

```

1  import chisel3._
2  import chisel3.util._
3
4  class Accelerator extends Module {
5      val io = IO(new Bundle {
6          val start = Input(Bool())
7          val done = Output(Bool())
8
9          val address = Output(UInt(16.W))
10         val dataRead = Input(UInt(32.W))
11         val writeEnable = Output(Bool())
12         val dataWrite = Output(UInt(32.W))
13     })
14
15     // State enum and register
16     //      0 ::      1 ::      2 ::      3 ::      4 ::      5 ::      6 ::
17     val idle :: regInit :: count :: done :: regUpdate :: checkPixel :: write :: Nil =
18         Enum(7)
19     val stateReg = RegInit(idle)
20
21     // Support registers
22     val addressReg = RegInit(0.U(16.W))
23     val dataReg = Reg(Vec(60, UInt(32.W)))
24     val xReg = RegInit(0.U(32.W))
25     val yReg = RegInit(0.U(32.W))
26     val varReg = RegInit(0.U(32.W))
27
28     // Default values
29     io.writeEnable := false.B
30     io.address := 0.U(16.W)
31     io.dataWrite := 0.U(16.W)
32     io.done := false.B
33
34     switch(stateReg) {
35         is(idle) {
36             when(io.start) {
37                 varReg := 20.U(32.W)
38                 stateReg := regInit
39             }.otherwise {
40                 stateReg := idle
41             }
42         }
43
44         is(regInit) {
45             when(varReg < 60.U) {
46                 io.address := varReg
47                 dataReg(varReg) := io.dataRead
48                 varReg := varReg + 1.U
49                 stateReg := regInit
50             }.otherwise {
51                 stateReg := count
52             }
53         }
54
55         is(count) {
56             when(yReg === 20.U) {
57                 stateReg := done
58             }.elsewhen(xReg === 20.U) {
59                 varReg := 0.U
60                 stateReg := regUpdate
61             }.elsewhen(
62                 xReg === 0.U || xReg === 19.U || yReg === 0.U || yReg === 19.U

```

```

63     ) {
64         addressReg := xReg + 20.U * yReg + 400.U
65         varReg := 0.U
66         stateReg := write
67     }.elsewhen(xReg < 20.U) {
68         addressReg := xReg + 20.U * yReg + 400.U
69         varReg := xReg + 20.U
70         stateReg := checkPixel
71     }
72 }
73
74 is(regUpdate) {
75     when(varReg < 40.U) {
76         dataReg(varReg) := dataReg(varReg + 20.U)
77         varReg := varReg + 1.U
78         stateReg := regUpdate
79     }.elsewhen(varReg < 60.U) {
80         addressReg := varReg + 1.U + 20.U * (yReg)
81         io.address := addressReg
82         dataReg(varReg) := io.dataRead
83         varReg := varReg + 1.U
84         stateReg := regUpdate
85     }.otherwise {
86         yReg := yReg + 1.U
87         xReg := 0.U
88         stateReg := count
89     }
90 }
91
92 is(checkPixel) {
93     when(dataReg(varReg) === 0.U) {
94         varReg := 0.U
95         stateReg := write
96     }.elsewhen(
97         dataReg(varReg - 20.U) === 0.U ||
98         dataReg(varReg - 1.U) === 0.U ||
99         dataReg(varReg + 1.U) === 0.U ||
100        dataReg(varReg + 20.U) === 0.U
101     ) {
102         varReg := 0.U
103         stateReg := write
104     }.otherwise {
105         varReg := 255.U
106         stateReg := write
107     }
108 }
109
110 is(write) {
111     io.writeEnable := true.B
112     io.address := addressReg
113     io.dataWrite := varReg
114     xReg := xReg + 1.U
115     addressReg := 0.U
116     varReg := 0.U
117     stateReg := count
118 }
119
120 is(done) {
121     io.done := true.B
122     stateReg := done
123 }
124 }
125 }

```