

چکیده:

در این پروژه می خواهیم نصب و کار با کتابخانه Chisel در زبان scala را بیاموزیم برای این کار نیاز است تا ابتدا سیستم عامل linux را نصب کنیم

مقدمه:

هذف از انجام این آزمایش یادگیری پایه ای Chisel و انجام دو ماژول ساده است برای این کار در ترمینال Linux قسمت های مورد نیاز را نصب کرده سپس Chisel را نصب می کنیم و در ان فایل Scala را ساخته و ماژول های خود را در ان می نویسیم. در این آزمایش می خواهیم دو ماژول ALU و Stack را بسازیم

ابزار ها:

در این آزمایش در محیط Linux با استفاده از زبان Scala و کتابخانه Chisel ماژول های مورد نیاز را می سازیم

قسمت اول:

در قسمت اول خواسته شده تا یک ALU ساده با استفاده از 4 ALU که هر کدام عمل خاص خود را دارند بسازیم

ابتدا بعد از نصب Chisel و باز کردن فایل ان به پوشه src/solution رفته و یک فایل Scala می سازیم و در ان پروژه را می سازیم

ابتده 4 ماژول ALU برای هر یک از عملیات های جمع تفریق ضرب و تقسیم را مانند شکل های زیر می سازیم

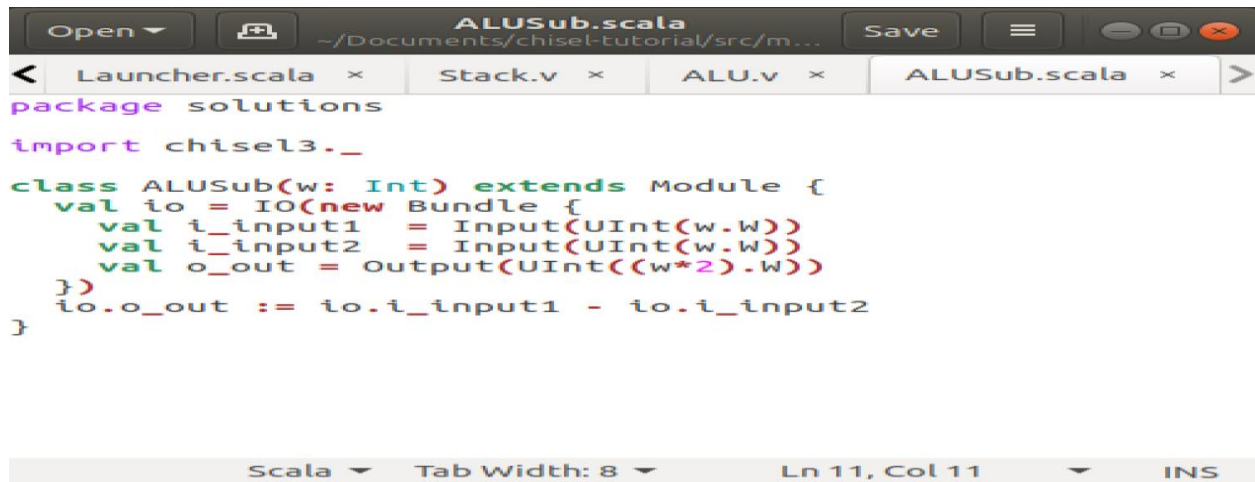
```

package solutions
import chisel3._

class ALUAdd(w: Int) extends Module {
  val io = IO(new Bundle {
    val i_input1 = Input(UInt(w.W))
    val i_input2 = Input(UInt(w.W))
    val o_out = Output(UInt((w*2).W))
  })
  io.o_out := io.i_input1 + io.i_input2
}
  
```

:ADDER:1 Figure

Summery



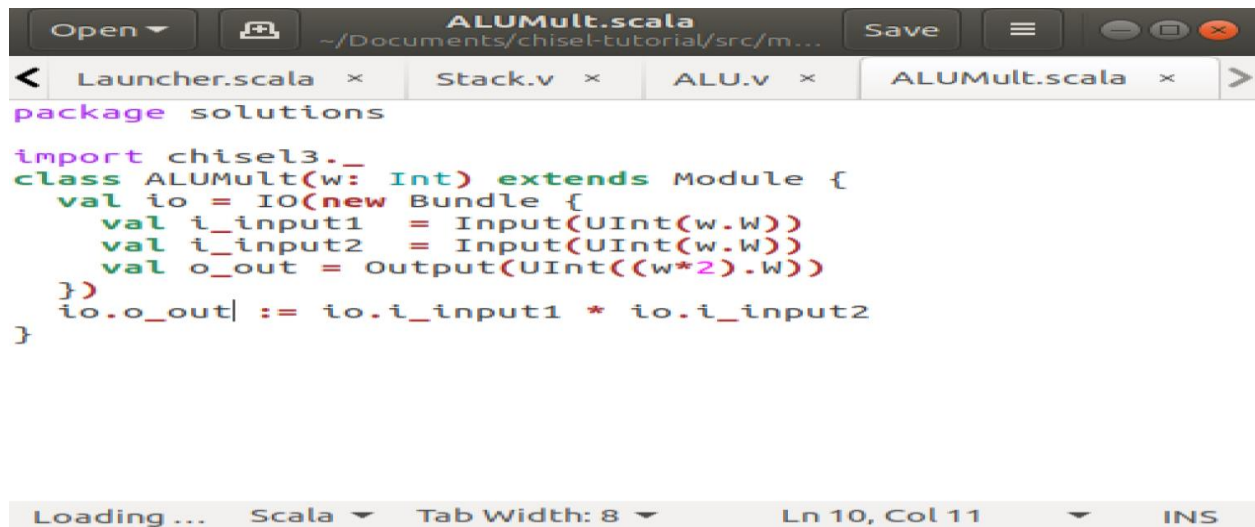
The screenshot shows a text editor window titled "ALUSub.scala" with a file path of "~/Documents/chisel-tutorial/src/m...". The editor has tabs for "Launcher.scala", "Stack.v", "ALU.v", and "ALUSub.scala". The code is as follows:

```
package solutions
import chisel3._

class ALUSub(w: Int) extends Module {
  val io = IO(new Bundle {
    val i_input1 = Input(UInt(w.W))
    val i_input2 = Input(UInt(w.W))
    val o_out = Output(UInt((w*2).W))
  })
  io.o_out := io.i_input1 - io.i_input2
}
```

The status bar at the bottom indicates "Scala", "Tab Width: 8", "Ln 11, Col 11", and "INS".

:SUB2 Figure



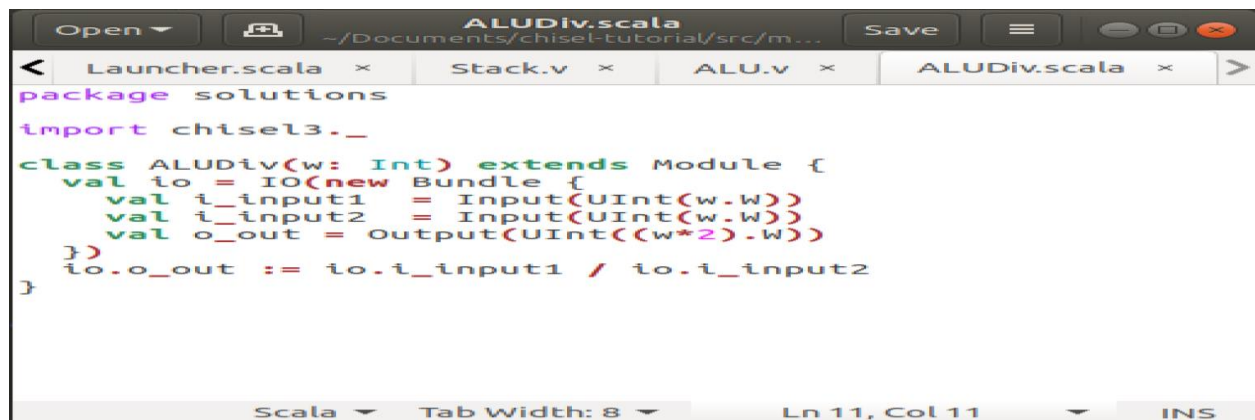
The screenshot shows a text editor window titled "ALUMult.scala" with a file path of "~/Documents/chisel-tutorial/src/m...". The editor has tabs for "Launcher.scala", "Stack.v", "ALU.v", and "ALUMult.scala". The code is as follows:

```
package solutions
import chisel3._

class ALUMult(w: Int) extends Module {
  val io = IO(new Bundle {
    val i_input1 = Input(UInt(w.W))
    val i_input2 = Input(UInt(w.W))
    val o_out = Output(UInt((w*2).W))
  })
  io.o_out := io.i_input1 * io.i_input2
}
```

The status bar at the bottom indicates "Loading ...", "Scala", "Tab Width: 8", "Ln 10, Col 11", and "INS".

:MULT3 Figure



The screenshot shows a text editor window titled "ALUDiv.scala" with a file path of "~/Documents/chisel-tutorial/src/m...". The editor has tabs for "Launcher.scala", "Stack.v", "ALU.v", and "ALUDiv.scala". The code is as follows:

```
package solutions
import chisel3._

class ALUDiv(w: Int) extends Module {
  val io = IO(new Bundle {
    val i_input1 = Input(UInt(w.W))
    val i_input2 = Input(UInt(w.W))
    val o_out = Output(UInt((w*2).W))
  })
  io.o_out := io.i_input1 / io.i_input2
}
```

The status bar at the bottom indicates "Scala", "Tab Width: 8", "Ln 11, Col 11", and "INS".

:Div4 Figure

سپس یک ماژولی می سازیم تا بتواند با گرفتن یک رشته حروف یکی از ماژول های عملیاتی را انتخاب کند

```
class ALUSelect(w: Int, Select:String) extends Module {
  val io = IO(new Bundle {
    val i_A = Input(UInt(w.W))
    val i_B = Input(UInt(w.W))
    val o_W = Output(UInt((w*2).W))
  })

  if(Select == "Add") {
    val ALUAdd = Module(new ALUAdd(w))
    ALUAdd.io.i_input1 := io.i_A
    ALUAdd.io.i_input2 := io.i_B
    io.o_W := ALUAdd.io.o_out
  }

  else if(Select == "Sub") {
    val ALUSub = Module(new ALUSub(w))
    ALUSub.io.i_input1 := io.i_A
    ALUSub.io.i_input2 := io.i_B
  }
}
```

Scala Tab Width: 8 Ln 37, Col 32 INS

Open ALUSelect.scala Save

~/Documents/chisel-tutorial/src/m...

Launcher.scala x Stack.v x ALU.v x ALUSelect.scala x

```
ALUSub.io.i_input2 := io.i_B
io.o_W := ALUSub.io.o_out
}

else if(Select == "Div") {
  val ALUDiv = Module(new ALUDiv(w))
  ALUDiv.io.i_input1 := io.i_A
  ALUDiv.io.i_input2 := io.i_B
  io.o_W := ALUDiv.io.o_out
}

else if(Select == "Mult") {
  val ALUMult = Module(new ALUMult(w))
  ALUMult.io.i_input1 := io.i_A
  ALUMult.io.i_input2 := io.i_B
  io.o_W := ALUMult.io.o_out
}
}
```

Scala Tab Width: 8 Ln 37, Col 32 INS

و سپس یک ماژول ساخته تا همه را به هم وصل کند ورودی خروجی ها را بگیرد و مقدار Delay دلخواه بدهد

```
class ALU(w: Int, Select: String, Delay: Int) extends
Module {
  val io = IO(new Bundle {
    val i_DataA = Input(UInt(w.W))
    val i_DataB = Input(UInt(w.W))
    val i_Valid_in = Input(UInt(1.W))
    val i_Tuser_in = Input(UInt(w.W))
    val o_out = Output(UInt((w*2).W))
    val o_Valid_out = Output(UInt(1.W))
    val o_Tuser_out = Output(UInt(w.W))
  })

  val s_init1 = Seq.fill(Delay) {0.U(1.W)}
  val s_Valid_Vec = RegInit(VecInit(s_init1))
  val s_init2 = Seq.fill(Delay) {0.U(w.W)}
  val s_Tuser_Vec = RegInit(VecInit(s_init2))
  val s_init3 = Seq.fill(Delay) {0.U((w*2).W)}
  val s_OutPut_Vec = RegInit(VecInit(s_init3))
  Scala Tab Width: 8 Ln 42, Col 46 INS

```

```
ALU1.io.i_B := io.i_DataB

s_OutPut_Vec(0) := ALU1.io.o_W
s_Tuser_Vec(0) := io.i_Tuser_in
s_Valid_Vec(0) := io.i_Valid_in

for(i <- 0 until (Delay-1) ){
  s_OutPut_Vec(i+1) := s_OutPut_Vec(i)
  s_Tuser_Vec(i+1) := s_Tuser_Vec(i)
  s_Valid_Vec(i+1) := s_Valid_Vec(i)
}

io.o_out := s_OutPut_Vec(Delay-1)
io.o_Tuser_out := s_Tuser_Vec(Delay-1)
io.o_Valid_out := s_Valid_Vec(Delay-1)
}
Scala Tab Width: 8 Ln 42, Col 46 INS

```

بعد از نوشتن ماژول ها ان را در ترمینال Compile می کنیم و از صحت ان مطمئن می شویم

```
ssa@ubuntu:~/Documents/chisel-tutorial$ #!/usr/bin/env bash
ssa@ubuntu:~/Documents/chisel-tutorial$ args=$@
ssa@ubuntu:~/Documents/chisel-tutorial$ sbt "test:runMain solutions.Launcher $a
rgs"
[info] Loading project definition from /home/ssa/Documents/chisel-tutorial/proj
ect
[info] Loading settings from build.sbt ...
[info] Set current project to chisel-tutorial (in build file:/home/ssa/Document
s/chisel-tutorial/)
[warn] Multiple main classes detected. Run 'show discoveredMainClasses' to see
the list
[info] Running solutions.Launcher
Available tutorials
Mux4
Accumulator
DynamicMemorySearch
MaxN
Stack
VecShiftRegisterSimple
Max2
VendingMachineSwitch
Mul
LFSR16
VecShiftRegister
RealGCD
ALU
VendingMachine
Memo
VecShiftRegisterParam
```

سپس به فایل تست رفته و تست ان را می نویسیم مانند شکل زیر:

Summery

```
ALUTest.scala
~/Documents/chisel-tutorial/src/te...
Open Save
Stack.v x ALU.v x ALU.scala x ALUTest.scala x
// See LICENSE.txt for license details.
package solutions

import chisel3.iotesters.{ChiselFlatSpec, Driver,
  PeekPokeTester}

class ALUTest(c: ALU) extends PeekPokeTester(c) {

  for (i <- 0 until 10) {
    val in0 = i
    val in1 = i
    poke(c.io.i_DataA, in0)
    poke(c.io.i_DataB, in1)
    step(10)
    expect(c.io.o_out, in0 + in1)
  }
}
```

سپس در فایل Luncher در پوشه ی تست یک instance از ماژول را گرفته و ان را تست می کنیم:

```
"ALU" -> { (manager: TesterOptionsManager) =>
  Driver.execute(() => new ALU(16,"Add",10),
manager) {
  (c) => new ALUTest(c)
}
},
```


Summery

```
ssa@ubuntu:~/Documents/chisel-tutorial$ ./run-solution.sh ALU
[info] Loading project definition from /home/ssa/Documents/chisel-tutorial/project
[info] Loading settings from build.sbt ...
[info] Set current project to chisel-tutorial (in build file:/home/ssa/Documents/chisel-tutorial/)
[info] Compiling 1 Scala source to /home/ssa/Documents/chisel-tutorial/target/scala-2.11/test-classes ...
[info] Done compiling.
[warn] Multiple main classes detected. Run 'show discoveredMainClasses' to see the list
[info] Packaging /home/ssa/Documents/chisel-tutorial/target/scala-2.11/chisel-tutorial_2.11-3.1.0-tests.jar ...
[info] Done packaging.
[info] Running solutions.Launcher ALU
Starting tutorial ALU
[info] [0.002] Elaborating design...
[info] [1.008] Done elaborating.
Total FIRRTL Compile Time: 703.2 ms
Total FIRRTL Compile Time: 275.3 ms
file loaded in 0.439996073 seconds, 79 symbols, 41 statements
[info] [0.001] SEED 1560162861565
test ALU Success: 10 tests passed in 105 cycles in 0.047722 seconds 2200.26 Hz
[info] [0.028] RAN 100 CYCLES PASSED
Tutorials passing: 1
[success] Total time: 14 s, completed Jun 10, 2019 3:34:24 AM
ssa@ubuntu:~/Documents/chisel-tutorial$
```

همان طور که در شکل نیز آورده شده است تست با موفقیت تمام شده است و ماژول کار می کند
سپس در ترمینال دستور برای ساخت کد وریرلاگ ان را می زنیم و کد را می گیریم. کد در پوشه ALU
در test_dir قرار می گیرد

```
module ALUAdd(  
  input  [15:0] io_i_input1,  
  input  [15:0] io_i_input2,  
  output [31:0] io_o_out  
);
```

Summery

کد را به Quartes برده و سنتز می کنیم و داریم:

Flow Summary	
Flow Status	Successful - Mon Jun 10 15:38:02 2019
Quartus II 64-Bit Version	13.0.1 Build 232 06...SP 1 SJ Web Edition
Revision Name	CA5
Top-level Entity Name	ALU
Family	Cyclone II
Device	EP2C5Q208C7
Timing Models	Final
Total logic elements	330
Total combinational functions	330
Dedicated logic registers	330
Total registers	330
Total pins	100
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0

قسمت دوم:

در این قسمت می خواهیم یک استک بسازیم:

همانند قسمت قبل ابتدا فایل ماژول را ساخته و Compile می کنیم

ماژول استک را مانند زیر می سازیم:

```
class Stack(val depth: Int) extends Module {
  val io = IO(new Bundle {
    val i_W_EN = Input(Bool())
    val i_RD_EN = Input(Bool())
    val o_Empty = Output(UInt(1.W))
    val o_Full = Output(UInt(1.W))
    val i_D_in = Input(UInt(32.W))
    val o_D_out = Output(UInt(32.W))
  })

  val stack_mem = Mem(depth, UInt(16.W))
  val sp = RegInit(0.U(log2Ceil(depth).W))
  val out = RegInit(0.U(32.W))
  io.o_Empty := 0.U
  io.o_Full := 0.U

  when(io.i_W_EN && (sp < (depth-1).U)) {
    stack_mem(sp+1.U) := io.i_D_in
  }

  when (sp == 0.U) {
    io.o_Empty := 1.U
  }
  when (sp == (depth-1).U) {
    io.o_Full := 1.U
  }
  io.o_D_out := out
}
```

سپس ان را در ترمینال Compile می کنیم و درستی ان را چک می کنیم

سپس به پوشه تست رفته و تست بنچ مربوط به آن را می نویسیم

```
println(s"o_Empty ${peek(c.io.o_Empty)}")
println(s"o_Full ${peek(c.io.o_Full)}")

poke(c.io.i_W_EN, 0)
poke(c.io.i_RD_EN, 1)
poke(c.io.i_D_in, 22)

println(s"o_D_out ${peek(c.io.o_D_out)}")
println(s"o_Empty ${peek(c.io.o_Empty)}")
println(s"o_Full ${peek(c.io.o_Full)}")
expect(c.io.o_D_out, o_D_out)
expect(c.io.o_Empty, o_Empty)
expect(c.io.o_Full, o_Full)
}

class StackTests(c: Stack) extends PeekPokeTester(c) {
  var nxto_D_out = 0
  var o_D_out = 0
  var o_Empty = 0
  var o_Full = 0

  poke(c.io.i_W_EN, 1)
  poke(c.io.i_RD_EN, 0)
  poke(c.io.i_D_in, 232)

  step(1)

  println(s"o_D_out ${peek(c.io.o_D_out)}")
  println(s"o_Empty ${peek(c.io.o_Empty)}")
  println(s"o_Full ${peek(c.io.o_Full)}")

  poke(c.io.i_W_EN, 1)
```

```

poke(c.io.i_RD_EN, 0)
poke(c.io.i_D_in, 90)

step(1)
step(1)

println(s"o_D_out ${peek(c.io.o_D_out)}")
println(s"o_Empty ${peek(c.io.o_Empty)}")
println(s"o_Full ${peek(c.io.o_Full)}")

poke(c.io.i_W_EN, 0)
poke(c.io.i_RD_EN, 1)
poke(c.io.i_D_in, 33)

step(1)
step(1)

println(s"o_D_out ${peek(c.io.o_D_out)}")

```

بعد از نوشتن تست بنچ یک Instance از آن را در فایل Luncher می‌گیریم:

```

"Stack" -> { (manager: TesterOptionsManager) =>
  Driver.execute(() => new Stack(10), manager) {
    (c) => new StackTests(c)
  }
},

```

و سپس آن را در ترمینال ران می‌کنیم:

```

[info] Packaging /home/ssa/Documents/chisel-tutorial/target/scala-2.11/chisel-tutorial_2.11-3.1.0-tests.jar ...
[info] Done packaging.
[info] Running solutions.Launcher Stack
Starting tutorial Stack
[info] [0.001] Elaborating design...
[info] [1.227] Done elaborating.
Total FIRRTL Compile Time: 970.9 ms
Total FIRRTL Compile Time: 290.3 ms
file loaded in 0.595060914 seconds, 43 symbols, 31 statements
[info] [0.001] SEED 1560164255426
[info] [0.003] o_D_out 0
[info] [0.003] o_Empty 0
[info] [0.004] o_Full 0
[info] [0.011] o_D_out 0
[info] [0.011] o_Empty 0
[info] [0.011] o_Full 0
[info] [0.016] o_D_out 90
[info] [0.018] o_Empty 0
[info] [0.020] o_Full 0
[info] [0.020] o_D_out 90
[info] [0.021] o_Empty 0
[info] [0.021] o_Full 0
test Stack Success: 3 tests passed in 10 cycles in 0.091977 seconds 108.72 Hz
[info] [0.026] RAN 5 CYCLES PASSED
Tutorials passing: 1
[success] Total time: 18 s, completed Jun 10, 2019 3:57:39 AM

```

همان‌طور که معلوم است تست با موفقیت تمام شده است

سپس در ترمینال دستور برای ساخت کد ورپلاگ ان را می زنیم و کد را می گیریم. کد در پوشه ALU در test_dir قرار می گیرد

```
module stack(
    input          clock,
    input          reset,
    input          io_i_W_EN,
    input          io_i_RD_EN,
    output         io_o_Empty,
    output         io_o_Full,
    input  [31:0]  io_i_D_in,
    output [31:0]  io_o_D_out
);
```

کد را به Quartes برده و ان را سنتز می کنیم و داریم:

Flow Summary	
Flow Status	Successful - Mon Jun 10 15:40:50 2019
Quartus II 64-Bit Version	13.0.1 Build 232 06...SP 1 SJ Web Edition
Revision Name	CA5
Top-level Entity Name	Stack
Family	Cyclone IV GX
Total logic elements	58
Total combinational functions	38
Dedicated logic registers	45
Total registers	45
Total pins	70
Total virtual pins	0
Total memory bits	160
Embedded Multiplier 9-bit elements	0
Total GXB Receiver Channel PCS	0
Total GXB Receiver Channel PMA	0
Total GXB Transmitter Channel PCS	0
Total GXB Transmitter Channel PMA	0
Total PLLs	0