Workshop on Open Source Hardware Development Tools and RISC-V

MohammadHossein AskariHemmat

Shahid Bahonar University of Kerman

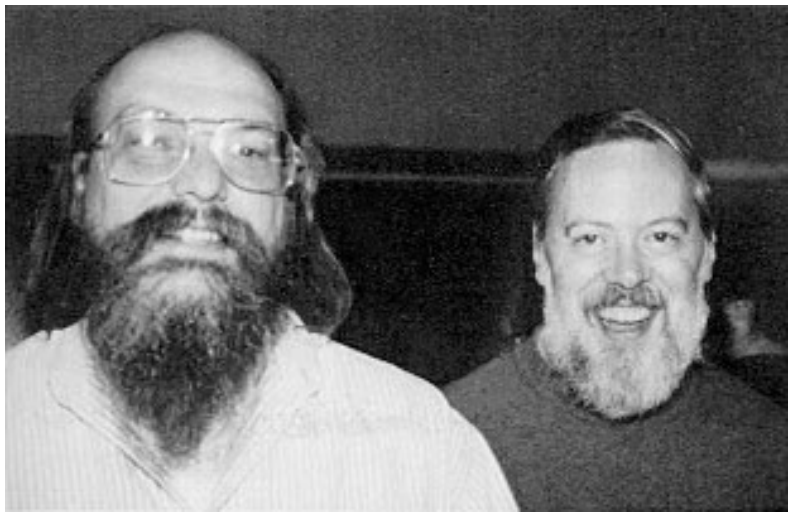
August 24, 2017

Agenda

1. Open Source Tools
   - History of Free and Open Source Software
   - Open Source Softwares
   - Open Source Softwares for Hardware Development
   - Demo: Open-Source Tools for Lattice FPGA Development
   - Lab1: Working with Open Source Hardware Development Tools
2. Working with Chisel and RISC-V
   - Why we need new hardware description language?
   - What is functional programming?
   - Scala and Chisel short intro
   - Chisel Demo: Full Adder
   - Lab2: Working with Chisel
   - RISC-V Intro
   - RISC-V Standard Base ISA Details
   - RISC-V Demo
   - Lab3: RISC-V Lab

History of Open Source Tools

- In early 1970 UNIX operating system was developed by Kenneth Thompson (Berkeley), Dennis Ritchie (Harvard) in AT&T Bell Labs



- UNIX came with no cost for researchers BUT no permission for redistribution was given
- Under UNIX terms, you were not even allowed to distribute a modified version of UNIX!

History of Open Source Tools

- In 1984 Richard Stallmans Free Software Foundation (FSF) began the GNU project

- The objective was to create a free version of UNIX operating system namely GNU operating system

History of Open Source Tools

- In 1984 Richard Stallmans Free Software Foundation (FSF) began the GNU project

- The objective was to create a free version of UNIX operating system namely GNU operating system
- By free, he meant a software that can be freely used, read, modified, and redistributed

History of Open Source Tools

- GNU Project is the base of many tools that we currently use!

- Some of the GNU tools that you might have heard of:
  - GCC: GNU Compiler Collection which is a suite of compilers for several programming languages
  - GDB: GNU DeBugger
  - GNU Binutils: A suite of tools including linker, assembler and other tools
  - And a lot more: Autoconf, Make, Bison ...

History of Open Source Tools

- By 1991, the only missing part for GNU Project to be a full operating system was the kernel
- In 1991, Linus Torvalds, began working on an operating system kernel

- His Kernel was able to be combined with FSF components to produce a usable operating system
- He named this combination Linux

History of Open Source Tools

- The name Linux is confused by many to be an operating system

- Yet, Linux is only the kernel
- Many use the term Gnu/Linux to point out that Linux is only the kernel and it needs GNU tool chain to be a fully usable

History of Open Source Tools

Nowadays, all big companies such as Google, Microsoft, Facebook and many more are promoting their users to use their Open Source products

History of Open Source Tools

As an example, currently, Google has 1055 repositories on Github some of which you use everyday!

Open Source Softwares

- Free Software was defined by FSF and it is often confused with programs whose executables are given away at no charge
- The other miss understanding is that the term "Open Source" is often replaced with "Free Software" and vice-versa
- Any Free Software, by definition, is an Open Source where the opposite may not be valid
- Distributing a program under the name of Free Software adds more moral aspects to your program

So What is an Open Source software?

Open Source Softwares

According to OpenSource.org, an Open Source software has the following criteria:

- The user must be free to redistribute it
- The source code must be available
- The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software

The full list of criteria can be found here:

https://opensource.org/osd.html

Open Source Softwares

Benefits of using Open Source tools:

- Customizability
- Auditability
- Cost
- Security

Hidden costs of using Open Source tools:

- Long learning curve
- Semi-restrictive Licenses

Open Source Softwares

Read more about Open Source and Free Software:

- https://opensource.org/
- http://www.fsf.org/
- https://en.wikipedia.org/wiki/GNU_Project
- https://www.gnu.org/

Open Source Softwares for Hardware Development

Unfortunately, there is not much interest in Open Source tools for Hardware Development:

- CAD companies like Cadence and Synopsys do not promote using open source tools for hardware development
- Almost all CAD tools used by silicon vendors are proprietary and requires expensive license agreement
- Researches around the world use proprietary tools which require expensive license agreement
- Sharing source code and releasing tools as open source is not a common practice at all

BUT, there is hope!

Open Source Softwares for Hardware Development

There are open source solutions for FPGA and ASIC development.

Icarus Verilog

VeriPool

CHISEL

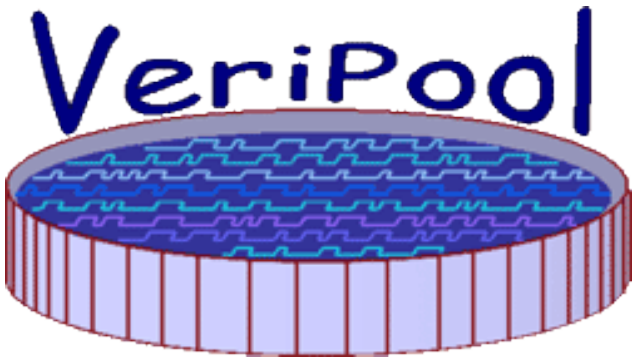
MyHDL

Open Source Softwares for Hardware Development

Icarus Verilog:

- Icarus Verilog is a Verilog simulation and synthesis tool
- It is developed and maintained by Stephen Williams
- Aimed to generate code by back-end tools (for example chisel)
- It does not fully support Systemverilog
- It is an open source tool under GPL
- Project page: http://iverilog.icarus.com/

Open Source Softwares for Hardware Development



Veripool:

- Veripool was established in 1998 as a repository of the authors' ASIC and electrical engineering tools
- A collection of open source tools for ASIC development for example:
  - Verilator is the fast Verilog to C/C++/SystemC compiler
  - Full list: https://www.veripool.org/projects
- It is an open source project licensed under GPL
- Project page: https://www.veripool.org/

Open Source Softwares for Hardware Development

GTKWave:

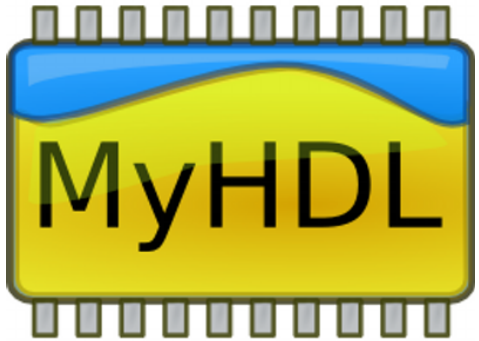
- A fully featured GTK+ based wave viewer
- It is published as a free software
- Project page: http://gtkwave.sourceforge.net/

Open Source Softwares for Hardware Development

Chisel:

- Constructing Hardware in a Scala Embedded Language
- Chisel is an **open source** hardware **construction** language developed at UC Berkeley
- Project page: https://chisel.eecs.berkeley.edu/

Open Source Softwares for Hardware Development



MyHDL:

- MyHDL is a free, open source package for using Python as a hardware description and verification language
- MyHDL converts design to Verilog or VHDL
- Project page: http://myhdl.org/

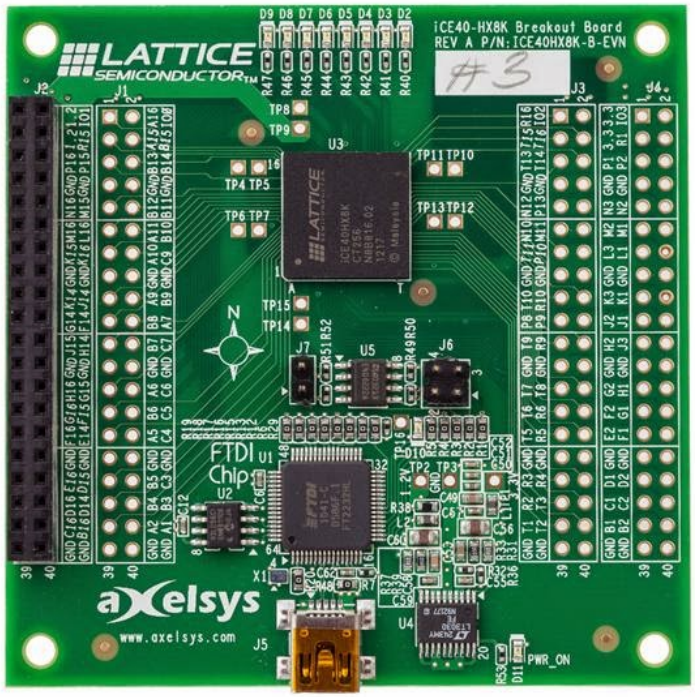
Open Source Softwares for Hardware Development

Tools developed by Clifford Wolf:

- YOSYS: A framework for RTL synthesis tools
- Project IceStorm: Lattice iCE40 FPGAs Bitstream Documentation (Reverse Engineered)
- riscv-formal: RISC-V Formal Verification Framework
- Projects page: https://github.com/cliffordwolf

Demo: Open-Source Tools for Lattice FPGA Development

In this demo, I will go through synthesis and implementation of a simple design on a Lattice FPGA



Lab1: Working with Open Source Hardware Development Tools

Follow instructions here

https://goo.gl/LctK9W

Why we need new hardware description language?

- Existing HDLs (Verilog/VHDL) are too low-level and closed source
- Depending on tool vendor, new HDLs such as Systemverilog and Bluespec have different language support
- Simulation languages such as SystemC are too far from synthesis
- Bluespec is closed source but high-level
- High Level Synthesis (HLSs) are ineffective for many domains

Chisel have solved these problems

Chisel is open source!

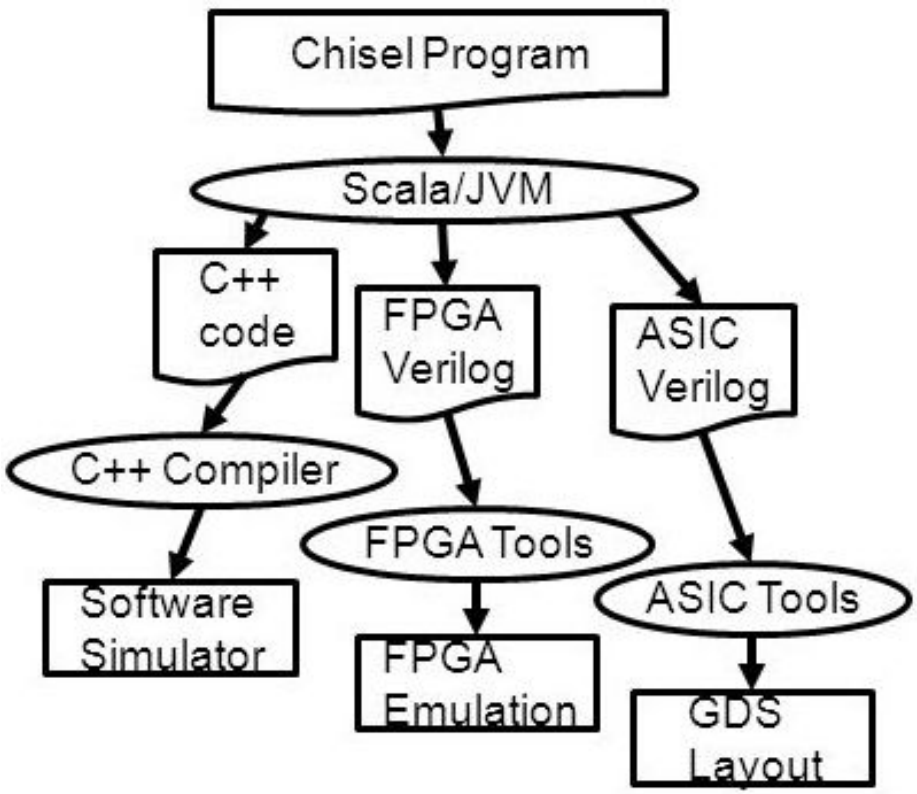
- Chisel source code is freely available and anyone can contribute to the project: https://github.com/ucb-bar/chisel
- Researchers are highly recommended to contribute to the project.
- Chisel has a Google user group, Join! https://goo.gl/EPnhnp

Chisel is embedded in the Scala programming language

- Traditional HDLs, do not use the most user friendly syntax. They also lack many advanced features that are currently adopted by many advanced languages such as object oriented, abstract data types, functional construction and etc...

Ability to choose different back-ends

- With a single Chisel source code, you can choose the back-end for implementation



The generated code from chisel is compatible to be used by any standard ASIC or FPGA tools

Chisel features

A full list of Chisel features can be found here:
https://chisel.eecs.berkeley.edu/

Learn more on why to learn Chisel:
https://goo.gl/44LzgZ

Chisel Pre-requisites

Unfortunately, Chisel has long learning curve and previous to learn Chisel, users need to know the following technologies:

- Object Oriented Programming: Inheritance, Abstract classes, Interfaces, Parameterization and Polymorphism
- Functional Programming: Higher-order functions, Recursion, Currying etc.
- Ability to design and understand RTL in any HDL

Chisel Pre-requisites

Usually, people with hardware design experience lack knowledge in Functional and Object Oriented Programming. The following are a good source for learning these pre-requisite:

- For Scala you can take this online course on Coursera: https://www.coursera.org/learn/progfun1
- For Object Oriented Programming, you take any advanced programming course. This one is in Farsi: https://maktabkhooneh.org/course/bazargan466

What is functional programming?

What is functional programming?

- In functional programming, programs are executed by evaluating expressions, in contrast with imperative programming where programs are composed of statements which change global state when executed.
- Functions are first class citizens (like any variable in other type of programming languages)
  This means that you can pass and return functions. This is called Higher Order Functions in Functional programming

There are other features that defines Functional Programming. For now we only rely on above features

Scala and Chisel short intro

The following slides are copied from Chisel Bootcamp which can be found here:
https://chisel.eecs.berkeley.edu/latest/chisel-bootcamp.pdf

Scala Bindings

```
// constant
val x = 1
val (x, y) = (1, 2)

// variable
var y = 2
y = 3
```

Scala Iteration

```
val tbl = new Array[Int](256)

// loop over all indices
for (i <- 0 until tbl.length)
  tbl(i) = i

// loop of each sequence element
val tbl2 = new ArrayBuffer[Int]
for (e <- tbl)
  tbl2 += 2*e

// loop over hashmap key / values
for ((x, y) <- vars)
  println("K " + x + " V " + y)
```

Scala Functions

```
// simple scaling function, e.g., x2(3) => 6
def x2 (x: Int) = 2 * x
```

```
// more complicated function with statements
def f (x: Int, y: Int) = {
  val xy = x + y;
  if (x < y) xy else -xy
}
```

Scala Functional

```
// simple scaling function, e.g., x2(3) => 6
def x2 (x: Int) = 2 * x
```

```
// produce list of 2 * elements, e.g., x2list(List(1, 2, 3)) => List(2, 4, 6)
def x2list (xs: List[Int]) = xs.map(x2)
```

```
// simple addition function, e.g., add(1, 2) => 3
def add (x: Int, y: Int) = x + y
```

```
// sum all elements using pairwise reduction, e.g., sum(List(1, 2, 3)) => 6
def sum (xs: List[Int]) = xs.foldLeft(0)(add)
```

Scala Object Oriented

```
class Blimp(r: Double) {
  val rad = r
  println("Another Blimp")
}

new Blimp(10.0)
```

```
class Zep(h: Boolean, r: Double) extends Blimp(r) {
  val isHydrogen = h
}

new Zep(true, 100.0)
```

Scala Singleton Objects

- like Java class methods
- for top level methods

```
object Blimp {
  var numBlimps = 0
  def apply(r: Double) = {
    numBlimps += 1
    new Blimp(r)
  }
}

Blimp.numBlimps
Blimp(10.0)
```

Scala, Things You Should Know

Refer to the following wiki to know what you need to know from Scala before working with Chisel:

https://goo.gl/CCETuC

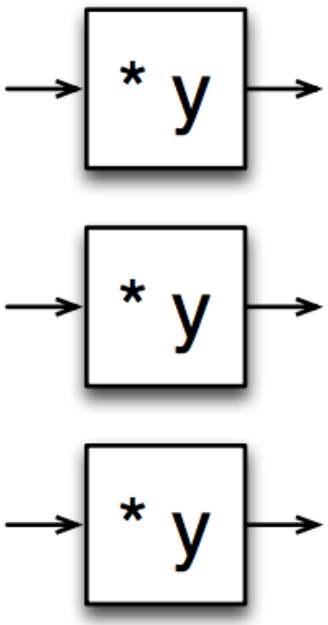
Chisel VS Scala

- Chisel is a Scala library
- Do not use var unless you are professional Chisel user and you are working on Chisel source code!
- Complete list can be found here: https://github.com/freechipsproject/chisel3/wiki/Scala-land-vs.-Chisel-land

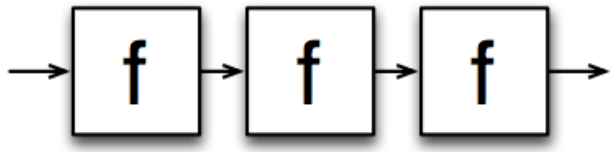
Using Constructional Blocks

There are three constructional blocks that are frequently used in Chisel:

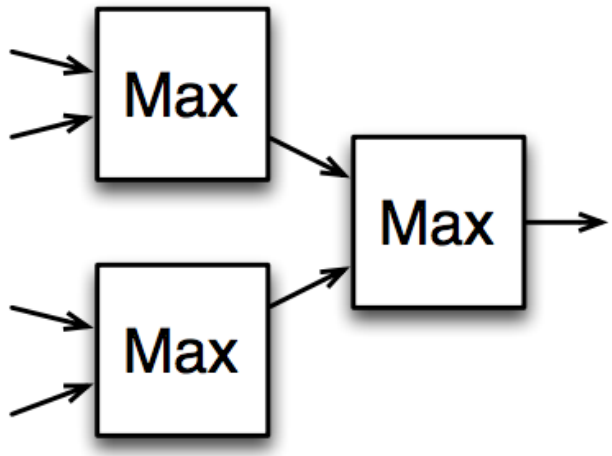
```
Map(ins, x => x * y)
```




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

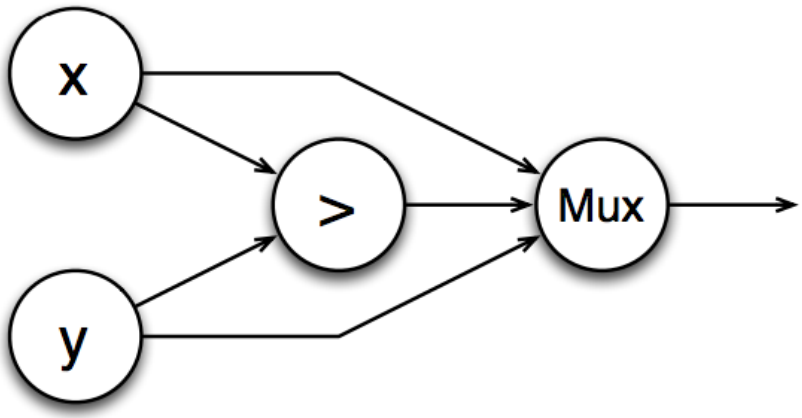



```
Reduce(data, Max)
```



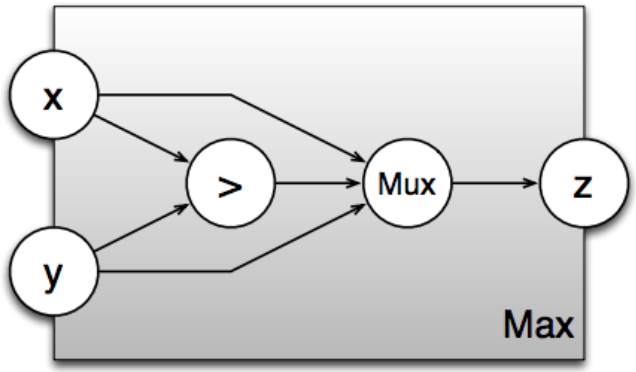
Algebraic Graph Construction

```
Mux(x > y, x, y)
```



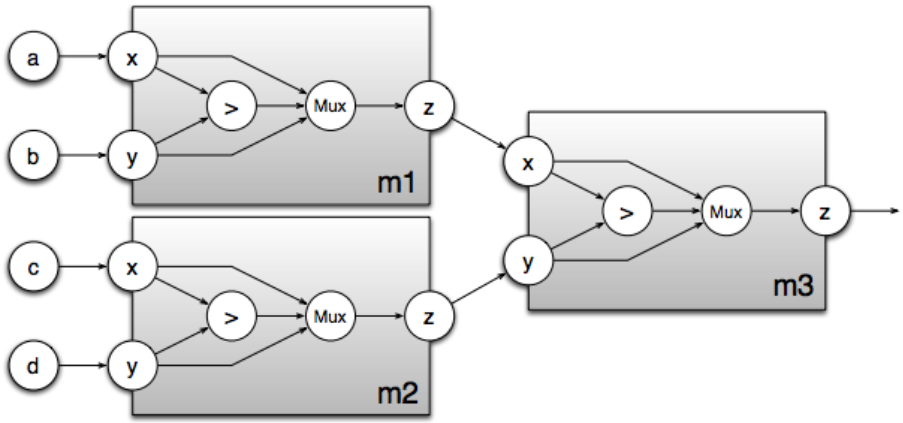
Creating Module

```
class Max2 extends Module {
  val io = new Bundle {
    val x = UInt(INPUT, 8)
    val y = UInt(INPUT, 8)
    val z = UInt(OUTPUT, 8) }
  io.z := Mux(io.x > io.y, io.x, io.y)
}
```



Connecting Module

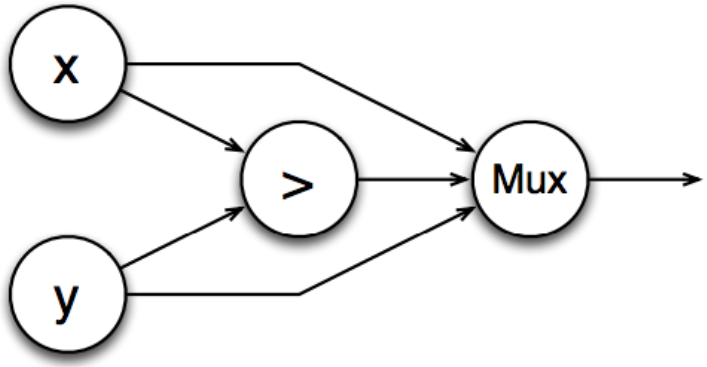
```
val m1 =
  Module(new Max2())
m1.io.x := a
m1.io.y := b
val m2 =
  Module(new Max2())
m2.io.x := c
m2.io.y := d
val m3 =
  Module(new Max2())
m3.io.x := m1.io.z
m3.io.y := m2.io.z
```



Defining Construction Functions

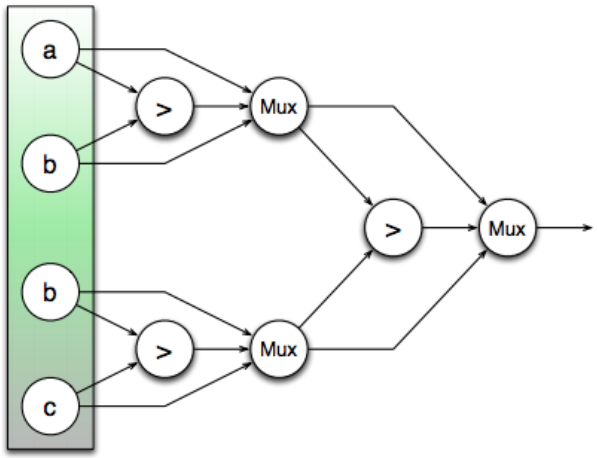
```
def Max2(x, y) = Mux(x > y, x, y)
```

```
Max2(x, y)
```



Functional Construction

```
class MaxN(n: Int, w: Int) extends Module {
  val io = new Bundle {
    val in  = Vec.fill(n){ UInt(INPUT, w) }
    val out = UInt(OUTPUT, w)
  }
  io.out := io.in.reduceLeft(Max2)
}
```



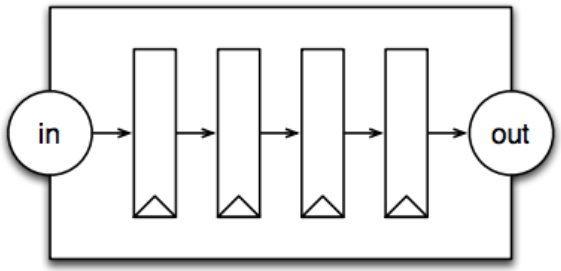
Using Registers

```
// clock the new reg value on every cycle
val y = io.x
val z = Reg(next = y)
```

```
// clock the new reg value when the condition a > b
val x = Reg(UInt())
when (a > b) { x := y }
.elsewhen (b > a) { x := z }
.otherwise { x := w }
```

Unconditional Register Update

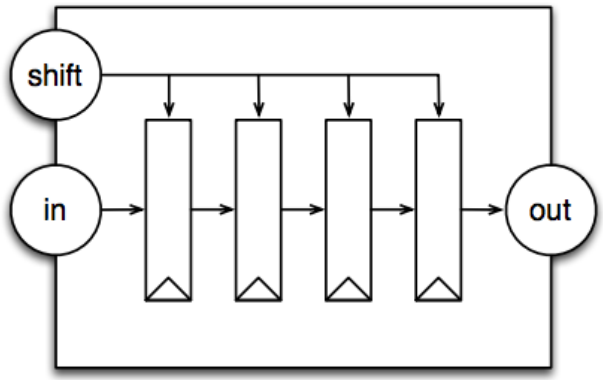
```
class ShiftRegister extends Module {
  val io = new Bundle {
    val in  = UInt(INPUT, 1)
    val out = UInt(OUTPUT, 1)
  }
  val r0 = Reg(next = io.in)
  val r1 = Reg(next = r0)
  val r2 = Reg(next = r1)
  val r3 = Reg(next = r2)
  io.out := r3
}
```




```
module ShiftRegister(input clk, input reset,
    input  io_in,
    output io_out);

  reg[0:0] r3;
  reg[0:0] r2;
  reg[0:0] r1;
  reg[0:0] r0;

  assign io_out = r3;
  always @(posedge clk) begin
    r3 <= r2;
    r2 <= r1;
    r1 <= r0;
    r0 <= io_in;
  end
endmodule
```

Conditional Register Update

```
class ShiftRegister extends Module {
  val io = new Bundle {
    val in    = UInt(INPUT, 1)
    val shift = Bool(INPUT)
    val out   = UInt(OUTPUT, 1)
  }

  val r0 = Reg(UInt())
  val r1 = Reg(UInt())
  val r2 = Reg(UInt())
  val r3 = Reg(UInt())

  when (io.shift) {
    r0 := io.in
    r1 := r0
    r2 := r1
    r3 := r2
  }
  io.out := r3
}
```



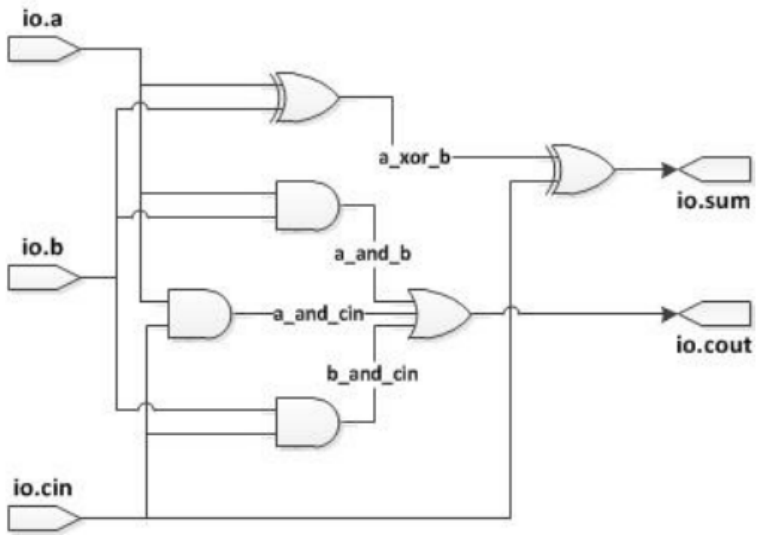
Chisel Cheat Sheet

You can find chisel cheat sheet here:
https://chisel.eecs.berkeley.edu/2.2.0/chisel-cheatsheet.pdf

Chisel Demo: Full Adder

Let's Design a Full adder in Chisel! The following code is in Chisel2, not compatible with docker image.

```
class FullAdder extends Module {
  val io = new Bundle {
    val a    = UInt(INPUT, 1)
    val b    = UInt(INPUT, 1)
    val cin  = UInt(INPUT, 1)
    val sum  = UInt(OUTPUT, 1)
    val cout = UInt(OUTPUT, 1)
  }
  // Generate the sum
  val a_xor_b = io.a ^ io.b
  io.sum := a_xor_b ^ io.cin
  // Generate the carry
  val a_and_b   = io.a & io.b
  val b_and_cin = io.b & io.cin
  val a_and_cin = io.a & io.cin
  io.cout :=
    a_and_b | b_and_cin | a_and_cin
}
```



Lab2: Working with Chisel

Follow instructions here

https://goo.gl/xEpiQM

RISC-V intro

- At Berkeley, in 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research, it was time to look at ISA for next set of projects
- Obvious choices: x86 and ARM
- x86 is too complex and ARM has lots of IP issues (story time!)

Some of RISC-V platinum members

RISC-V has attracted many companies

Who are the key people working on this project?




Intel x86 AAA Instruction

- ASCII Adjust after Addition
- AL register is default source and des/na/on
- If the low nibble is >9 decimal, or the auxiliary carry flag AF $= 1$, then
  - Add 6 to low nibble of AL and discard overflow
  - Increment high byte of AL
  - Set CF and AF
- Else
  - CF $=$ AF $= 0$
- Single byte instruction

Totally useless!

What is RISC-V?

- A new free and open ISA developed at UC Berkeley starting in 2010 (ParLab and ASPIRE)
- It is a FREE ISA (as in free software)
- Designed for
  - research
  - education
  - commercial use

Whats Different about RISC-V?

- Simple
  - Far smaller than other commercial ISAs
- Clean-slate design
  - Clear separation between user and privileged ISA
  - Avoids micro architecture or technology-dependent features
- A modular ISA
  - Small standard base ISA
  - Multiple standard extensions
- Stable
  - Base and standard extensions are frozen
  - Additions via optional extensions, not new versions

Whats Different about RISC-V?

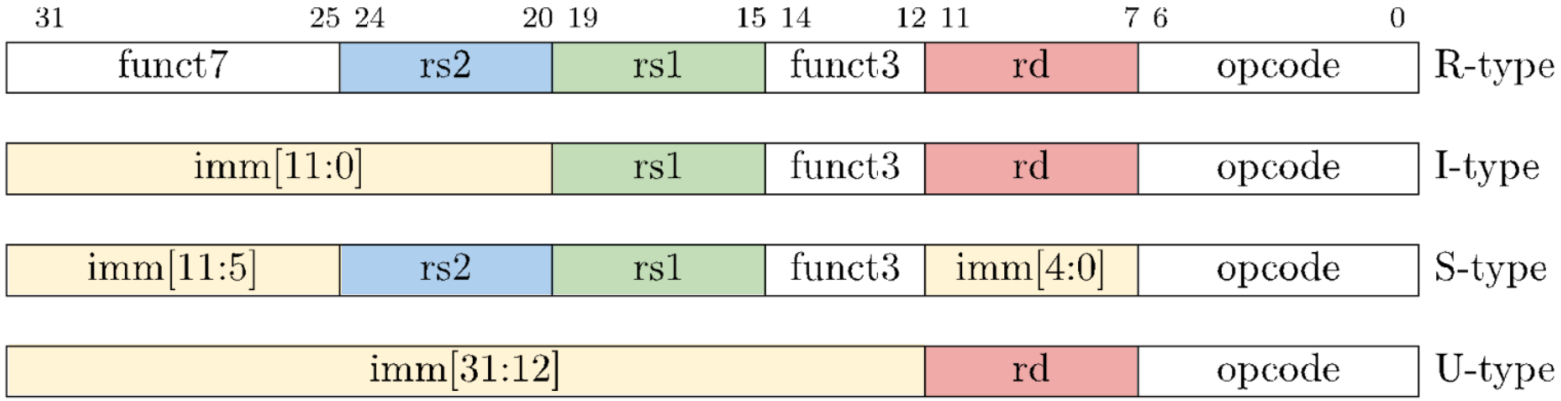
RISC-V has reduced IP licensing much simpler. As an example, on SiFive website, you can buy an IP based on RISC-V in a matter of minutes!

The process is much longer and much more expensive if other ISAs (x86 or ARM) is chosen

RISC-V is NOT an Open-Source Processor!

- RISC-V is an ISA specification
- Want to encourage both open-source and proprietary implementations of the RISC-V ISA specification
- Most of cost of hardware design is software, so make sure software can be reused across many chip designs
- Expand to have open specifications for whole platforms, including I/O and accelerators

RISC-V Standard Base ISA Details



- 32-bit fixed-width, naturally aligned instructions
- 31 integer registers x1-x31, plus x0 zero register
- rd/rs1/rs2 in fixed location, no implicit registers
- Immediate field (instr[31]) always sign-extended
- Floating-point adds f0-f31 registers plus FP CSR, also fused mul-add four-register format

ARMv8 ISA VS RISC-V ISA

Copied from risc-v workshop 2

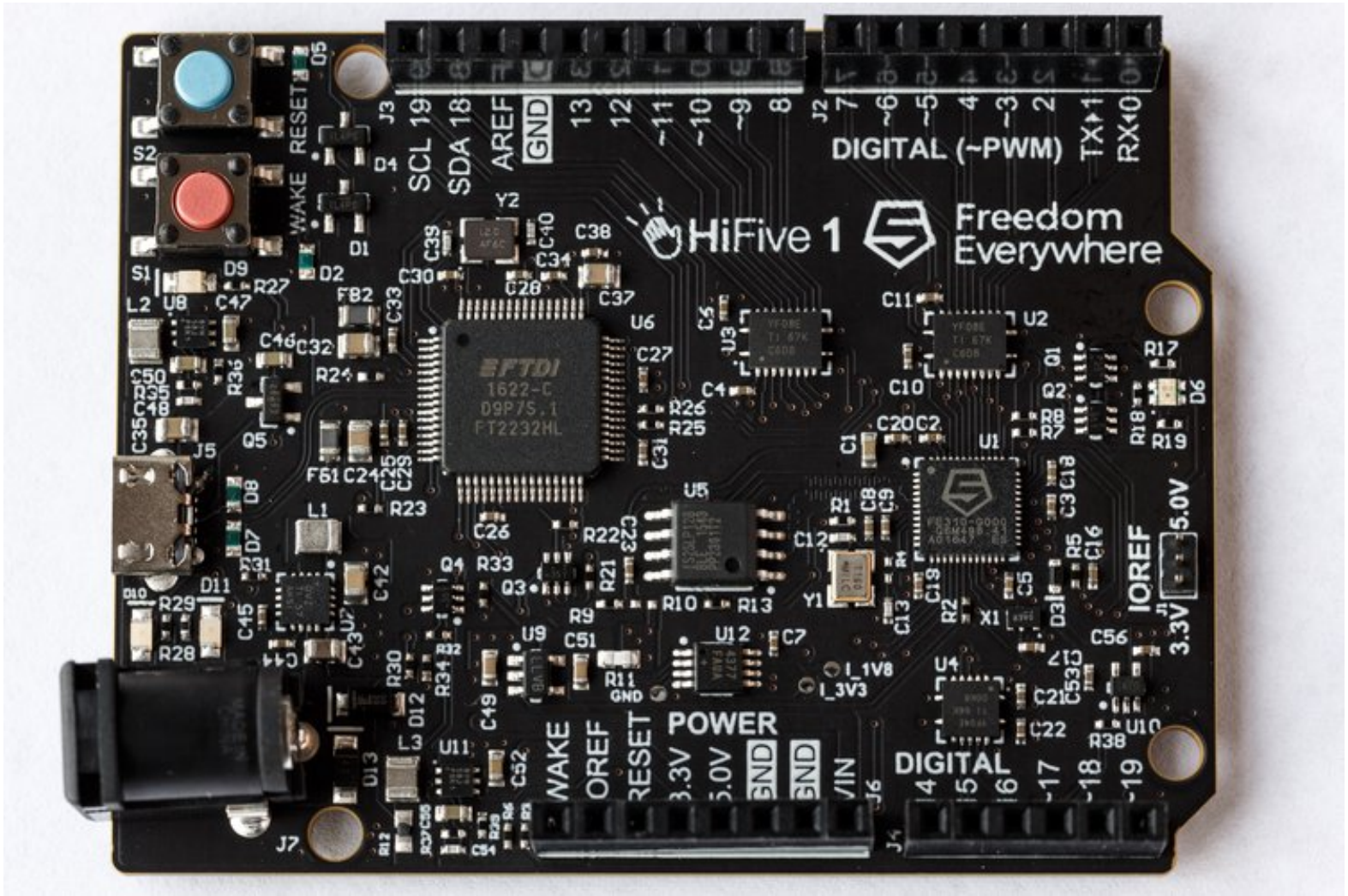
Category	ARMv8	RISC-V	ARM/RISC
Year announced	2011	2011	--
Address sizes	32 / 64	32 / 64 / 128	--
Instruction sizes	32	16† / 32	--
Relative code size	1	0.8†	--
Instruction formats	53	6 / 12†	4X-8X
Data addressing modes	8	1	8X
Instructions	1070	177†	6X
Min number instructions to run Linux, gcc, LLVM	359	47	8X
Backend gcc compiler size	47K LOC	10K LOC	5X
Backend LLVM compiler size	22K LOC	10K LOC	2X
ISA manual size	5428 pages	163 pages	33X

MIPS manual 700 pages
80x86 manual 2900 pages

†With optional Compressed RISC-V ISA extension

RISC-V is not a toy!

In fact risc-v is now commercially available!

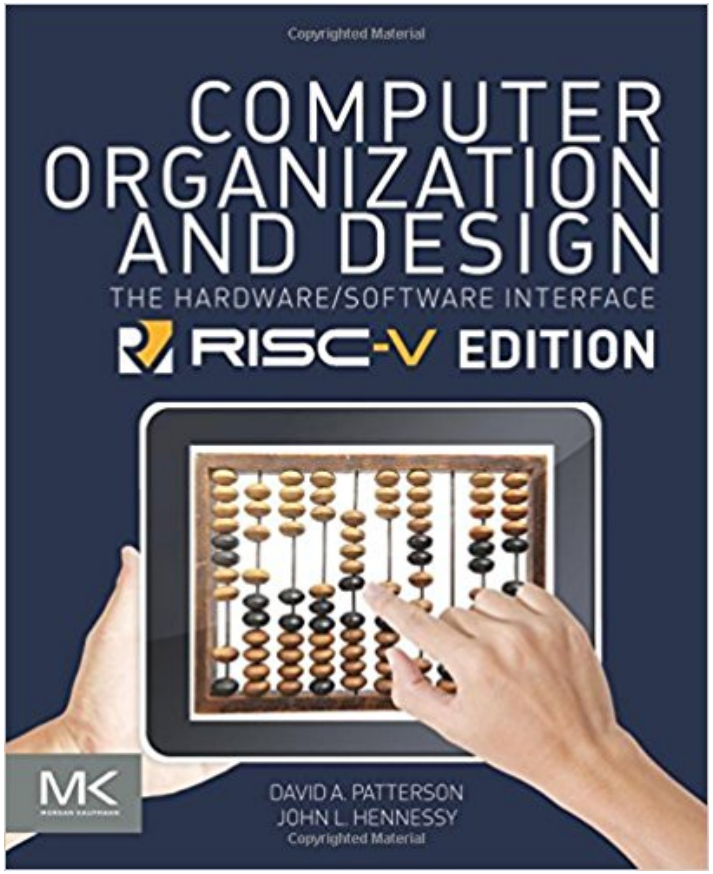


Where to start?

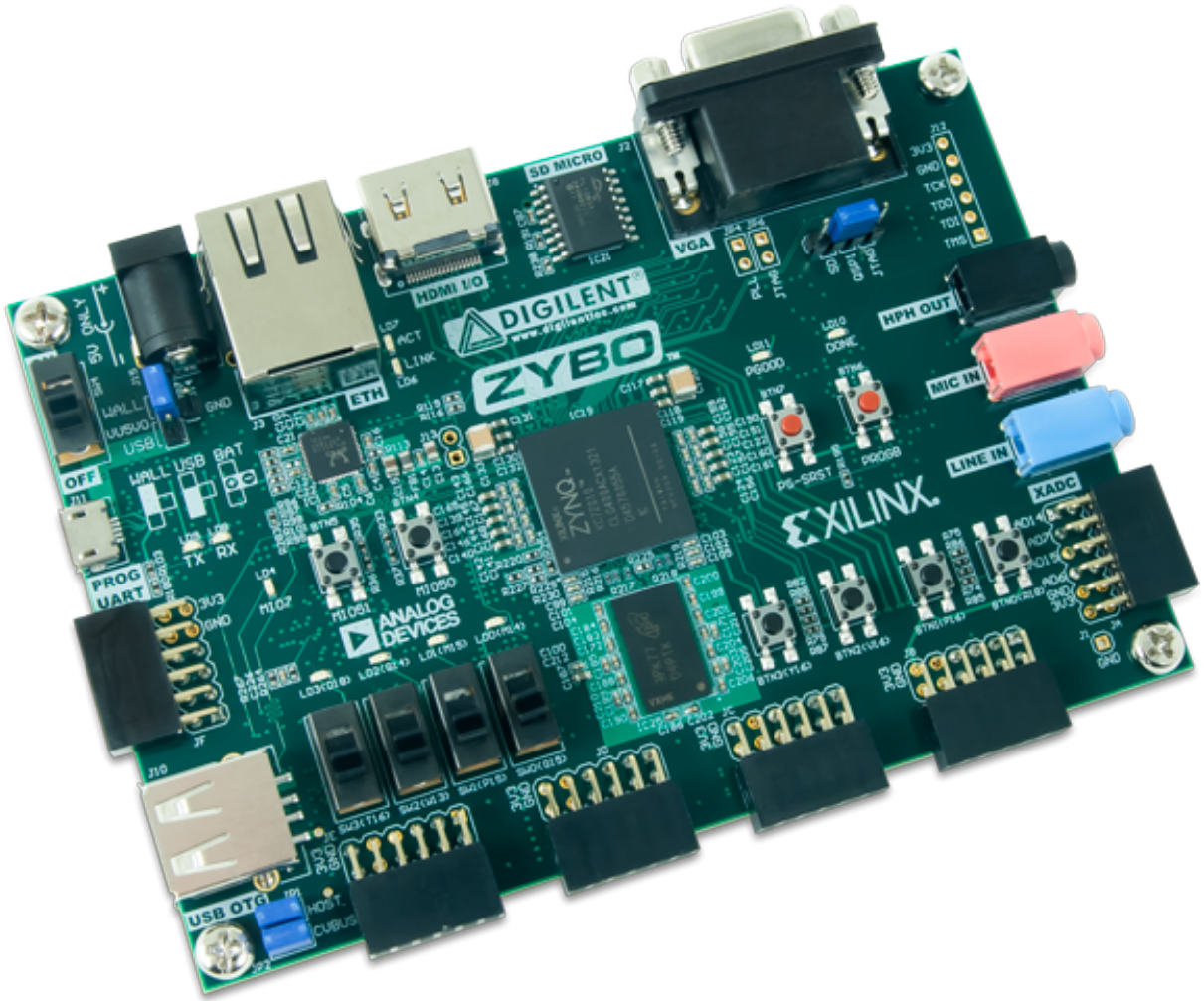
- You can start by reading RISC-V ISA:
  https://riscv.org/specifications/
- Then, you start studying simple implementations:
  riscv-mini is highly recommended
- You can then read other implementations. Just search for risc-v on Github! Here are a few:
  - RocketChip project
  - PulPino
  - picorv32
  - and many more, just search!

RISC-V is great for research

- RISC-V is a great option for research
- The ISA is free to use and open
- There are upcoming courses based on risc-v
- The new Paterson Book is on RISC-V



Booting Linux On RISC-V on ZYBO



Lab3: RISC-V Lab

Follow instructions here

https://goo.gl/QVqwmq

The End