Chisel: Sequential Circuits

Muhammad Tahir*

Lecture 6

*Professor, Electrical Engineering Department, University of Engineering and Technology Lahore

Contents

Sequential Circuits

- Sequential circuits are used to implement states and state elements
- State machines and memories can be constructed from sequential circuits

Reg

- Registers are fundamental elements to build sequential circuits
- Register implementation using single or multiple D-type flip-flops
- Chisel provides object `Reg` for constructing hardware registers

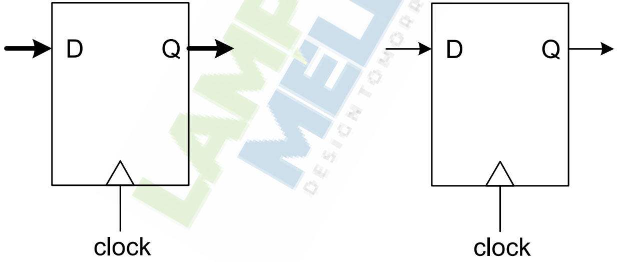


Figure: Single- or multi-bit register.

RegNext

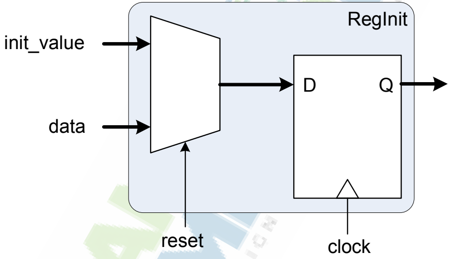
- Width can be inferred or specified
- Can be used to get one cycle delayed version of the signal

```
// following uses of Reg and RegNext are valid
val reg1 = Reg(UInt(8.W))
val reg2 = RegNext(reg1)       // width is inferred from reg1
val reg3 = RegNext(3.U(8.W))   // width is specified

// following uses are invalid
val reg4 = Reg(3.U(8.W))
val reg5 = RegNext(UInt(8.W))
```

RegInit

- Register with initialized value
- Initialization occurs on reset



```
// following uses of RegInit are valid
val reg1 = RegInit(24.U(8.W))
val reg2 = Reg(UInt(8.W))
val reg3 = RegInit(reg2)

// following uses are invalid
val reg1 = RegInit(0.U(UInt(8.W)))
val reg2 = RegInit(UInt(8.W))
```

RegEnable

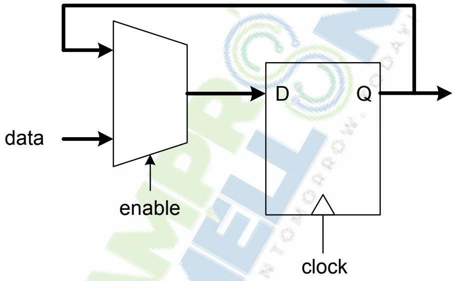
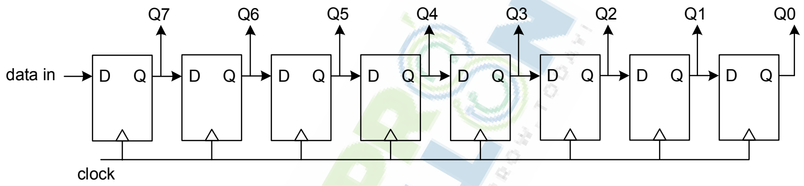


Figure: Register with enable input.

```
val regWithEnable = RegEnable(nextVal, ena)
```

Shift Register: Serial In Parallel Out



```
// shift register (serial in and parallel out)
val shiftReg = RegInit(0.U(n.W))

// shift register implementation
shiftReg := Cat(data_in , shiftReg (n-1, 1))
val Q = shiftReg
```

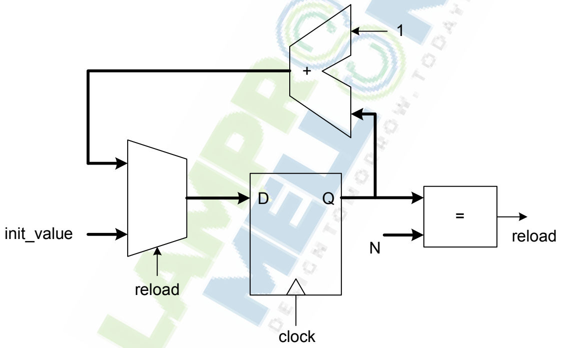
Shift Register: Implementation

```
// shift register example
import chisel3._

class shift_reg(val init: Int = 1) extends Module {
    val io = IO(new Bundle{
        val in = Input(Bool())
        val out = Output(UInt(4.W))
    })
    // register initialization
    val state = RegInit(init.U(4.W))

    // serial data in at LSB
    val nextState = (state << 1) | io.in
    state := nextState
    io.out := state
}
println((new chisel3.stage.ChiselStage).emitVerilog(new
    shift_reg))
```

Counter



Counter Cont'd

```
// Optimized counter example
import chisel3._
import chisel3.util._

class counter(val max: Int, val min: Int = 0) extends Module
     {
    val io = IO(new Bundle{
        val out = Output(UInt(log2Ceil(max).W))
    })
    val counter = RegInit(min.U(log2Ceil(max).W))

    // If the max count is of power 2 and the min value = 0,
    // then we can skip the comparator and the Mux
    val count_buffer = if (isPow2(max) && (min == 0))
      counter + 1.U
    else Mux(counter === max.U, min.U, counter + 1.U)
    counter := count_buffer
    io.out  := counter
}
println((new chisel3.stage.ChiselStage).emitVerilog(new
    counter(32)))
```

One-Shot Timer

```
// one shot timer implementation
val timer_count = RegInit(0.U(8.W))
val done = timer_count === 0.U
val next = WireInit(0.U)

when (reload){
    next := din                // load the data from input
}
.elsewhen (!done){
    next := timer_count - 1.U  // decrement the timer
}
timer_count := next           // update the timer
```

PWM Generation

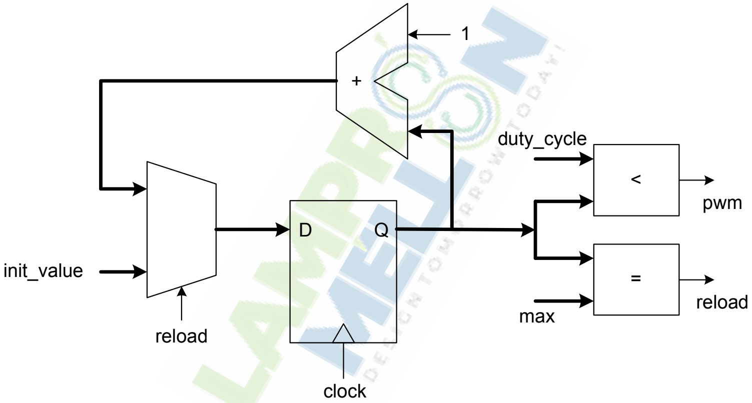


Figure: Block diagram for PWM generation.

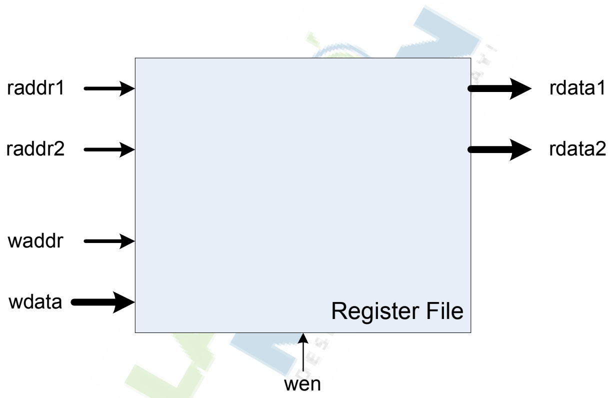
PWM Generation Cont'd

```
// PWM example
import chisel3._
import chisel3.util._

class PWM(val max: Int = 2, val duty_cycle: Int = 1) extends
      Module {
     val io = IO(new Bundle{
          val out = Output(Bool())
     })
     val counter = RegInit(0.U(log2Ceil(max).W))
     counter := Mux(counter === max.U, 0.U, counter+1.U)
     io.out := duty_cycle.U > counter
}

println((new chisel3.stage.ChiselStage).emitVerilog(new PWM
     (15)))
```

Register File



Implementing Register File

```
import chisel3._

class RegFileIO  extends Bundle with Config {
     val raddr1 = Input(UInt(5.W))
     val raddr2 = Input(UInt(5.W))
     val rdata1 = Output(UInt(XLEN.W))
     val rdata2 = Output(UInt(XLEN.W))
     val wen    = Input(Bool())
     val waddr  = Input(UInt(5.W))
     val wdata  = Input(UInt(XLEN.W))
}

class RegFile extends Module with Config {
     val io = IO(new RegFileIO)
     val regs = Reg(Vec(REGFILE_LEN, UInt(XLEN.W)))

     io.rdata1 := Mux((io.raddr1.orR), regs(io.raddr1), 0.U)
     io.rdata2 := Mux((io.raddr2.orR), regs(io.raddr2), 0.U)

     when(io.wen & io.waddr.orR) {
          regs(io.waddr) := io.wdata
     }
}
```

Queues

- Queue interface using Decoupled
- Implements queue with 16 elements

```
import chisel3._
import chisel3.util._

class User_Queue extends Module {
    val io = IO(new Bundle {
        //valid is Input, ready is Output, bits is Input
        val in = Flipped(Decoupled(UInt(8.W)))
        //valid is Output, ready is Input , bits is Output
        val out = Decoupled(UInt(8.W))
    })
    // 16 element queue
    val queue = Module(new Queue(UInt(), 16))
    queue.io.enq <> io.in
    io.out <> queue.io.deq
}

println(chisel3.Driver.emitVerilog(new User_Queue))
```

BlackBox

- Integration of existing Verilog IP is an essential requirement
- Chisel solution to this problem is `BlackBox`
- BlackBox is instantiated in the generated Verilog
- No implicit clock or reset in BlackBox, explicit connectivity required for this purpose

BlackBox Cont'd

Adder implementation using BlackBox

```
class BlackBoxAdder extends BlackBox with
    HasBlackBoxResource {
    val io = IO(new Bundle() {
        val in1 = Input(UInt(32.W))
        val in2 = Input(UInt(32.W))
        val out = Output(UInt(33.W))
    })
    setResource("/Adder.v")
}
```

BlackBox Cont'd

Adder implementation using BlackBox with inline Verilog

```
class BlackBoxAdder extends BlackBox with HasBlackBoxInline
     {
    val io = IO(new Bundle() {
        val in1 = Input(UInt(32.W))
        val in2 = Input(UInt(32.W))
        val out = Output(UInt(33.W))
    })
    setInline("BlackBoxAdder.v",
    s"""
    |module BlackBoxAdder(
    |  input [32:0] in1,
    |  input [32:0] in2,
    |  output [33:0] out
    |);
    |always @* begin
    |  out <= ((in1) + (in2));
    |end
    |endmodule
    """.stripMargin)
}
```

Reading List I

- Read the relevant sections from Chapter 6 of [Schoeberl, 2019]
- Consult [chisel3, 2020] for further details

References

chisel3 (2020).
Chisel3 library reference.
https://www.chisel-lang.org.

Schoeberl, M. (2019).
Digital Design with Chisel.
Kindle Direct Publishing.