Sketching (in) Hardware

Jonathan Bachrach +
Huy Vo + Andrew Waterman + Christopher Celio
Patrick Li + Ben Keller + Palmer Dabbelt +
Sebastian Mirolo + John Wawrzynek + Krste Asanović +
many more

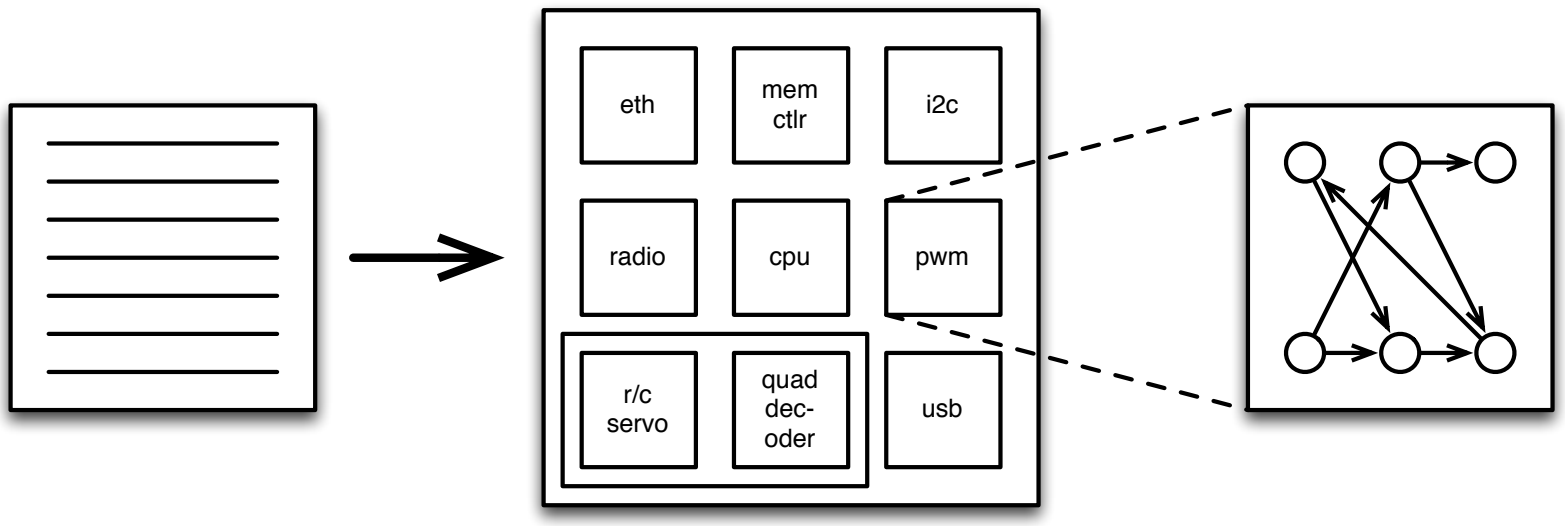
faculty @ EECS UC Berkeley
cofounder @ Otherlab

July 21, 2013

I Have a Hardware Sketching Dream

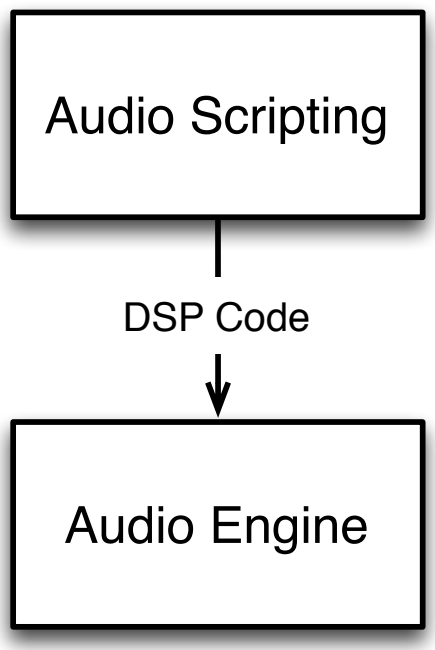
i want to sketch

- arbitrary hardware building blocks
- bigger blocks from smaller blocks
- all the down to digital logic



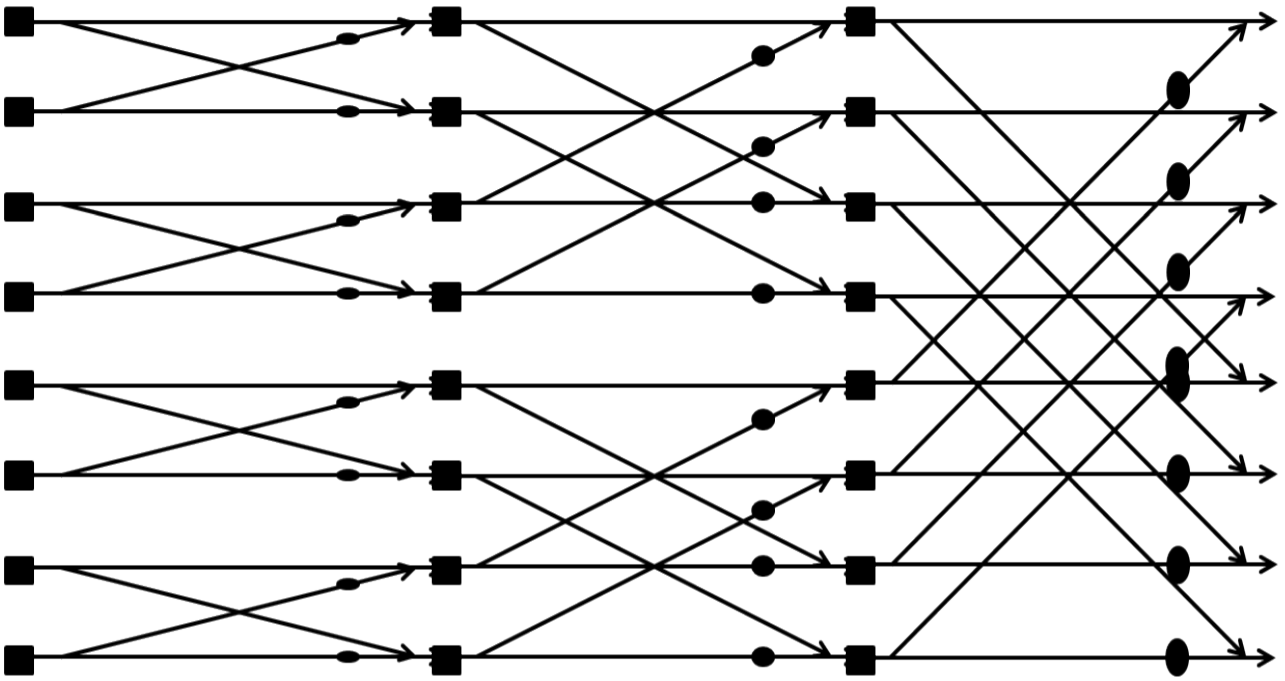
Sketching All The Way Down

- Can sketch both audio scripts and engines
- Can delay decision of what's script and what's engine



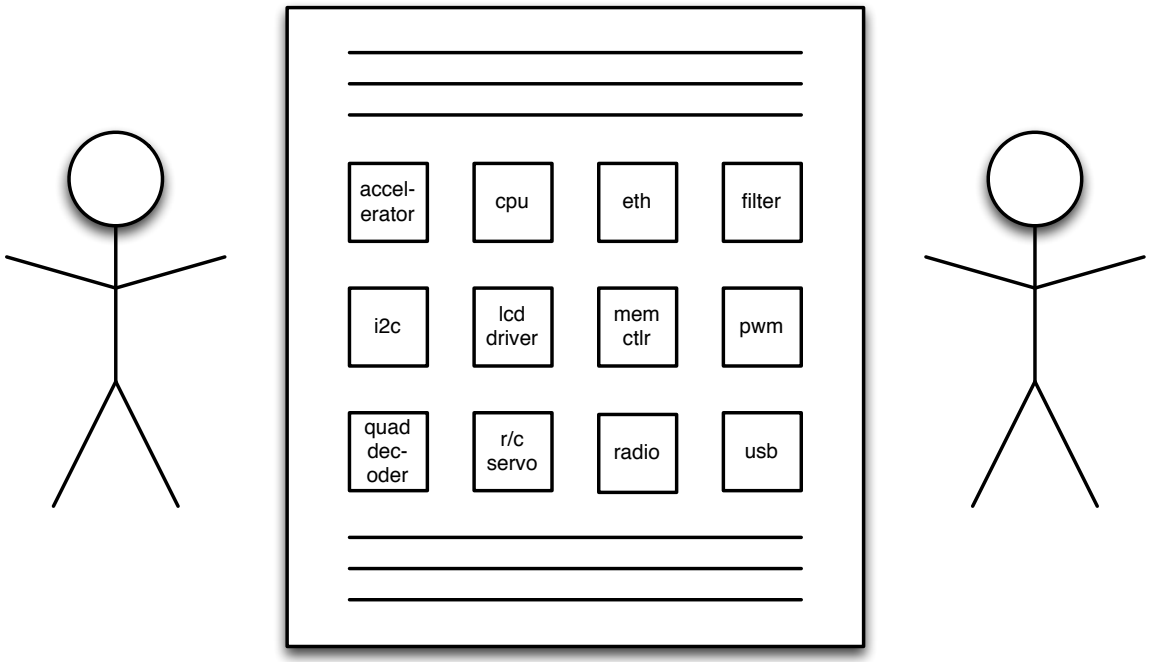
Can Sketch Truly Reusable Modules

- sketch as succinct specification as generator
- parameterized by numbers, types, functions
- abstract data types
- procedural construction



Open Source and Networkable

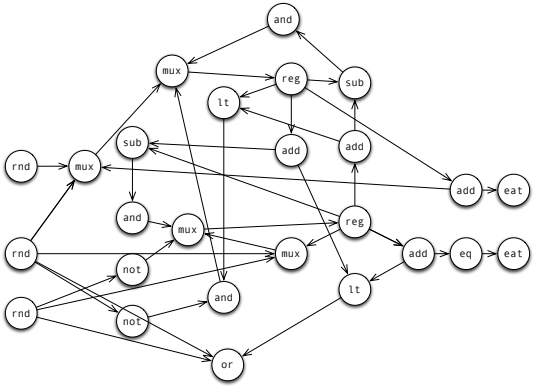
- open source
- complete library of all components
- apt-get interface
- common interface



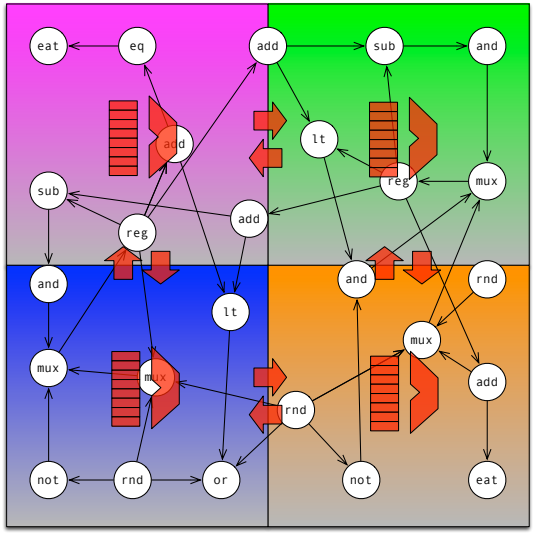
Want Powerful + Inexpensive Logic Substrate


```
import Chisel._

class GCD extends Component {
  val N = 16
  val io = new Bundle {
    val a  = UFix(INPUT,  N)
    val b  = UFix(INPUT,  N)
    val e  = Bool(INPUT)
    val z  = UFix(OUTPUT, N)
    val v  = Bool(OUTPUT)
  }
  val x  = Reg(){ UFix() }
  val y  = Reg(){ UFix() }
  when (io.e)          { x := io.a; y := io.b }
  .elsewhen  (x < y) { x := x - y }
  .otherwise         { y := y - x }
  io.z := x
  io.v := y === UFix(0)
}
```

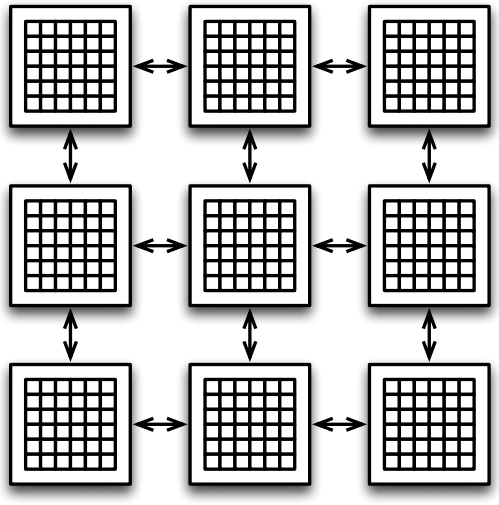
=>



=>



- fast clock rates
- scalable parallelism
- fast compilation
- automatically mapped
- logic, blocks, chips
- sketchable



Specification

- C
  - too high level
  - not enough parallelism
- Verilog
  - clumsy and longwinded
  - minimal abstraction
- Simulink
  - limited parameterization
  - WYSIWYG wiring
- limited reusability!
- lots of manual steps!

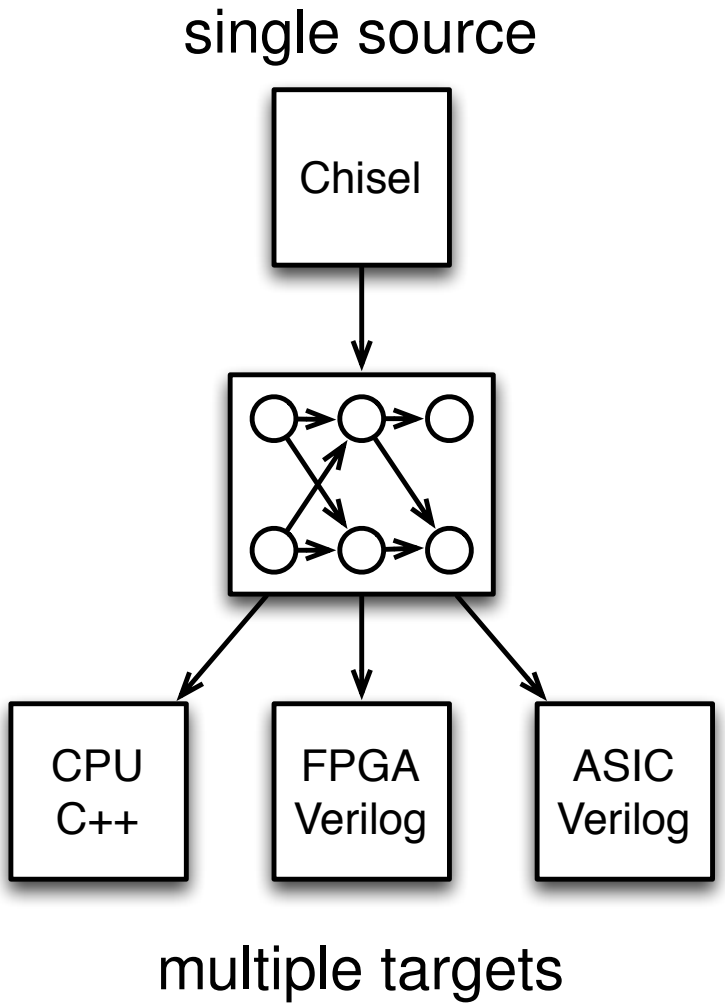
Realization

- Network of DSPs
  - limited hardware choices
  - hard to meet timing
- FPGA
  - slow to compile for
  - no virtualization
- ASIC
  - complex
  - expensive
- tedious to program
- slow to compile

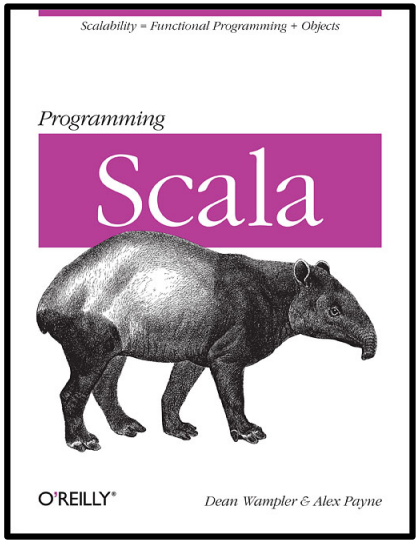
- chisel
  - design hw like software
  - soup to nuts
- DREAMER
  - new highly programmable hardware fabric
  - fast, cheap and scalable

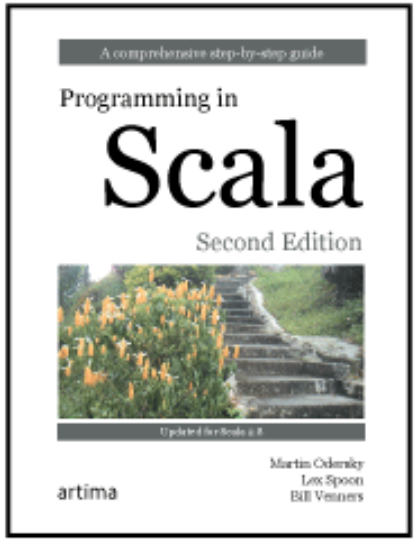
Chisel is ...

- Best of hardware and software design ideas
- Embedded within Scala language to leverage mindshare and language design
- Not Scala -> Verilog
- Algebraic construction and wiring
- Hierarchical, object oriented, and functional construction
- Abstract data types and interfaces
- Bulk connections
- Multiple targets
  - Simulation and synthesis
  - Memory IP is target-specific



- Compiled to JVM
  - Good performance
  - Great Java interoperability
  - Mature debugging, execution environments
- Object Oriented
  - Factory Objects, Classes
  - Traits, overloading etc
- Functional
  - Higher order functions
  - Anonymous functions
  - Currying etc
- Extensible
  - Domain Specific Languages (DSLs)





- Chisel has 3 primitive datatypes
  - UInt – Unsigned Integer
  - SInt – Signed Integer
  - Bool – Boolean value
- Can do arithmetic and logic with these datatypes

Example Literal Constructions

```
val sel = Bool(false)
val a   = UInt(25)
val b   = SInt(-35)
```

where `val` is a Scala keyword used to declare variables whose values won't change

Bundle

- User-extendable collection of values with named fields
- Similar to structs

```
class MyFloat extends Bundle {
  val sign        = Bool()
  val exponent    = UInt(width=8)
  val significand = UInt(width=23)
}
```

Vec

- Create indexable collection of values
- Similar to array

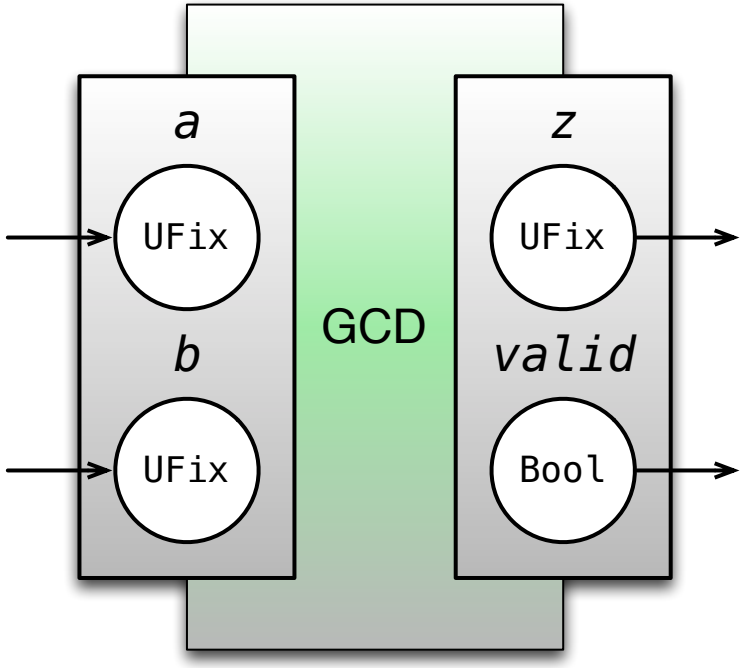
```
val myVec = Vec(5){ SInt(width=23) }
```

- The user can construct new data types
  - Allows for compact, readable code
- Example: Complex numbers
  - Useful for FFT, Correlator, other DSP
  - Define arithmetic on complex numbers

```
class Complex(val real: SInt, val imag: SInt)
    extends Bundle {
  def + (b: Complex): Complex =
    new Complex(real + b.real, imag + b.imag)
  ...
}
val a = new Complex(SInt(32), SInt(-16))
val b = new Complex(SInt(-15), SInt(21))
val c = a + b
```

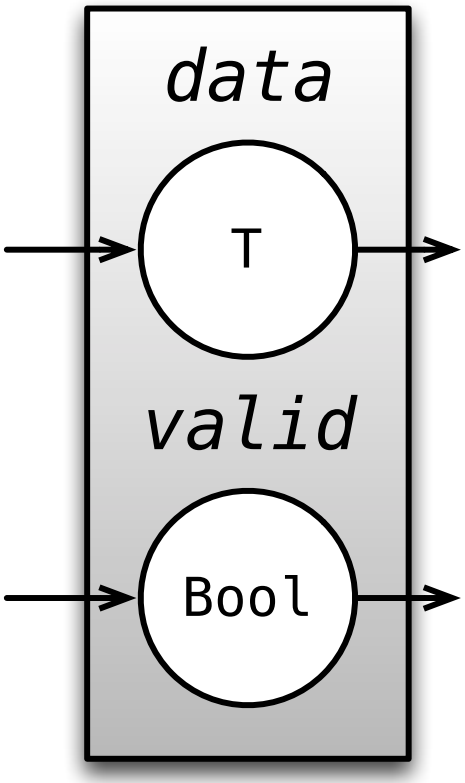
Example


```
class GCD extends Module {
  val io = new Bundle {
    val a     = UInt(INPUT, 16)
    val b     = UInt(INPUT, 16)
    val z     = UInt(OUTPUT, 16)
    val valid = Bool(OUTPUT) }
  val x = Reg(resetVal = io.a)
  val y = Reg(resetVal = io.b)
  when (x > y) {
    x := x - y
  } .otherwise {
    y := y - x
  }
  io.z     := x
  io.valid := y === UInt(0)
}
```



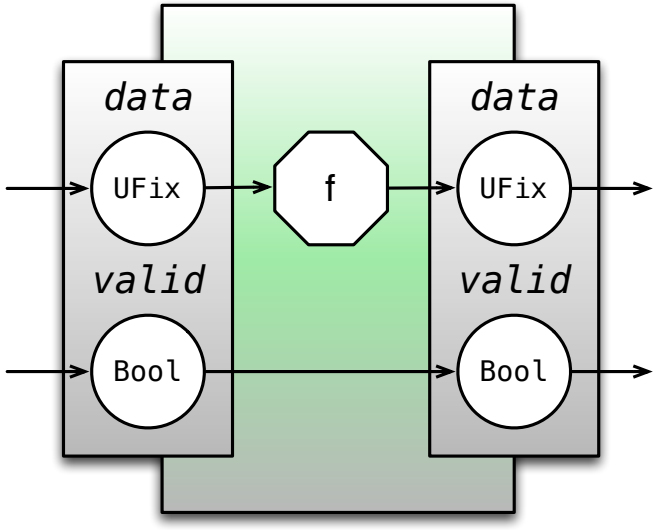
Valid Wrapper


```
class Valid[T <: Data](dtype: T) extends Bundle {
  val data  = dtype.clone
  val valid = Bool()
  override def clone = new Valid(dtype)
}

class GCD extends Module {
  val io = new Bundle {
    val a   = UInt(INPUT, 16)
    val b   = UInt(INPUT, 16)
    val out = new Valid(UInt(OUTPUT, 16))
  } }
  ...
  io.out.data  := x
  io.out.valid := y === UInt(0)
}
```



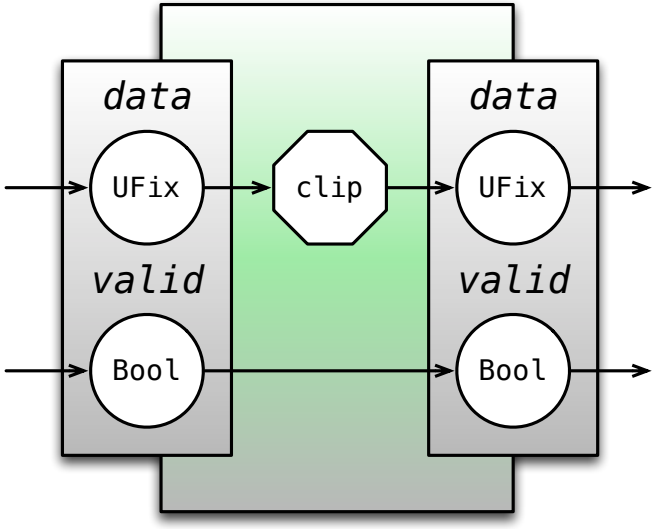
Function Filters


```
abstract class Filter[T <: Data](dtype: T) extends Module {
  val io = new Bundle {
    val in  = new Valid(dtype).asInput
    val out = new Valid(dtype).asOutput
} }

class FunctionFilter[T <: Data](f: T => T, dtype: T) extends Filter(dtype) {
  io.out.valid := io.in.valid
  io.out       := f(io.in)
}
```



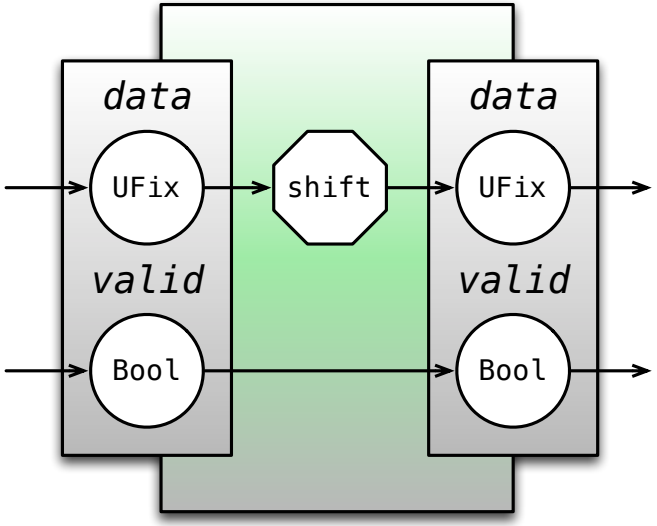
Clipping Filter


```
def clippingFilter[T <: Num](limit: Int, dtype: T) =
  new FunctionFilter(min(limit, max(-limit, _)), dtype)
```



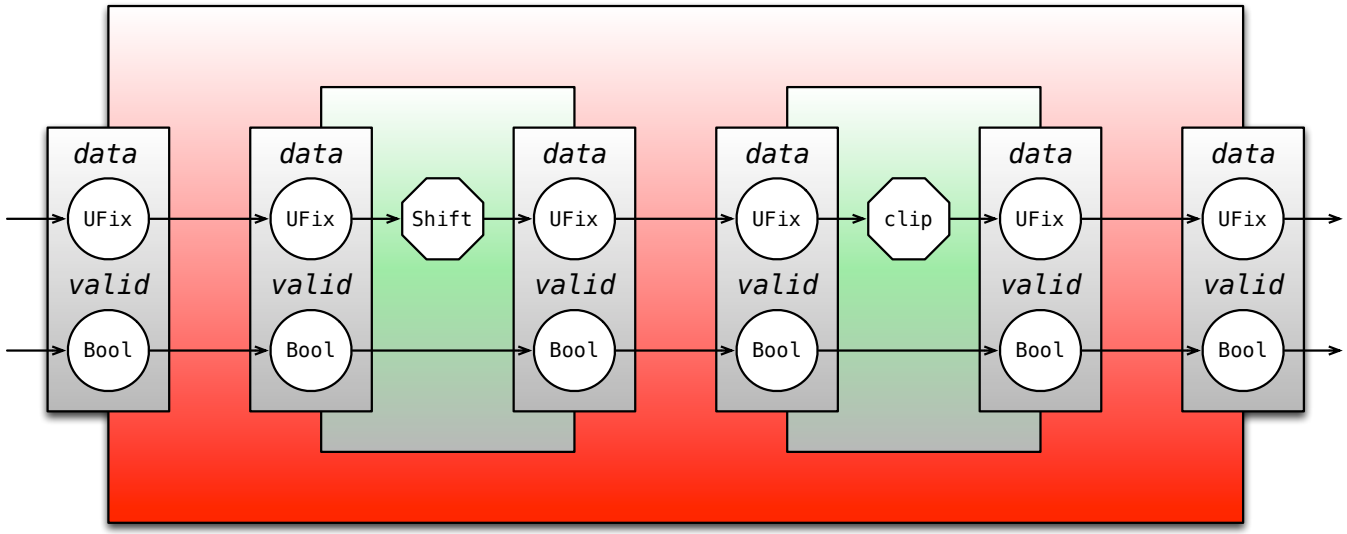
Shifting Filter


```
def shiftingFilter[T <: Num](shift: Int, dtype: T) =
  new FunctionFilter(_ >> shift, dtype)
```

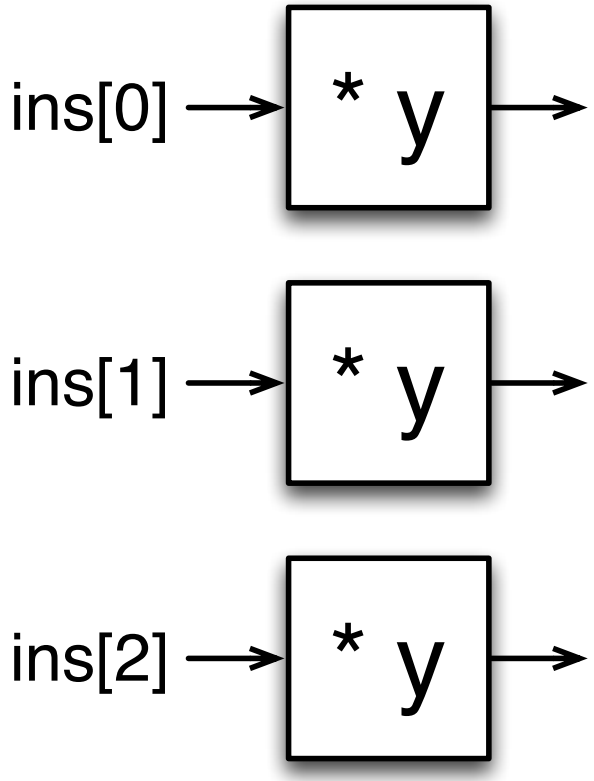


Chained Filter

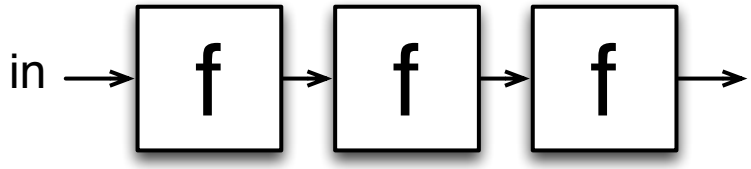

```
class ChainedFilter[T <: Num](dtype: T) extends Filter(dtype) = {
  val shift   = new ShiftFilter(2, dtype)
  val clipper = new ClippingFilter(1 << 7, dtype)
  io.in          <> shift.io.in
  shift.io.out   <> clipper.io.in
  clipper.io.out <> io.out
}
```



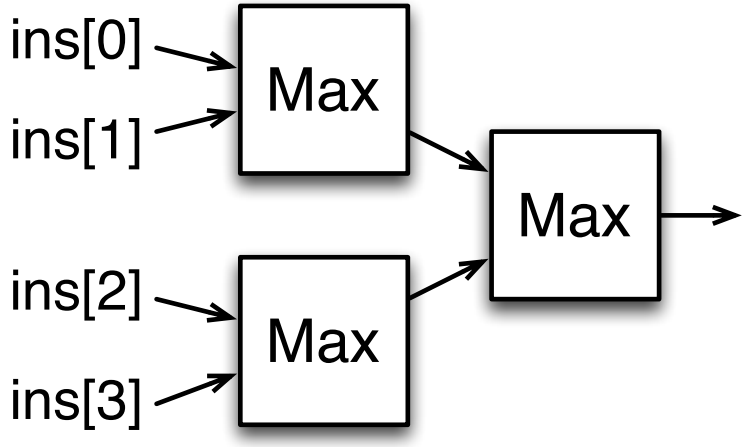
```
Map(ins, x => x * y)
```




```
Chain(n, in, x => f(x))
```




```
Reduce(ins, Max)
```



```
def delays[T <: Data](x: T, n: Int): List[T] =
  if (n <= 1) List(x) else x :: taps(Reg(x), n-1)

def FIR[T <: Num](hs: Seq[T], x: T): T =
  (hs, delays(x, hs.length)).zipped.map( _ * _ ).reduce( _ + _ )

class TstFIR extends Filter(SInt(width = 8)) {
  val io = new Bundle{ val x  = SInt(INPUT, 8); val y = SInt(OUTPUT, 8) }
  val h  = Array(SInt(1), SInt(2), SInt(4))
  io.y  := FIR(h, io.x)
}
```

$$y[n] = \sum_{k=0}^{N-1} x[n-k]h[k]$$

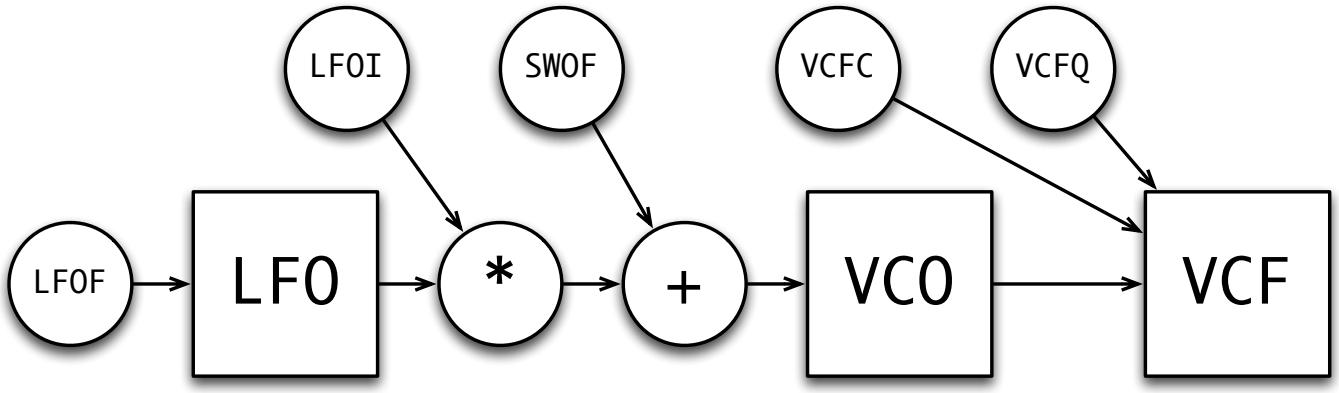
- Flo and Dbl data types and ops
- Add FP support in C++ backend
- Audio harness with mics, speakers, and controls

Emulated Korg Monotron

- Monotron is a portable classic analog synth
- Built out of SawWave, LFO, mixer, and VCF
- Use laptop / C++ for emulation
- Use BCF-2000 USB based mixer for controls

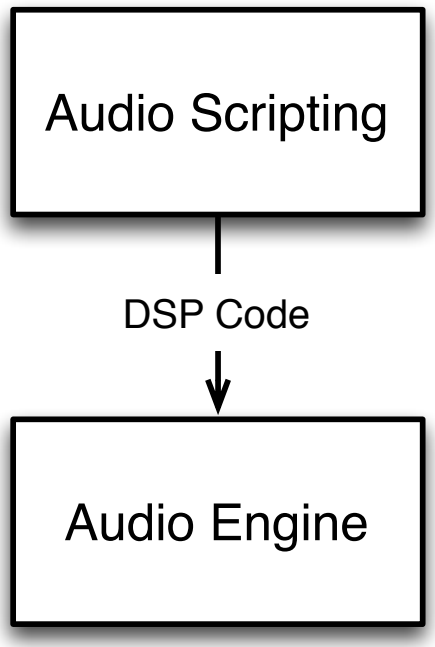
Chiseled Korg Monotron


```
class Monotron extends Module {
  val io = new Bundle {
    val swof = Dbl(INPUT);
    val lfof = Dbl(INPUT); val lfoi = Dbl(INPUT);
    val vcfc = Dbl(INPUT); val vcfq = Dbl(INPUT);
    val out  = Dbl(OUTPUT);
  }
  val lfo = io.lfoi * SawWave(io.lfof);
  val vco = SawWave(io.swof + lfo)
  val vcf = VCF(io.vcfc, io.vcfq, vco);
  io.out := vcf
}
```



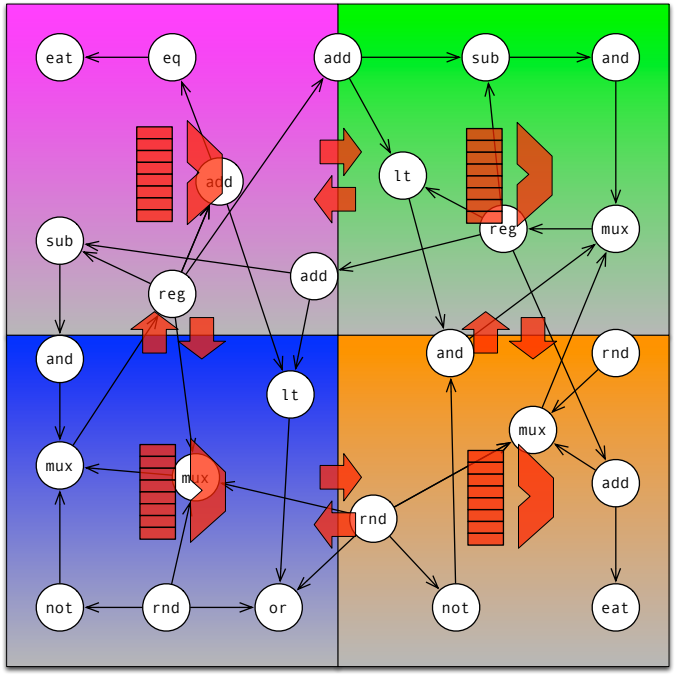
Wiring All The Way Down

- Can write both audio scripts and engines in Chisel
- Can choose which part is baked into hardware
- For example, can map entire DSP to FPGA or ASIC



Chisel Graph Execution on DREAMER

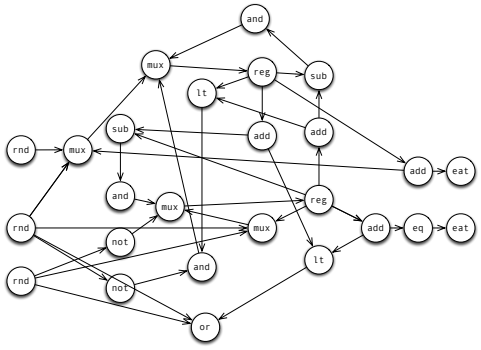
- spatial fabric of graph execution tiles
- map piece of graph to each core
- have network route intertile dataflow values
- use dataflow scheduling to hide latency
- coarser grained high level chisel instructions



```
import Chisel._

class GCD extends Component {
  val N = 16
  val io = new Bundle {
    val a  = UFix(INPUT,  N)
    val b  = UFix(INPUT,  N)
    val e  = Bool(INPUT)
    val z  = UFix(OUTPUT, N)
    val v  = Bool(OUTPUT)
  }
  val x  = Reg(){ UFix() }
  val y  = Reg(){ UFix() }
  when (io.e)        { x := io.a; y := io.b }
  .elsewhen  (x < y) { x := x - y }
  .otherwise         { y := y - x }
  io.z := x
  io.v := y === UFix(0)
}
```

chisel

=>



graph

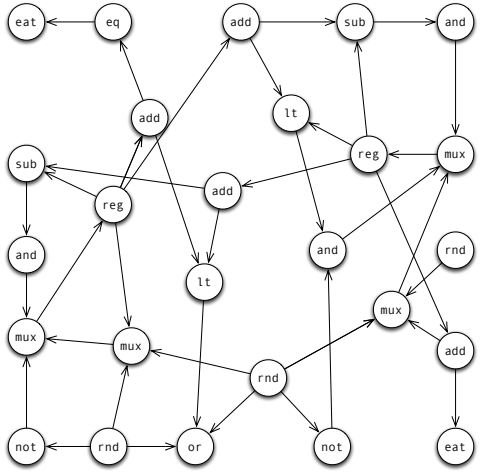
=>

```
GCD__io_z = eat GCD__x
GCD__io_a = rnd/16
GCD__io_e = rnd/1
T5 = mux GCD__io_e GCD__io_a GCD__x
T12 = sub/16 GCD__x GCD__y
T6 = and T12 65535
T13 = lt/16 GCD__x GCD__y
T9 = not/1 GCD__io_e
T7 = and T9 T13
T4 = mux T7 T6 T5
GCD__x = reg 1 T4
GCD__io_b = rnd/16
T2 = mux GCD__io_e GCD__io_b GCD__y
T14 = sub/16 GCD__y GCD__x
T3 = and T14 65535
T8 = lt/16 GCD__x GCD__y
T11 = or GCD__io_e T8
T10 = not/1 T11
T1 = mux T10 T3 T2
GCD__y = reg 1 T1
T0 = eq GCD__y 0
GCD__io_v = eat T0
```

netlist

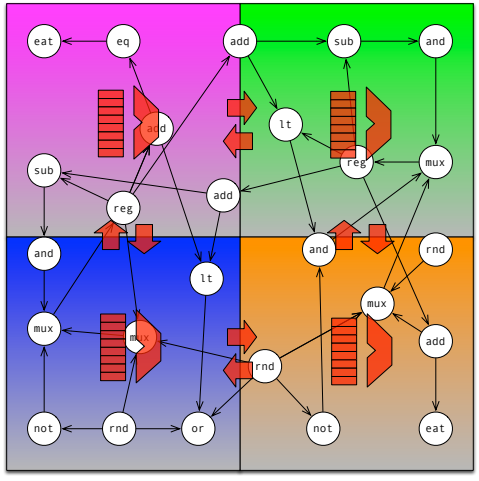
```
GCD__io_z = eat GCD__x
GCD__io_a = rnd/16
GCD__io_e = rnd/1
T5 = mux GCD__io_e GCD__io_a GCD__x
T12 = sub/16 GCD__x GCD__y
T6 = and T12 65535
T13 = lt/16 GCD__x GCD__y
T9 = not/1 GCD__io_e
T7 = and T9 T13
T4 = mux T7 T6 T5
GCD__x = reg 1 T4
GCD__io_b = rnd/16
T2 = mux GCD__io_e GCD__io_b GCD__y
T14 = sub/16 GCD__y GCD__x
T3 = and T14 65535
T8 = lt/16 GCD__x GCD__y
T11 = or GCD__io_e T8
T10 = not/1 T11
T1 = mux T10 T3 T2
GCD__y = reg 1 T1
T0 = eq GCD__y 0
GCD__io_v = eat T0
```

=>



layout

=>



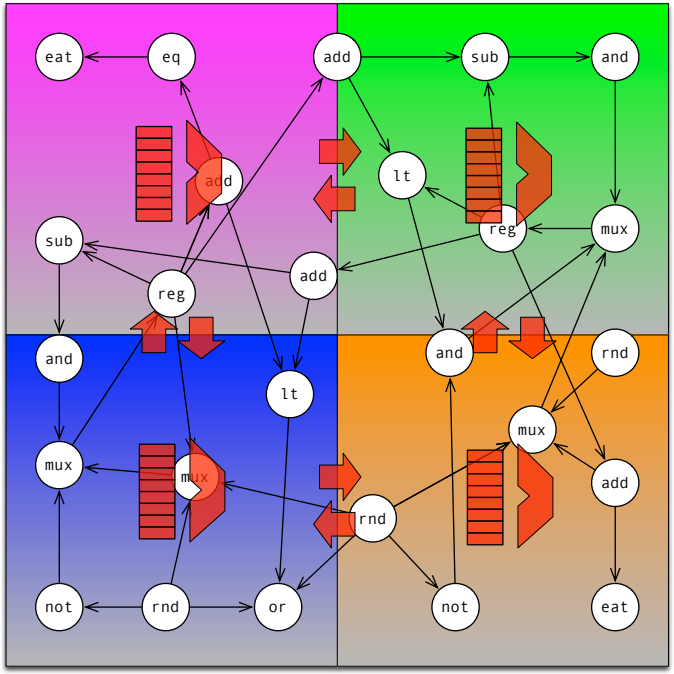
execution

DREAMER Properties

- **efficient to compile to** – 10-100x faster than FPGA
- **efficient to run** – nearly as fast as FPGAs
- **quick to probe any signal** – no recompile necessary
- **easily scalable** – multiple chips
- **easy to map large designs** – auto FAME + nice DRAM interface

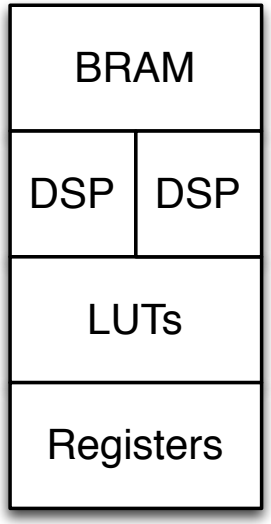
additional facilities

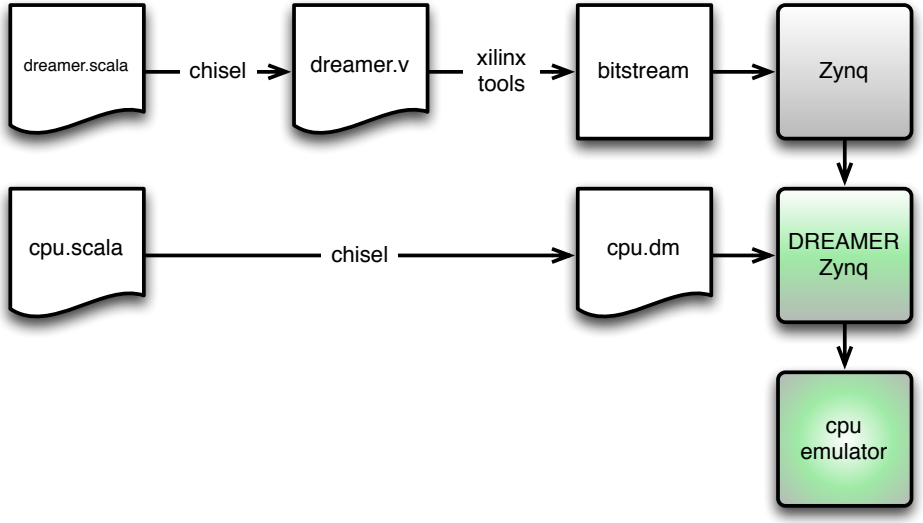
- debugging and tracing
- activity counters for energy
- fault injection



FPGA Mapping Opportunity

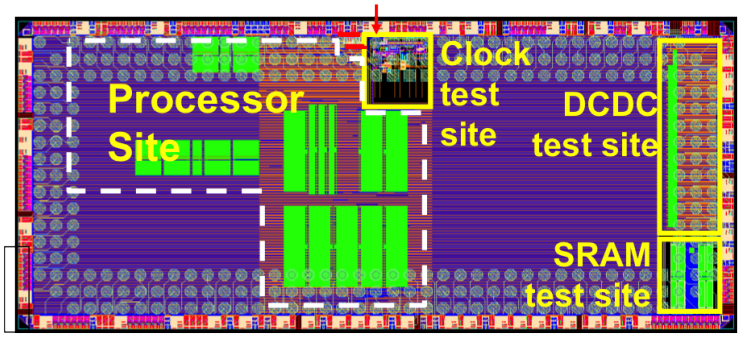
- FPGAs have great density and economies of scale
- program FPGA with DREAMER once
- then throw away Xilinx tools
- match DSP + BRAM density
- map to few BRAMs using port scheduling
- double pump BRAM for extra ports

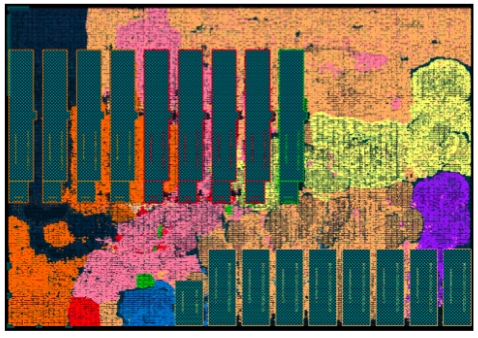


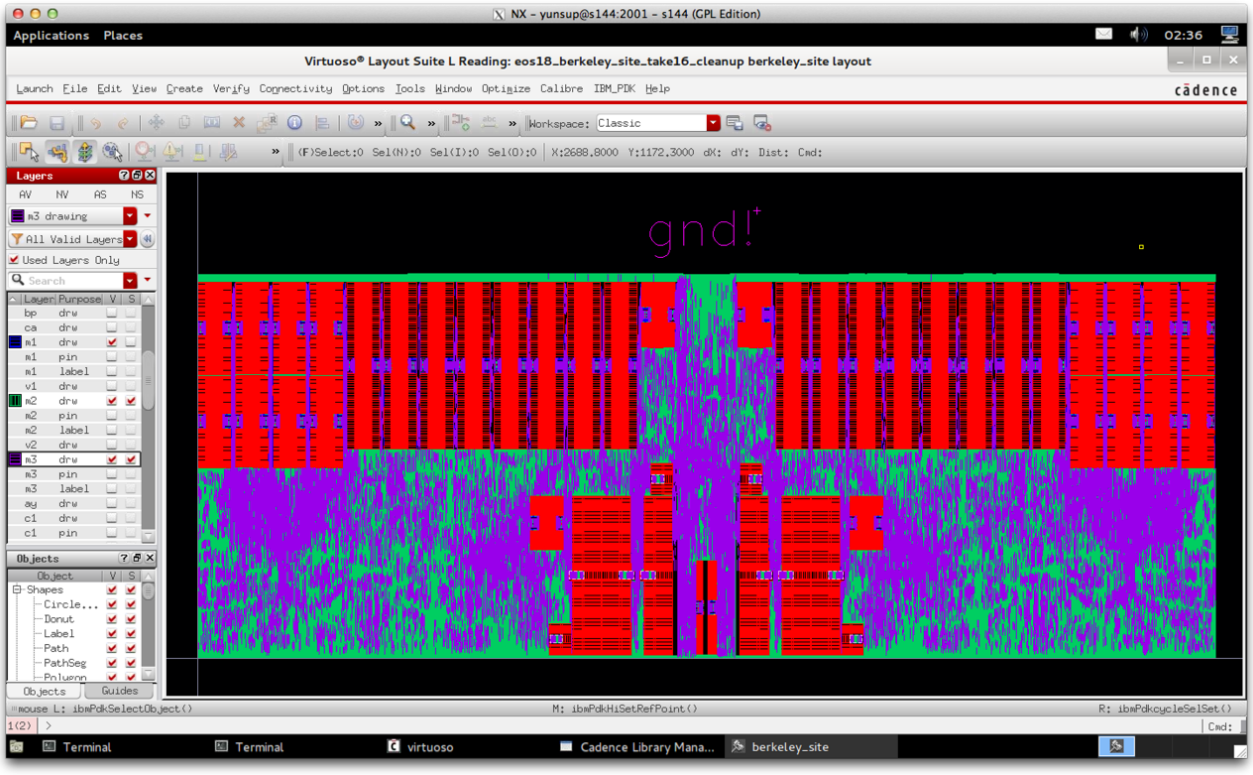


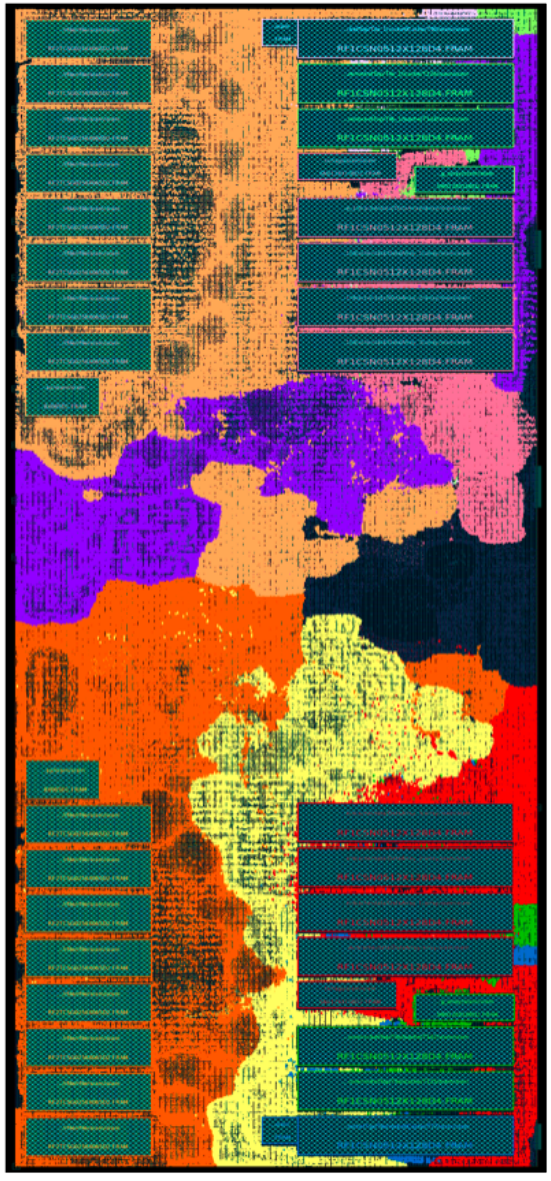
Chisel is Real

Digital Circuits Written in Chisel









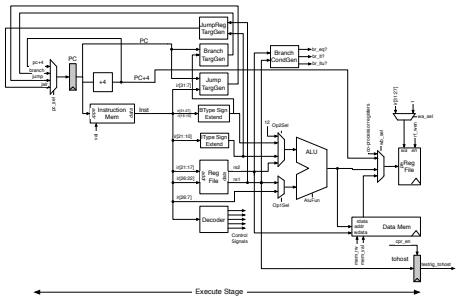
Chisel is Open Source

- `chisel.eecs.berkeley.edu`
- BSD License
- complete set of documentation
- one goal is creation of library of high level and reusable components

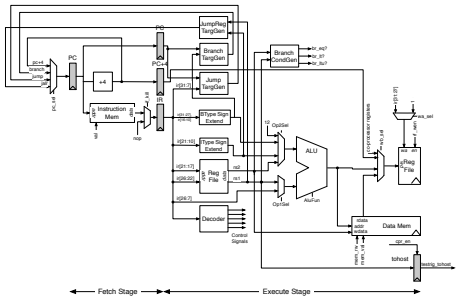
Chisel Contains Library of Modules

- queues, pipe,
- prioritymux, decoders, encoders,
- fixed-priority arbiters, round-robin arbiters,
- popcount, scoreboards
- ROMs, RAMs, CAMs, TLB, caches, prefetcher,
- integer ALUs, LFSR, Booth multiplier, iterative divider
- IEEE-754/2008 floating-point units

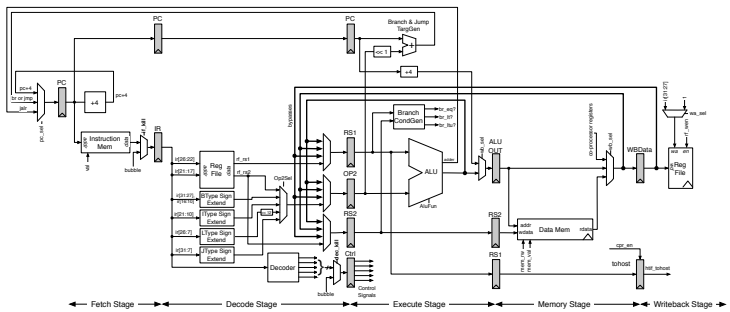
- fifth Berkeley RISC ISA
- open source specification
- fast functional simulator
- boots linux
- lots of open source implementations



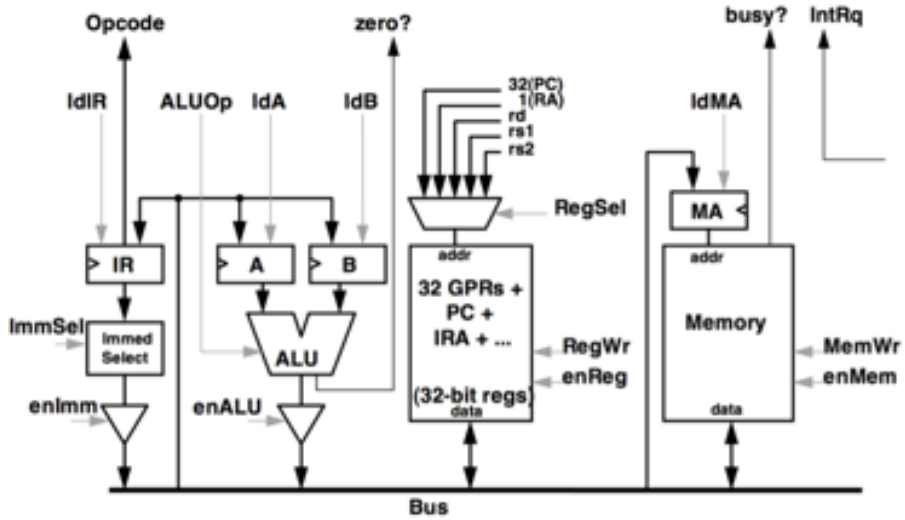
1 stage



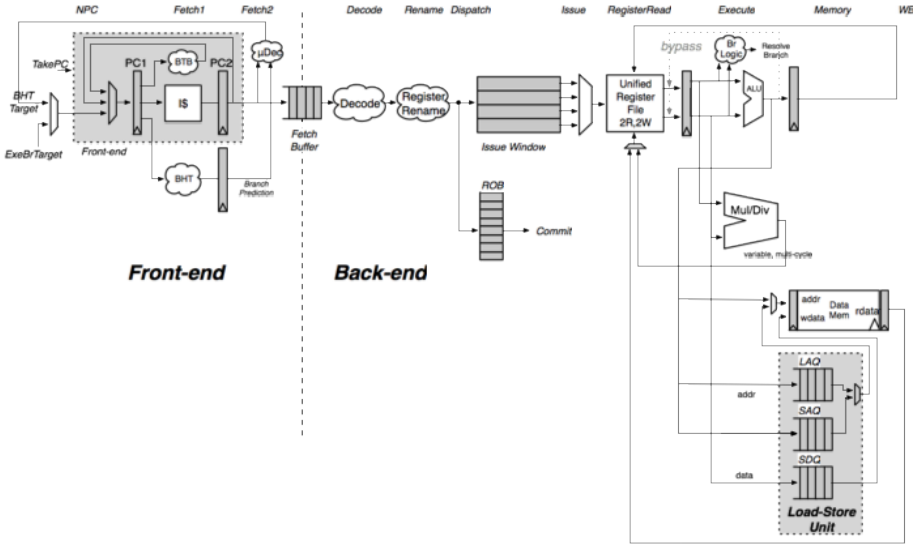
2 stage



5 stage



microcode



out of order

UC Berkeley Classes

- 2x CS152 – Undergraduate Computer Architecture
  - Sodor
  - Multicore and Vector
- 2x CS250 – VLSI System Design
  - Processors
  - Image Processing
- 1x CS294-88 – Declarative Design Seminar
  - High Level Specification
  - Automated Design Space Exploration

Outside Projects

- NOC generator – MSR
- Monte Carlo Simulator – TU Kaiserslautern
- Precision Timed Machine (PRET) – Edward Lee's Group
- Chisel-Q – Quantum Backend – John Kubiatowicz's Group

- sketching all the way down
- powerful new hardware substrate
- truly open source reusable hardware
- printable electronics ready

funding

- Project Isis: DoE Award DE-SC0003624.
- Par Lab: Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support came from Par Lab affiliates Nokia, NVIDIA, Oracle, and Samsung.
- ASPIRE: DARPA PERFECT program, Award HR0011-12-2-0016.