Universidade de Coimbra
Faculdade de Ciências e Tecnologias
Departamento de Engenharia Eletrotécnica e de Computadores

Digital Systems Project
Real Time Sobel Edge Dectector on FPGA Board

Bruno Gonçalves Oliveira, 2014228470

Ricardo Barreto, 2014191164

9 de Julho de 2018

1 Abstract

Edges are one of the most important features that can be extracted from image processing, because obtaining them allows to discover elements of a scene, observe disparities between frames and understand their structure. There is a a lot of research and methods associated, but in this project was implemented a real time Sobel operator using Altera DE2 and a TRDB-D5M Terasic camera.

For that we used the Terasic drivers for the camera. So, first we studied the hardware manual so that every block of the camera design became understandable, allowing that necessary changes could be made during the realisation of the project.

Every step of the implementation phase was strictly planned by the developers in order to accomplish better results and organisation. After the executions the results were analysed to facilitate the localisation of the faults.

The final outcome produce the expected behaviour of the Sobel operator, with the advantage of being in real time.

Conteúdo

2 Introduction

The purpose of this project is to develop a real time Sobel Operator. The material necessary for the implementation was a Altera DE2 Cyclone II FPGA board used for image storing and processing, a TRDB-D5M CMOS Sensor for image acquisition and a VGA monitor to display the filtered image in real time.

Choosing this project was conditioned by the algorithms learned in computer vision during this semester. In this manner, we could conciliate two courses in one time.

Sobel Operator is used in computer vision as a edge detector where it creates an image by emphasising his edges. At each pixel we need to get the neighbour pixels to perform a convolution with the two masks of size 3x3. Then, according do the magnitude of the calculated values we decide if the pixel belongs to an edge or not. This step addresses us to a problem since we do not have libraries that calculate the root square of a value.

In this document, we started to declare the system's functionalities in terms of user's perspective, followed by the explanation of the High Level Architecture with the particular interest in the interactions between the main modules that make up the system.

After that, we address our discussion through the implemented Modules as well as the modified Modules from the camera demo.

In section 6, we exhibit the obtained results and their observations. We see a lot of potential in this work, so we suggest possible expansions to our work.

At last, we make considerations about the general process of implementation during this project.

3 System Operation

The user is able to choose the output of the VGA Monitor by pressing the keys listed in table 1.

Function	**Key**
Stop Camera	Key[2]
Stream Mode	Key[3]
Grey Scale	SW[0]
Sobel Mode	SW[1]

Tabela 1: User interface keys.

When the switches **SW[0]** and **SW[1]** are at the state zero, the VGA Monitor will display the direct image from the Camera.

4 System Architecture

In order to describe the solution to our project, we will first expose the architecture design with a high level of abstraction containing the principal modules and the data flow of the system.

In the figure 1, we can observe the main modules of the system. The modules represented in green are the ones that were created/modified by the developer team. The arrows correspond to the paths that the data can take.

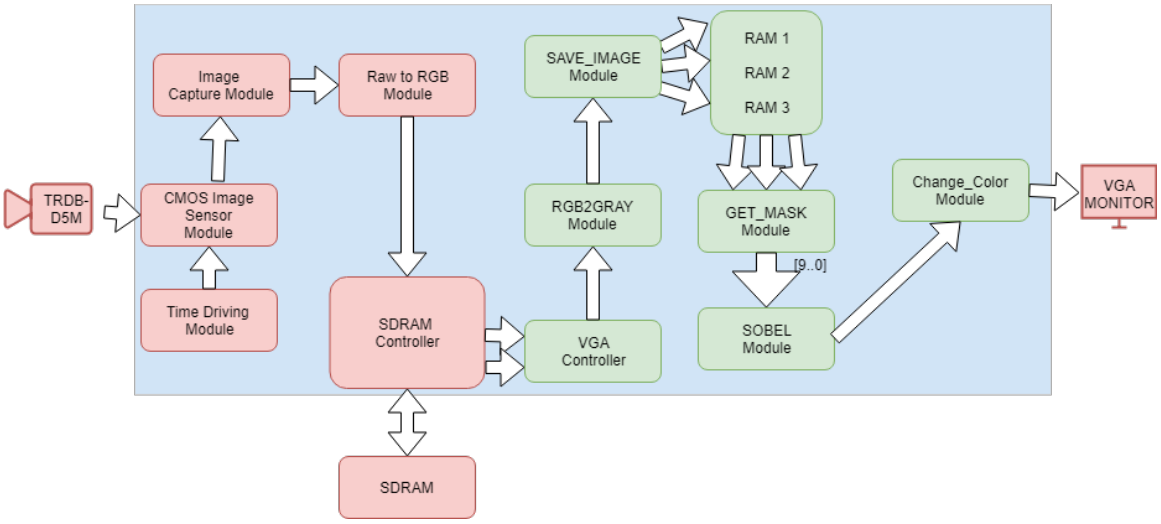


Figura 1: High level Architecture.

The components CMOS Image Sensor, Time Driving, Image Capture, RawToRGB and SDRAM Controller are provided in the Altera demo. Cause of that, we can find more documentation about this components in the *TRDB-D5M Hardware Specification* [1].

In the next part of this section we will explain how the functional Modules that compose the system are integrated and their role.

4.1 CMOS Image Sensor

Captures the raw data and transfers that to the system.

4.2 Timing Driving Module

Generate the several clocks that some system modules work.

4.3 RAW to RGB conversion

Pixels are outputted in a Bayer pattern format consisting of four colours Green1, Green2, Red, and Blue. So, before we write the values at the SDRAM, is necessary to converting them to a format that is compatible with the Module who makes the access. In this case, the Module that reads the values from the SDRAM is the VGA Controller and he works with a RGB format.

4.4 SDRAM Controller

When the camera information is in the appropriate format the values are ready to be stored in the SDRAM memory. At this part, we have that the modules of the camera raw data, SDRAM and the VGA operates on different clocks and they have a high dependency. This problem is overcome with the use of FIFO's for reads and writes. More specifically the RGB data are stored in two banks, one for the red information and halve of the green, and the another bank for the other halve of the green and for the blue. The operations of the SDRAM are governed by a Controller.

4.5 VGA Controller

The data from the banks is sent to this block that has the responsibility of making the interface with the VGA Monitor. Basically, this controls the moments of the data display. From this block, we see when the sweep is in region of active image and his position. The parameters of the VGA are the default ones, that is , 640x480.

4.6 RGB to Gray

This module has the function of converting the RGB data format to grey scale because the Sobel Operator is only applicable in that. After this conversion the data is ready to be stored.

4.7 SAVE IMAGE

Enables the specific RAM that should store the grey scale color, and keeps a address pointer to that RAM.

4.8 Three external RAM's

Each RAM stores one line of the image frame and the lines stored are constantly updated according to the position of the VGA sweep.

4.9 Get Mask

This block has the important responsibility of receiving the values stored in the RAM memories and ordinate them according to the mask for that specified time. Then, the values are delivered to the calculations. Note, that the order of the values are important to make the filtering.

4.10 Sobel

When this module receives the elements of the mask, it performs the respective calculations according to the formula. The data are now ready to be sent to the monitor.

4.11 Change Color

To choose the output of the VGA Monitor between normal mode, grey scale and Sobel filter according to the switches.

5 Comparative Analysis with other Architectures

A system Design is a very complex task, therefore this process becomes iterative. In other words, we can come across with a part of a solution that does not do the expected behaviour or that there is more efficient ways of implementation, being necessary to go back in the process.

Our first approach, consisted to apply the filter to only one square in the VGA Monitor. We implemented the square boundaries and we store the data in a RAM with sufficient memory. But, when we implemented the filter, there was a need to store the new values in a new RAM memory so the values could be sent to the VGA. This became a problem, because the FPGA board that we used didn't have enough memory to save to frames at one time.

Our second approach, was based on safeguarding some previous lines prior to the data sweeping. And then, we applied the filter to the values stored in the RAM, using a fastest clock to implement all calculus, with a bit of delay to the current value. This approach showed up some results, but we saw a lot of interference. Very likely there was a "clock problem", however after a few attempts we could not come up with a solution, and this approach was discarded.

That way, we arrived at the architecture presented in figure 1.

6 Implementation

In this section is discuss the technical details about developed system modules. There is no reference to some of the modules presented in system High Level Architecture, because they aren't created/modified in the current project. Their information can be consulted in *Hardware Specification* [1].

6.1 VGA Controller

This Module has been modified to also output the X,Y positions of the sweep. For that, we initialises the X and Y pointers with zero, and then we increment each one, when we have an horizontal or vertical sweep.

6.2 RGB to Gray

Here we convert the RGB pattern grey scale. There is a lot of ways to do that, but we decided to do a simple and efficient conversion doing the average of the 3 values.

$$Output = \frac{R + G + B}{3} \tag{1}$$

6.3 Save Image

This block controls the 3 RAM memories. It has as input the X, Y, and the GREY value. With the X and Y positions we can control everything. At start (X = 0 and Y = 0) we initialises the variables at zero. Then, we will have to control the address to write and which memory to write. For the address, each time the Y is changed (row changed), the address start at zero and increment at every clock cycle. The variable that controls the RAM to write, goes from 0 to 2 (3 RAM's), also increments when the Y is changed, but when it is equal to 2, it start over from 0. So it overwrite the oldest RAM. This behaviour is described in the fine state machine below 2.

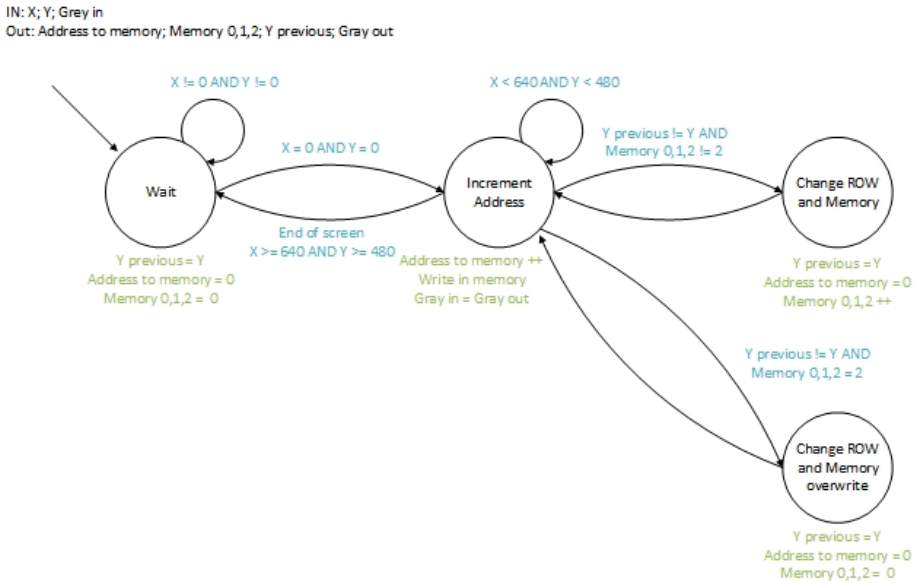


Figura 2: Fine state machine of the Save Image module

As we can see the module Save Image as another output, the $save_addr\_mem\_read[9..0]$. This output is the $addresstomemory - 1$ to read the previous value.

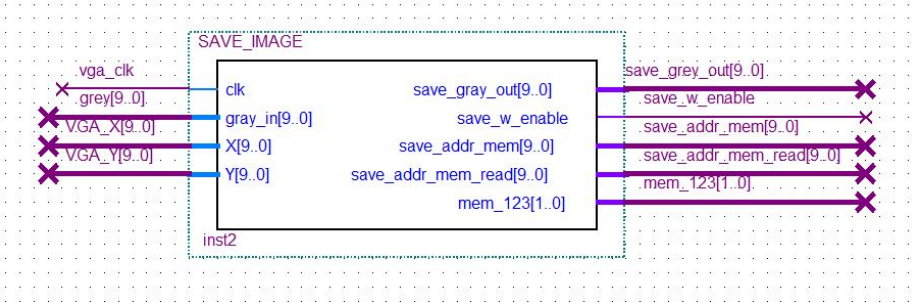


Figura 3: Save Image Module

6.4 Three RAM's

This project uses three RAM's. Whit this configuration we save a line of the screen in one memory, having the total of three lines. Then we can read from the three memories one value, getting three values in one clock cycle.

6.5 Get Mask

This module has some restrictions, because of the amount of conditions it could have. As we work with only 3 memories, we overwrite the oldest when changing the row, so it becomes the newest. And in this block we, need to take in consideration the memory that is being written at the moment, to map the corresponding values to the Sobel matrix.

To do that we get the values from the RAM and save in 9 different variables, V0 to V8. At each clock cycle we do two things, we save the new pixel received (in the Save Image module), and read from the 3 memories, 3 different pixels, corresponding to the same column, and to the rows r, $r-1$, and $r-2$.

In the figure 4 we can see what happens when we are writing in the different memories, the values we get are mapped to the variables V0-V9.

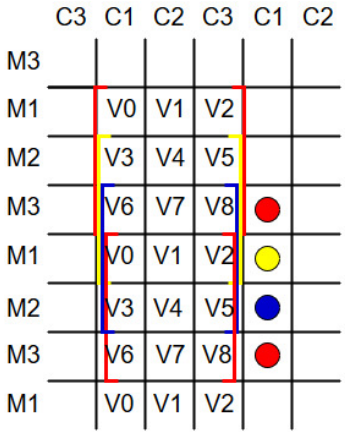


Figura 4: Get Mask according to the writing memory

The Red circle represents the writing position, so when we are writing there, in the memory 3 (M3), we would get the values V2, V5, and V8. At this point we have the mask represented bellow (the previous two columns, V0 V3 V6, V1 V4 V7, comes from the previous clock cycles).

$$Mask = \begin{bmatrix} V0 & V1 & V2 \\ V3 & V4 & V5 \\ V6 & V7 & V8 \end{bmatrix} \tag{2}$$

If we are writing in the yellow circle, in the memory 1 (M1), we would get the values V5, V8, and

V2. At this point we have the mas represented bellow.

$$Mask = \begin{bmatrix} V3 & V4 & V5 \\ V6 & V7 & V8 \\ V0 & V1 & V2 \end{bmatrix} \tag{3}$$

If we are writing in the blue circle, in the memory 2 (M2), we would get the values V8, V2, and V5. At this point we have the mas represented bellow.

$$Mask = \begin{bmatrix} V6 & V7 & V8 \\ V0 & V1 & V2 \\ V3 & V4 & V5 \end{bmatrix} \tag{4}$$

At this point this sequence repeats every 3 rows.

Then we have to deal with the columns. When the column change, we also save different values, and at that point, we have a different matrix to map. In the figure 5, we assume that we are writing in the memory 3 (M3), and the columns change.

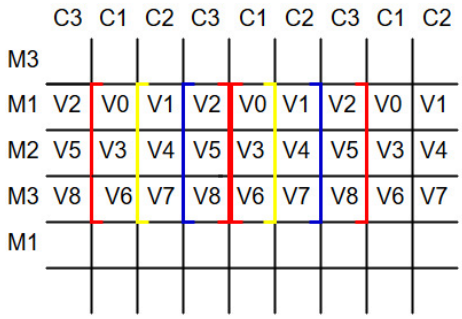


Figura 5: Get Mask according to the column position

Assuming that we are in the column 3 (C3) we have the following values:

$$Mask = \begin{bmatrix} V0 & V1 & V2 \\ V3 & V4 & V5 \\ V6 & V7 & V8 \end{bmatrix} \tag{5}$$

Then if we are in column 1 (C1) we ave the following values:

$$Mask = \begin{bmatrix} V1 & V2 & V0 \\ V4 & V5 & V3 \\ V7 & V8 & V6 \end{bmatrix} \tag{6}$$

And if we are in column 2 (C2) we have the following values:

$$Mask = \begin{bmatrix} V2 & V0 & V1 \\ V5 & V3 & V4 \\ V8 & V6 & V7 \end{bmatrix} \tag{7}$$

So when we change the column we change the order of the columns, and when we change a row, we change the rows order. With this approach we can efficient map to the sobel module the received values, according to the current memory and column.

Also we are sending to the screen a pixel with a delay of one row and 2 columns behind of the real VGA sweep.

After this process of getting the right values to construct the right matrix, we send the values to the Sobel Module.

6.6 Sobel

This module gets the values from the Get Mask module, and applies the following accounts. The values obtained are represented bellow.

$$A = \begin{bmatrix} A0 & A1 & A2 \\ A3 & 1 & A5 \\ A6 & A7 & A8 \end{bmatrix} \tag{8}$$

Then we use the sobel filters bellow and multiply each one with A.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \qquad G_y = \begin{bmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A \tag{9}$$

Then, because of the complexity of doing a square root in digital systems, we simplified the following equation.

$$G = \sqrt{G_x^2 + G_y^2} \Leftrightarrow G = |G_x| + |G_y| \tag{10}$$

With this simplification we only have to convert to positive if the number is negative. Then with the G value, we only have to compare with a threshold.

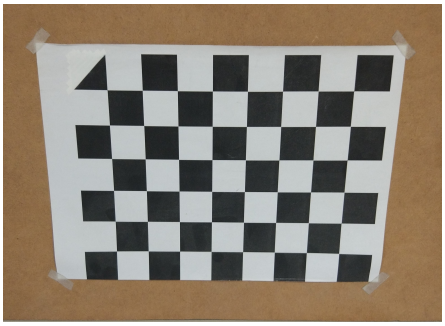
6.7 Change Color

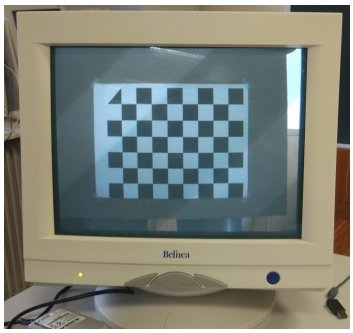
This module is a simple Verilog module to commute the different modes depending of the switch (SW[1..0]) used.

7 Results and Analysis

Through some photographs taken in the laboratory we present the obtained results, but is important to think that the photos were captured by a telephone camera.

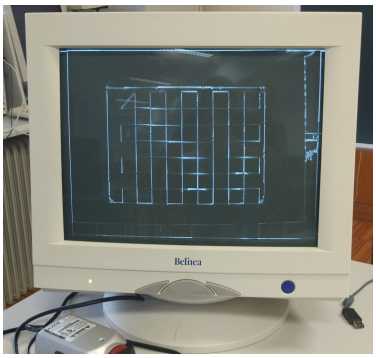
A well know test in Computer Vision Algorithms is the chess board examination, so we put a chess board in front of a camera and present the results in the next four pictures.


(a) Chess Board.



(b) Normal Mode.

(a) Gray Scale Mode.

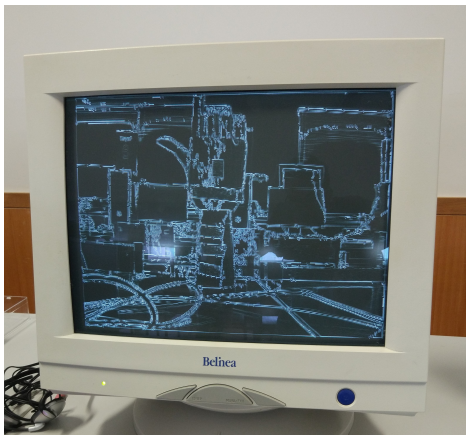


(b) Sobel Mode.

The lines of the squares are well defined, being that the results are satisfactory.

We continue by setting higher the test difficulty, but this time we used the wooden hand that is placed in the laboratory.


(a) Normal Mode.



(b) Sobel Mode.

The hand is easily identified as well as the background, although we can see a bit of noise in the display.

Now we will show the edge detector on a person.

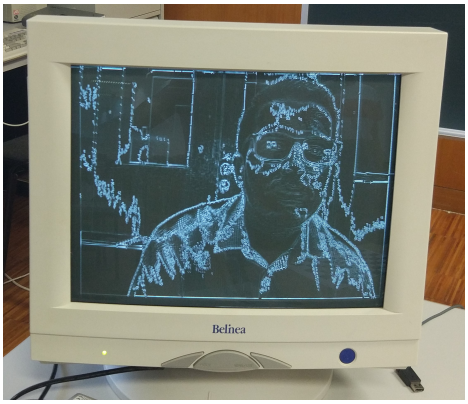


Figura 9: Sobel Mode.

The edge detector has a good behaviour on the face's contours, even allowing understand that the person has glasses. However, the filter did not worked well in the person's shirt, because we can see a lot of noise.

We conclude that the filter has the same problems like the typical Sobel operator, more specifically poor background detection and noise interference. The results could be improved if we had passed a noise removal filter before the edge detection.

Note, that there is only used 19% of the total memory, so the implementation is optimised, but reducing the amount of bits of the greys scale, we can reduce even more the memory used.

8 Future Work

The results could be more significant if the data image were filtered by a noise attenuator. In the future would be interest the addition of Gaussian Operator or a Mean filter to that purpose.

From the current project and with the same strategy, the implementation of different size operators could be easily done. This would allow to compare the precision of masks with different size.

The Module Sobel can be change to others operators like Robert Cross, or others, so would be a really nice project if each switch of the FPGA could enable different operators and the filter's name showed up on top of the VGA Monitor.

9 Conclusions

The final result of the project is very positive, since we obtained the desired behaviour in real time with only a small fraction of memory use. This project is currently open, because the future possibilities from our work developed are huge.

During this project, we had to spent a lot of time in the laboratory doing the implementation, so the "open lab"helped us a lot.

We had to have a good time management to conciliate study for finals with a project from this caliber, but in the end, we could develop a lot of skills that made us one step closer of became successful Engineers.

Referências

[1] Terasic Technologies, *Terasic TRDB-D5M Hardware Specification*, Document Version 0.2, June 10, 2009.

[2] Sobel Code
http://edge.kitiyo.com/2009/codes/sobel-core-verilog-module.html

[3] CHOUCHENE, Marwa, et al. Efficient implementation of Sobel edge detection algorithm on CPU, GPU and FPGA. International Journal of Advanced Media and Communication, 2014, 5.2-3: 105-117.

[4] User Manual, *CMOS TRDB-D5M Camera and 8MB SDRAM A2V4S40CTP*.

[5] Nazma Nausheen, Ayan Seal, Pritee Khanna, Santanu Halder, A FPGA based implementation of Sobel edge detection, Microprocessors and Microsystems, Volume 56, 2018, Pages 84-91, ISSN 0141-9331