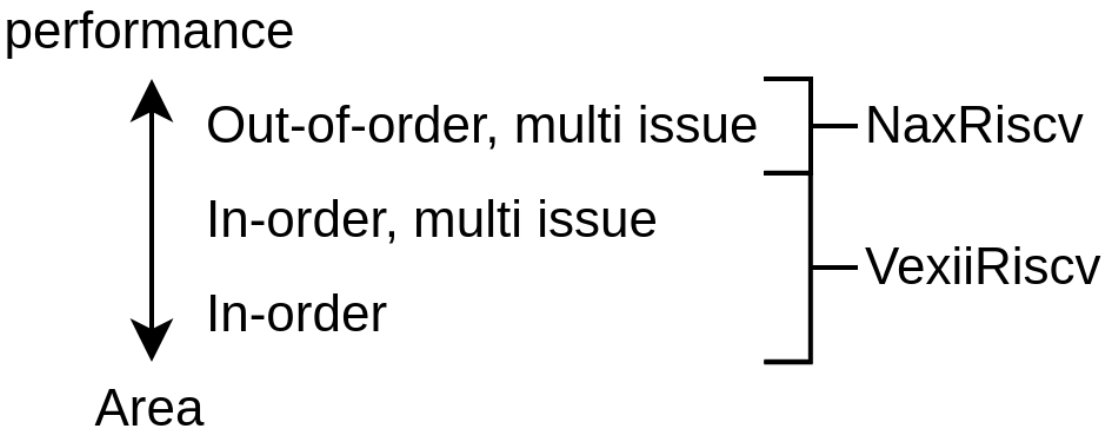
Moving toward VexiiRiscv

Charles Papon

Why VexiiRiscv

- VexRiscv limits
  - Missing features : Multi-issue / RV64 / Writeback cache
  - Could have been better : Branch prediction / framework
  - The legacy : Technical debt / Ossification
- Open-source gap between single-issue in-order / out-of-order
- Expending the CPU design space you get from one design



VexiiRiscv design

VexiiRiscv feature set

- Mainly :
  - RV32/RV64, optional MAFDCBSU
  - Can run linux / buildroot / Debian
  - Optional multi-issue
  - Optional late-ALU
  - Up to 5.24 coremark/Mhz 2.50 dhystone/Mhz
  - Free / open-source
- Aswell :
  - Optional I$ D$
  - Optional BTB / RAS / Gshare branch prediction
  - RISC-V official jtag debug

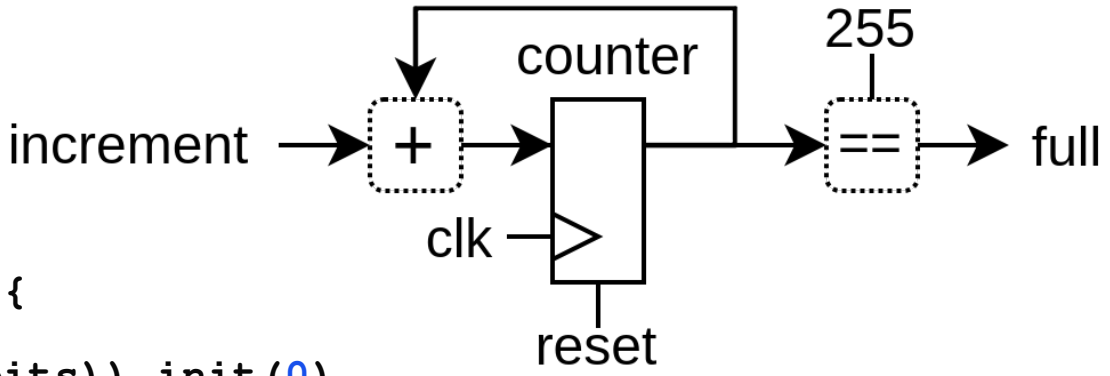
SoC / Interconnect

- Coherent interconnect via Tilelink, optional L2
- SoC and peripherals via Litex
  - https://github.com/SpinalHDL/VexiiRiscv/blob/dev/doc/litex/debian/README.md
- USB host via OHCI
  - https://spinalhdl.github.io/SpinalDoc-RTD/master/SpinalHDL/Libraries/Com/usb_ohci.html
  - 3.3V GPIO (x2) @ 48Mhz + X*12Mhz

VexiiRiscv Hardware description

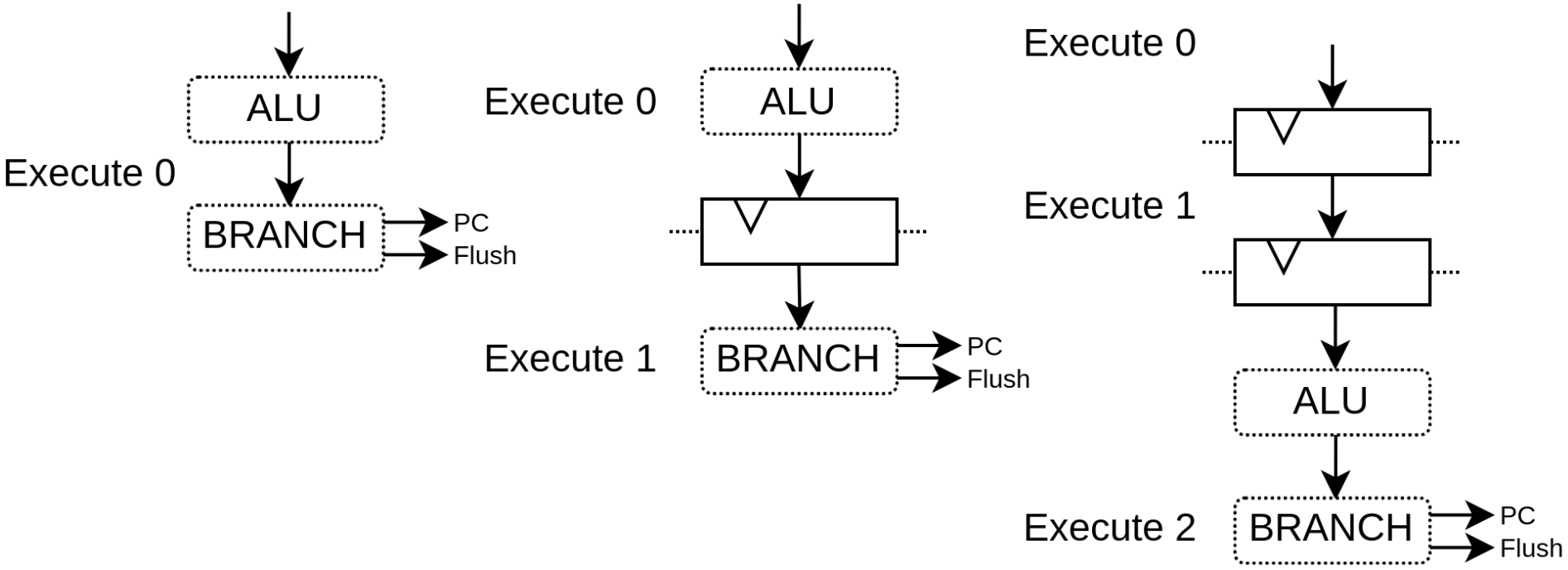
Let’s use an hardware description library

```
import spinal.core._

object Main extends App{
  SpinalVerilog(new Timer)
}

class Timer extends Component {
  val increment = in(Bool())
  val counter   = Reg(UInt(8 bits)) init(0)
  val full      = out(counter === 255)
  when(increment){
    counter := counter + 1
  }
}
```

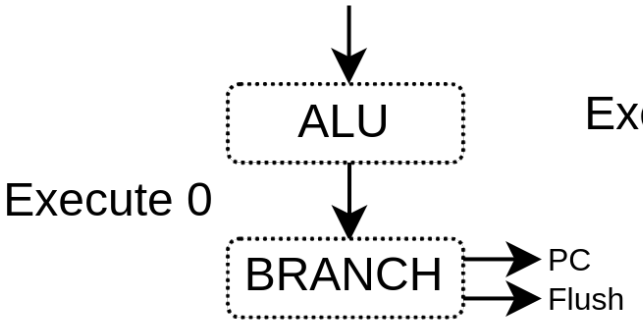


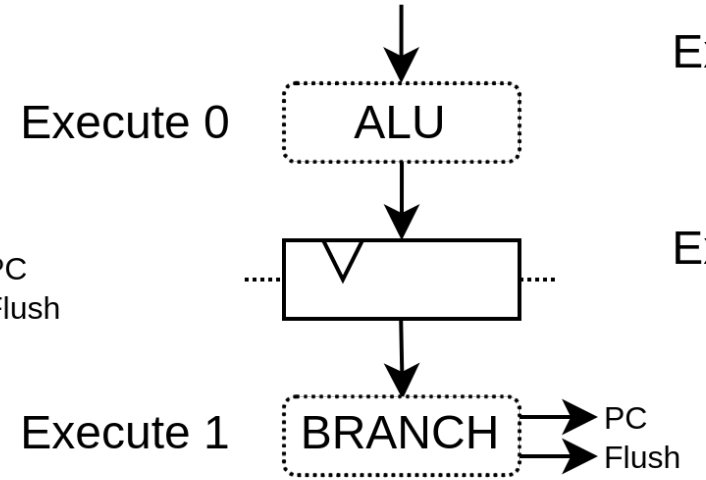

```
module Timer (
  input  wire increment,
  output wire full,
  input  wire clk,
  input  wire reset
);
  reg         [7:0]    counter;
  assign full = (counter == 8'hff);
  always @(posedge clk or posedge reset) begin
    if(reset) begin
      counter <= 8'h00;
    end else begin
      if(increment) begin
        counter <= (counter + 8'h01);
      end
    end
  end

endmodule
```

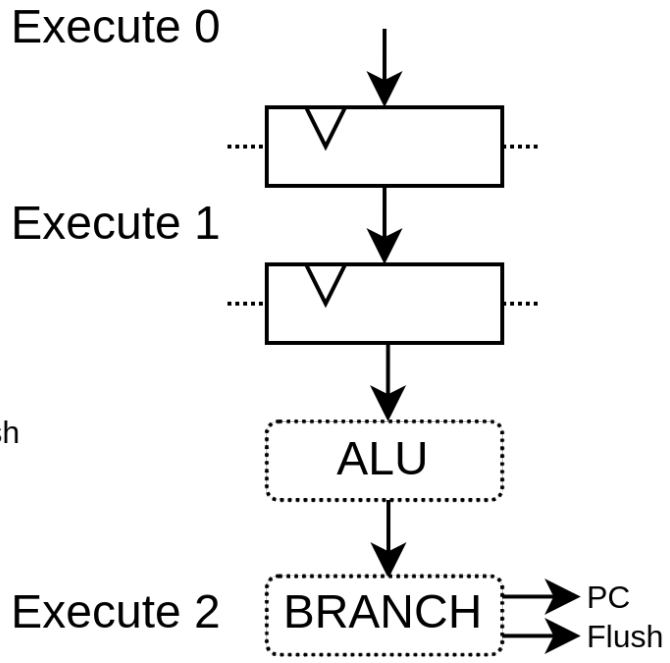
VexiiRiscv : Pipeline API



VexiiRiscv : Pipeline API

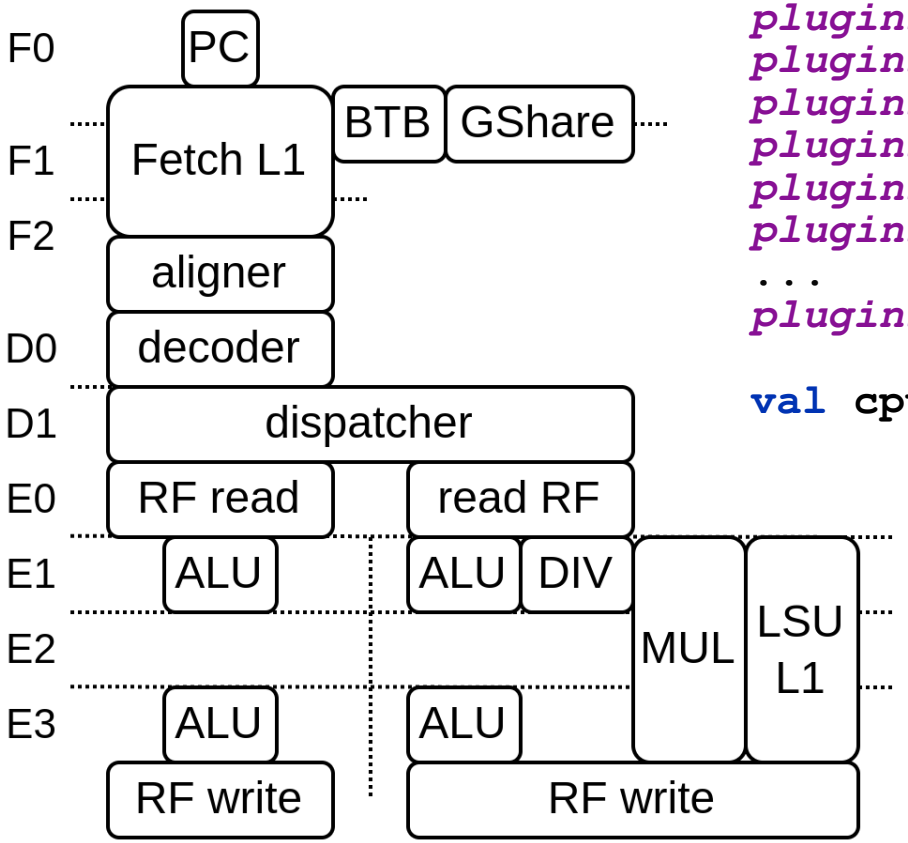






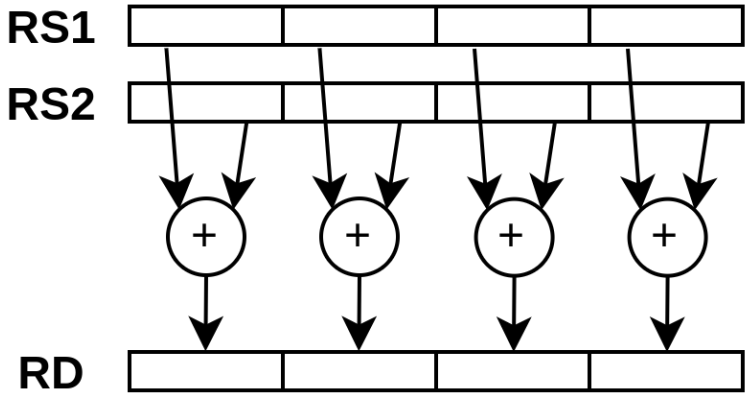

```
val aluAt = 0
val branchAt = 1
...
val alu = new pipeline.Execute(aluAt) {
  val TAKEN = insert(RS1 === RS2);
}
val branch = new pipeline.Execute(branchAt) {
  flushPort.valid := alu.TAKEN

  pcPort.valid := alu.TAKEN
  pcPort.pc := ..
}
```

No more toplevel / design space




```
class VexiiRiscv(...) extends Component{
  val database = ...
  val host     = ...
}


val plugins = ArrayBuffer[Hostable]()
plugins += new fetch.FetchPipelinePlugin()
plugins += new fetch.PcPlugin(resetVector)
plugins += new fetch.FetchL1Plugin(...)
plugins += new prediction.BtbPlugin(...)
plugins += new prediction.GSharePlugin (...)
plugins += new prediction.HistoryPlugin(...)
...
plugins += new execute.SimdAddPlugin(...)

val cpu = VexiiRiscv(plugins)
```

SIMD add plugin



```
class SimdAddPlugin(val layer : LaneLayer) extends FiberPlugin{
  ...

  val add4 = layer.add(
    IntRegFile.TypeR(M"0000000----------000-----0001011")
  )

  add4.addRsSpec(RS1, executeAt = 0)
  add4.addRsSpec(RS2, executeAt = 0)
  add4.setCompletion(0)

//add4.mayFlushUpTo(0)
//add4.dontFlushFrom(0)
//add4.reserve(something, at=0)

  val wbp = host.find[WriteBackPlugin](p =>
    p.rf == IntRegFile && p.lane == layer.lane
  )
  val wb = wbp.createPort(at = 0)
  wbp.addMicroOp(wb, add4)

  val SEL = Payload(Bool())
  layer.lane.setDecodingDefault(SEL, False)
  add4.addDecoding(SEL -> True)

  ...
}
```

```
class SimdAddPlugin(val layer : LaneLayer) extends FiberPlugin{
  ...

  val process = new layer.Execute(id = 0) {
    //Get the RISC-V RS1/RS2 values from the register file
    val rs1 = layer.lane(IntRegFile, RS1).asUInt
    val rs2 = layer.lane(IntRegFile, RS2).asUInt

    //Do some computation
    val rd = UInt(32 bits)
    rd( 7 downto  0) := rs1( 7 downto  0) + rs2( 7 downto  0)
    rd(16 downto  8) := rs1(16 downto  8) + rs2(16 downto  8)
    rd(23 downto 16) := rs1(23 downto 16) + rs2(23 downto 16)
    rd(31 downto 24) := rs1(31 downto 24) + rs2(31 downto 24)

    //Provide the computation value for the writeback
    wb.valid := isValid && SEL
    wb.payload := rd.asBits
  }

  ...
}
```

Looking for contributions

- Already a few
  - RVB (Andreas Wallner)
  - Compact divider (Thomas Kramer)
- Looking for help
  - Features additions
  - Alternative implementations
  - Performance improvements
  - Testing
  - ASIC port
  - …
- https://github.com/SpinalHDL/VexiiRiscv