


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.

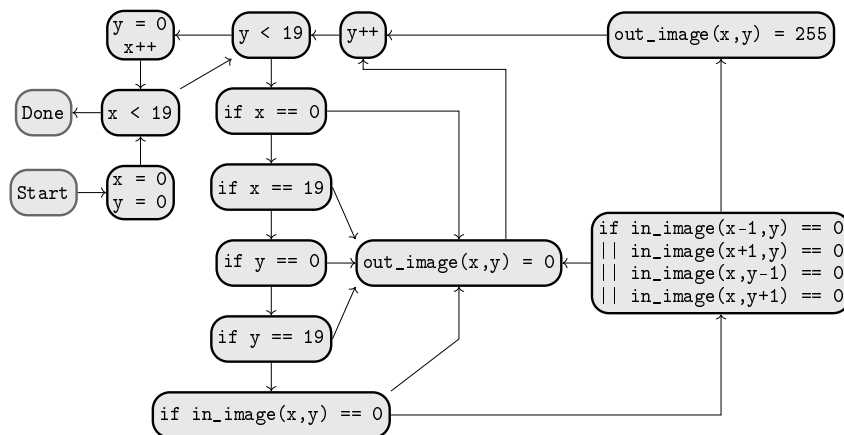
Table 1: Work distribution on the project

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

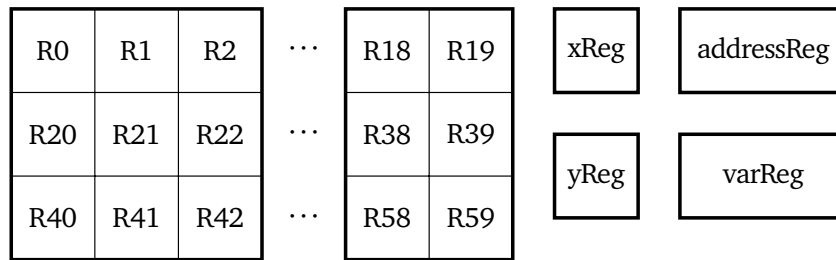
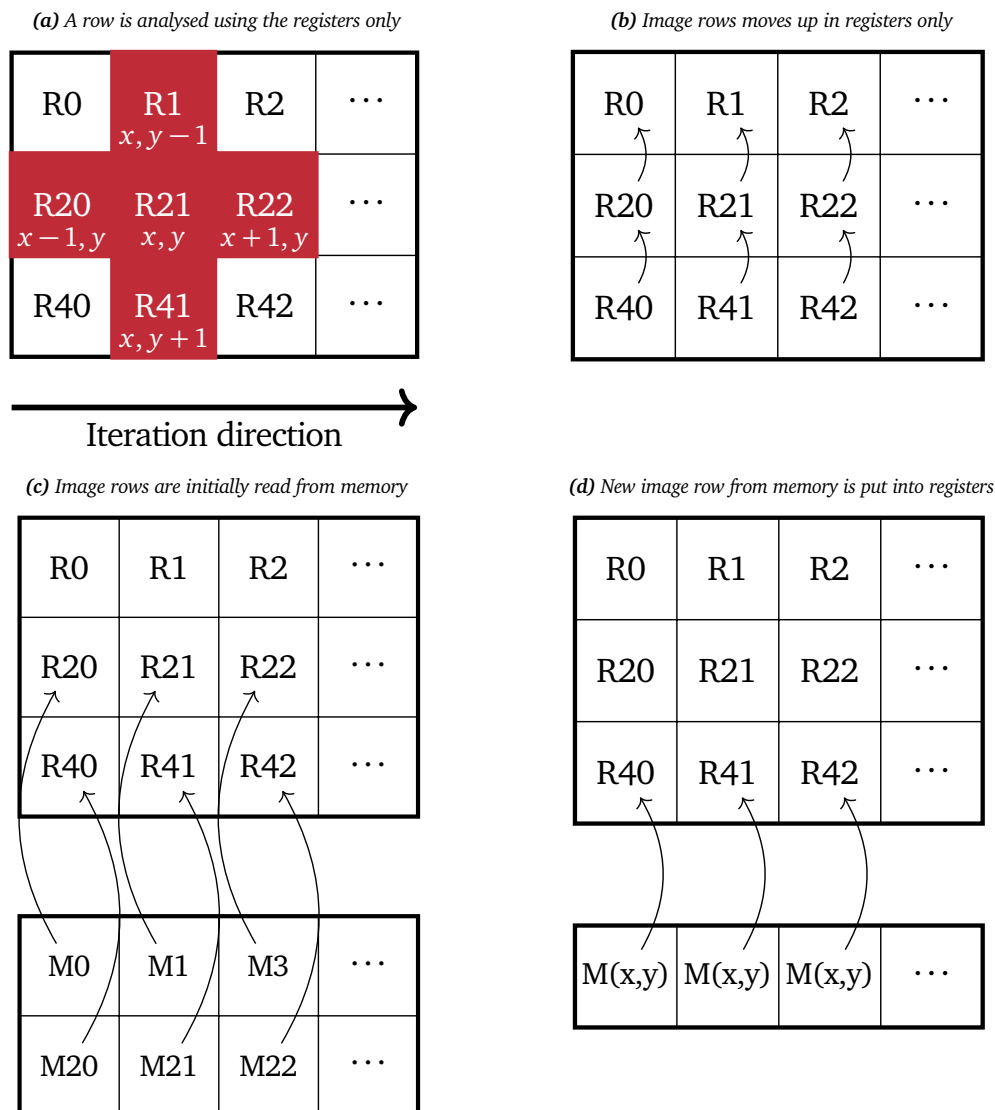


Figure 3: Data register handling

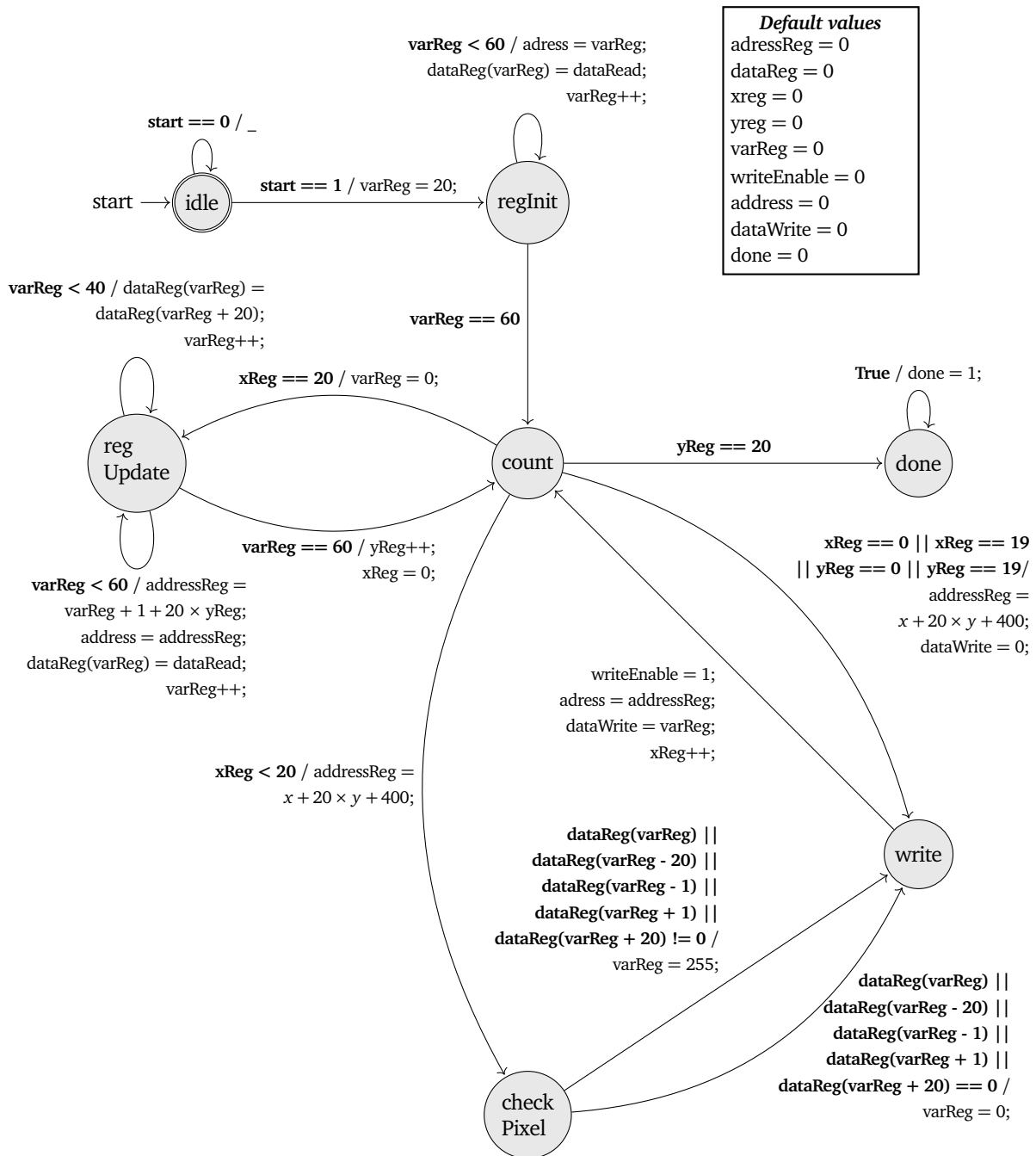


1. Registers needs to be initialised.
2. Registers needs to be updated on changing rows.

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 FSM



- 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 FSM is shown on the state diagram in Fig. 4.

3 IMPLEMENTATION

Briefly discuss the implementation in Chisel of your design. You can include some code snippets if these are relevant to explain certain aspects of the implementation. In other words, try to answer the question “What does a reader need to know about your Chisel implementation?”

We implemented a feature where we could control the states with a Enum called stateReg which we make starts on idle. The way we use it its that all our functions is some when loops, that checks if the state is certain state, if so then run the specific function corresponds to that state. when everything have been ran, then the state becomes done.

Listing 1: stateReg

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

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 FSM/D 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.