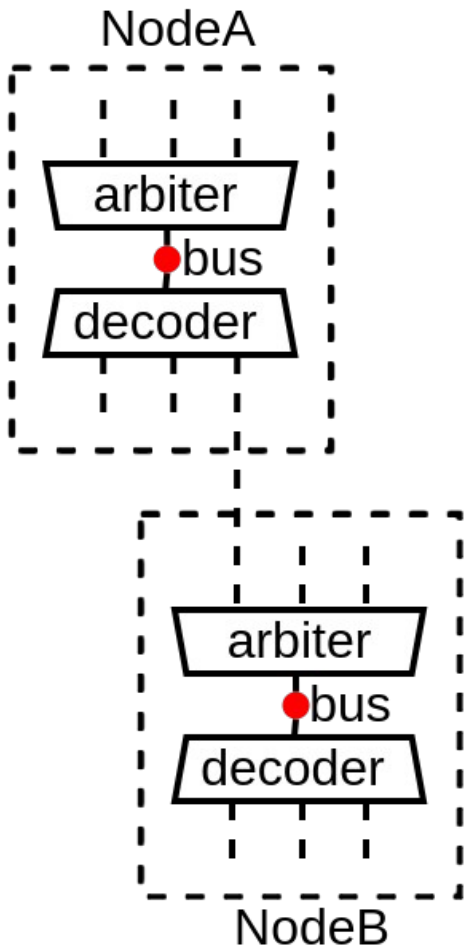
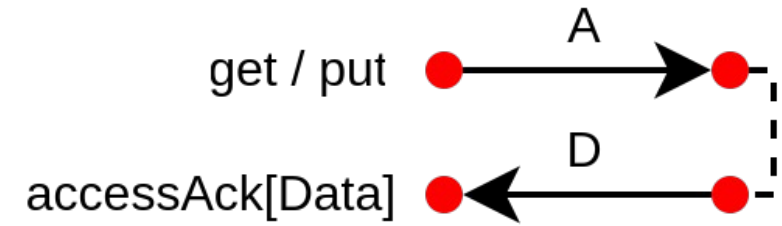
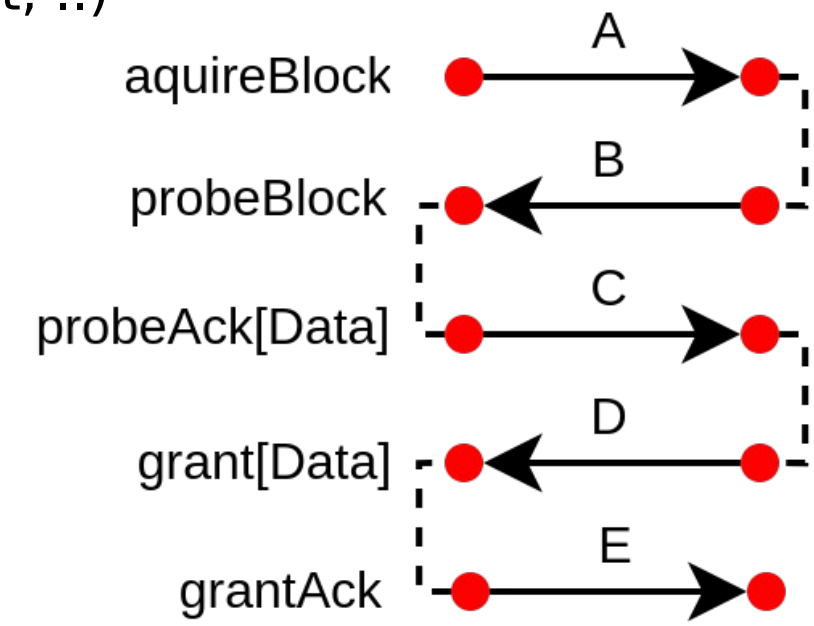
spinal.lib.bus.tilelink
An Tilelink interconnect generator based on fiber



Tilelink in short

- Memory bus specification
- Free / Open source
- Few specificities
  - No feature creep (always aligned, no wrap burst, ..)
  - Out of order
- Optional memory coherency
  - Memory block ownership (probe)





Parameter propagation / negotiation

- Source / Sink identifier width
  - Expends / mutate
- Address / Data width
  - Inference
- Memory region attributes
  - Cacheable, IO, Speculative, ...
- Atomic support
- Coherent memory block size
- ...

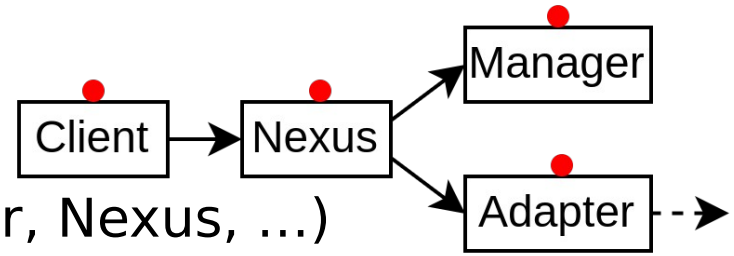
Some context

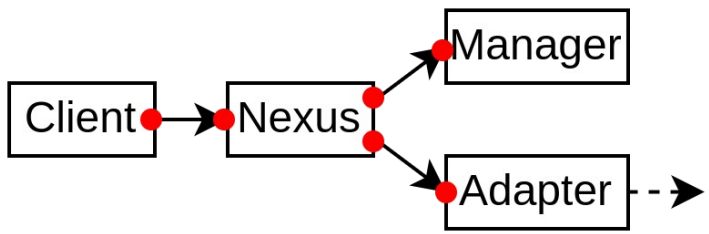
- Hardware Description Library (HDL) (SpinalHDL, Chisel, Migen, Amaranth)
  - Embedded in a general purpose programming language (Scala, Python)
  - Sequential hardware elaboration

```
// Define hardware signals
val a,b    = UInt(8 bits)
val result = UInt(8 bits)

// Define some behaviour
result := a + b
```

Propagation / negotiation API design possibilities

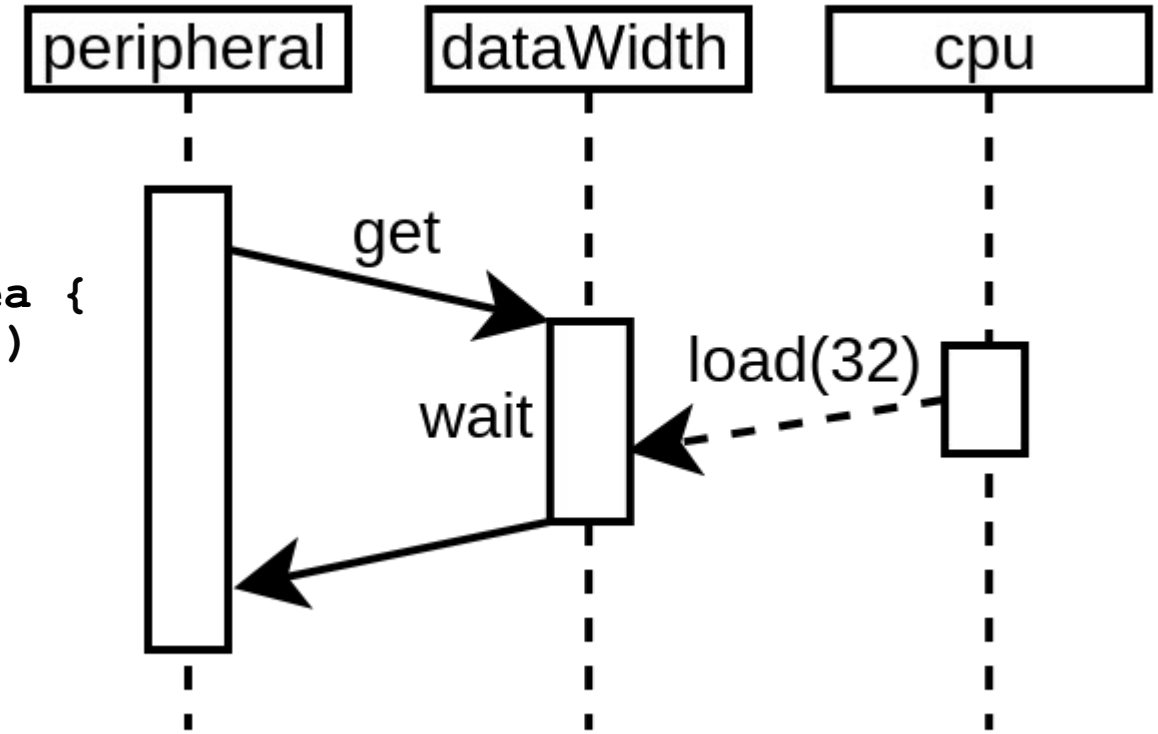
- Callback based (ex : Litex)
  - Agents implements a software interface and are registered in a centralized elaboration scheduler
- Rocket-Chip diplomacy
  - Based on scala lazy val
  - Module centric (Client, Manager, Adapter, Nexus, …)
- spinal.lib.bus.tilelink
  - Interface centric
  - Based on fiber





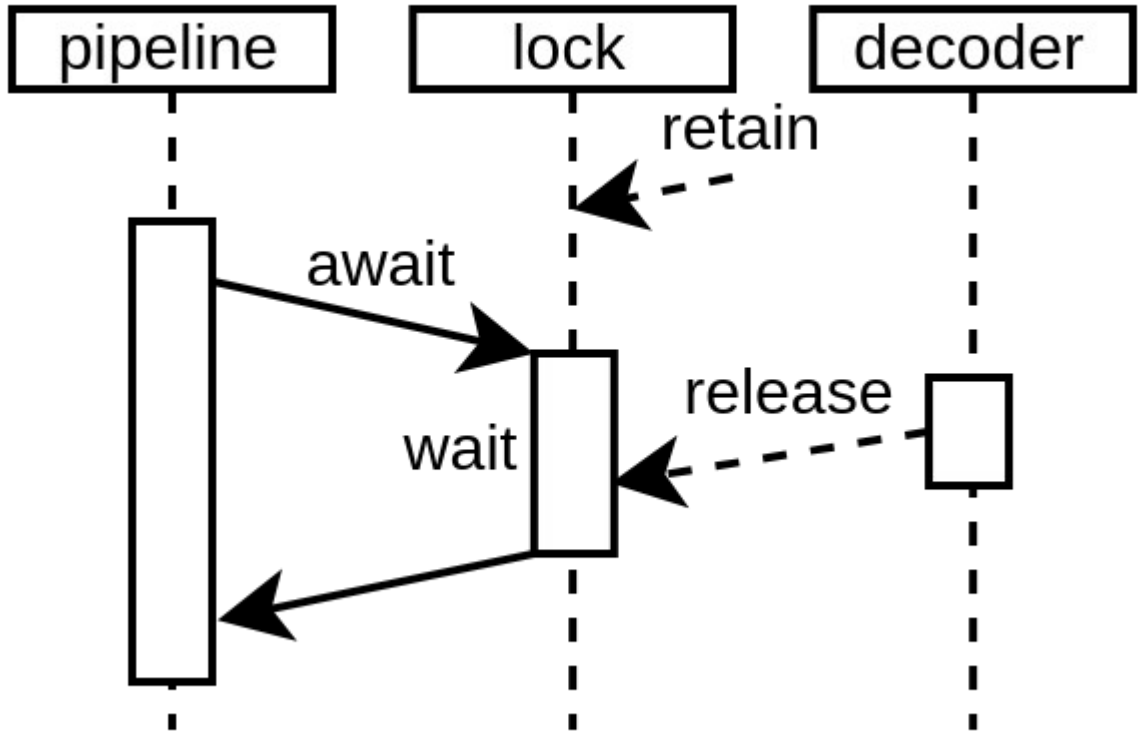
Fiber : late parameter

```
val dataWidth = Handle[Int]

val peripheral = Fiber build new Area {
  val data = UInt(dataWidth.get bits)
}

val cpu = Fiber build new Area {
  dataWidth.load(32)
}
```



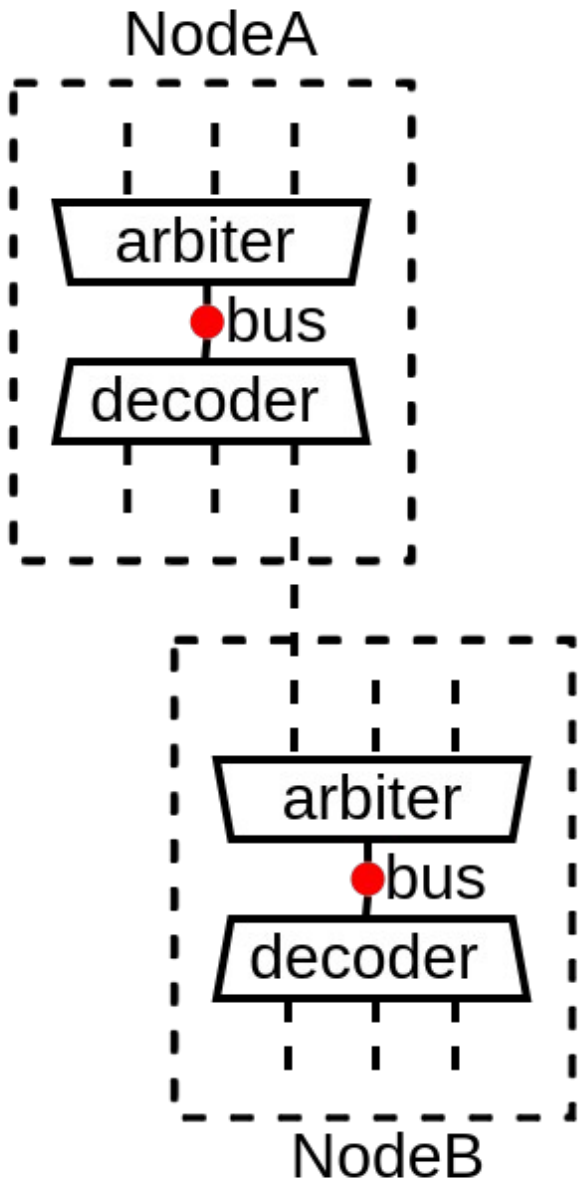
Fiber : delayed elaboration

```
val lock = Lock()
lock.retain()

val pipeline = Fiber build new Area{
  lock.await()
  this.build()
}

val decoder = Fiber build new Area {
  // ...
  lock.release()
}
```



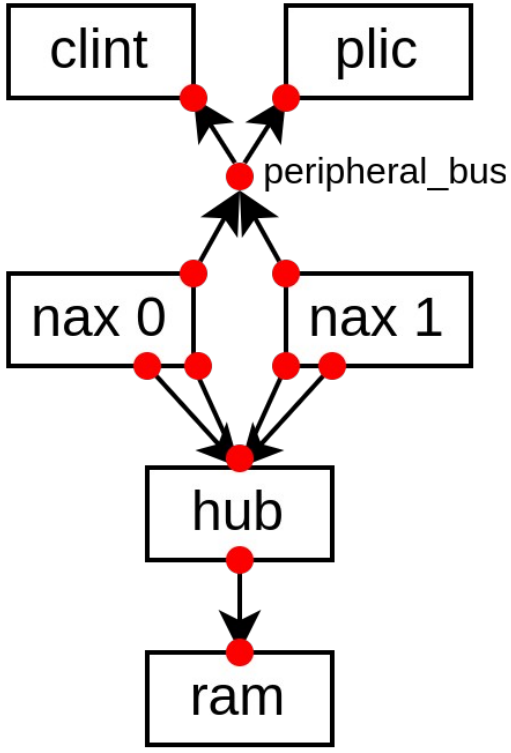
Tilelink Node

- What is it
  - A Tilelink bus instance
  - Can optionally have masters
  - Can optionally have slaves
- Automatically :
  - Negotiate / propagate Tilelink parameters
  - Add arbiter / decoder
  - Add Width adapter
  - Add Cross clock domain adapter



Little SoC

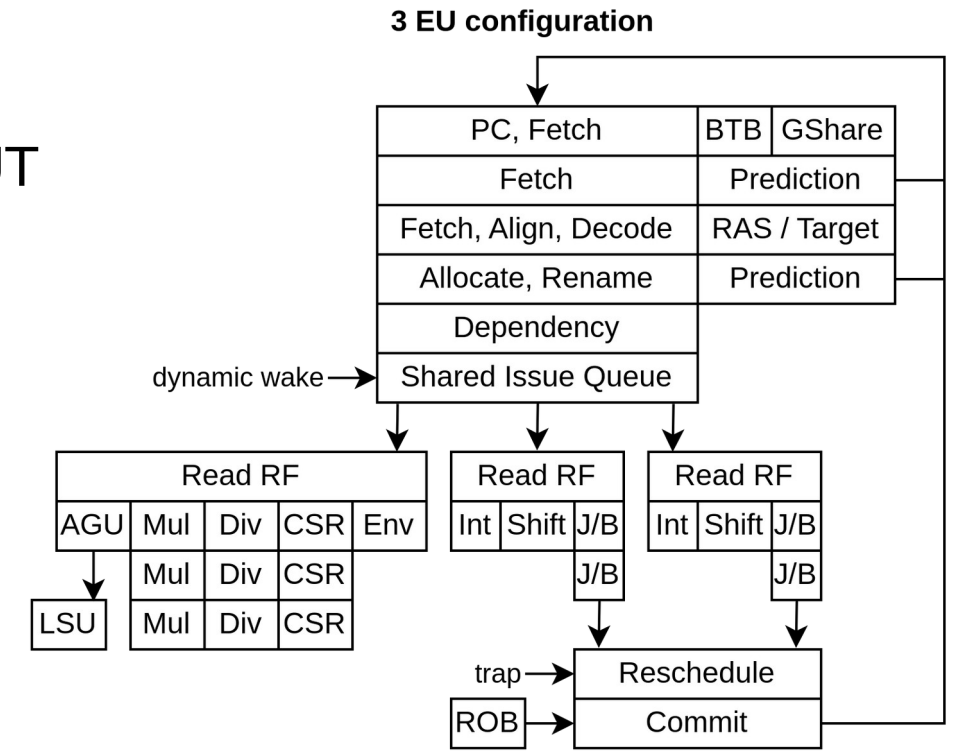
(with memory coherency)




```
class SocDemoSmall(cpuCount : Int) extends Component {
  val naxes = List.fill(cpuCount)(new TilelinkNaxRiscvFiber())

  for((nax, hartId) <- naxes.zipWithIndex)
    nax.setCoherentConfig(hartId)

  val hub = new HubFiber()
  for(nax <- naxes) hub.up << (nax.iBus, nax.dBus)

  val ram = new RamFiber()
  ram.node at (0x80000000, 64 kB) of hub.down

  val peripheral = new Area {
    val bus = Node()

    val clint = new TilelinkClintFiber()
    clint.node at 0x10000 of bus

    val plic = new TilelinkPlicFiber()
    plic.node at 0xC00000 of bus

    for(nax <- naxes) {
      nax.bind(clint)
      nax.bind(plic)
      bus at(0x10000000, 16 MB) of nax.pBus
    }
  }

  peripheral.bus.setUpConnection(a = StreamPipe.FULL)
}
```

NaxRiscv

- An Out Of Order, multi issue, RISC-V CPU
- Target FPGA
  - Artix7-3 : RV32IMASU 155 Mhz 13.3 KLUT
    2.93 DMIPS/Mhz 5.02 Coremark/Mhz
- Can run Debian (64 bits config)
- Plugin based, like VexRiscv
  - But those plugins are based on Fibers

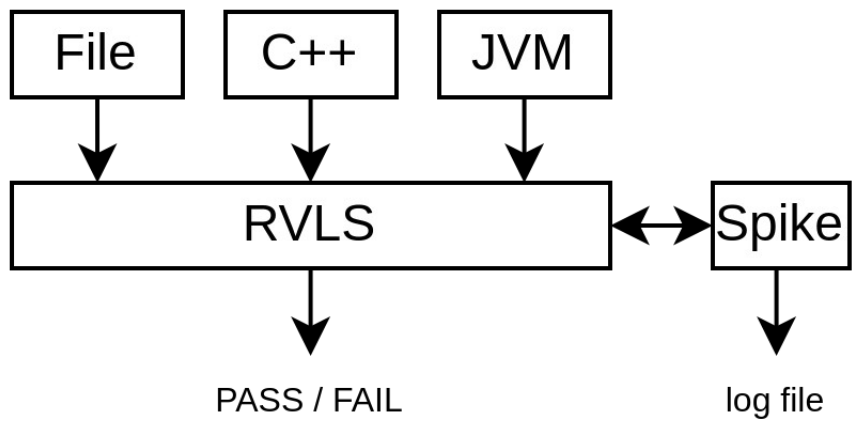


https://spinalhdl.github.io/NaxRiscv-Rtd/main/NaxRiscv/abstraction/index.html

RVLS

```
memview new 0 17 16
rv new 0 RV32IMA MSU 32 0
rv set pc 0 ffffffff80000000
rv region add 0 0 0000000080000000
elf load  0000000000000000 example/
rv int set 0 7 1
rv rf w 0 0 32 0000000000000001
rv commit 0 ffffffff80000000
rv rf w 0 0 32 0000000000000002
rv commit 0 ffffffff80000004
rv rf w 0 0 32 0000000000000003
rv commit 0 ffffffff80000008
```

- RVLS (Risc-V Lock Step) is a RISC-V simulation trace checker
- Spike used as RISC-V software model
- Traces Frontends
  - Human-readable text file
  - C++ / Java JNI calls
- Support multi-core systems (including some memory coherency traces)
  - Dual core SMP NaxRiscv linux boot OK



https://github.com/SpinalHDL/rvls

Question ?

- Open Discussion about the tilelink interconnect API / framework :
  - https://github.com/SpinalHDL/SpinalHDL/discussions/1115
- Multicore NaxRiscv / tilelink cluster integrated in Litex
  - Can run doom, Buildroot and Debian
- L2 coherent cache one the way
- Looking for buddies :)