The Design of an Image Converting and Thresholding Hardware Accelerator

Rafael M. Macieira, Lucas F. S. Cambuim,
Luiz L. Souza, Luiz A. Oliveira, Marcus F. R. Rios, Edna Barros
CIn - Centro de Informatica
UFPE - Universidade Federal de Pernambuco - Brazil
Email: {rmm2,lfsc,lls,laoj2,mfrs,ensb}@cin.ufpe.br

***Abstract*—The increasing amount of images and videos accesses by social networks users is demanding devices with more efficiency on image processing. On the other hand, such devices cannot be expensive and must have reduced power consumption due to the need for mobility. To cope with this scenario, this paper proposes the improvement of an image processing application by implementing some computation intensive functions in hardware and the others in software. The platform comprises a ATOM processor and FPGAs. After a profiling application, the time critical parts of image processing application are implemented as hardware modules in FPGAs and, thereby, speeding up the processing power of the platform. As case study we present the hardware-software implementation of the RGB converter and thresholding algorithm, an important function of image segmentation. The function for converting images from RGB format to YCrCB format is very time consuming. The implementation of this function in hardware results in speedup of about 13% up to 99% in this image processing approach, with an average error of 0.000410 for the Y component, 0.000293 for the Cr component and 0.0002333 for the Cb component.**

I. Introduction

Vision is one of the most important senses of some living beings. For the human beings, this sense is fundamental to perform work on a day-to-day such as product quality inspection, public safety and aircraft control. However, in certain cases, the amount of visual information and the speed at which this one should be processed are incompatible with the processing capacity of the human brain.

Thus, the computational systems are widely used to perform the high treatment and processing of images. Image processing has emerged basically from 2 interests: improvement of visual information for human interpretation and data processing scenes for perception through automatic machines [1].

The use of image processing in general-purpose processor systems is typically performed using the OpenCV library [2]. But, using this library could make the desired degree of processing unreachable, since most of image processing algorithms are carried out with a great use of calculations in parallel and their performance in software is slower than the performance using dedicated hardware. Also, another problem is the portability of OpenCV for different embedded system platforms.

The implementation of image processing components in embedded hardware arises as an alternative to increase the speed of these kinds of algorithms, once they are built to a specific purpose and, consequently, have a short response time. Thus, this approach increases the portability to other embedded system platform projects in which the image processing libraries are currently in software, such as OpenCV, and, because of this, they are far of achieving the degree of processing desired.

Two algorithms that have many applications in image processing and require a high computational processing time are the color space conversion and the thresholding algorithm. These two algorithms are widely used in skin detection applications. Thus, the color space conversion transforms images from RGB color space to YCrCb color space and the thresholding algorithm highlights the skin colors.

This paper proposes the implementation of color space conversion and thresholding algorithms using a Terasic DE2i-150 Board, that has the Intel Atom processor connected to a Altera's Cyclone IV FPGA through PCI express bus.

Compared to the implementation of these two algorithm in software, the solution in hardware provided a speedup from 13% up to 99%. However, the combined approach of hardware/software using this respective board resulted in a low performance boost. This happens due to the low data throughput between the Atom processor and the FPGA, provided by the physical implementation of the PCI express.

The paper is organized as follows: section II presents the related works, section III shows the algorithms' implementation in software approach, followed by the hardware modules implementation, in section IV. Section V shows the experiments and the obtained results and, finally, the section VI the conclusion.

II. Related Works

There are many works in the literature which propose hardware modules for converting RGB images into YCrCb color spaces with FPGAs [3], [4], [5], [6], [7], [8]. Most of them use a fixed point notation to represent real numbers. However, the thresholding operation in YCrCb color space is very sensitive to any variation in pixels values, and the implementation of such operations using fixed point representation can reduce the accuracy of the final result despite its simplicity.

The converter module proposed by [3] was implemented in VHDL in the FPGA 2v250fg256-6 and synthesized by Xilinx ISE 10.1i. This module operates over fixed point values and has a throughput of one pixel per clock cycle with the maximum

frequency of 106.741 MHz. Although it works at a high clock frequency, the use of fixed point can result in a considerable error rate.

Faycal Bensaali and Abbes Amira [4] propose an architecture based on Distributed Arithmetic (DA) ROM accumulator; and has a throughput of one pixel after eight clock cycles with frequency of 128 MHz. The architecture was implemented using the FPGA Celoxica RC1000-PP. In spite of having a high clock frequency, depending on the system architecture, the throughput of one pixel after 8 clock cycles can be a system bottleneck.

The work described in [5] proposes a conversion module that performs operations in fixed point representation with frequency of 5 MHz, it was implemented using the FPGA 2vp30ff1152-5. As said before, using fixed point can generate mismatches at the thresholding operation. Besides, the clock frequency can also be the bottleneck of the whole system.

The approach detailed by [6] developed an integrated architecture implemented in VHDL. It has been designed to operate as a 3 stage pipeline which executes the conversion from RGB to YCrCb and the down sampling; and is used in JPEG compression. Multiplications were developed through the use of shift operations. The module was tested in the 10KA Altera's FPGA family, in this way the module reached the frequency of 20.12 MHz and gives one pixel every 7 or 9 clock ticks depending if the pipeline is empty or not. Again, the low clock frequency and throughput might be the bottleneck of the system.

Muhammad Bilal and Shahid Masud at [7] proposed a hardware based on Distributed Arithmetic (DA) approach, which works with fixed point and was incorporated as a functional unit in a RISC processor. The architecture has been synthesized using Xilinx ISE 5.3i and was implemented in the FPGA Spartan IIe XC2s300e. Using fixed point data might generate a high data error rate, which can easily lead to errors in the thresholding results.

The work described in [8] proposed a converter module which performs operations in fixed point and with clock frequency of 55.179MHz and throughput of one pixel per clock cycle. Despite this approach proposes a high throughput with a significant clock frequency, it also uses fixed point units, and has the same issues of high error rate in the conversion results.

Considering the need of a mechanism for color space conversion and thresholding, with high performance and low error rate, the proposed approach provides a solution including a hardware module with a throughput of one pixel per clock cycle and running at 50 MHz, but that was already synthesized to a clock frequency of 116.6 MHz.

III. Software Approach

The focus of this paper is to show the improvement of performance obtained by the hardware-software co-design of an image application using one embedded processor and hardware modules in the prototyping board DE2i-150 from Terasic.

Image processing applications demand a high computational effort. However, they can, in most of the cases, take advantage of the support given by hardware to explore their great parallelism capacity. In this paper two algorithms, which are commons in applications for skin color detection, were optimized. They are the color space conversion from RGB to YCrCb and the thresholding algorithm. Those algorithms are often used before other techniques that refine the result, but, for the purpose of this paper, they are enough to demonstrate the enhancement of performance obtained when implemented in hardware. Besides, since YCrCb color system needs representing pixels intensity by floating point numbers, the high overhead of operations with this type of numerical representation was solved by implementing them in hardware.

The algorithm to detect skins used in this paper works as follows in Figure 1: First of all, an image of resolution of 640x480 is captured from a Terasic's TRDB-D5M camera. Since pixels comes out of the camera in RAW format, a step of conversion from RAW to RGB is necessary. Once the conversion is done, the converted pixel goes to a RGB to YCrCb conversor, aiming to explore the smaller space where the skin tones are confined, thus making the thresholding process easier. The next step is a thresholding which aims to detect skin tones in image. Finally the image is shown in a screen.

Fig. 1. Application Flow

The conversion from RGB to YCrCb was implemented by using the equation in the Figure 2, which has been extracted from OpenCV's documentation of the function *cvtColor* [2].

```
Y  <- 0.299*R + 0.*587*G + 0.114*B | Δ= 128   , for 8-bits;
Cr <- (R - Y)*0.713 +Δ             |   32768 , for 16-bits;
Cb <- (B - Y)*0.564 +Δ             |   0.5   , for Floating-point
                                   |           Images.
```

Fig. 2. RGB to YCrCb Conversion Formula [2]

Many research works [9], [10], [11] indicate that the best color space to represent human skin considering different human races is YCrCb. In this paper, the threshold has been applied at images in this color space using a different threshold for each channel. The used values were obtained experimentally and allow identifying the human skin in an efficient way. The threshold used in this paper is defined by the expression below, where $g(x, y)$ represents an image pixel in the position (x, y) in the image matrix. The elements y, cr, cb represent the components of the color space of the pixel $g(x, y)$.

- $g(x, y) = 1$, if $(803 < y < 2890) \wedge (2007 < cr < 2569) \wedge (1365 < cb < 2168)$
- $g(x, y) = 0$, if $(803 \geq y \geq 2890) \vee (2007 \geq cr \geq 2569) \vee (1365 \geq cb \geq 2168)$

IV. The Hardware Implementation of Converter and Thresholding Technique

The *GrecoRGB-CT* is the proposed hardware module for accelerating image processing functions. It has two main

features: (i) to convert images from the *RGB* color space to the *YCrCb* color space and (ii) to apply the thresholding technique for segmenting elements into the image.

Figure 3 shows, the *GrecoRGB-CT* subsystem that is composed by two main modules: *PixelConverterYCrCb* and *PixelThresholding*. The converter comprises four instances of PixelConverter modules for exploring parallelism. The first module, called *PixelConverterYCrCb*, converts a pixel from the RGB color system, represented by three 12-bits word, to the YCrCb color system, represented by three single precision floating point numbers, each one is 32 bits length. The second module, called *PixelThresholding*, is responsible for the pixel thresholding. This module performs five floating point comparisons at one time. These two modules will be better explained further.

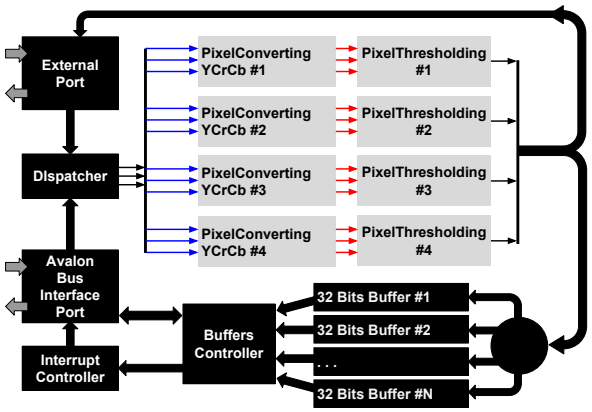

Fig. 3. The Architecture of the *GrecoRGB-CT* Module

As it can be seen in Figure 3, the *GrecoRGB-CT* module also contains infrastructure modules as the Avalon [12] bus interface controller port, the external port, the pixel dispatcher, the buffer bank with its controller and the interrupt controller.

The Avalon bus interface is responsible for the communication between the module and the CPU. In the Intel Atom based prototyping platform there is a PCI express controller attached to the Avalon bus (explained in Section V). The pixel dispatcher receives from the Avalon bus interface a pixel in the RGB color system, represented by three 12-bits words (the red, green and blue color), and returns the pixel to the next idle *PixelConverterYCrCb* module.

The external port permits attaching directly to the *GrecoRGB-CT* module an external image source, such as a digital camera. This port uses an external clock to define the color input frequency. It also support RGB format up to 12-bits. This port also has as output the result of the thresholding. The pixel dispatcher also receives data from this port.

The dispatcher module contains an index register that points to the last conversion unit. Thus, it informs to a multiplex which converting unit must receive the input signals. The pixel processing is done in a sequential and cyclic way, what means that the next converting unit will always be available.

The buffer bank and buffers controller units are responsible for selecting the next free buffer for storing the thresholding result, represented by one bit, and informing the interrupt controller when a result set was stored and a full buffer is ready to be read. Then, the interrupt controller triggers an interrupt to the CPU through the Avalon interface. As well as the resulting pixel from the converting unit, the thresholding units send their result to the next free buffer in a sequential and cyclic way. If all buffers are full, the whole unit stops working and receiving more pixels from the CPU until it read a result set, making available one buffer at least.

The *PixelConverterYCrCb* module is a pixel converter that works in pipeline with 6 stages and each stage has a latency of four clock cycles. As it can be seen in Figure 4, the converting unit is composed by several floating point adders, floating point subtraction units and floating point multipliers units. Except for the second and third stage, all the stages perform parallel floating point arithmetic for calculating each component of the YCrCb color system.

As can be seen in Figure 4, the inputs of the conversion unit are three 12-bits words representing the red color, green color and blue color of the RGB color system. However, the floating point units inside the converter require floating point numbers as inputs. Thus, a combinational module called *Int2FP* was developed for converting the integer number to a floating point number at the input.

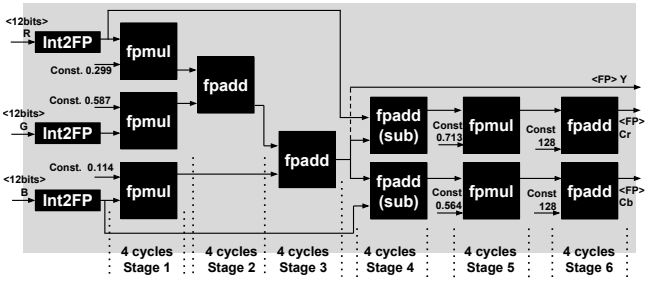

Fig. 4. The Architecture of the *ConversorYCRCB* Module

Once the pipeline is full, the converter module produces a YCrCb color system pixel every 4 clock cycle. Considering the four instances of the *PixelConverterYCrCb*, the entire *GrecoRGB-CT* module works with a throughput of 1 pixel per clock cycle. Because a conversion unit will be used again at in a period of at least four clock cycles, the pixel dispatcher can guarantee that the converter will be available when needed.

Each pipeline stage signalizes when it ends a pixel conversion, notifying the next stage informing when there is a new input pixel. Thus, the converting unit is ready, even in the case of some input delay, introducing gaps between the pixels into the pipeline.

The adders and multipliers modules operate on normalized single precision floating point numbers specified by IEEE-754 floating point standard [13]. These modules are able to process a pixel in three clock cycles. For reducing the error of each operation, it has been developed a rounding module which rounds the result to the nearest representable value according to IEEE-754 specification. The rounding algorithm needs three bits called guard, round and sticky (GRS). It is needed that both adders and multipliers result correct GRS bits to the rounding module.

The adder module performs the addition and subtraction operation according the algorithm described on [14]. For the multiplier module was used the version of the traditional shift-and-add algorithm to multiply the operands mantissa as described on [14]. To guarantee that the multiplier operation finishes after three clocks cycles, eight partial sums of multiplicand shifted based on the multiplier value were calculated in each clock cycle .

The underflow and overflow exceptions are treated without interrupting the pipeline. If an underflow takes place, i.e. the negative exponent is too large to be represented in the exponent field of the normalized single precision floating point number, a zero value is resulted. Similarly, in the case of an overflow, i.e. the positive exponent is too large to be represented in the exponent field of the normalized single precision floating point number, then the largest representable number is resulted. For this application it was noticed that and invalid exception will never take place because the data provided is always in the normalized floating point format.

It is important to know that all the modules are synchronized. Therefore when a module is in the data processing phase, the forward module is also running the same phase and it is not ready to receive any data. Thus, we put in each pipeline stage a delay adding one more cycle, totaling four cycles per stage. This mechanism ensures that the forward module is always going to be ready to receive the data coming from previous module.

Every time the conversion unit finishes a pixel, it signalizes to the forward thresholding unit that there is a new converted pixel, which is ready to be used as input for the *PixelThresholding* module.

Despite the overhead associated to the multiplication operation, the normalization operation inside the adder and multiplier module, the sticky-bit computation and the control delay, the Synplify tool has obtained the frequency of 116.6 MHz for the YCrCb converter module.

To guarantee the correct operation of the entire system with the PCI Express and Atom processor, some test vectors were created. Some of these test vectors were specified in SystemVerilog using Modelsim to validate the converter module, the thresholding module and the global modules. In the converter test vector, the R,G and B components of each pixel from many images was stored in a *.txt* file.

The test vector source reads a pixel from the *.txt* file and send its values to the converter. Then, the test vector source waits for the converter to raise the output ready signal, and when it does, the resulting YCrCb values is written in a *.txt* file in hexadecimal format.

The test vector for the global module is very similar to the converter's test vector. However, it has to simulate the signals sent through the Avalon bus and write in the output file the pixel values of the binaries image. The test vector of the thresholding module is also similar to the converter test vector. Besides, since the module is combinational, when the input data is ready and sent to the module, the result can already be written in the output file.

In order to implement data transfers between the processor and the converter module, a driver was developed. This driver performs the communication between the module *PixelConverterYCrCb* and the operating system through the prototyping platform used in this paper. A driver for the hardware module was specified in C, and it has 5 functions that are transparent to user program. The driver implemented runs over another driver, for the PCI Express, that Terasic releases with the prototyping board.

The PCI Express driver has some limitations. For example, although the PCI allows 256 bits for data transfer, the Terasic's driver supports the reading and writing of only 32 bits in the bus for each call. Besides, the driver from Terasic only works in Yocto [15] and Windows Operating Systems and it is not open-source. So, it was not possible to use PCI Express with its full capacity.

The driver's functions implemented in this paper are: *initPCIE*, *WriteRGB*, *ReadRGB*, *processImage* and *freePCIE*.

The *initPCIE* is used to load in memory a dynamic library for the PCI express and to grant access to the bus. The *WriteRGB* is used to write the R, G and B component words in the converter module, through a DMA module.

On the other hand, the *ReadRGB* function performs the reading of 32 bits from the *GrecoRGB-CT* module, also through a DMA module. Each bit represents the result from the thresholding of a pixel.

The *processImage* function uses the functions *WriteRGB* and *ReadRGB* to send and receive, respectively, from the module all the pixels of the image. Finally *freePCIE* closes the dynamic library and releases PCI Express bus.

V. Experiments and Results

As mentioned before, one of the goals of this work is to analyze the performance and quality of the conversion, comparing an implementation only in software and a hardware-software mixed implementation.

Before running any comparison experiments, the time consumed by critical functions implemented in software was measured with the purpose of defining the functions, for which a hardware implementation could leads to a performance improvement. For this, 1000 images were converted and thresholded and the *GNU gprof* tool were used to measure how much time each function spent during the execution of the application.

The measurement showed that the function called *converteYCRCB*, responsible for the color space converting, and the function called *limiarizacaoYCRCB*, responsible for the pixel thresholding, spent respectively 42.43% and 25.61% of the application runtime. Thus, these two functions were selected to be implemented in hardware, in order to speed up the performance.

The first experiment performed was the converting and thresholding of several images with low resolution (640x480) and high resolution (1280x720). For each image, it was measured the time to perform the functions using the *time.h* library of Yocto kernel, which counts the clock cycles for executing a piece of code. On average, the Atom processor took 131 ms to convert and threshold each low resolution images and 395 ms for each high resolution image.

The second experiment was done using the implemented hardware module. For this experiment we performed the measurement using the *GrecoRGB-CT* module isolated. The module was running on a clock frequency of 50 MHz and the image input was directly delivered to it at the same rate. Every 32 written pixels, one reading was done to get the result stored into the module buffers.

This experiment shows that the *GrecoRGB-CT* module, executing on that clock frequency, can perform the converting and thresholding of a low resolution image in 7 ms and a high resolution image in 20 ms. The real time of converting and thresholding of a low and high resolution image at this clock frequency is respectively 6,144 ms and 18.432 ms, however, some readings to recover the converting and thresholding result must be performed.

That experiment also shows that the average error rate for each YCrCb color space is of 0.000410 for the Y component, 0.000293 for the Cr component and 0.0002333 for the Cb component. Notice that this error is negligible and it does not have any influence for the thresholding result, once all values as compared with integer thresholds.

The third experiment was made using the DE2i-150 Terasic prototyping board environment, running the Windows 7 operating system into an Intel N2600 Atom Processor. A digital camera Altera D5M was connected to the GPIO port of the board and the *GrecoRGB-CT* hardware module was downloaded into the FPGA. The D5M camera was connected to the *GrecoRGB-CT* through the external port (Figure 3), providing one 12-bits RGB pixel in a frequency of 25 MHz.

As mentioned before, the DE2i-150 board uses PCI express bus to connect the Atom processor and the FPGA chip. Internally in the FPGA, the PCI express controller is connected to an Avalon bus and so the hardware IPs are connect to this Avalon bus.

The PCI express runs in a clock frequency of 100 MHz or 125 MHz and provides to the Avalon peripheral a clock frequency of 50 Mhz. Thus, through the *GrecoRGB-CT*'s Avalon interface (Figure 3), the *GrecoRGB-CT* IP is connected to the Avalon bus, transferring 32-bits packages using the 50 Mhz clock. Due to the implementation of the PCI express in the DE2i-150, only low resolution images were used in the third experiment.

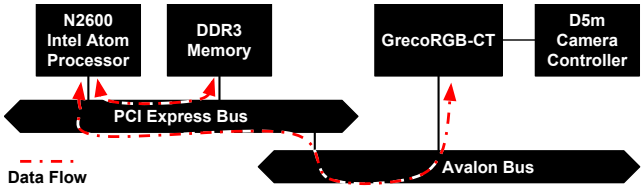

Fig. 5. A Simple Diagram of the Platform Architecture of DE2i-250 Board without using the DMA. The Dash Line Shows the Writing's Data Flow

This last experiment was performed using two different approaches. The first approach does not use the DMA. As it can be seen through the dashed line in Figure 5, the Atom processor must execute each reading and writing request directly to the *GrecoRGB-CT* device. The second approach uses the DMA to perform the burst data transfers, represented by the dashed line in Figure 6.

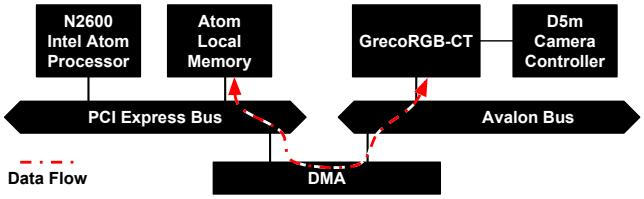

Fig. 6. A Simple Diagram of the Platform Architecture of DE2i-250 Board using the DMA. The Dash Line Shows the Writing's Data Flow

The Figures 5 and 6 also show a simple schematic of the DE2i-150 board architecture with the *GrecoRGB-CT* module. The released PCIe's device driver and hardware IP available only implements the access to the FPGA devices through an Avalon interface.

For the both approaches of the third experiment, four scenarios were mounted and performed to extract the results. The Scenario I (S-I) represents an application running into the Atom processor and executing the conversion and thresholding of images coming from the D5m Camera linked to the FPGA. In this scenario the *GrecoRGB-CT* module is bypassed, thus the Atom receives the raw 12-bits RGB image from the camera through the PCI express. So, for each low resolution image, this scenario needs to request 307200 pixels of 36-bits each. For this it must to perform 345600 reading requests of 32-bits bus word each.

The Scenario II (S-II) also represents an application running into the Atom, however the *GrecoRGB-CT* is not bypassed and, because of this, the conversion and thresholding step is fulfilled by the hardware layer. Because the result of a thresholding is represented as a single bit, the *GrecoRGB-CT* compact the result of 32 pixels for each bus word. Every 32-bits word read performed by the software represents 32 thresholded pixels. Thereby, the software application only needs to fulfill 9600 reading requests. In this scenario, the software layer needs to remount the image, reading the compressed bits and writing the complete pixel format into the Atom memory.

The last two scenarios (S-III, S-IV) are similar. They also represent an application running into the Atom using the *GrecoRGB-CT* to execute the conversion and thresholding, however they use a different compression of the thresholding result data. For each sequence of the same consecutive bits, the *GrecoRGB-CT* mounts a 32-bits bus word, in which the thirty-second bit represents the color of the pixel sequence (black or white) and the rest of the bus word represents the number of pixels in the sequence. So, when a sequence of the same color is broken or when the end of the frame is reached, the module closes the sequence and mount the 32-bits word, as specified. For these scenarios the software layer also must remount the image. For each bus word read from the hardware module, the software must write a sequence of complete pixel format into the Atom memory, depending on the color and length of the same pixels sequence.

The S-III represents the best case of this data compression. This means a thresholded image completely black or completely white. In this case, there is only one reading per frame; however the module mounts the 32-bits word only in

the end of the frame, which means that it only must wait the conversion and thresholding of the 307200 pixels of the image, in a clock frequency of 25 MHz. This takes approximately 12,3 ms.

The S-IV represents the worst case. In this case, every threshold pixel sequence only contains only one pixel. Thus, it needs to transfer 307200 32-bits word, once there is 307200 sequences of pixels.

The Tables I and II show the results of the third experiment for both approaches, without and with the DMA respectively, including all scenarios. They also show the speed up of the S-II, S-III and S-IV compared to the S-I.

Metrics	S-I	S-II	S-III	S-IV
Transferred Bus words	345600	9600	1	307200
Bus words Transf. time (ms)	4810.752	133.632	12.301	4276.224
SW Layer time (ms)	131	7.6	1.02	1.8
Total time (ms)	4941.752	141.232	13.321	4278.024
Speed Up	-	97.14%	99.73%	13.43%

TABLE I. EXECUTION TIME OF THE SCENARIO OF THE THIRD EXPERIMENT, WITHOUT DMA, COMPARED TO THE FIRST EXPERIMENT

Metrics	S-I	S-II	S-III	S-IV
Transferred Bus words	345600	9600	1	307200
Bus words Transf. time (ms)	4147.2	115.2	12.28	3686.4
SW Layer time (ms)	131	7.6	1.02	1.8
Total time (ms)	4278.2	122.8	13.30	3688.2
Speed Up	-	97.13%	99.69%	13.79%

TABLE II. EXECUTION TIME OF THE SCENARIO OF THE THIRD EXPERIMENT, WITH DMA, COMPARED TO THE FIRST EXPERIMENT

Mainly due to the reduction of the data transference, the S-II, III and IV are always better than the S-I. Even if the amount of transferred data is the same, the conversion and thresholding performed in hardware speed up the application time in at least 2.5%.

Comparing the two approaches, the Tables I and II show the reduction of the data traffic delay. This happens because, using the DMA, the overhead due to reading and writing function call and operating system *syscall* for each bus word is reduced, once the DMA execute blocks of readings and writings calls.

It is also important to notice that the DMA approach provides a better speed up only in scenarios where exists high data traffic.

VI. CONCLUSION

This work proposes an approach for speedup the converting and thresholding of images using a dedicated hardware module. To achieve this goal, it was necessary firstly to identify the most expensive functions of this application.

Thus, after identifying the time consuming functions, the hardware module was developed, tested and used in some experiments to validate and analyze the performance improvement.

The first conclusion was that, with dedicated data transfer, this image processing hardware accelerator can speed up the images' converting and thresholding in about 13% to 99%, compared to the application fully implemented in software. It is important to know that, as said before, after these experiments another version of the module was synthesized for a clock frequency of 116.6 MHz, what should speedup even more the performance of the image processing hardware accelerator. Besides, some improvements can be done in the multiply module, e.g. the implementation using carry-save adders [16], booth encoded [17] and so, what shall increases the clock frequency of the hardware module.

Another conclusion extracted from this work was the feasibility of using floating point units in dedicated hardware with a high performance and, consequently, a low associated average error.

Analyzing the whole platform, it becomes clear that, for a burst data processing at the FPGA, this architecture should be improved, once the delay in the data transfer between the system memory and the FPGA modules can be prohibitive.

REFERENCES

[1] R. T. Chin and C. A. Harlow, "Automated visual inspection: A survey," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 557–573, 1982.

[2] Opencv documentation. [Online]. Available: http://docs.opencv.org/

[3] E. Prathibha, S. Yellampalli, and P. A. Manjunath, "Design and Implementation of Color Conversion Module RGB to YCbCr and Vice Versa," vol. 1, no. 1, pp. 13–18, 2011.

[4] F. Bensaali and A. Amira, "Design and Efficient FPGA Implementation of an RGB to YCrCb Color Space Converter Using Distributed Arithmetic," 2004.

[5] B. Ahirwal, M. Khadtare, R. Mehta, and A. F. Implementation, "FPGA based system for Color Space Transformation RGB to YIQ and YCbCr," no. I, pp. 1345–1349, 2007.

[6] L. Agostini and S. Bampi, "Arquitetura Integrada para Conversor de Espaço de Cores e Downsampler para a Compressão de Imagens JPEG," 2002.

[7] M. Bilal and S. Masud, "Efficient color space conversion using custom instruction in a risc processor," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, May 2007, pp. 1109–1112.

[8] A. Sapkal, M. Munot, and M. Joshi, "R'g'b' to y'cbcr color space conversion using fpga," in *Wireless, Mobile and Multimedia Networks, 2008. IET International Conference on*, Jan 2008, pp. 255–258.

[9] D. Chai and K. N. Ngan, "Face segmentation using skin-color map in videophone applications," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 9, no. 4, pp. 551–564, 1999.

[10] T. M. Mahmoud, "A new fast skin color detection technique." *Proceedings of World Academy of Science: Engineering & Technology*, vol. 45, 2008.

[11] S. K. Singh, D. Chauhan, M. Vatsa, and R. Singh, "A robust skin color based face detection algorithm," *Tamkang Journal of Science and Engineering*, vol. 6, no. 4, pp. 227–234, 2003.

[12] Altera, *Development and Education Board - User Manual*, 2006.

[13] I. C. Society, *IEEE Standard for Floating-Point Arithmetic*, August 2008, sponsored by the Microprocessor Standards Committee.

[14] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software interace*, 3rd ed. Morgan Kaufmann Publishers, 2005.

[15] Yocto project. [Online]. Available: https://www.yoctoproject.org/documentation

[16] M. Ortiz, F. Quiles, J. Hormigo, F. Jaime, J. Villalba, and E. Zapata, "Efficient implementation of carry-save adders in fpgas," in *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*, July 2009, pp. 207–210.

[17] D. Chandel, G. Kumawat, P. Lahoty, V. V. Chandrodaya, and S. Sharma, "Booth multiplier: Ease of multiplication," 2013.