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

class Accelerator extends Module {
  val io = IO(new Bundle {
    val start = Input(Bool())
    val done = Output(Bool())

    val address = Output(UInt(16.W))
    val dataRead = Input(UInt(32.W))
    val writeEnable = Output(Bool())
    val dataWrite = Output(UInt(32.W))
  })

  // State enum and register
  //     0 ::       1 ::     2 ::    3 ::         4 ::          5 ::     6 ::
  val idle :: regInit :: count :: done :: regUpdate :: checkPixel :: write :: Nil =
    Enum(7)
  val stateReg = RegInit(idle)

  // Support registers
  val addressReg = RegInit(0.U(16.W))
  val dataReg = Reg(Vec(60, UInt(32.W)))
  val xReg = RegInit(0.U(32.W))
  val yReg = RegInit(0.U(32.W))
  val varReg = RegInit(0.U(32.W))

  // Default values
  io.writeEnable := false.B
  io.address := 0.U(16.W)
  io.dataWrite := 0.U(16.W)
  io.done := false.B

  switch(stateReg) {
    is(idle) {
      when(io.start) {
        varReg := 20.U(32.W)
        stateReg := regInit
      }.otherwise {
        stateReg := idle
      }
    }

    is(regInit) {
      when(varReg < 60.U) {
        io.address := varReg
        dataReg(varReg) := io.dataRead
        varReg := varReg + 1.U
        stateReg := regInit
      }.otherwise {
        stateReg := count
      }
    }

    is(count) {
      when(yReg === 20.U) {
        stateReg := done
      }.elsewhen(xReg === 20.U) {
        varReg := 0.U
        stateReg := regUpdate
      }.elsewhen(
        xReg === 0.U || xReg === 19.U || yReg === 0.U || yReg === 19.U
```

```
      ) {
        addressReg := xReg + 20.U * yReg + 400.U
        varReg := 0.U
        stateReg := write
      }.elsewhen(xReg < 20.U) {
        addressReg := xReg + 20.U * yReg + 400.U
        varReg := xReg + 20.U
        stateReg := checkPixel
      }
    }

    is(regUpdate) {
      when(varReg < 40.U) {
        dataReg(varReg) := dataReg(varReg + 20.U)
        varReg := varReg + 1.U
        stateReg := regUpdate
      }.elsewhen(varReg < 60.U) {
        addressReg := varReg + 1.U + 20.U * (yReg)
        io.address := addressReg
        dataReg(varReg) := io.dataRead
        varReg := varReg + 1.U
        stateReg := regUpdate
      }.otherwise {
        yReg := yReg + 1.U
        xReg := 0.U
        stateReg := count
      }
    }

    is(checkPixel) {
      when(dataReg(varReg) === 0.U) {
        varReg := 0.U
        stateReg := write
      }.elsewhen(
        dataReg(varReg - 20.U) === 0.U ||
          dataReg(varReg - 1.U) === 0.U ||
          dataReg(varReg + 1.U) === 0.U ||
          dataReg(varReg + 20.U) === 0.U
      ) {
        varReg := 0.U
        stateReg := write
      }.otherwise {
        varReg := 255.U
        stateReg := write
      }
    }

    is(write) {
      io.writeEnable := true.B
      io.address := addressReg
      io.dataWrite := varReg
      xReg := xReg + 1.U
      addressReg := 0.U
      varReg := 0.U
      stateReg := count
    }

    is(done) {
      io.done := true.B
      stateReg := done
    }
  }
}
```