02132 Assignment 3 report

Implementation of an FSMD-based hardware accelerator for the image erosion in Chisel

Group: 22

Niclas Juul Schæffer **s224744**
Rasmus Kronborg Finnemann Wiuff **s163977**
github.com/rwiuff/02132Assignment3

December 1st

1 Work distribution

Table 1 shows the work distribution for this project.

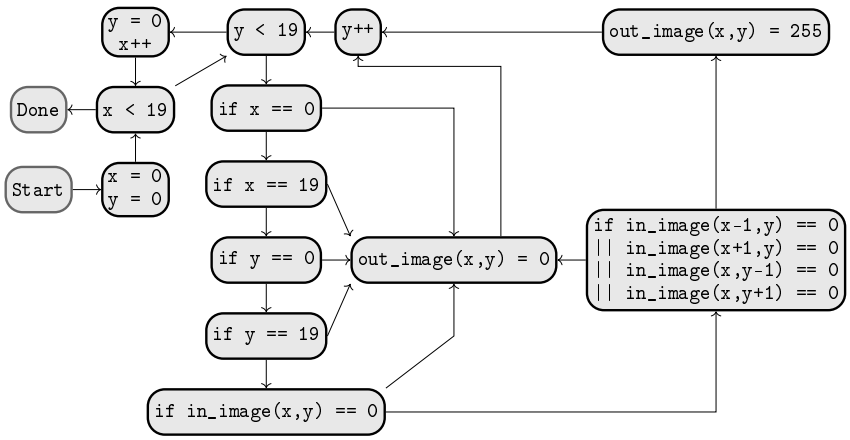
Table 1: *Work distribution on the project*

Name	Development tasks	Report tasks
Niclas Juul Schæffer		
Rasmus Wiuff		

2 Design

First a CFG was generated from the pseudocode in the assignment material, yielding the graph in Fig. 1.

Figure 1: *Control flow graph of erosion algorithm*



The diagram was then optimised:

1. All the logical comparisons regarding the inner pixel and neighbours can be merged into a single state.
2. Incrementers and loop conditions can be merged into a single state.
3. Border check can be merged into the loop condition state.
4. Write operations can be merged into a single state.

Next step focused on registers. The assignment allows O(x_size) registers, and thus registers for three image rows are chosen. Once the rows are stored in the registers, a row can be processed with neighbouring pixels without reading from memory, limiting memory access. Furthermore registers for `x`, `y`, the memory address and a variable was decided upon. The variable register frees up tampering with `x` or `y` throughout the erosion (other than incrementing). The register layout is shown in Fig. 2.

This gives two development tasks:

Figure 2: *The registers in the accelerator*

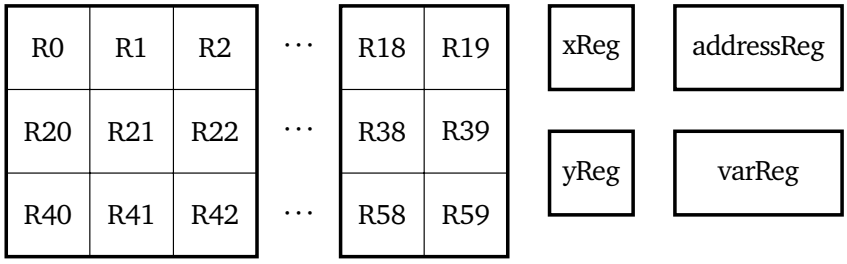
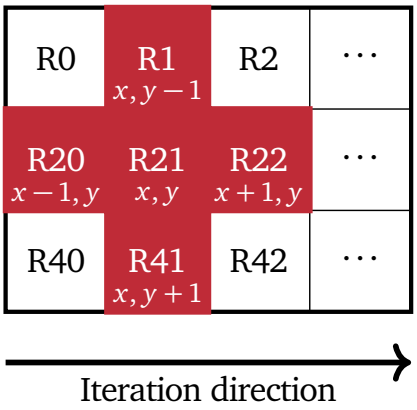
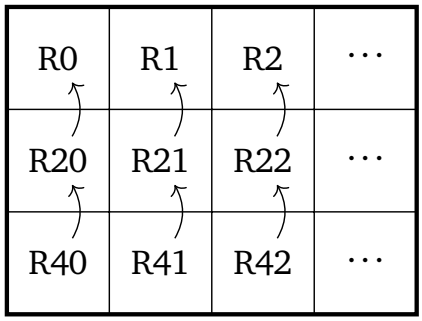


Figure 3: *Data register handling*

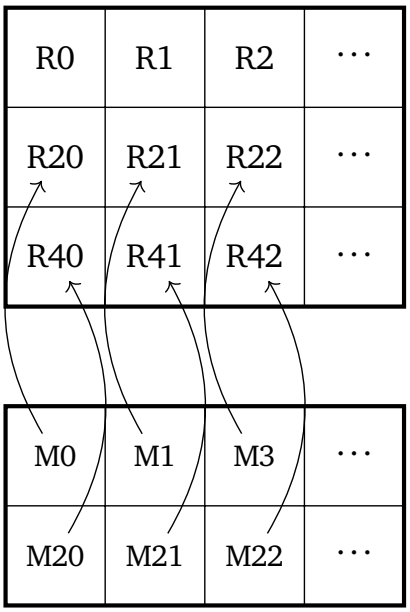
(a) *A row is analysed using the registers only*



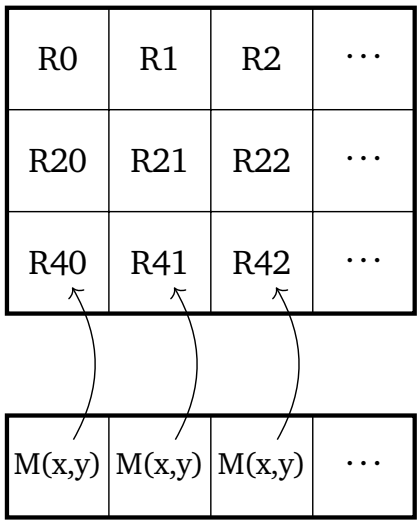
(b) *Image rows moves up in registers only*



(c) *Image rows are initially read from memory*



(d) *New image row from memory is put into registers*

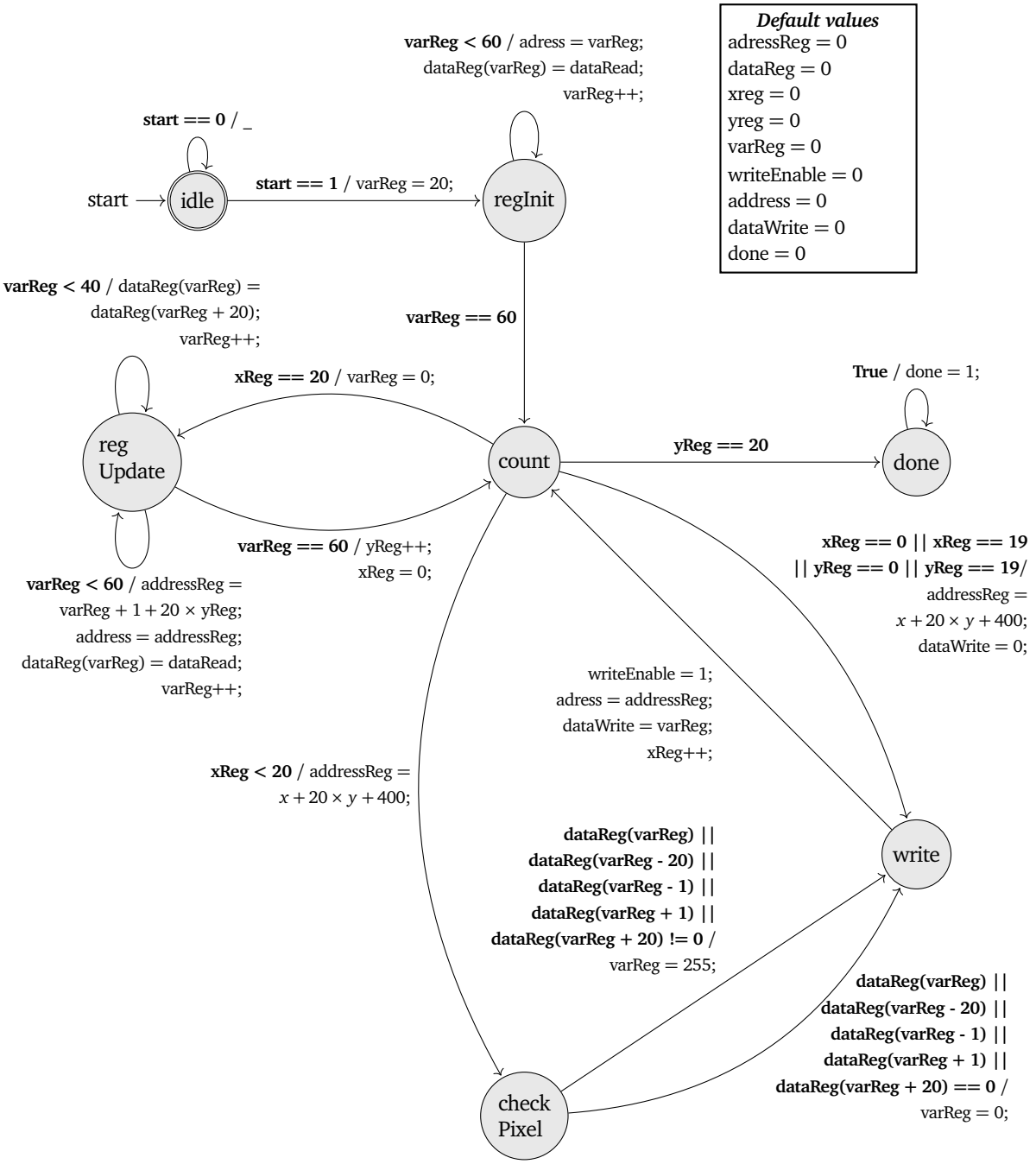


1. Registers needs to be initialised.
2. Registers needs to be updated on changing rows.

Each task is solved by a separate state. An initialisation state will store pixels in the registers R20 to R59 as shown in Fig. 3c. On every row only the registers are used as shown in Fig. 3a. Once the registers needs updating the register at position x is updated with the data in $x + 20$ for the first 40 registers, as shown in Fig. 3b. Once done the last 20 registers are updated with the next row from memory as shown in Fig. 3d.

To recapitulate: The following states are needed

Figure 4: *State diagram of the erosion accelerator FSMD*



- A state to initialise the registers with image rows.
- A state to update the image rows when a row have been analysed and eroded.
- A state to check if a pixel should be eroded.
- A state to write the output image to memory.
- A state to control incrementers.

The resulting FSMD is shown on the state diagram in Fig. 4.

3 Implementation

Listing 1: *stateReg*

```
1        val dataReg = Reg(Vec(60, UInt(32.W)))
```

regvec

Listing 2: *stateReg*

```
1        addressReg := varReg + 1.U + 20.U * (yReg)
```

address + 1

4 Test And Evaluation

Report here the results from the test you have carried out. Present how you have tested (paper and pencil testing) the FSMD you have designed. Present the tests you have developed (if any). Remember to discuss the relusts and the test you have carried out, do not just present them, but explain and argue their meaning. Adress the design evaluation questions in Task 6 in the Assignment 3 document.

4.1. Execution time and speed-up

Assignment 1 clock cyclese cell image 950x950:
Assignment 2 clock cycles cell image 20x20: 6815

4.2. Utilisation of resources

The number of needed functional units are derived from the state diagram in Fig. 4. Table 2 depicts units used by various states.

Table 2: *Number of functional units used in states*

State	=	<	||	+	−	×
idle	1					
done						
regInit	1	1		1		
regUpdate	1	2		3		1
count	6	1	3	2		1
checkPixel	4		4	2	2	
write				1		

The highest number of units are summarised in Table 3.

Table 3: *Total number of functional units in the FSMD*

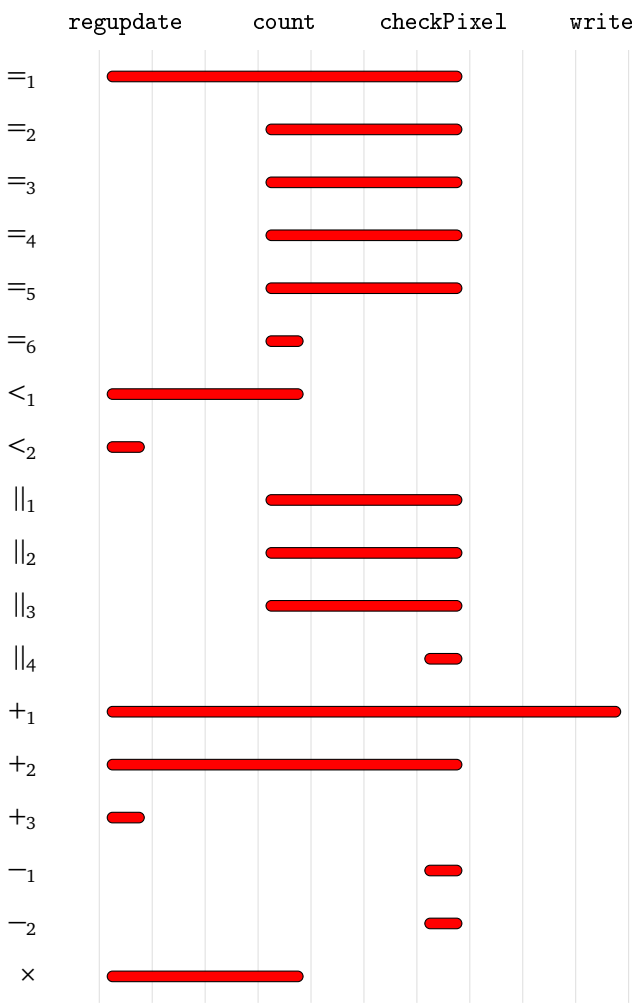
=	<	||	+	−	×
6	2	4	3	2	1

The longest path begins when the pixel registers are updated and during analysis throughout the row. The path is defined as `regUpdate` → `count` → `checkPixel` → `write` → `count`. The utilised resources in this path is shown in Fig. 5.

Using the formula for the functional units,

$$\frac{\#\text{ of states when its used}}{\#\text{ of states when it is used} + \#\text{ of states when it is not used}} \tag{4.1}$$

Figure 5: *Schedule of functional units in the longest FSMD path*



the utilisation of the functional units is determined. For the path in Fig. 5 the utilisation is given in Table 4.

Table 4: *Utilisation of functional units in the longest FSMD path*

Functional unit	$=_1$	$=_2$	$=_3$	$=_4$	$=_5$	$=_6$	$<_1$	$<_2$	$\|\|_1$
Utilisation	0.75	0.5	0.5	0.5	0.5	0.25	0.5	0.25	0.5
Functional unit	$+_1$	$+_2$	$+_3$	$-_1$	$-_2$	$\times$	$\|\|_2$	$\|\|_3$	$\|\|_4$
Utilisation	1	0.75	0.25	0.25	0.25	0.5	0.5	0.5	0.25

4.3. Hardware size