Getting Started with Chisel

CA 2025Fall PA2

TA: 田庭瑄 r14943176@ntu.edu.tw

Outline

- Introduction to Chisel
- PA2 Overview
- Setup Environment
- Writing Chisel Modules
- Testing with ChiselTest
- Generating Verilog (not included in PA2)
- References

Introduction to Chisel

- What is Chisel?
  - Chisel: **C**onstructing **H**ardware **i**n a **S**cala **E**mbedded **L**anguage
  - A hardware construction language (HCL) built on top of Scala

    Verilog is a kind of hardware description language (HDL)
  - Use modern programming language features (object-oriented, functional programming, libraries) to describe circuits
- Why Chisel?
  - High abstraction: ex. parameterization, reusability
  - Less repetitive code: loops and functions
  - Scalability: complex datapaths, pipelines

Introduction to Chisel - compared to Verilog

Verilog

```
genvar i;
generate
  for (i=0; i<WIDTH; i=i+1) begin
    assign out[i] = a[i] & b[i];
  end
endgenerate
```

- separate **generate** block
- **WIDTH** is determined early (when elaborating) by user or constant
- may need to repeat a lot due to different parameter

Chisel

```
val out = VecInit((0 until width).map(i => a(i) & b(i)))
```

- can combine with other features like *map, reduce, filter, List*
- **WIDTH** can be calculated by itself

Introduction to Chisel - compared to Verilog

Verilog

```
function [7:0] adderN;
  input [7:0] a, b;
  begin
    adderN = a + b;
  end
endfunction
assign sum = adderN(in1, in2);
```

- bitwidth should be fixed
- function: no delay, combinational only
- task: unsynthesizable sometimes

Chisel

```
def adderN(n: Int): (UInt, UInt) => UInt = {
  (a: UInt, b: UInt) => (a +& b)(n-1,0)
}
val sum = adderN(8)(io.in1, io.in2)
```

- better scalability
- steep learning curve

Introduction to Chisel - Syntax

Core Components

- **class** – Defines a hardware module or data structure
- **object** – Contains definitions or helper functions
- **def** – Defines a function
- **package** – Groups related Chisel files/modules

Modules and I/O

- **Module** – Represents a hardware block
- **IO()** – Defines input/output interface for a module
- **Input()** / **Output()** – Specify signal direction
- **Bundle** – Groups related I/O or signals into one structure

```
class Adder(val width: Int) extends Module {
  val io = IO(new Bundle {
    val a = Input(UInt(width.W))
    val b = Input(UInt(width.W))
    val y = Output(UInt(width.W))
  })
  val offset = WireDefault(0.U(8.W))
  offset := 5.U
  io.y := io.a + io.b + offset
}
```

```
usage:

val adder8  = Module(new Adder(8))
val adder32 = Module(new Adder(32))
```

Introduction to Chisel - Syntax

Basic Types

- **UInt** – Unsigned integer
- **SInt** – Signed integer
- **Bool** – Boolean (1-bit value)

Type Modifiers

- **.W** – Width specifier
- **.U** – Unsigned literal, you can convert it by **.asUInt**
- **.S** – Signed literal, you can convert it by **.asSInt**

Collections

- **Vec** – Vector (fixed-size collection of elements)
- **Array** – Scala software array (not synthesizable hardware)

```
class Adder(val width: Int) extends Module {
  val io = IO(new Bundle {
    val a = Input(UInt(width.W))
    val b = Input(UInt(width.W))
    val y = Output(UInt(width.W))
  })
  val offset = WireDefault(0.U(8.W))
  offset := 5.U
  io.y := io.a + io.b + offset
}
```

```
usage:

val adder8  = Module(new Adder(8))
val adder32 = Module(new Adder(32))
```

Introduction to Chisel - Syntax

```
val regB = RegInit(0.U(8.W))
when (io.en) {
  regB := regB + 1.U
}
io.out := regB
```

Registers

- **Reg()** – Creates a hardware register (stores value across cycles)
- **RegInit(value)** – Register with an initial/reset value
- **RegNext(signal)** – Register that automatically stores the next-cycle value of a signal

```
val stage1 = io.in1 + io.in2        // Stage 1: Add
val stage2 = RegNext(stage1)        // Pipeline register
io.out := stage2 * 2.U              // Stage 2: Multiply
```

Wires

- **Wire()** – Defines a combinational signal (no storage)
    - Used for intermediate logic connections

Introduction to Chisel - Syntax

Control flow

- **when(cond) {...}** – if
- **.elsewhen(cond2) {...}** – else if
- **.otherwise {...}** – else
- **switch(sel) { is(0.U) {...} ... }** – Multi-way conditional (like case)

```
when (io.enable) {
  reg := reg + 1.U
} .otherwise {
  reg := 0.U
}
```

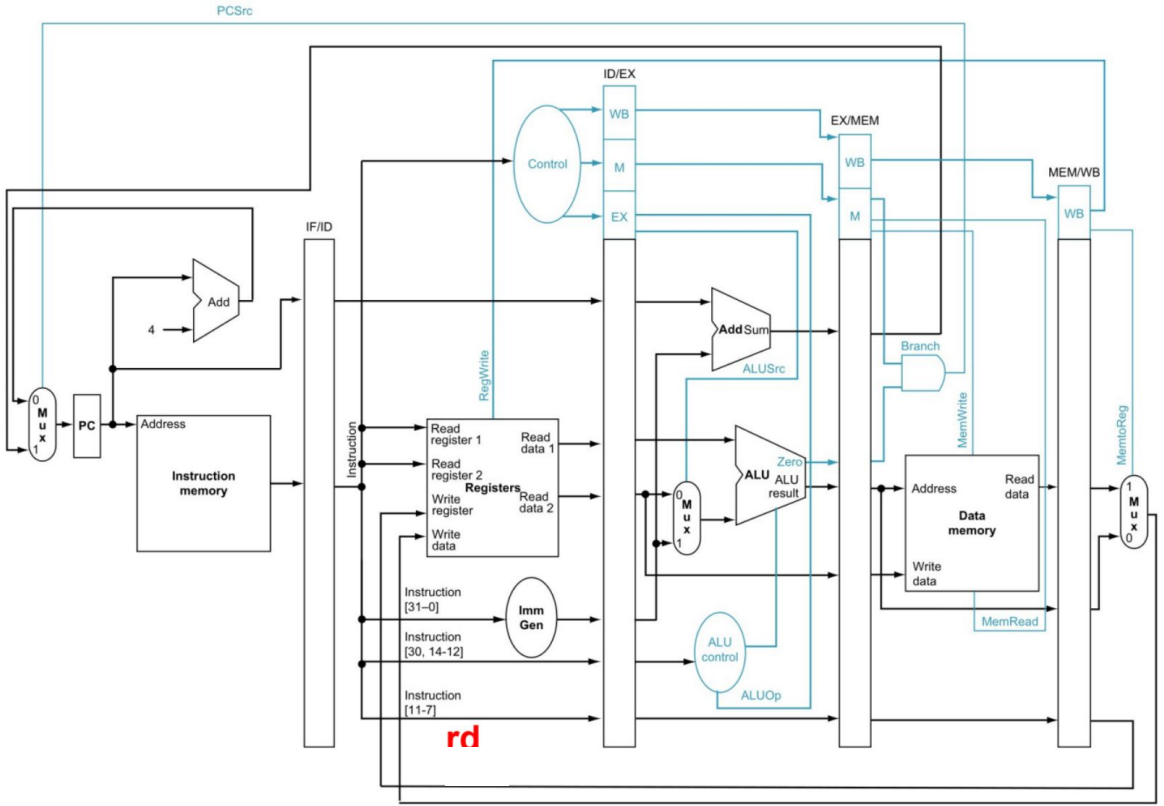
Memory

- **Mem(n, type)** – combinational
- **SyncReadMem(n, type)** - delay a cycle

```
val mem = Mem(1024, UInt(32.W))
// combinational read
val syncMem = SyncReadMem(1024, UInt(32.W))
// synchronous read
```

PA2 Overview

- a RISC-V CPU in Chisel
- either **single-cycle** or **pipeline** is accepted, and you will get corresponding points



PA2 Overview

```
CA2025_PA2/
│
├── project/
│   ├── build.properties
│   └── plugins.sbt
├── src/
│   ├── main/scala/cpu
│   │   ├── Core.scala
│   │   └── (other design files)
│   └── test/
│       ├── scala/cpu
│       │   └── CoreTest.scala
│       └── pattern/
│           ├── p1.hex
│           └── (other test patterns)
└── build.sbt
```

	pattern	baseline CPI	points	bonus CPI	points
p1, h1	no hazard	single-cycle	6	pipeline	4
p2, h2	data hazard (alu)	single-cycle	6	stall	4
p3, h3	data hazard (load-use)	single-cycle / stall	6	forwarding	4
p4, h4	control hazard (branch, jump)	single-cycle	6	forwarding	4
p5, h5	general	single-cycle / stall	6	forwarding	4

Setup Environment

Install **JDK 17** and **sbt** (Scala Build Tool)

You can verify your installations using the commands:

```
(base) circle@LAPTOP-VVPR01JP:~/CA_2025fall/PA2$ java -version
openjdk version "17.0.16" 2025-07-15
OpenJDK Runtime Environment (build 17.0.16+8-Ubuntu-0ubuntu122.04.1)
OpenJDK 64-Bit Server VM (build 17.0.16+8-Ubuntu-0ubuntu122.04.1, mixed mode, sharing)
(base) circle@LAPTOP-VVPR01JP:~/CA_2025fall/PA2$ sbt --version
sbt runner version: 1.11.5

[info] sbt runner (sbt-the-shell-script) is a runner to run any declared version of sbt.
[info] Actual version of the sbt is declared using project/build.properties for each build.
```

Writing Chisel Modules

- Regfile.scala
- Bundles.scala

```scala
// collect all control signals
class CtrlSignal extends Bundle {
    val ctrlJump = Output(Bool())
    val ctrlBranch = Output(Bool())
    val ctrlRegWrite = Output(Bool())
    val ctrlMemRead = Output(Bool())
    val ctrlMemWrite = Output(Bool())
    val ctrlALUSrc = Output(Bool())
    val ctrlALUOp = Output(UInt(5.W))
    val ctrlMemToReg = Output(Bool())
}

// collect all register addresses
class RegAddr extends Bundle {
    val rs1_addr = Output(UInt(5.W))
    val rs2_addr = Output(UInt(5.W))
    val rd_addr = Output(UInt(5.W))
}
```

```scala
package cpu

import chisel3._
import chisel3.util._

class Regfile_io extends Bundle {
    val ctrlRegWrite = Input(Bool())
    val r                                  input
    val d
    val d
    val data_read2 = Output(UInt(32.W))
}

class Regfile extends Module {
    val io = IO(new Regfile_io())

    // declare 32 registers each 32 bits
    val regs = Reg(Vec(32, UInt(32.W)))

    // read
    when(io.reg_addr.rs1_addr === 0.U) {
        io.data_read1 := 0.U
    }
    when(io.reg_add                .U) {
        io.data_rea
    }

    io.data_read1 := regs(io.reg_addr.rs1_addr)
    io.data_read2 := regs(io.reg_addr.rs2_addr)

    // write
    when(io.ctrlRegWrite && io.reg_addr.rd_addr =/= 0.U) {
       regs(io.reg_addr.rd_addr) := io.data_write
    }
}
```

collect I/O of module

define module

function

Testing with ChiselTest

- command

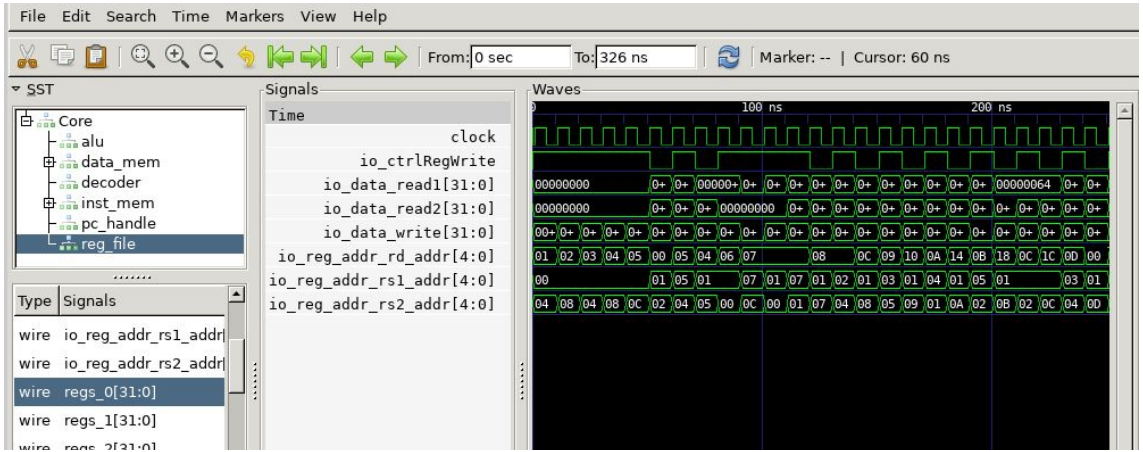
```
PIPELINE=0 INST_FILE=pattern/p1.hex GOLDEN_FILE=pattern/p1_golden.hex sbt test
```

- environment variable
  - PIPELINE: default 0
  - INST_FILE: default p1.hex
  - GOLDEN_FILE: default p1_golden.hex

Testing with ChiselTest

- It is easier to debug with waveform sometimes

```
gtkwave test_run_dir/<test_name>/<module_name>.vcd
```



Generating Verilog (not in PA2)

- **Chisel -> FIRRTL** (intermediate representation) **-> Verilog** -> synthesis -> FPGA/ASIC
- in “CA2025_PA2/src/main/scala/cpu/Core.scala”

```
20  class Core extends Module {
72
73  }
74
75  /// You can add the following code
76  //  and generate verilog by command: sbt "runMain cpu.main" ///
77
78  // object main extends App {
79
80  //     (new ChiselStage).execute(
81  //     Array("--target-dir", "verilog_output"),
82  //     Seq(ChiselGeneratorAnnotation(() => new Core()))
83  //     )
84  // }
```

- You can try the fascinating function in chisel-template

References

- Syntax: https://www.chisel-lang.org/api/latest/index.html
- Official site: https://www.chisel-lang.org/
- Cheatsheet: https://github.com/freechipsproject/chisel-cheatsheet
- Exercise: https://github.com/freechipsproject/chisel-bootcamp