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

November 29, 2018

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: 2019-02-02, on canvas.

Remarks on delivery

- We have provided 6 files, in 3 directories.
  - clash/CCenterOfMass.hs – Main file for `clashi`, extend this file with your code
  - clash/Image.hs – Image file containing the image and helper functions for CλaSH
  - ghc/HCenterOfMass.hs – Main file for `ghci`, extend this file with your code
  - ghc/Image.hs – Image file containing the image, and helper functions for Haskell
  - images/original.png – The original image
  - images/convert.exe – Windows binary for converting `pgm` files to `png` files
- Add your names and student numbers at the top of both the `HCenterOfMass.hs` and `CCenterOfMass.hs` file.
- 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/)
- For all assignments you are asked to deliver CλaSH code. Please include your CλaSH code for that specific assignment in a text box in your report.
- There is no need to deliver VHDL code.
- Use vector functions can be found in the 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 about clock cycles, we are looking for comments describing the amount of clock cycles for latency and throughput.
- You can score 10 points in this assignment. This is 1 points of the final grade.
- Deliver your report (pdf), CλaSH file and the Haskell file.
- Report name: `ECA2_com_lastname1_lastname2.pdf`.
- The grading also depends on the style and readability of the code.

Preliminary Remarks on Haskell and CλaSH

- A filename of a Haskell/CλaSH program should be of the form `<Filename>.hs`, starting with a capital letter.
- You can transform Haskell to CλaSH by the following steps:
  - add the line

    import `Clash.Prelude`

    as the second line of your file. Among others, this redefines many standard list functions in Haskell towards corresponding vector functions in CλaSH. For example, in Haskell functions such as *take* and *map* work for lists, whereas in CλaSH they work for vectors. If you need such a function for lists, use *Data.List.take*, *Data.List.map*, etc. However, this only works in the `clash` interpreter, and not in a program file.
  - define the hardware types needed, using `Signed`, `Unsigned`, `Vec`, etc.
  - Instructions how to install CλaSH on your own system can be found on `clash-lang.org`.
- Finally, to generate VHDL code using CλaSH, you need to define the variable *topEntity* and make sure that its type is not polymorphic and fully determined. Note that Haskell (and thus CλaSH also) can derive the type of an expression, to be checked in Haskell/CλaSH with:

  ```
  :t <expression>
  ```

  VHDL code is generated by CλaSH using the command[1]

  ```
  :vhdl
  ```

  This will put the resulting VHDL code in a subdirectory `vhdl/<Filename>`. We assume that you have access to *Quartus* for synthesizing the VHDL generated by CλaSH.

[1] Don't bother about possible "`Can't make testbench`" errors, they are not relevant in our context.

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 CλaSH. The Haskell files are in the directory (`./ghci/`) and the CλaSH files are in the (`./clashi/`) directory. Both directories contain a file called (`Image.hs`). This file contains the image that needs to be processed and helper functions. The helper functions are:

- `wf`: for writing an image with a `filename` to a `path`.
- `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 `filename` and `path` are pre-defined to place the output image in the (`./image/`) directory. 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[2]. In Linux, this tool is often already include in the distribution. The windows portable binary is located in the `./image/` directory.

(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 install split` Use the following steps to get the initial setup working

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

[2] 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	$\sum cm_y = 27$ $\quad y = \frac{27}{9} = 3$ $\quad x = \frac{25}{9} = 2$

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 (0.5 pts):

Write a function `threshold` that takes a threshold `t` 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 125 as threshold value. Show the threshold function in your report, and include the generated picture.

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 (3 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`). Show the implementation of these functions in your report. 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 image 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.

Assignment 3 (1 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 gray (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 only puts borders around the output image (`addBorders 2`). Show the implementation of both functions in your report and the resulting pictures.

Clashi: Intro and setup

To transform the Haskell code to CλaSH-code the lists and numbers need to be bounded by using `Vectors` and bounded number types. A pixel can be presented as a 16 bit `Unsigned` as used in the `Image.hs` file. You can use the function `fromIntegral` to go from Integral numbers (like Int) 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 VHDL.

4 Clashi: changing a pixel in an image

In CλaSH 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.

Assignment 4 (1 pts):

Implement `changePixelInImage` function using the `replace` function and show it in your report.

5 Clashi: Center of mass of entire image

If you used higher order functions in the previous exercises then transforming the code from Haskell to CλaSH should be relatively easy.

Assignment 5 (1 pts):

Copy the `threshold`, `comRows`, `com` and `imageWithCom` from the Haskell specification and turn it into a CλaSH specification. Show your function implementations and the result of the following expression:

```
*CCenterOfMass> com $ threshold 125 image
```

6 Clashi: Center of mass of parts of the image

The functions `blocks2D`, `addBorders`, `unblocks2D` are also available in CλaSH (`./clashi/Image.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 CλaSH, these Ints need to be converted to SNats[3]. SNats are represented in CλaSH with a `d` followed by a number, d10 would be the natural number 10.

Assignment 6 (1 pts):

Copy the `comParts` and `comPartsWB` from the Haskell specification and turn in into a CλaSH specification. Show your function implementation and the resulting pictures: with and without borders.

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

7 Clashi: time vs area

Calculating the center off mass off a 128x128 picture in blocks of 8 requires a large amount of resources. The first time-area trade-off is quite obvious; perform all calculations for 1 block in 1 clock cycle. We can simulate that using the following definition of the `topEntity` function:

```
topEntity :: Vec 8 (Vec 8 Pixel) -> Vec 8 (Vec 8 Pixel)
topEntity = (imageWithCom 2)
```

The `imageWithCom` function calculates the center of mass of both the rows and columns. Simulate the topEntity with the `simulate` function:
`simulate (topEntity <$>) $ toList $ blocks2D d8 im`
where `im` is the black-and-white image.
The next time-area trade-off step could be to split the calculations of the rows and columns over two clock cycles.

Assignment 7 (2.5 pts):

Implement a `mealy` machine that, depending on the state, calculates the center of mass of the rows or columns. Show your implementation in the report, include the simulation output. There is no need to create a picture, you can just show the simulation output as a list of (Maybe tuples) that contain the indices of the center of mass of the row and column. Synthesize your `topEntity`, show (parts) of the RTL schematic, and comment on resource usage and combinational path.