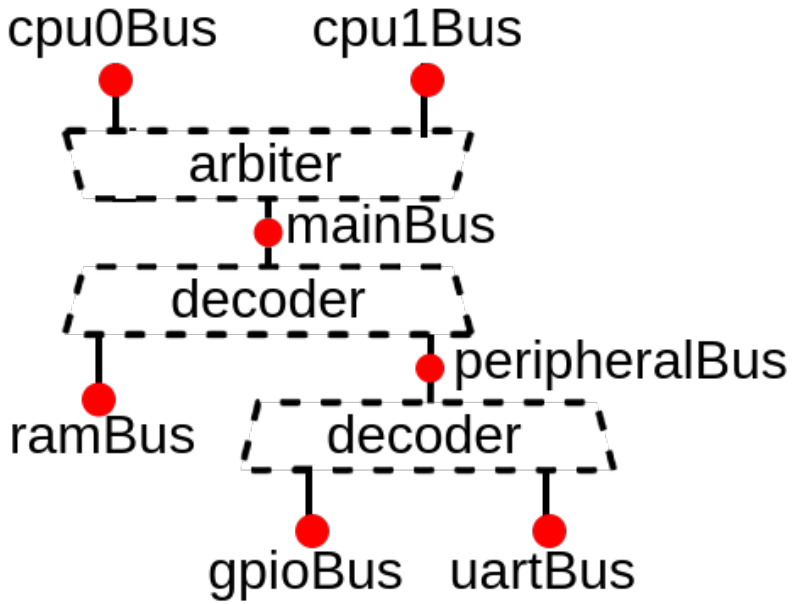
Introspection

```
trait MemoryConnection extends SpinalTag {
  def m : SpinalTagReady
  def s : SpinalTagReady
  def mapping : SizeMapping
  override def toString = s"$m $s $mapping"
}
```

```
val tag = new MemoryConnection{
  def m = peripheralBus
  def s = gpioBus
  def mapping = SizeMapping(0x5000, 0x1000)
}
peripheralBus.addTag(tag)
gpioBus.addTag(tag)
```







Introspection

```
def visit(node : SpinalTagReady, level : Int){
  node.foreachTag{
    case mc : MemoryConnection if mc.m == node => {
      println("  " * level + mc)
      visit(mc.s, level + 1)
    }
    case _ =>
  }
}

visit(cpu0Bus, 0)
```

Introspection

```
def visit(node : SpinalTagReady, level : Int){
  node.foreachTag{
    case mc : MemoryConnection if mc.m == node => {
      println("  " * level + mc)
      visit(mc.s, level + 1)
    }
    case _ =>
  }
}

visit(cpu0Bus, 0)
```

Introspection

```
def visit(node : SpinalTagReady, level : Int){
  node.foreachTag{
    case mc : MemoryConnection if mc.m == node => {
      println("  " * level + mc)
      visit(mc.s, level + 1)
    }
    case _ =>
  }
}

visit(cpu0Bus, 0)
```

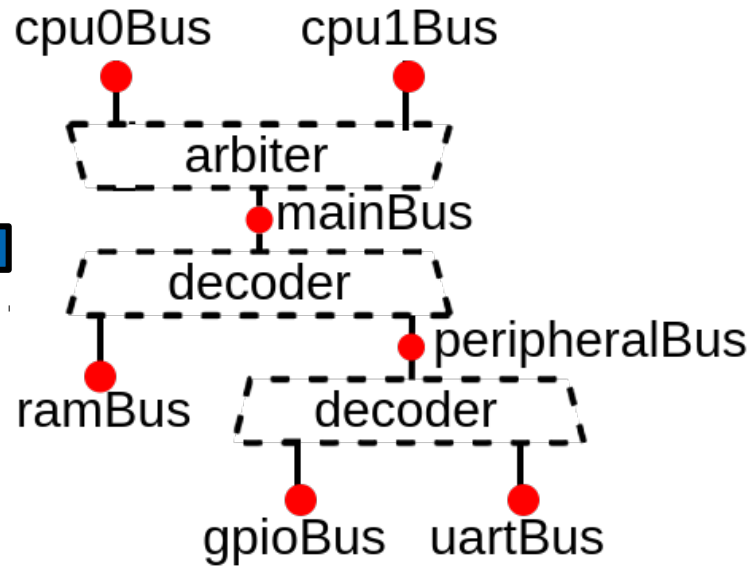
Introspection

```
def visit(node : SpinalTagReady, level : Int){
  node.foreachTag{
    case mc : MemoryConnection if mc.m == node => {
      println("  " * level + mc)
      visit(mc.s, level + 1)
    }
    case _ =>
  }
}

visit(cpu0Bus, 0)
```

Introspection

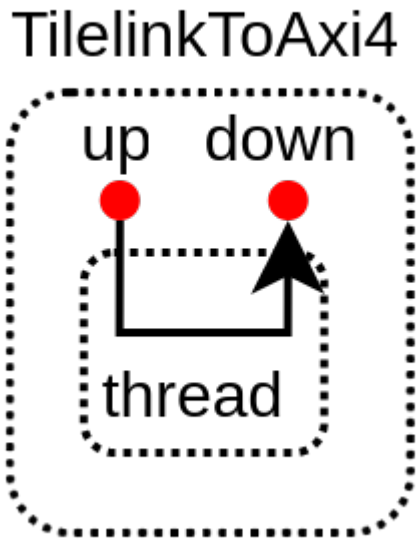
```
def visit(node : SpinalTagReady, level : Int){
  node.foreachTag{
    case mc : MemoryConnection if mc.m == node => {
      println("  " * level + mc)
      visit(mc.s, level + 1)
    }
    case _ =>
  }
}

visit(cpu0Bus, 0)
```

```
cpu0Bus mainBus 0 100000000
  mainBus ramBus 40000000 10000
  mainBus peripheralBus 10000000 100000
    peripheralBus gpioBus 2000 1000
    peripheralBus uartBus 5000 1000
```



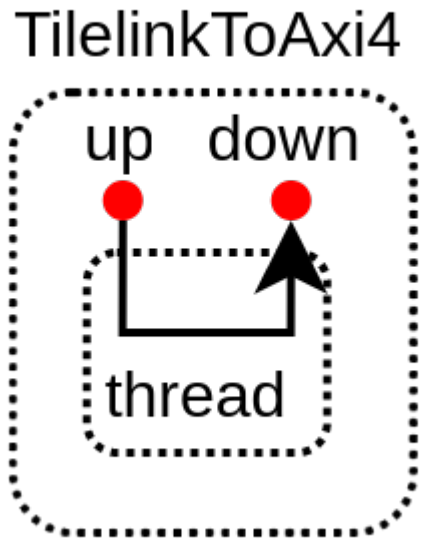
Interoperability

```
class TilelinkToAxi4() extends Area{
  val up   = tilelink.Node.slave()
  val down = axi4.Node.master()

  val tag = new MemoryConnection {
    def m = up
    def s = down
    def mapping = ...
  }
  up.add(tag)
  down.add(tag)

  val thread = Fiber build new Area{
    // Handle the negotiation from Tilelink to AXI
    // ...
    // Generate the hardware
    // ...
  }
}
```



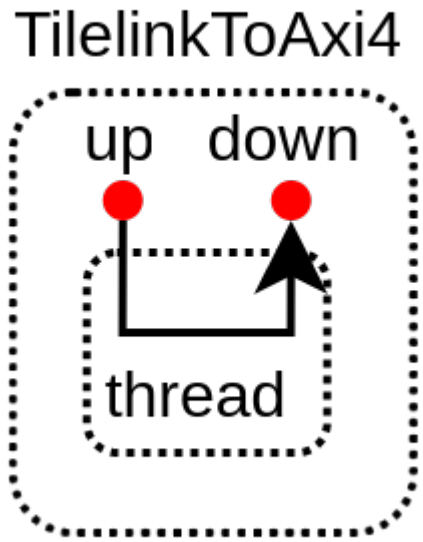
Interoperability

```
class TilelinkToAxi4() extends Area{
  val up   = tilelink.Node.slave()
  val down = axi4.Node.master()

  val tag = new MemoryConnection {
    def m = up
    def s = down
    def mapping = ...
  }
  up.add(tag)
  down.add(tag)

  val thread = Fiber build new Area{
    // Handle the negotiation from Tilelink to AXI
    // ...
    // Generate the hardware
    // ...
  }
}
```



Interoperability

```
class TilelinkToAxi4() extends Area{
  val up   = tilelink.Node.slave()
  val down = axi4.Node.master()

  val tag = new MemoryConnection {
    def m = up
    def s = down
    def mapping = ...
  }
  up.add(tag)
  down.add(tag)

  val thread = Fiber build new Area{
    // Handle the negotiation from Tilelink to AXI
    // ...
    // Generate the hardware
    // ...
  }
}
```



Summarize

- Paradigms (keeps things clean)
- Elaboration thread
- Decentralized negotiation / elaboration
- Introspection
- Interoperability

Question ?

- Open Discussion about the tilelink interconnect API / framework :
  - https://github.com/SpinalHDL/SpinalHDL/discussions/1115
- Roadmap
  - Tilelink interconnect with
    - memory coherency
    - L2 cache
  - NaxRiscv with tilelink memory coherency
- Looking for buddies :)