Practical assignment ECA2:
Center of mass design using Haskell and Clash

Introduction

Below is the set of homework exercises for ECA-2 regarding center of mass calculation of images, to be handed in by groups of two students. We assume you successfully completed the tutorial (tutorial.pdf).

deadline for handing in your solutions: 1 February 2026, 23:59, on canvas. **Deliver as group**, submission after the deadline means -5pts subtracted for each day.

Remarks on delivery

- We have provided a number of files, in 3 directories.

  images/convert.exe Windows binary for converting `pgm` files to `png` files

  images/convertToPGM.bat bat file for converting to PGM in Windows.

  images/convertToPNG.bat bat file for converting to PNG in Windows.

  images/lightHouse.png The original image of the light house.

  images/lightHouse.pgm The pgm image for of the light house.

  images/Rocky.png The original image of rocky.

  images/Rocky.pgm The pgm image of rocky.

  ghci/HCenterOfMassXX.hs Main file for `ghci`, extend this file with your code, change the `XX` to your group number in two digits.

  ghci/Image.hs The image, and helper functions for Haskell.

  clashi/Axi.hs Axi4Stream definitions.

  clashi/AxisImageGenerator.hs Axi4Stream image genrator functions.

  clashi/AxisImageReader.hs Axi4Stream image reader from file functions.

  clashi/CCenterOfMassXX.hs Main file for `clashi`, extend this file with your code, change the `XX` to your group number in two digits.

  clashi/ComAxisTb.hs Testbench files that has Axi4Stream testbench input functions, inspect this file to understand the axi input.

  clashi/Common.hs All helper functions for image, both List and Vector variants are here.

  clashi/ImageData.hs Raw image data in a clash vector.

clashi/ReadPGM.hs function to read a .pgm file.

clashi/RunImage.hs full testbench functions.

- Add your names and student numbers at the top of both the `HCenterOfMassXX.hs` and `CCenterOfMassXX.hs` file and rename the `XX` to your group number in two digits, so group 3 becomes 03.
- Add your names and student numbers to the front page of your report.
- In some assignments you are asked to draw a diagram, you may use any drawing tool you like. We used the free tool yEd (https://www.yworks.com/).
- Only deliver code when asked for.
- There is no need to deliver Verilog/VHDL code.
- Use vector functions, reference documentation of Vectors:
  `hackage.haskell.org/package/clash-prelude/docs/Clash-Sized-Vector.html`
- In some assignments you are asked to describe/show the combinational path, here you need to draw this path in a figure.
- If you are asked to **comment on clock cycles**, we are looking for comments describing the **amount of clock cycles for latency and throughput**.
- You can score 15 points in this assignment.
- Deliver 3 files on canvas, make sure that your code file loads in either `ghci` or `clashi`.
  - report (pdf)
  - `CCenterOfMassXX.hs`
  - `HCenterOfMassXX.hs`
- The grading also depends on the style and readability of the code.
- Keep your answers short and concise.

GHCI: Intro and setup

In image processing, the center of mass is used as an estimation of an objects center. In this assignment you will create an implementation of the center of mass calculation in Haskell and Clash. The Haskell files are in the directory `./ghci/` and the Clash files are in the `./clashi/` directory. The `./ghci/` directory contains an `Image.hs` file that contains the image that needs to be processed and the following helper functions:

- `wf`: for writing an image with to a `FilePath`
- `blocks2D`: takes an image (list of lists of grayscale pixels) and chops it into different blocks.
- `unblocks2D`: reverse of `blocks2D`.
- `addBorders`: takes a grayscale value and blocks of an image and adds a border with the grayscale value around every block.
- `changePixelInImage`: takes an image, and produces a new image with a value replaced.
- `changePixelInRow`: takes a row, and produces a new row with a value replaced.
- `image`: contains the original image

The function `wf` writes an image (list of lists of grayscale pixels (Int)) to a `.pgm`-file. The `.pgm`-file can be converted to a `.png` using the `convert` command line tool from ImageMagic[1]. In Linux, this tool is often already include in the distribution. The Windows portable binary is located in the `./images/` directory, including some `.bat` files that use `convert` to convert to PNG or PGM files. In Windows, you can drag your file in your filebrowers on this .bat file, and it will automatically convert the image.

(if you get an error that the library `Data.List.Split` cannot be found you need to install the cabal package split (command line: `cabal v1-install split`)) Use the following steps to get the initial setup working

- Open a terminal
- Go to the `./ghci/` directory
- `$ ghci HCenterOfMassXX.hs`
- In GHCi: `*HCenterOfMassXX> wf "../images/output.pgm" image`
- Go to the `./image/ directory`
- Use the `convert` tool to convert the .pgm to png (`convert output.pgm output.png`)
- View the png (some image viewers can also show .pgm files, so conversion is not necessary)

[1] https://www.imagemagick.org

		col (c)			
		1	2	3	4
row (r)	1	0	0	0	1
	2	0	1	0	1
	3	1	1	1	1
	4	0	0	1	1

m_x	rm_x
1	1
2	4
4	12
2	8
$\sum m_x = 9$	$\sum rm_x = 25$

m_y	1	2	2	4	$\sum m_y = 9$
cm_y	1	4	6	16	$\sum cm_y = 27$

$y = \frac{27}{9} = 3 \quad x = \frac{25}{9} = 2$

Table 1: Center of mass calculation on a small image (yellow table represents an example image)

1 GHCI: From grayscale to black-and-white

If you have successfully completed the steps as described above, you may continue with the assignments. In a grayscale image the center of mass is often in the center, that is why we first convert the image from grayscale to black-and-white.

Assignment 1 (0.25 pts):

Write a function `thresholdIm` that takes a threshold and an image, and produces a black-and-white image. An image is a list of lists containing grayscale values, while a black-and-white image is a list of lists containing a 0 for black and 1 for white. Take the value `threshold` as a threshold, it is defined as 128 in the file `Common.hs`. Use the `wf` function from `Image.hs` to generate a picture for your report.

2 GHCI: Center of mass of entire picture

One way of calculating the center of mass is shown in the Table 1 where m_x and m_y are the sums of the weights of the rows and columns in the image. All the m_x and m_y values are multiplied with the row r or column c that they are in. In order to get the row index the $\sum rm_x$ is divided by $\sum m_x$, and in order to get the column index $\sum cm_y$ is divided by $\sum m_y$.
Use higher-order functions in this assignment. Be careful with the indices (0-based, and confusion about rows and columns is likely). You can use the `div` function to divide Ints.

Assignment 2 (2 pts):

Write a function `comRows` which takes an image and calculates the center of mass horizontally (produces a row index which is the center of mass). Write a function `com` which takes an image and calculates the center of mass both horizontally and vertically (hint: use `transpose`). Write a function `imageWithCom` which takes a color (Int) and an image, and produces a new image with the pixel at the center of mass changed into a different color (hint: use `changePixelInImage`). Test it with the black-and-white image. You can use the numeric value 2 to color the pixel that indicates the center of mass. The resulting image will then consist of three different "colors" namely, black, grey and white. Show the resulting picture in your report and the coordinates of the center of mass.

3 GHCI: Center of mass of parts of the image

The function `blocks2D` subdivides the image into blocks with a specific width and height. For these assignments you must use the width and height of 8. The function `addBorders` adds a border with a selected color to the blocks generated from the `blocks2D` function. You should use the number 2 for the color of the borders (this is the same as the pixel color which shows the center of mass). The `unblocks2D` function is a reverse of the `blocks2D` and constructs an image from the height of the resulting image and the blocks. Note: if every pixel in the sub image is the same color, then use the center pixel as center of mass.

Assignment 3 (0.5 pts):

Implement a function `comParts` which takes an image as argument, and calculates the center of masses of every sub block. The output should be an image with all the center of mass pixels changed to white (color value: 2). Use the `blocks2D` function to create those sub parts with a width and height of 8. Implement a function `comPartsWB` that has the same behaviour as `comParts` but puts borders around the sub images (`addBorders 2`). Show the resulting picture in your report.

Clashi: Intro and setup

To transform the Haskell code to Clash-code the lists and numbers need to be bounded by using `Vectors` and bounded number types. A pixel can be presented as a 8-bit `Unsigned`, this is defined in the `Common.hs` file. You can use the function `fromIntegral` to go from Integral numbers (like `Int`, `Signed`) to a number of the `Num` type class. In this part of the assignment you have to be careful to not make mistakes in indices from rows, columns, x and y. Unless asked for, there is no need to generate Verilog/VHDL.

4 Clashi: changing a pixel in an image

If you used higher order functions in the previous exercises then transforming the code from Haskell to Clash should be relatively easy. In Clash the `replace` function can be used to replace an element in a vector. If we want to replace a certain pixel with another to show the center of mass we need a function that changes a pixel in an image. The import for `ImageData` is commented, (line 9 of `CCenterOfMassXX.hs`. In order to acces the hardcoded image, uncomment the line and imiport `ImageData`. The loading time of Clash is a bit longer, therefore, you can comment this import after finishing this questions.

Assignment 4 (0.25 pts):

Implement `changePixelInImage` function using the `replace` function. Copy the `thresholdIm`, `comRows`, `com` and `imageWithCom` from the Haskell specification and turn it into a Clash specification. Show the result of the following expression:

```
*CCenterOfMassXX> com $ thresholdIm threshold image
```

Note: If you experience overflow of values you may change need to change the types in your functions, you can use `fromIntegral` to convert something to a `Num` typeclass.

5 Clashi: Center of mass of parts of the image

The functions `wf`, `blocks2D`, `addBorders`, `unblocks2D` are also available in Clash (`Common.hs`). In Haskell, both the `blocks2D` and `unblocks2D` functions required an `Int` for the width of the window or height of the new image. However, in Clash, these Ints need to be converted to SNats[2]. SNats are represented in Clash with a `d` followed by a number, `d10` would be the natural number 10.

Assignment 5 (2 pts):

Copy the `comParts` and `comPartsWB` from the Haskell specification and turn in into a Clash specification. Show the resulting pictures in your report (use `wf` from `Common.hs` to write a pgm file). NOTE: use the thresholded image as input image.

[2]Singleton value for a type-level natural number

6 Clashi: Serial Axi4Stream of an image

Usualy pictures are not stored in FPGA logic, but are streamed in from other parts on a chip. This streaming is often over a bus that adheres to a certain protocol. AXI is a well known bus, note that there are multiple protocols AXI4-Streaming, AXI4-Full and AXI4-Lite. The latter two are control/status protocols, while AXI4-Streaming is for data streaming. AXI4-Streaming contains a master (transmitter) and slave (receiver). Only a few signals in AXI4-Streaming are mandatory to implement e.g. `tdata`, `tvalid`, `tready`, most others are design-specific. For our CoM accelerator, the required signals are specified in Table 2 and Figure 1 is for visual understanding. The widths are as follows: `tdata` is of Signed 32 and tkeep of Unsigned 4. Although the data is a single word and does not need a keep, the DMA sees the data as 4 bytes, thus requires 4 keep bits. A more elaborate tutorial can be found in the Appendix A.

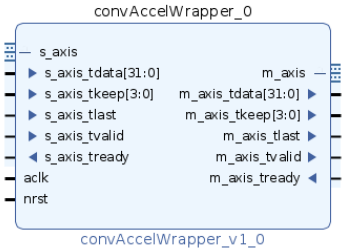

Figure 1: Accelerator IP.

Signal	Source	Description
ACLK	Clock	Global AXI4-Stream Clock
ARESETn	Reset	Global reset is active-LOW
TDATA[(8n-1):0]	Master	Data of the packet
TLAST	Master	Last packet of burst flag
TKEEP[(n-1):0]	Master	Which bytes are part of packet
TVALID	Master	Master is driven a valid transfer
TREADY	Slave	Slave can accept transfer in current cycle

Table 2: Accelerator AXI4-Stream signals.

Assignment 6 (6 pts):

Implement `axisComSer` function that operates on the AXI4-Streaming protocol. Make sure that the function has a mealy compatible type signature `s -> i -> (s, o)`. Note that `axisComSer` is both a slave and a master, make sure that it adheres to the AXI4-Streaming protocol (Appendix A). It receives data as slave as an `Axi4Stream`, but it also sends the computed coordinates as an `Axi4Stream`, so it is also a master. The AXI4-Streaming input of `tdata`, `tlast`,`tkeep` uses the `Axi4Stream a n` data type, noticed that a `Maybe` is wrapped around it for the `tvalid` flag. You can use the `spsAxisComSerTb` function to simulate the `axisComSer` and print the internal state for every clock cycle. As simulation input the `mAxisComSerInp` from the file `ComAxisTb.hs` is used. This input is a list of tuples that consist of the following:

- `Maybe Axi4Stream` from the master that holds the image data (1 pixel per clock cycle).
- `Bool` that indicates wheter the slave is ready to receive the calucalated coordinates).

The `Maybe Axi4Stream` contains data, but also a `tLast` flag that indicates the end of a subimage. In this assignment you are allowed to compute the Center of Mass in one clock cycle. This means that data is streamed in, once the last pixel of subimage is received (`tLast` flag), you compute the center of mass and place it on the output (if the slave is ready to receive). Note: The image streamed in is a grayscale image, this means that you need to apply a threshold first before computing the center of mass.

Implement mAxisComSer to put the axisComSer in a mealy machine. Make sure to not change the type of mAxisComSer, since it will be used in other simulation functions. simMAxisComSerTbPrint simulates the mealy machine with the same input (`mAxisComSerInp`). Run simMAxisComSerTbReport and put the output in your report. Load the file `RunImage.hs` and change the line: `import CCenterOfMassXX` to your group number and execute the following command:
`runImage SER d1 d8 lightHouseImgPath`
This will generate an output image in the `./images/` directory called `LightHouseSerOut.pgm`. Place this image in your report.

7 Clashi: Serial Axi4Stream synthesis

synthAxisComSer has a synthesis annotation, this means that the clash-compiler will generate Verilog/VHDL for this specific function.

Assignment 7 (0.5 pts):

Generate Verilog/VHDL, show the RTL schematic of synthAxisComSer in your report. Give an overview of the resources used (Stats viewer in EDAcation plugin, or Flow summary in Quartus) and comment on the number of input/output pins, registers, and DSPs.

8 Clashi: Parallel Axi4Stream of an image

In the previous assignment, the data inside the `Axi4Stream` was a `Signed 32` and contained a single pixel. This is a very inefficent way of streaming in data. An AXIS bus can be in different widths, in this assignment we will store a pixel inside a `Unsigned 8` and stream a Vector of length 16 every clock cycle. This means that the type of this parallel Axi4Stream is as follows:

```
Maybe (Axi4Stream (Vec 16 (Unsigned 8)) (BitVector 16))
```

The test data for this assignment can also be found in `ComAxisTb.hs` and is defined as mAxisComParInp.

Assignment 8 (3 pts):

Implement axisComPar that can handle a parallel Axi4Stream. Implement mAxisComPar that wraps axisComPar in a mealy machine. As with the previous assignment, you are allowed to compute the center of mass in one clock cycle. Run simMAxisComParTbReport and place the output in your report. Load the file `RunImage.hs`, and execute the following command:

`runImage PAR d16 d8 lightHouseImgPath`
This will generate an output image in the `./images/` directory called `LightHouseSerOut.pgm`. Note: The image streamed in is a grayscale image, this means that you need to apply a threshold first before computing the center of mass. Place this image in your report.

9 Clashi: Parallel Axi4Stream synthesis

`synthAxisComPar` has a synthesis annotation, this means that the clash-compiler will generate Verilog/VHDL for this specific function.

Assignment 9 (0.5 pts):

Generate Verilog/VHDL, show the RTL schematic in your report. Give an overview of the resources used (Stats viewer in EDAcation plugin, or Flow summary in Quartus) and comment on the number of input/output pins, registers, and DSPs.

A AXI4-Streaming Protocol Description

AXI4-Streaming protocol makes use of a slave and master channel. The local slave channel (convolution accelerator) receives data from an external master (stream generator) while the local master (convolution accelerator) channel transmits data to an external slave (stream verifier). See Figure 2 for an overview of the relevant signals for the convolution assignment, the clock and reset are left out for simplicity.

The `clock`, `reset`, `tdata`, `tvalid` and `tready` signals are always mandatory, while `tkeep` and `tlast` are added to be compatible with the AXI-DMA IP, which can be used in combination with our accelerator. The `tvalid` and `tready` are used for the handshake. The `tkeep` and `tlast` are control signals, `tdata` contains the actual content in the stream. It contains the kernel or subimage item on the slave and the convolved feature on the master.

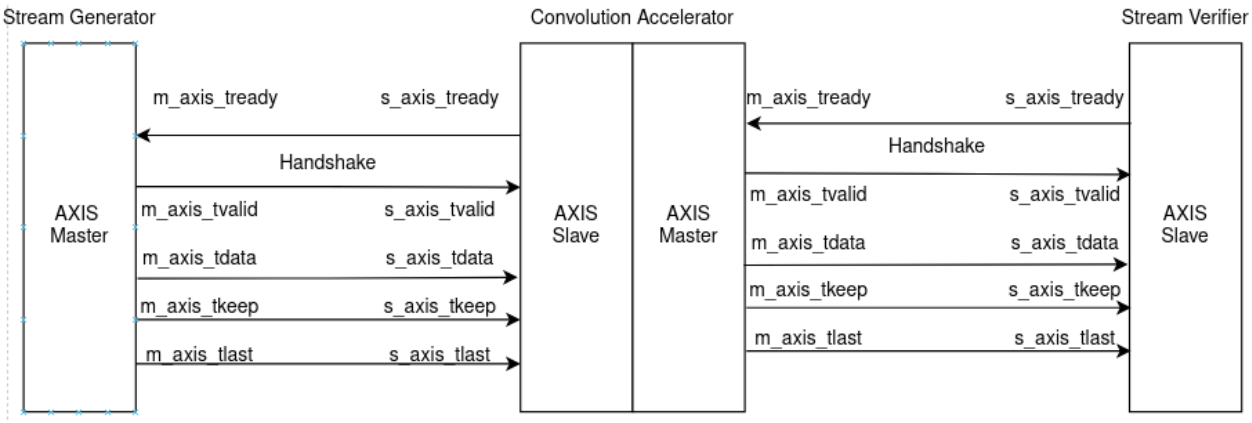

Figure 2: Test setup convolution accelerator.

A.1 Slave receive

The handshake determines when information is passed across the interface. In order for a transfer to occur, both `tready` and `tvalid` must be asserted. Either `tvalid` or `tready` can be asserted first or both can be asserted in the same clock cycle. The local slave can request data by putting its `tready` high. The external master will notice this and put the `tvalid` high, this is a *tready before tvalid handshake*. It is also possible that the master already has the `tvalid` asserted, this is called a *tvalid before tready handshake*. If this is the case it cannot deassert it.

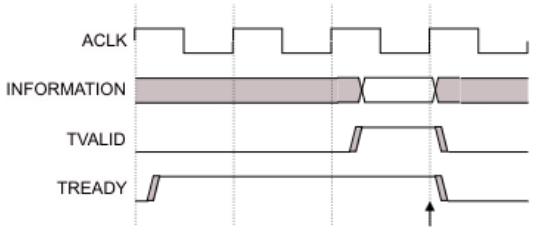

Figure 2-2 TREADY before TVALID handshake

Figure 3: *tready before tvalid hanshake* [?]

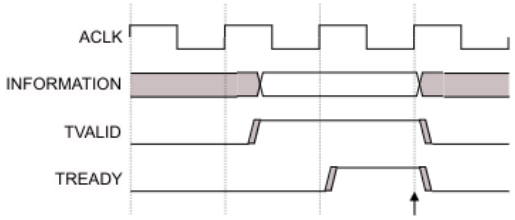

Figure 2-1 TVALID before TREADY handshake

Figure 4: *tvalid before tready handshake* [?]

To simplify the code we can define a local signal which is high in case the `tvalid` and `tready` are high.

```
handshake = case (s_axis, s_axis_tready) of
  (Just _, True) -> True -- Incoming packet is valid & local slave is ready
  _              -> False
```

The local slave can now read out `tdata`, `tkeep` and `tlast`. In case the data is valid (`Maybe` is used as valid flag), each field can be accessed as shown below, notice that in case it is not valid default values must be inserted.

```
(s_axis_tdata, s_axis_tlast, s_axis_tkeep) = case s_axis of
  (Just x) -> (tData x, tLast x, tKeep x) -- extract data from axi record
  _        -> (0, False, 0) -- default data
```

The `tready` should be low in case the external slave cannot receive any data or if the local buffer is full. This only happens if we are in the `CONV` state and the kernel and image registers are full. Otherwise, the incoming data from the stream generator, will not be retransmitted next clock cycle and it will be lost forever, as it is not stored in the local kernel or subimage buffer during the `CONV` state. As we need to keep the current buffer, because the state is still `CONV` next clock cycle.

```
s_axis_tready = not (state == CONV && not m_axis_tready)
```

A.2 Master transmit

The local master should assert the `tvalid` in the `CONV` state as the convolved feature will be available on tdata. It cannot leave the `CONV` state until the tready from the external slave is asserted, which means that it has completed the handshake and holds our transmitted convolved feature.

```
m_axis_tvalid
  | state == CONV = True -- Convolutional feature available
  | otherwise = False -- Convolutional feature not available
```

The `tkeep` and `tlast` signals can be forwarded from the local slave to the local master without any changes, as our block does not support any control logic for these signals. Do not hard code them to zero as another IP in the stream could be using them and would thus discard all the data or introduce a deadlock.

```
m_axis | m_axis_tvalid = Just Axi4Stream
          {tData = cf, tLast = s_axis_tlast, tKeep = s_axis_tkeep}
       | otherwise = Nothing
```