02132 Assignment 3: Implementation of an FSMD-based hardware accelerator for the image erosion in Chisel

Luca Pezzarossa and Jan Madsen

November 7, 2023

Preface

In this assignment, you are required to implement a small hardware accelerator for the erosion step from the algorithm for detecting and counting cells, which we presented in Assignment 1 and compiled by hand in Assignment 2. The hardware accelerator is based on an FSMD (Finite-State Machine with Datapath) and should be implemented using the Chisel language. This assignment aims to let you practice with pure hardware design (not programmable) and give you an understanding of how hardware accelerator can be used to speed-up computation-intensive sections of an algorithm.

As mentioned in the course introduction, this assignment is probably the easiest of the three assignments of the course since you have already worked with Chisel and no new complex concepts and design paradigms are introduced. Nevertheless, do not underestimate the workload for this assignment and take into account that it will overlap with the preparation for the exam.

1 General information

This assignment should be carried out in **groups of two or three people** and you are free to keep your previous group or change it. In any case, you need to **re-register** the group in DTU-Learn. In addition to the design tasks, you are also required to prepare a short report describing the approach used in its implementation (further details are provided in Section 5). All the material related to this assignment can be found on the DTU-Learn course page at the following location:

`DTU-Learn/Course content/Content/Assignments/Assignment 3`

The deadline for this assignment is **Friday 1st December 2023 at 11:59**. By this date, you have to hand-in an electronic version of the short report in PDF format and the source files as ZIP archive using the assignment utility in the DTU-Learn course page at the following location:

`DTU-Learn/Course content/Assignments/Assignment 3`

Before the deadline, during the last laboratory session dedicated to this assignment held on **Wednesday 29th November 2023**, you are asked to demonstrate the functionality of your implementation to the teacher or to the TAs. More information regarding the DEMO session will follow as DTU-Learn announcements.

This document is divided into 4 sections and 1 appendix:

- Section 2 provides the general background needed for the assignment and describes the provided code as well as the overall hardware accelerator specifications.
- Section 3 presents the developing and testing tasks you need to carry out in this assignment.
- Section 4 lists important notes and hints for this assignment.
- Section 5 discusses time-management, report requirements, and evaluation criteria.
- Appendix A lists useful pointers to Chisel documentation and material online.

Similarly to Assignment 1 and 2, we are not enforcing a particular approach (besides the already implemented modules and tests), rather we would like you to explore. For this reason, it may be difficult for the teachers to understand your implementation and thus, help us read/debug your design by taking into account the following hints:

- Make sure you understand the FSMD model and its implementation in hardware.
- Structure your hardware design such that it is clear what the different parts/modules are doing.
- Test your hardware modules individually with a dedicated tester before connecting everything together.
- Comment your hardware design (when needed).
- Deliver all the source code needed to test the implementation. Add a README file with instructions on how to run the test cases including the ones you developed.

For this assignment, we assume that you have a functional Chisel environment already installed on your personal computer. If this is not the case, please take a look at the section *Setting up and working with Chisel* in the Assignment 2 document.

Feel free to ask or send an e-mail to the course teacher if you have questions regarding practical matters and the assignment in general.

2 Background and specifications

Background

Your work on the cell detection and counting algorithm at the fictive company Bioware System continues. After having investigated and implemented a small CPU running the erosion step of the algorithm, your research and development group wants to increase the execution time performance even more to be able to process in real-time the frames of a video stream of moving cells. Therefore, your last task in this project is to implement and evaluate a hardware accelerator to speed-up the erosion step of the algorithm, which seems to be the most computation-intensive step.

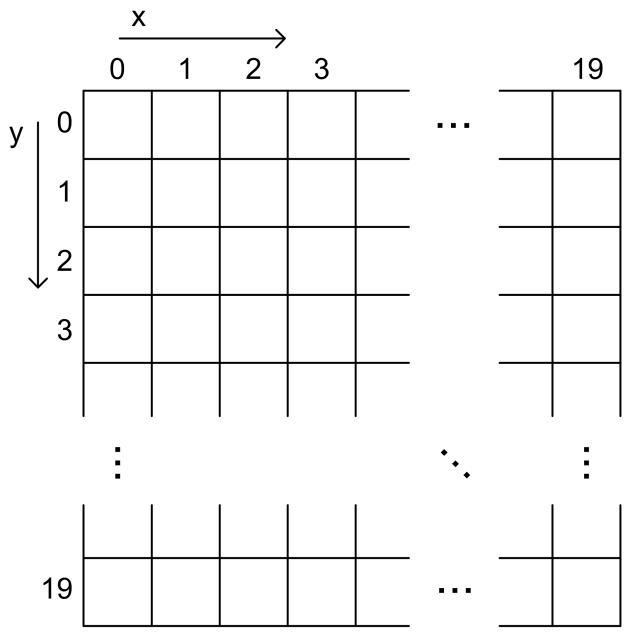

Figure 1: The pixel mapping of the 20-by-20 binary image.

To carry out this task you have to:

1. Design an FSMD version of the erosion step and test its functionality.
2. Implement the FSMD in hardware using Chisel.
3. Run and test your design.
4. Evaluate the performance of the accelerator in terms of speed and area.

For this assignment, we continue using a reduced size of 20-by-20 pixels for the binary image on which we run the erosion step. The pixels are mapped as shown in Figure 1. Black pixels correspond to the value 0 and white pixels correspond to the value 255.

In the following, we describe the accelerator specifications and the provided code/infrastructure.

Full system overview

Figure 2 shows the full system and its tester. The names of the modules match the names of the files/classes of the provided code. The modules outlined in red are provided to you already implemented.

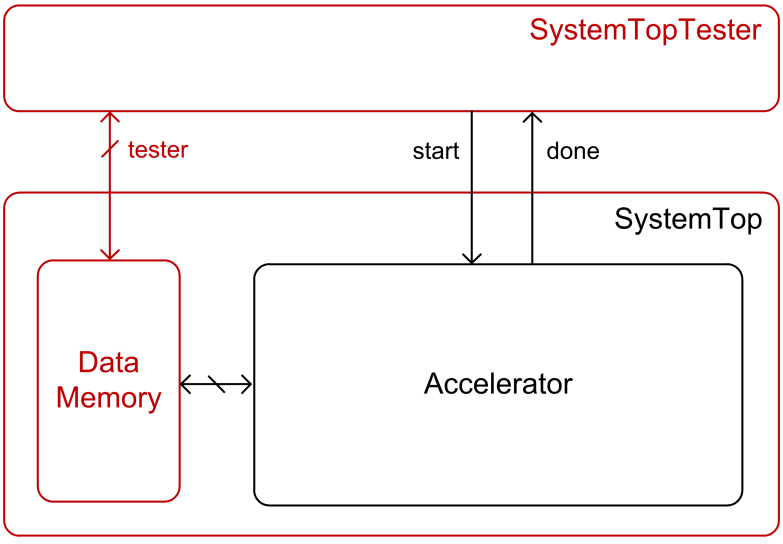

Figure 2: An overview of the full system and its tester.

Figure 3 shows the folder/file structure of the provided code, which is structured in a similar way as Assignment 2.

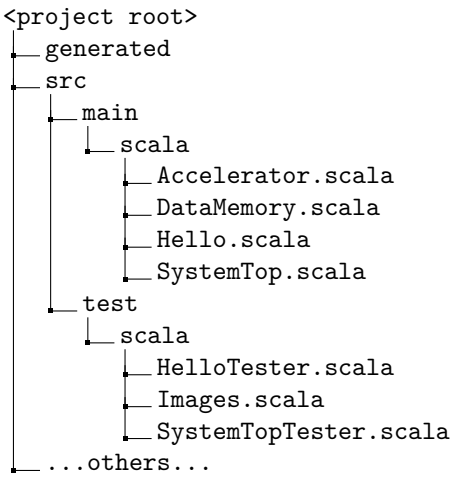

Figure 3: The folder/file structure of the provided code.

In the following, we describe the purpose and functionality of each module.

SystemTop module

This is the top-level module of your system. In this module, your hardware accelerator and other sub-modules are connected together.

The interface of this module consists of a set of `tester` inputs and outputs, a `start` boolean input and a `done` boolean output. The `tester` inputs and outputs are used by the SystemTopTester module to load and dump the content of the data memory with the binary image to be processed. **Do not touch these signals.**

The input signal `run` interacts with accelerator you are developing. When `start` is true (1) for at least 1 clock cycle, the accelerator should start performing the erosion step. When the accelerator is finished, it drives the signal `done` true (1) to signal the end of execution and it is ready to start again.

SystemTopTester module

This module is the tester of the full system and it does not describe hardware. It contains sequential Scala, Java, and Chisel code that interacts with the SystemTop module in order to load and dump the memories and to print/show the images in the terminal.

The SystemTopTester module performs the following operation in sequence:

1. **Load the data memory with image data:** During this phase, the tester loads the selected 20-by-20 pixels image in the data memory (DataMemory module). The test images are the same as Assignment 2: blackImage, whiteImage, cellsImage, and borderCellsImage. These images are stored in the file `.../src/test/scala/Images.scala` and you can select

which image to load by changing the following line of code in the SystemTopTester:

```
var image = Images.cellsImage
```

You are free to add your own image to erode in the file `.../src/test/scala/Images.scala`.

The selected image is loaded in the data memory from address 0 to address 399. Each 32 bits entry of the data memory is used to store one pixel. A pixels pixel at coordinates (x, y) is stored at the address $x + 20 * y$ in the DataMemory.

2. **Run the simulation of the accelerator:** During this phase, the tester sets the `start` signal of the accelerator to true (1) for one clock cycle to begin the erosion step processing. This phase continues until the output signal `done` driven by your accelerator becomes true (1) signalling end of execution or if a maximum of 20000 clock cycles have been simulated.

   To simulate more that 20000 clock cycles, change the following line of code in the SystemTopTester:

   ```
   var maxCycles = 20000
   ```

3. **Dump the data memory content:** During this phase, the tester reads the data memory from address 0 to 799 to retrieve the original image and the processed one. The original image is expected to be stored from address 0 to 399 as explained earlier. The processed image is expected to be stored by your program from address 400 to 799. This means that the processed pixel at coordinates (x, y) is stored at the address $x+20*y+400$.

4. **Show the original and processed images:** During this phase, a graphical representation of the original and processed images is printed in the terminal. Use this to ensure that your program did not affect the original image stored in memory and that your accelerator works as expected.

To run the SystemTopTester in order to debug and test your implementation, run the following command in the IntelliJ terminal:

```
sbt "test:runMain SystemTopTester"
```

When you look at the waveforms in the generated file `SystemTop.vcd` in the folder `.../generated/`, the part you are interested in is from when the signal `start` is set to true (1) for one clock cycle to when your accelerator sets the signal `done` to true (1). This is the interval of time when your accelerator is running. Outside this time span, the waveforms show the loading and dumping procedures of the data memory performed by the tester and should be ignored.

DataMemory module

The DataMemory module is also provided to you already implemented and it is exactly the same module used by the CPU in assignment 2. It stores the input

image and the processed images. The DataMemory has a size 65536 words of 32 bits and the accelerator can write or read to this memory. Figure 4 shows the interface of the memory. When the `writeEnable` input is false (0) the memory can be read in the same way as the ProgramMemory. When the `writeEnable` input is true (1), the 32 bits data provided to the `dataWrite` input is written in the specified `address`. Figure 5 shows the timing diagram for a read and for a write operation.

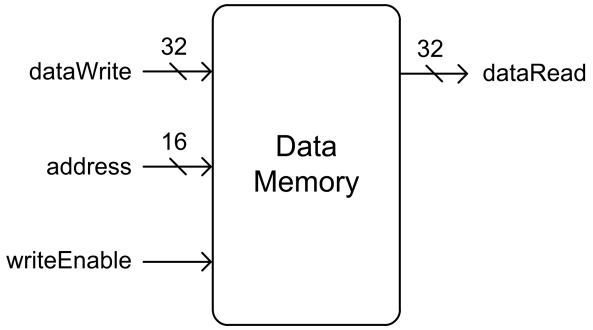

Figure 4: Interface of the DataMemory module.

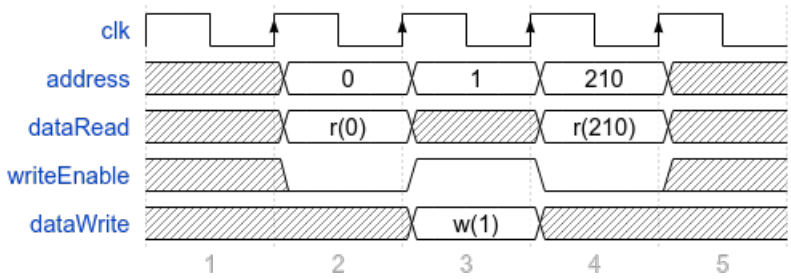

Figure 5: Timing diagram of the DataMemory module for read and write operations. In cycle 2 and 4, reading operations to address 0 and 210 are carried out. In cycle 3, a writing operation to address 1 is carried out.

Accelerator

The Accelerator module needs to be implemented by you to perform the erosion step. Section 3 provides more information with regards to the development and testing of these modules.

3 Development and testing tasks

In this section, we list and describe the tasks you need to perform for the development and testing of your accelerator.

Task 1: Preparation

In this task, you should carefully read the assignment text and have a clear idea of what you need to do. Refresh your knowledge about FSMD from the lectures and the textbook and review how to implement an FSMD in Chisel.

In addition, we recommend that you inspect and run the provided HelloTester. The Hello example loads a 20-by-20 pixel binary image and produces the inverted version of it (negative image). Please note that this is not the same code provided in Assignment 2. The Hello example uses an FSMD to perform the inversion of the input image. The diagram of the FSMD is shown in Figure 6. Explore the code in the files `.../src/main/scala/Hello.scala` and feel free to use it as a starting point for the design and implementation of your FSMD.

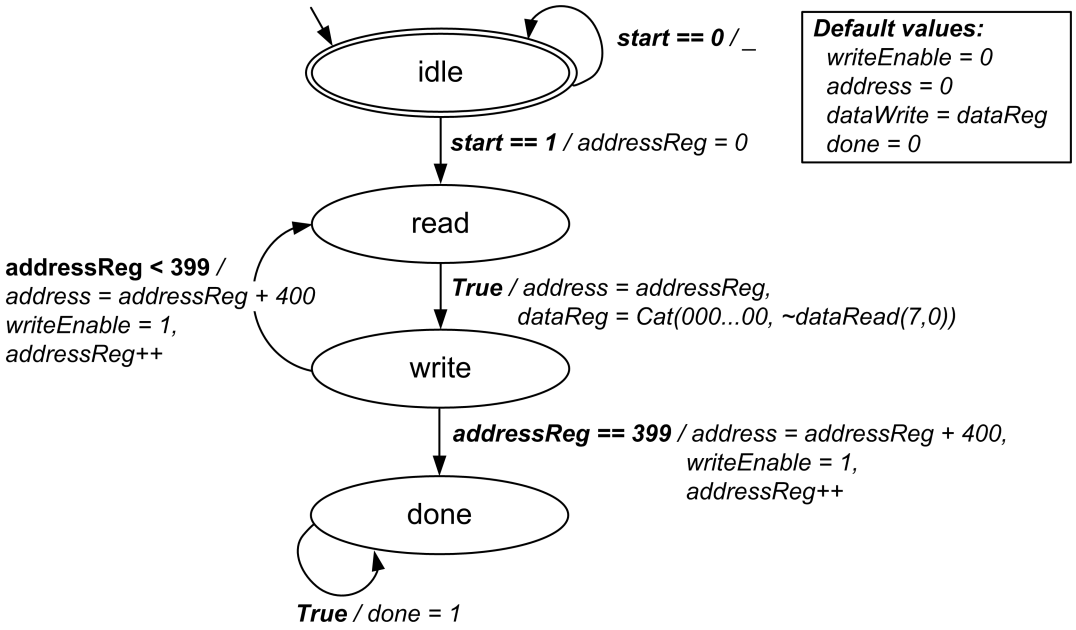

Figure 6: FSMD diagram for the Hello example.

To run the simulation of the Hello module, you need to run in the IntelliJ terminal the command:

```
sbt "test:runMain HelloTester"
```

Task 2: FSMD design

In this task, you should design the FSMD of your hardware accelerator. The purpose of the FSMD is to perform the erosion step. The starting point for the FSMD design is the erosion step algorithm provided as pseudo-code in Listing 1. This is the same as in Assignment 2.

Listing 1: Pseudo-code for the erosion step.

```
for (x from 0 to 19) {
  for (y from 0 to 19) {
    //Processin border pixel
    if (x==0) { out_image(x,y)=0; continue; }
    if (y==0) { out_image(x,y)=0; continue; }
    if (x==19) { out_image(x,y)=0; continue; }
    if (y==19) { out_image(x,y)=0; continue; }
    //Processing inner pixel
    if (in_image(x,y)==0) {
      //Black pixel
      out_image(x,y)=0;
    } else {
      //White pixel, cheking neighboring pixels
      if (in_image(x-1,y)==0 or
          in_image(x+1,y)==0 or
          in_image(x,y-1)==0 or
          in_image(x,y+1)==0) {
        //Erode
        out_image(x,y)=0;
      } else {
        //Do not erode
        out_image(x,y)=255;
      }
    }
  }
}
```

A suggestion for the design process is to use the FSMD/control-flow-graph templates presented in the lectures for the fundamental programming constructs. After having done that, you can then merge the redundant stated. In general, it is good to minimize the number of states since it can lead to a reduction in the execution time.

In addition to designing the FSMD as a state diagram, you should also define what registers you plan to use (e.g., to hold variables) and what additional logic you may want to include to reduce the execution time of the accelerator. In principle, you can directly implement advanced operation in hardware. For example, you can implement a combinational circuit that works on 5 input pixels (loaded from memory and stored in registers) for detecting if a pixel is eroded or not in a single clock cycle. Or, you can store previously loaded pixels in registers in order not to spend time loading from memory again.

Please show your FSMD design to the teacher or the TAs before continuing with the next tasks.

Task 3: FSMD testing (paper and pencil)

Once you have designed your FSMD, make sure that it works as expected by doing some paper and pencil testing. Run through the FSMD by hand using a very small image (e.g., 4-by-4 pixels) and remember to test corner cases. For example, make sure that the FSMD is functional for both inner pixels and border pixels.

Task 4: Implementation in Chisel

Once you are confident that your design works, implement it in Chisel in the Accelerator module. This is the only module you need to implement in this assignment. The interface of the Accelerator module is shown in Figure 7. Please note that the Accelerator module is already wired-up with the DataMemory module in the SystemTop module.

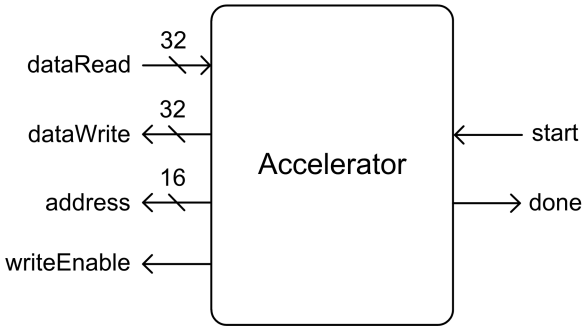

Figure 7: Interface of the Accelerator module.

As previously mentioned, you can use the code for the Hello example listed in the file `.../src/main/scala/Hello.scala` as a starting point for the design and implementation of your FSMD. Also, refer to the FSM and FSMD design examples in the Chisel textbook (see Appendix A).

Task 5: Simulation

At this point, your accelerator is ready and your full system can be tested in simulation. Start the simulation by running the following command:

```
sbt "test:runMain SystemTopTester"
```

When the simulation is finished, you can examine the output in the terminal (created using 'prints' in the tester file) and the waveforms file `SystemTop.vcd` generated in the folder `.../generated/` by opening it with GTKWave.

Task 6: Performance evaluation

Analyze the performance of the erosion step accelerator you have designed by addressing the following points:

- **Execution time and speed-up:** Compare the execution time in term of clock cycles of your implementation with the execution time of the implementation of Assignment 1 and Assignment 2. Calculate the speed-up.

- **Utilization of resources:** Not all the functional units (adder, multiplier, etc.) of your FSMD are used all the time. For example, the multiplier is used only in certain states, while the adder is used in others. Analyze your FSMD and calculate the utilization percentage of each functional unit. Plot the utilization of the functional units over time during execution for the different states. Can some of the functional units be shared in order to increase utilization?

  Hint: At first, examine your state machine and identify how many of each functional unit (adder, multiplier, comparators, etc.) you need. Functional units used in the same state cannot be shared. Once you know how many of each functional unit you need, identify the longest and the most common paths in your state machine. For these paths, shows in a table or in a plot what functional units are used state by state. You can plot the results or show them in a table. Of course you are welcome to examine more/different paths in your state machines if it makes sense (depends on the state machine you built). Depending on how much each unit is used you can calculate the utilization (# states when it is used / (# states when it is used + # states when it is not used)).

- **Hardware size:** Estimate the size of your design? How may bits of registers, adders, multipliers, etc.? Do not consider multiplexers when accounting for the hardware resources.

Task 7: Prepare the report

This is the task where you finalize the report. Use the report requirements available in Section 5.

4 Important notes and hints

Please consider the following notes and hints for this Assignment.

- You can use a number of registers that is proportional to the image side size. This is O(x_size) or O(y_size). It is **not** allowed to have a number of registers that grows proportionally to the total amount of pixels, which is O(#pixels). This enables you to store in registers few image rows, but not the full images. This is due to the fact that it is not realistic to have O(#pixels) registers when the image is 950x950 pixels.
- Using the modulo operation in hardware is not allowed. This is because the circuit for modulo is very big and slow. However, you can use modulo by a power of 2, which is just bit-masking. If you really need to use modulo to implement a very interesting functionality, please discus it with the teacher. The same applies for the division operation.
- Assume the memory space of the output image as dirty. This means you need to write both black and white pixels.

- The amount of clock cycles that your accelerator takes is written before the two images are printed in the IntelliJ terminal. This is:

```
Running cycle: XXX
```

5 Time management, report requirements, and evaluation

As usual, in order for you to check if you are on track or behind, we provide the following plan with reference to the tasks.

Date	**Expected task completion**
8th November	Completed: T1 - Work on: T1, T2, T3
15th November	Completed: T2 - Work on: T3, T4
22nd November	Completed: T3 - Work on: T4, T5
29th November	Completed: T4, T5 - Work on: T6, T7
1st December	Completed: T6, T7

The short report should not be longer than 8 pages (everything included) and a mandatory template for the report is available in DTU-Learn. Similar to Assignment 1 and 2, write "directly-to-the-point", without re-explaining and presenting the basic ideas already presented in this document. Use the report to explain, justify, and validate your design decisions. Do not include the full code in the report, but you can include some code snippets if these are relevant to explain certain aspects of the implementation.

Your work will be evaluated according to the following task-based criteria on a scale of 100 points.

Achievement	**Points**
Task 2: FSMD design	20
Task 3: FSMD testing (paper and pencil)	10
Task 4: Implementation in Chisel	15
Task 5: Simulation (working design)	25
Task 6: Performance evaluation	25
Report	5
Total	100

Appendix A:
Online Chisel documentation and material

This appendix lists useful pointers to Chisel documentation and material online.

The main reference book for learning Chisel is:

- *Martin Schoeberl, Digital Design with Chisel, 2nd edition, 2019* (free, open-access book available here). The book source is available here.

In addition, you can take a look at the following online documentation and material:

- The official Chisel website. Click Chisel3 for documentation.
- A collection of FAQ from another course using Chisel at DTU.
- A collection of examples from another course using Chisel at DTU.
- The Chisel Jupyter Notebooks: an online introduction to the Scala language and Chisel.
- The Chisel Cookbook: large FAQ and introduction to Chisel.