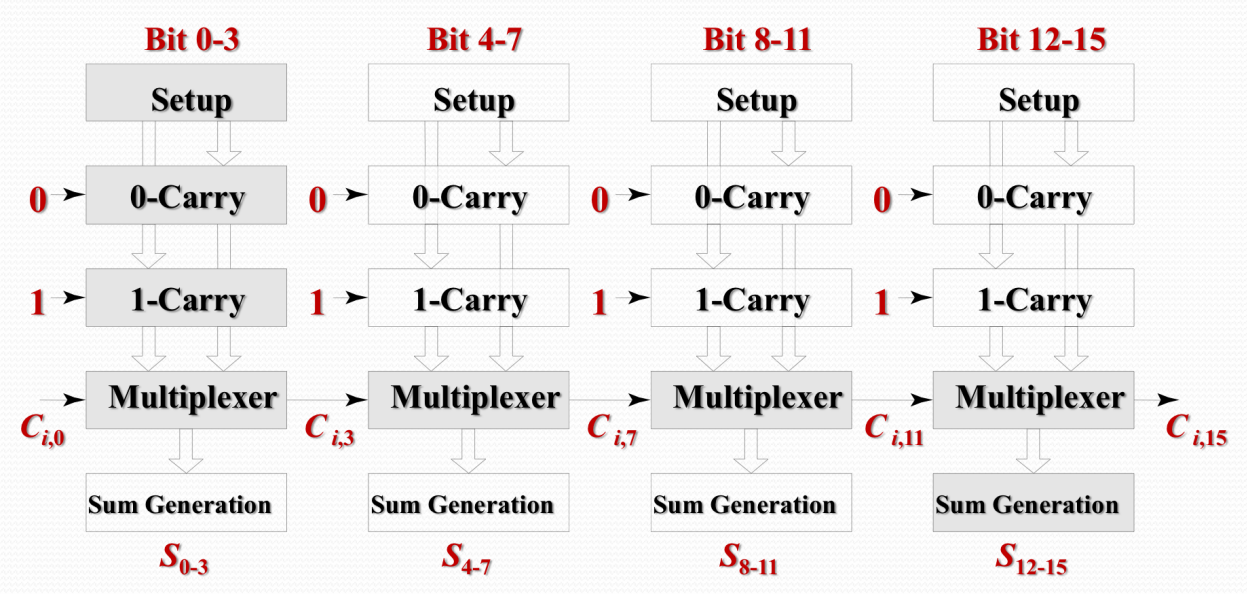
第三次讨论课

实现了一个可变长度(4N)位的线性进位选择加法器




```
package adder

import chisel3._
import chisel3.util._

class Adder1() extends Module {
  val io = IO(new Bundle {
    val A = Input(UInt(1.W))
    val B = Input(UInt(1.W))
    val Cin = Input(UInt(1.W))
    val Sum = Output(UInt(1.W))
    val Cout = Output(UInt(1.W))
  })

  io.Cout := io.A & io.B | io.A & io.Cin | io.B & io.Cin
  io.Sum := io.A ^ io.B ^ io.Cin

}

object Adder1{
  def apply (A:UInt,B:UInt,Cin:UInt) = {
    val m = Module(new Adder1)
    m.io.A := A
    m.io.B := B
    m.io.Cin := Cin
    (m.io.Sum,m.io.Cout)
  }
}

class Adder4Cin0 extends Module {
  val io = IO(new Bundle {
    val A = Input(UInt(4.W))
    val B = Input(UInt(4.W))
    val Cin = Input(UInt(1.W))
    val Sum = Output(UInt(4.W))
    val Cout = Output(UInt(1.W))
  })

  val temp0 = Adder1(io.A(0), io.B(0), 0.U(1.W))
  val temp1 = Adder1(io.A(1), io.B(1), temp0._2)
  val temp2 = Adder1(io.A(2), io.B(2), temp1._2)
  val temp3 = Adder1(io.A(3), io.B(3), temp2._2)

  io.Cout := temp3._2
  io.Sum := Cat(temp3._1, temp2._1, temp1._1, temp0._1)

}

object Adder4Cin0{
  def apply (A:UInt,B:UInt,Cin:UInt) = {
    val m = Module(new Adder4Cin0)
```

```
    m.io.A := A
    m.io.B := B
    m.io.Cin := Cin
    (m.io.Sum,m.io.Cout)
  }
}

class Adder4Cin1 extends Module {
  val io = IO(new Bundle {
    val A = Input(UInt(4.W))
    val B = Input(UInt(4.W))
    val Cin = Input(UInt(1.W))
    val Sum = Output(UInt(4.W))
    val Cout = Output(UInt(1.W))
  })

  val temp0 = Adder1(io.A(0), io.B(0), 1.U(1.W))
  val temp1 = Adder1(io.A(1), io.B(1), temp0._2)
  val temp2 = Adder1(io.A(2), io.B(2), temp1._2)
  val temp3 = Adder1(io.A(3), io.B(3), temp2._2)

  io.Cout := temp3._2
  io.Sum := Cat(temp3._1, temp2._1, temp1._1, temp0._1)

}

object Adder4Cin1{
  def apply (A:UInt,B:UInt,Cin:UInt) = {
    val m = Module(new Adder4Cin1)
    m.io.A := A
    m.io.B := B
    m.io.Cin := Cin
    (m.io.Sum,m.io.Cout)
  }
}

class Adder4 extends Module {
  val io = IO(new Bundle {
    val A = Input(UInt(4.W))
    val B = Input(UInt(4.W))
    val Cin = Input(UInt(1.W))
    val Sum = Output(UInt(4.W))
    val Cout = Output(UInt(1.W))
  })

  val temp1 = Adder4Cin1(io.A, io.B, 1.U(1.W))
  val temp2 = Adder4Cin0(io.A, io.B, 0.U(1.W))

  io.Cout := Mux(io.Cin.asBool, temp1._2, temp2._2)
//io.Cout := Mux(io.Cin === 1.U, temp1._2, temp2._2)
  io.Sum := Mux(io.Cin.asBool, temp1._1, temp2._1)
}

object Adder4{
  def apply (A:UInt,B:UInt,Cin:UInt) = {
    val m = Module(new Adder4)
    m.io.A := A
    m.io.B := B
    m.io.Cin := Cin
    (m.io.Sum,m.io.Cout)
  }
}

class Adder4N(n: Int) extends Module {
  require(n > 0, "n must be a positive integer")

  val io = IO(new Bundle {
    val A = Input(UInt((4 * n).W))
    val B = Input(UInt((4 * n).W))
    val Cin = Input(UInt(1.W))
    val Sum = Output(UInt((4 * n).W))
    val Cout = Output(UInt(1.W))
  })

  // 递归地构建加法器
  def generateAdders(levels: Int, a: UInt, b: UInt, carryIn: UInt): (UInt, UInt) = {
    if (levels == 1) {
      val adder4 = Module(new Adder4())
      adder4.io.A := a
      adder4.io.B := b
      adder4.io.Cin := carryIn
      (adder4.io.Sum, adder4.io.Cout)
    } else {
      val halfLevels = levels / 2
      val (lowerSum, lowerCout) = generateAdders(halfLevels, a((4 * halfLevels) - 1, 0), b((4 * halfLevels) - 1, 0), carryIn)
      val (upperSum, finalCout) = generateAdders(halfLevels, a((4 * levels) - 1, 4 * halfLevels), b((4 * levels) - 1, 4 * halfLevels), lowerCout)
      val sum = Cat(upperSum, lowerSum)
      (sum, finalCout)
    }
  }
```

```
  val (sum, cout) = generateAdders(n, io.A, io.B, io.Cin)
  io.Sum := sum
  io.Cout := cout
}

class testTc1_counterTop extends FlatSpec with Matchers {
  (new chisel3.stage.ChiselStage).execute(
    Array("-X", "verilog"),
    Seq(ChiselGeneratorAnnotation(() => new Adder4N(4)),
      TargetDirAnnotation("Verilog")))
}
```

12位的ChiselTest

```
package adder

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

class AdderTest(c: Adder4N) extends PeekPokeTester(c) {
  val rnd2 = rnd.nextInt(2)
  for (t <- 0 until 100) {
    val rnd0 = rnd.nextInt(16)
    val rnd1 = rnd.nextInt(16)
    val rnd2 = rnd.nextInt(2)
    poke(c.io.A,   rnd0)
    poke(c.io.B,   rnd1)
    poke(c.io.Cin, rnd2)
    step(1)
    val rsum = (rnd0 & 0xFFF) + (rnd1 & 0xFFF) + (rnd2 & 0x1)
    expect(c.io.Sum, (rsum & 0xFFF))
    expect(c.io.Cout, rsum >> 12)
  }
}

class AdderTester extends ChiselFlatSpec {
  behavior of "Adder4N"
  backends foreach {backend =>
    it should s"correctly add randomly generated numbers $backend" in {
      Driver(() => new Adder4N(3), backend)(c => new AdderTest(c)) should be (true)
    }
  }
}
```

```
[info] [0.002] SEED 1710722549074
test Adder4N Success: 200 tests passed in 105 cycles taking 0.406089 seconds
[info] [0.331] RAN 100 CYCLES PASSED
```

16位的Testbench如下

```
`timescale 1ns/1ps

// `include "booth_mult.v"

module adder_tb ();

    reg [15:0]ain;
    reg [15:0]bin;
    reg cin;
    wire [15:0]sum;
    wire cout;

    wire signed [16:0] calculated_result;
    assign calculated_result = ain + bin;
    reg error;

    Adder16 adder_init(
        .io_A(ain),
        .io_B(bin),
        .io_Cin(cin),
        .io_Sum(sum),
        .io_Cout(cout)
    );

    initial
    begin
        error <= 0;
        ain <= 0;
        bin <= 0;
        cin <= 0;

        #100;
        ain <= 2;
        bin <= 4;

        #100;

        repeat(10000)
```

```
        begin
                ain <= $random();
                bin <= $random();
            #100;

            if ({cout, sum} != calculated_result)
            begin
                $display("ERROR: mismatch, ain=%d, bin=%d, out=%d, should be %d", ain, bin, {cout, sum}, calculated_result);

                error <= 1;
            end
        end

        if (error == 0)
            $display("NO ERROR");

    end
endmodule
```

Modelsim仿真10000次后

```
VSIM 4> run
# NO ERROR
```