8 Finite-State Machines

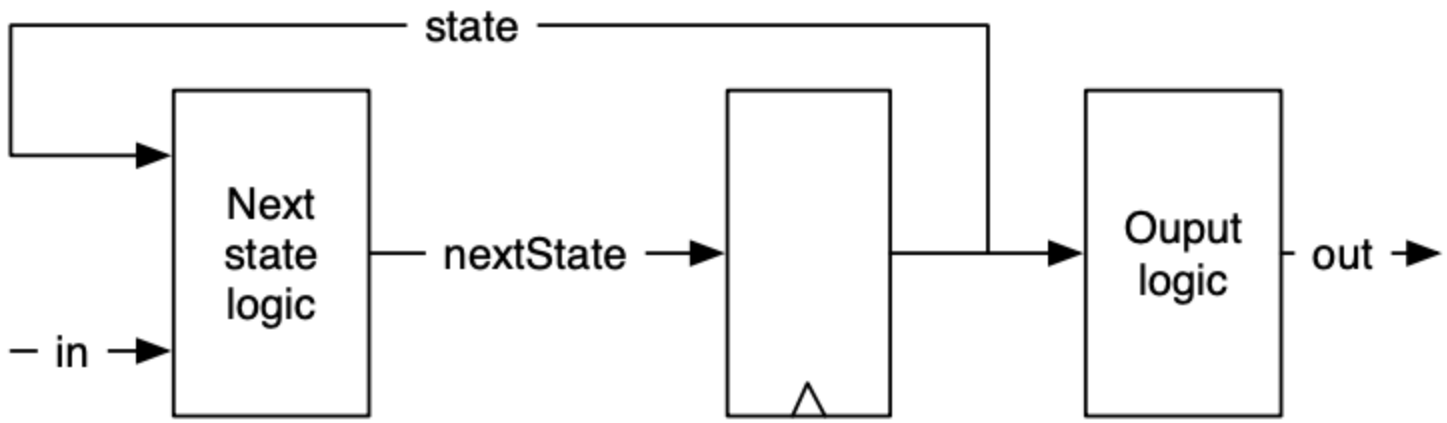
8.1 Basic Finite-State Machine

8.2 Faster Output with a Mealy FSM

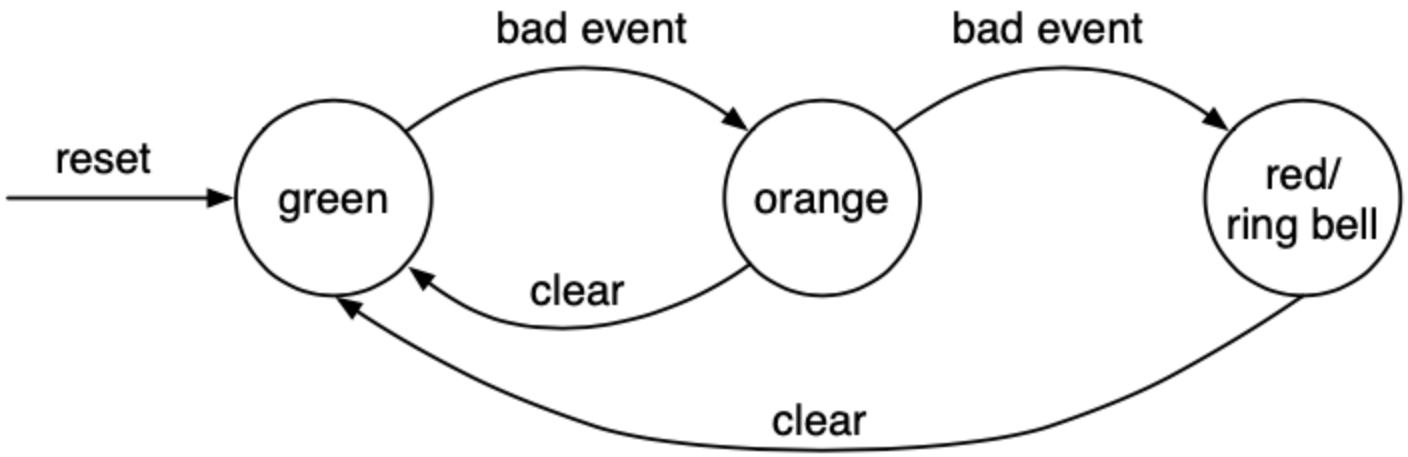
8.3 Moore versus Mealy

8.4 Exercise

8.1 Moore Machine



The state diagram of an alarm FSM



State table for the alarm FSM

	Input			
State	Bad event	Clear	Next state	Ring bell
green	0	0	green	0
green	1	-	orange	0
orange	0	0	orange	0
orange	1	-	red	0
orange	0	1	green	0
red	0	0	red	1
red	0	1	green	1

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

class SimpleFsm extends Module {
  val io = IO(new Bundle{
    val badEvent = Input(Bool())
    val clear = Input(Bool())
    val ringBell = Output(Bool())
  })

  // The three states
  val green :: orange :: red :: Nil = Enum(3)

  // The state register
  val stateReg = RegInit(green)

  // Next state logic
  switch (stateReg) {
    is (green) {
      when(io.badEvent) {
        stateReg := orange
      }
    }
    is (orange) {
      when(io.badEvent) {
        stateReg := red
      } .elsewhen(io.clear) {
        stateReg := green
      }
    }
    is (red) {
      when (io.clear) {
        stateReg := green
      }
    }
  }

  // Output logic
  io.ringBell := stateReg === red
}
```

Chisel Bundle:

```
val io = IO(new Bundle{
  val badEvent = Input(Bool())
  val clear = Input(Bool())
  val ringBell = Output(Bool())
})
```

FSM States

```
val green :: orange :: red :: Nil = Enum(3)
```

```
val stateReg = RegInit(green)
```

Next state logic

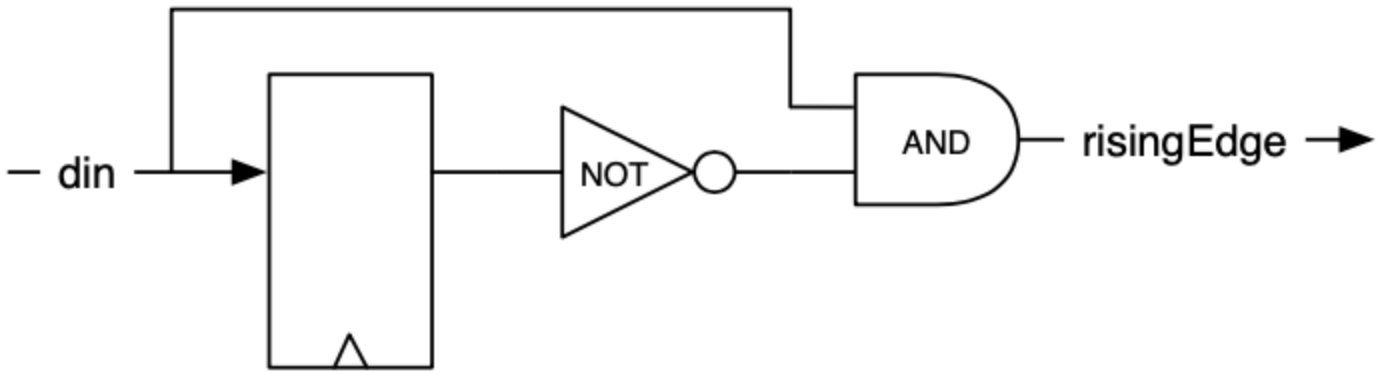
```
switch (stateReg) {
  is (green) {
    when(io.badEvent) {
      stateReg := orange
    }
  }
  is (orange) {
    when(io.badEvent) {
      stateReg := red
    } .elsewhen(io.clear) {
      stateReg := green
    }
  }
  is (red) {
    when (io.clear) {
      stateReg := green
    }
  }
```

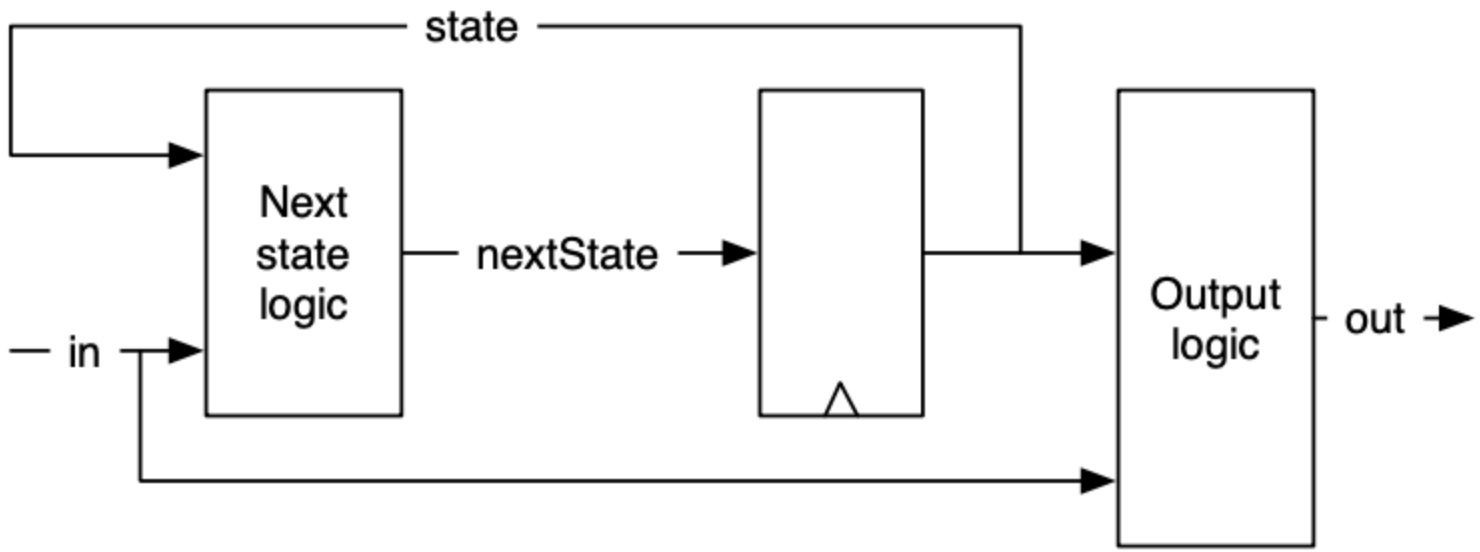
Output Logic

```
io.ringBell := stateReg === red
```

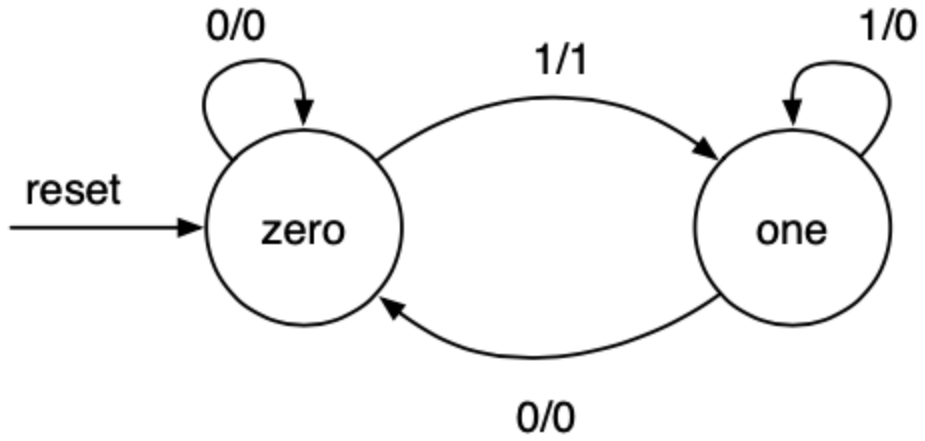
8.2 Faster Output with a Mealy FSM

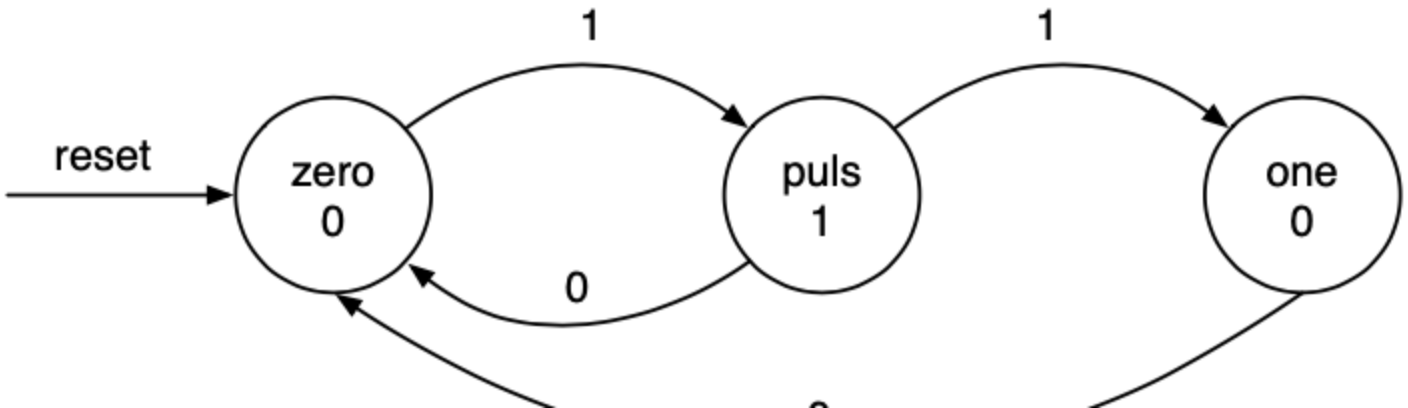
```
val risingEdge = din & !RegNext(din)
```



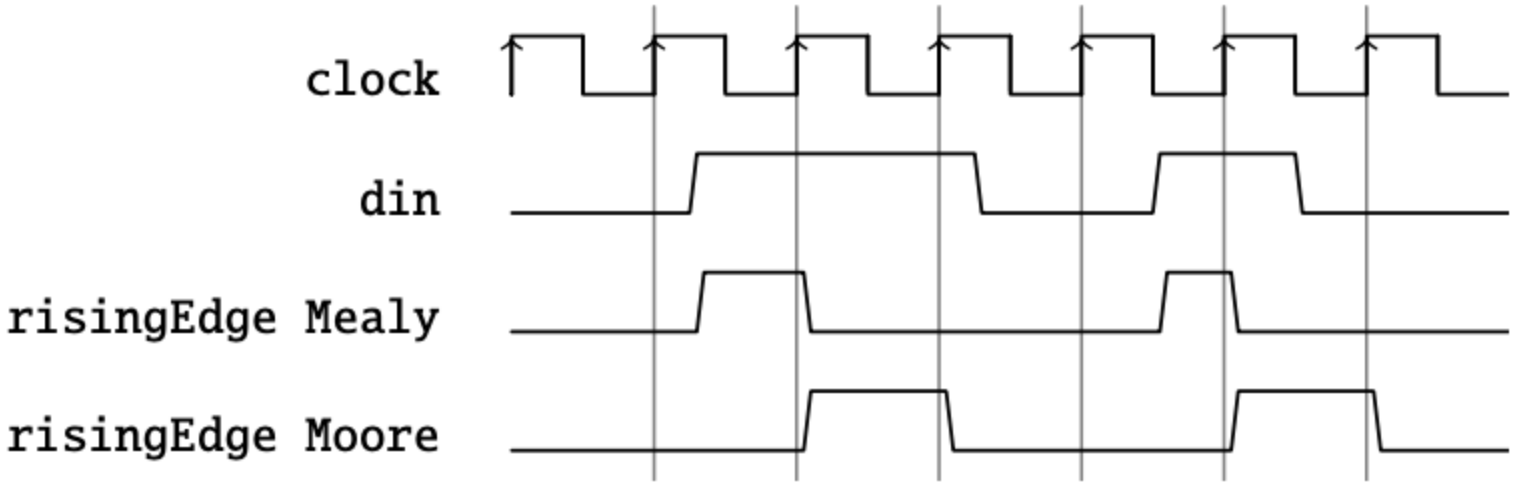

state
Next
state
logic
nextState
in
Output
logic
out

8.3 Moore versus Mealy





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

class RisingFsm extends Module {
  val io = IO(new Bundle{
    val din = Input(Bool())
    val risingEdge = Output(Bool())
  })

  // The two states
  val zero :: one :: Nil = Enum(2)

  // The state register
  val stateReg = RegInit(zero)

  // default value for output
  io.risingEdge := false.B

  // Next state and output logic
  switch (stateReg) {
    is(zero) {
      when(io.din) {
        stateReg := one
        io.risingEdge := true.B
      }
    }
    is(one) {
      when(!io.din) {
        stateReg := zero
      }
    }
  }
}
```


clock
din
risingEdge Mealy
risingEdge Moore

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

class RisingMooreFsm extends Module {
  val io = IO(new Bundle{
    val din = Input(Bool())
    val risingEdge = Output(Bool())
  })

  // The three states
  val zero :: puls :: one :: Nil = Enum(3)

  // The state register
  val stateReg = RegInit(zero)

  // Next state logic
  switch (stateReg) {
    is(zero) {
      when(io.din) {
        stateReg := puls
      }
    }
    is(puls) {
      when(io.din) {
        stateReg := one
      } .otherwise {
        stateReg := zero
      }
    }
    is(one) {
      when(!io.din) {
        stateReg := zero
      }
    }
  }

  // Output logic
  io.risingEdge := stateReg === puls
}
```

8.4 Exercise

이 장에서는 매우 작은 FSM의 많은 예를 보았습니다. 이제 실제 FSM 코드를 작성할 시간입니다. 조금 더 복잡한 예를 선택하고 FSM을 구현하고 이에 대한 테스트 벤치를 작성하십시오.

FSM의 전형적인 예는 신호등 컨트롤러입니다([3, 섹션 14.3] 참조). 신호등 컨트롤러는 빨간색에서 녹색으로 전환할 때 교차로의 두 도로 사이에 금지 신호등(빨간색 및 주황색)이 있는 단계가 있는지 확인해야 합니다. 이 예를 조금 더 흥미롭게 만들려면 우선 순위를 고려하십시오. 작은 도로에는 두 대의 차량 감지기가 있습니다(교차로로 들어가는 두 입구 모두). 차가 감지될 때만 작은 도로에 대해 녹색으로 전환한 다음 우선 도로에 대해 다시 녹색으로 전환합니다.