7 Input Processing

7.1 Asynchronous Input

7.2 Debouncing

7.3 Filtering of the Input Signal

7.4 Combining the Input Processing with Functions

7.5 Synchronizing Reset

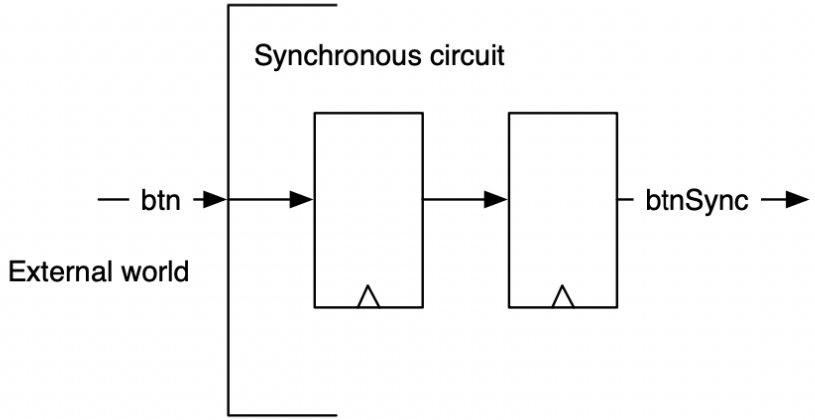
7.6 Exercise

외부 세계에서 동기 회로로의 입력 신호는 일반적으로 클록과 동기되지 않습니다. 그것들은 비동기적입니다. 입력 신호는 0에서 1 또는 1에서 0으로의 깨끗한 전환이 없는 소스에서 올 수 있습니다. 예를 들어 버튼이나 스위치가 튀는 것입니다. 입력 신호는 동기 회로에서 전환을 유발할 수 있는 스파이크와 함께 잡음이 있을 수 있습니다. 이 장에서는 이러한 입력 조건을 처리하는 회로에 대해 설명합니다.

후자의 두 가지 문제인 디바운싱 스위치와 노이즈 필터링은 외부 아날로그 구성 요소로도 해결할 수 있습니다. 그러나 디지털 영역에서 이러한 문제를 처리하는 것이 (비용)효율적입니다.

7.1 Asynchronous Input

```
val btnSync = RegNext(RegNext(btn))
```



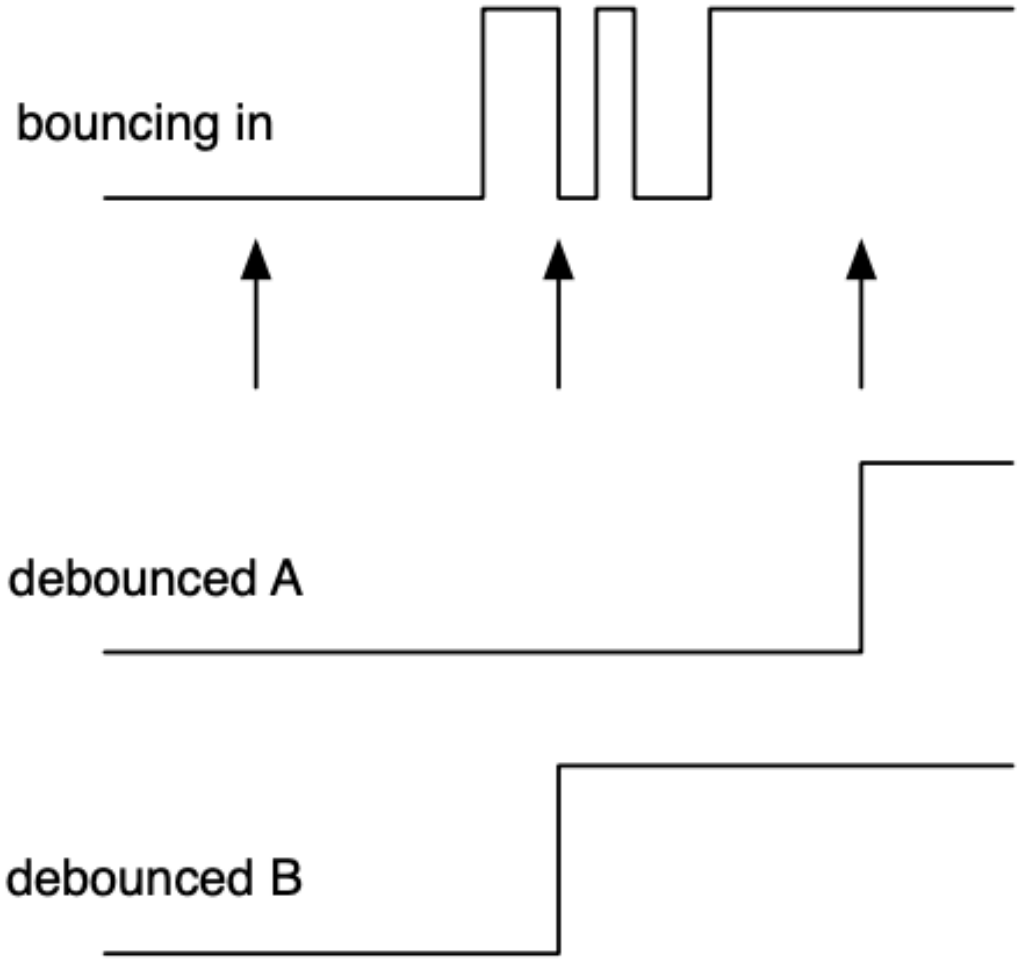
7.2 Debouncing

```
val FAC = 100000000/100

val btnDebReg = Reg(Bool())

val cntReg = RegInit(0.U(32.W))
val tick = cntReg === (fac-1).U

cntReg := cntReg + 1.U
when (tick) {
  cntReg := 0.U
  btnDebReg := btnSync
}
```

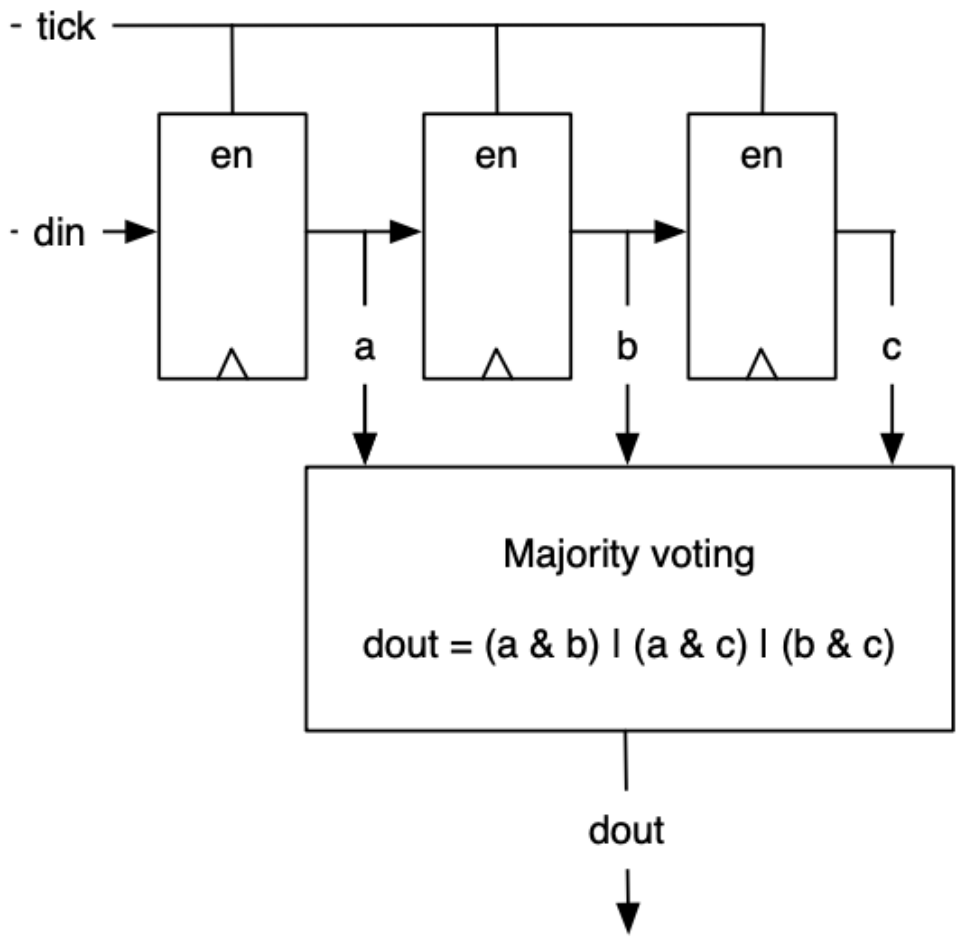


7.3 Filtering of the Input Signal

```
val shiftReg = RegInit(0.U(3.W))
when (tick) {
  // shift left and input in LSB
  shiftReg := shiftReg(1, 0) ## btnDebReg
}
// Majority voting
val btnClean = (shiftReg(2) & shiftReg(1)) | (shiftReg(2) &
    shiftReg(0)) | (shiftReg(1) & shiftReg(0))
```

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

// Use the rising edge of the debounced and
// filtered button to count up
val reg = RegInit(0.U(8.W))
when (risingEdge) {
  reg := reg + 1.U
}
```



7.4 Combining the Input Processing with Function

```
def sync(v: Bool) = RegNext(RegNext(v))

def rising(v: Bool) = v & !RegNext(v)

def tickGen() = {
  val reg = RegInit(0.U(log2Up(fac).W))
  val tick = reg === (fac-1).U
  reg := Mux(tick, 0.U, reg + 1.U)
  tick
}
```

```
def filter(v: Bool, t: Bool) = {
  val reg = RegInit(0.U(3.W))
  when (t) {
    reg := reg(1, 0) ## v
  }
  (reg(2) & reg(1)) | (reg(2) & reg(0)) | (reg(1) & reg(0))
}
```

```
val btnSync = sync(io.btnU)

val tick = tickGen()
val btnDeb = Reg(Bool())
when (tick) {
  btnDeb := btnSync
}

val btnClean = filter(btnDeb, tick)
val risingEdge = rising(btnClean)

// Use the rising edge of the debounced
// and filtered button for the counter
val reg = RegInit(0.U(8.W))
when (risingEdge) {
  reg := reg + 1.U
}
```

7.5 Synchronizing Reset

```
class SyncReset extends Module {
  val io = IO(new Bundle() {
    val value = Output(UInt())
  })

  val syncReset = RegNext(RegNext(reset))
  val cnt = Module(new WhenCounter(5))
  cnt.reset := syncReset

  io.value := cnt.io.cnt
}
```

7.6 Exercise

입력 버튼에 의해 증가되는 카운터를 작성하십시오. FPGA 보드의 LED를 사용하여 카운터 값을 바이너리로 표시합니다. (1) 입력 싱크로나이저, (2) 디바운싱 회로, (3) 노이즈를 억제하는 다수결 회로, (4) 카운터 증가를 트리거하는 에지 감지 회로로 완전한 입력 처리 체인을 구축하십시오.

7.6 Exercise

최신 버튼이 항상 바운스된다는 보장은 없으므로 버튼을 수동으로 빠르게 연속해서 누르고 낮은 샘플 주파수를 사용하여 바운스와 스파이크를 시뮬레이션할 수 있습니다. 예를 들어 1초를 샘플 주파수로 선택합니다. 즉, 입력 클럭이 100MHz에서 실행되는 경우 100,000,000으로 나눕니다. 안정적인 프레스에 도달하기 전에 빠르게 여러 번 연속해서 눌러 튀는 버튼을 시뮬레이션합니다. 1Hz에서 디바운싱 회로 샘플링을 사용하거나 사용하지 않고 회로를 테스트합니다. 과반수 투표의 경우 카운터의 안정적인 증가를 위해 1초에서 2초 사이를 눌러야 합니다. 또한 버튼의 릴리스는 과반수 투표입니다. 따라서 회로는 릴리스가 1-2초보다 길 때만 릴리스를 인식합니다.