# On the Implementation of the Gamma Function for Image Correction on a Endoscopic Camera

Sheikh Shanawaz Mostafa[1], L. Natércia Sousa[1], Nuno Fábio Ferreira[1,2], Ricardo M. Sousa[3], Joao Santos[3], F. Morgado-Dias[1,2], Martin Wäny[3]
[1]*Madeira Interactive Technologies Institute, Madeira Tecnopolo 9020-105, Funchal, Portugal.*
[2]*University of Madeira, Rua dos Ferreiros 9000-082, Funchal, Portugal.*
[3]*Awaiba Lda, Madeira Tecnopolo 9020-105, Funchal, Portugal.*

***Abstract*—This paper describes part of project that implemented the image processing of a CMOS sensor for endoscopic purposes. The sensor is a small sized device of 1x1mm2 and the image processing has been done inside a FPGA. This part of the work describes the implementation of the Gamma function with a balance between the resources needed and the accuracy. A linear piecewise solution was used that stores the values for 31 gamma functions with values ranging from 1 to 4 with 0.1 steps. The solution developed is 10 bit based, was coded in VHDL and is implemented in a Spartan 6 FPGA. The results show that it is an accurate solution that has a small footprint in terms of used resources.**

***Keywords—gamma function; FPGA; VHDL; Endoscopy***

## I. INTRODUCTION

Endoscopic cameras are a relevant tool to support many medical procedures. In the United States of America alone, in 2009, more than eighteen million endoscopies were performed with a cost of more than thirty two million dollars [1]. The endoscopy procedure produces an image that is richer (and allows continuous image) than other techniques but it requires several steps from the image sensor to the screen monitored by the physician. These steps for the current work are represented in figure 1, from the sensor used, NanEye to the output stage.

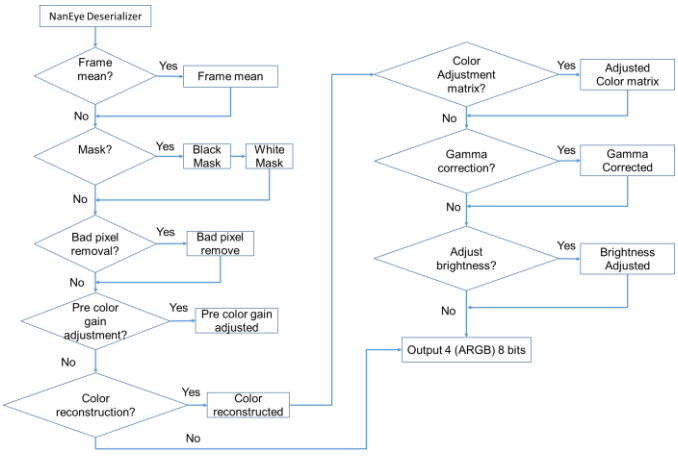


Fig. 1. Flowchart of image processing pipeline for endoscopy.

One of the necessary steps necessary to process the sensor output is to correct the image using the Gamma function, in spite of the improvement achieve in the last few years in the display characteristics, which are now almost linear. At present this correction is still mostly done through coding [2], [3] and matching the nonlinearity of the human vision system [4].

Considering the digital implementation, the inverse Gamma function can be implemented according to eq.1:

$$\text{Output pixel value} = (\text{Input pixel value})^{(1/\text{gamma})} \quad (1)$$

This work presents the hardware implementation of gamma function for small size camera that is used for endoscopy purposes. The camera used is the NanEye sensor, produced by Awaiba. This sensor can be used in an environment with multiple sensors, for 3D image and the processing of the image extracted from the sensors requires heavy calculations, making it hard to do in a PC in real time. As an alternative, an Field Programmable Gate Array (FPGA) was chosen as the platform but all the resources need to be minimized in order to reduce the cost. Nevertheless the gamma function is quite resource consuming and a new solution needs to be developed.

This solution is based in linear polynomial approximations principles. The final solution is able to apply the gamma correction on three color channels simultaneously.

This paper is organized as follows: the second section, equipment presents the FPGA, the camera and the board; the third section, implementation of the gamma function, presents the options taken to design the piece wise linear solution; the fourth section is about the results where the image corrected with the different gamma functions are acompared and finnaly the conclusions summarize the paper.

## II. EQUIPMENT

The equipment used for this paper is a 1.0x1.0x1.65mm camera, called NanEye-RGB, shown in figure 2a, together with the platform chosen, an FPGA on development board CESYS EFM-02. The NanEye is in a protective casing and the casing external size is naturally bigger. The camera has the following characteristics: LVDS 10 bit serial data transmission, 250 x

250 pixels (0.0625MP) resolution, 42Fps – 55 Fps frame rate, rolling shutter and adjustable power supply [5], [6].

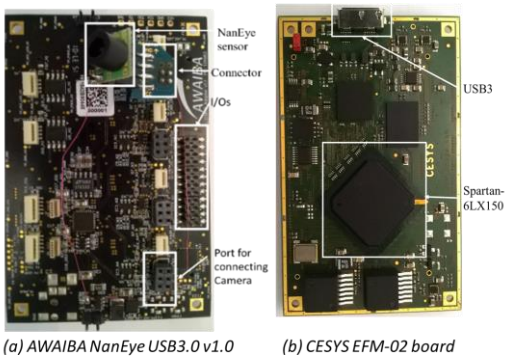


Figure 2: Hardware system used for design and verification of gamma correction.

The NanEye is connected to the FPGA through the NanEye USB3.0 v1.0 board, that can be seen on Fig. 2 (a), and supports the connection of up to four NanEye cameras.

The CESYS EFM-02 board contains a Spartan 6 XC6LX150 FPGA with 147,443 configurable logic blocks, 1,335kb of distributed RAM and 4,824 kb of block RAM. This board is also responsible for powering the FPGA and the sensors.

## III. Implementation of the Gamma Function

There are some simple general options for implementing non-linear functions inside a FPGA, such as Look Up Table (LUTs) or Read-only memory (ROMs) [8]. Nevertheless these options do not present the a good resolution or will result in a large amount of used resources.

The objective is, considering the hardware being used, to create an adjustable gamma function with values ranging from 1 to 4 with 0.1 intervals and 10 bit pixels. Using the above mentioned solutions this results in 31 gamma functions whose values need to be stored, using a large amount of memory.

A potential alternative can be obtained representing the gamma functions as piece-wise linear polynomial approximation [9]. This approach requires a few decisions to be made: how many linear sections to use and when to change for a new linear section.

Analyzing the gamma function it can be seen that the slope changes a lot, so to minimize the error with a small number of sections, placing the segments evenly is not the best solution. In fact, in the literature different papers suggest not using evenly placed segments as a better solution [9], [10], [2].

After detailed analysis and empirical testing, it was decided to use ten linear segments. The approach chosen is stated in Eq. 2, with $C_{1x}$ representing the slope and $C_{2x}$ represents the intercept, while the index x stands for the segmentation number. It was also decided to increment the sections are in two's power to better fit the non-linearity of the gamma function. Both coefficients for each function and each are previously calculated with 95% confidence bounds..

$$\text{Output pixel value} = \text{Input pixel value} \times C_{1x} + C_{2x} \quad (2)$$

### A. Hardware implementation

The hardware implementation proposed in this work is composed of three main parts: ROM, multiplexer and mathematical operators. The implementation receives as input the coded gamma values upon which the gamma correction will be applied. The inputs are in the RGB format (Red, Green and Blue) and for each pixel a 10 bit value is transmitted corresponding to each of the colors that need gamma correction

Data transmission is coordinated using two signals: Fval, and Lval that correspond to vertical and horizontal synchronization. At the same time the control data transmission, they will be used in the hardware to make the different blocks work correctly. The Reset and Clock inputs are driven by the global reset and the main clock is used for the global FPGA design.

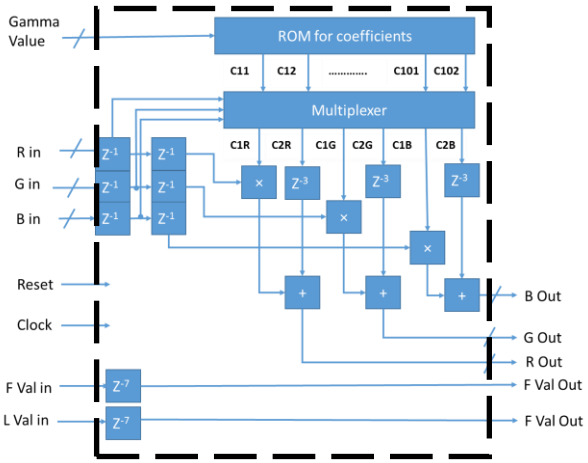


Figure 3: Block diagram of the implemented System in the FPGA.

#### *ROM*

The ROM is used as a LUT to store the slope and intercept values for the 10 segments of each of the 31 possible gamma values, as shown in Fig.3. The user decides the value to be used for the gamma correction and for each choice there are 20 coefficients to be supplied to the Multiplexer.

#### *Multiplexer*

The multiplexer is connected in such a way that it is able to choose the appropriate linear section to be applied for gamma correction in each of the inputs supplied for each of the color channels, following what is represented in Eq. 2.

#### *Arithmetic Operations*

The arithmetic operations implement Eq. 2 simultaneously for the three color channels, using a multiplier [11] and an adder [12] implemented with the LogiCORE IP from Xilinx (Fig.3). The parallel solution chosen is able to reduce the processing time working on the three components of the RGB data at the same time.

## IV. RESULTS

After the hardware implementation has been completed and tested, results are collected and analyzed in three different approaches:

- Calculate error between the true gamma function and the hardware implementation
- Compare the quality of image between the true gamma function and the hardware implementation
- Compare the hardware resources used by the hardware implementation developed and a Intellectual Property (IP) core supplied by Xilinx.

***Error Analysis***

The error analysis is made over 10bits, considering that this is the format used for input and output. Considering this and Eq.3, where: $M$ is the mathematically gamma corrected pixel, $H$ is the hardware corrected pixel and $i$ is the input pixel, the the Mean Absolute Error (MAE) was calculated and is presented in Table I.

$$\text{MAE} = \frac{1}{1024}\sum_{i=0}^{1023}\left|\begin{array}{l}\text{corrected pixel}_{\text{i,M}}\\ -\text{corrected pixel}_{\text{i,H}}\end{array}\right| \quad (3)$$

Also considering a common gamma function value for endoscopy settings, 1.2, the error between the gamma function and its hardware implementation in this project is shown in Fig.4. In this figure the pixel values are represented for the 10 bits.

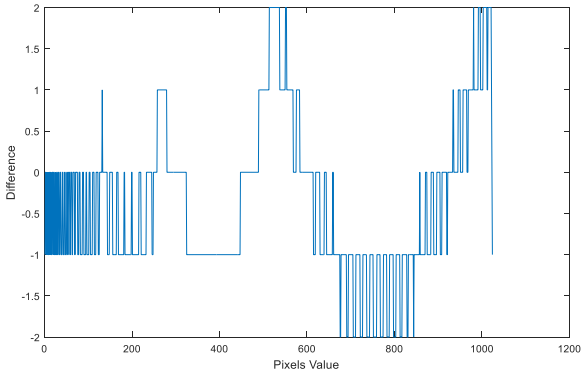


Figure 4: The difference between mathematical gamma and piecewise linear gamma in FPGA for the gamma value of 1.2

TABLE I. MEAN ABSOLUTE ERROR FOR GAMMA RANGE 1 TO 4 FOR 10 BITS

| Gamma Value | MAE |
|---|---|
| 1 | 0 |
| 1.1 | 0.775390625000000 |
| 1.2 | 1.13378906250000 |
| 1.3 | 1.47167968750000 |
| 1.4 | 1.74218750000000 |
| 1.5 | 1.94628906250000 |
| 1.6 | 2.10839843750000 |
| 1.7 | 2.15820312500000 |
| 1.8 | 2.30468750000000 |
| 1.9 | 2.36523437500000 |
| 2 | 2.46972656250000 |
| 2.1 | 2.45312500000000 |
| 2.2 | 2.46972656250000 |
| 2.3 | 2.54492187500000 |
| 2.4 | 2.52441406250000 |
| 2.5 | 2.52050781250000 |
| 2.6 | 2.49316406250000 |
| 2.7 | 2.49902343750000 |
| 2.8 | 2.52734375000000 |
| 2.9 | 2.47949218750000 |
| 3 | 2.46386718750000 |
| 3.1 | 2.45605468750000 |
| 3.2 | 2.45605468750000 |
| 3.3 | 2.45605468750000 |
| 3.4 | 2.41210937500000 |
| 3.5 | 2.42773437500000 |
| 3.6 | 2.38183593750000 |
| 3.7 | 2.38183593750000 |
| 3.8 | 2.34570312500000 |
| 3.9 | 2.32324218750000 |
| 4 | 2.32519531250000 |
| **Average MAE** | **2.1747** |

The values presented in Table I have an average error of 2.1747. The NanEye camera has a default value of 1.2. It can be seen that for this value the implementation error is lower than the average error.

Figure 5 shows the box whiskers representation of the error per segment. As can be concluded: the Least Significant Bits (LSB) (pixel value 0 and 1) have the greatest contribution to the error. As it is common in these situations, it might be the case that the LSB is no longer significant.

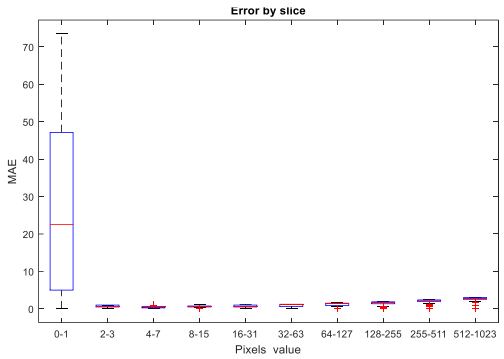


Figure 5. Box whiskers representation of MAE by slice for all the 31 gamma choices.

Another solution that can be found in the literature and that was also tested is the LUT interpolation [13]. No solution of this type has been reported for 10 bits inputs, but the principle is that it stores every 4th sample, being able to save 75% of the memory.

The comparison is shown in Fig. 6, through the MAE of the LUT interpolation and the piecewise linear method, considering all 31 gamma values, with the gamma values on the horizontal axis and the MAE in the vertical axis. It can be concluded that up to the value of 1.4 of gamma the piecewise linear method presents less error, after that the LUT interpolation presents less error. It should be considered that the most common choice for the use of this type of camera is 1.2. The biggest error for the piecewise linear approximation is obtained for 2.3 and reaches the value of 2.545. In the case of LUT interpolation, the maximum error is 1.779 reached for 3.9.

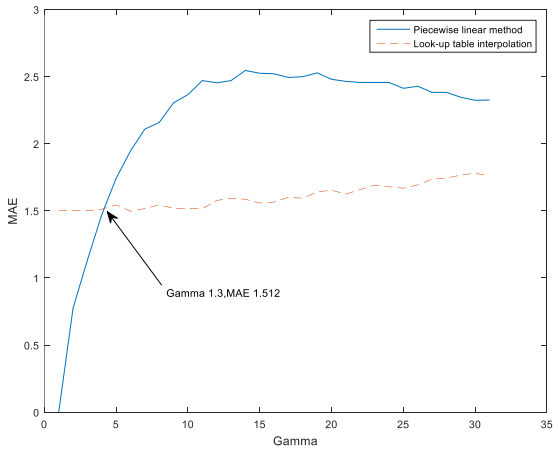


Figure 6. Comparison between look-up table interpolation (which stores every 4th sample) and piecewise linear method in 10bits.

For a different test a RGB picture of 250×250 pixels with 10bits is generated using uniformly distributed pseudorandom integers from 0 to 1023. It results in a quite flat histogram that ensures that the picture does not have very dark or very lighted areas. The corresponding corrected histogram for 1.2 gamma function is shown in Fig.7. As can be seen, the general trend of the curve is followed by both solutions, but the piecewise linear solution has less difference when compared to the with the correct mathematical function.

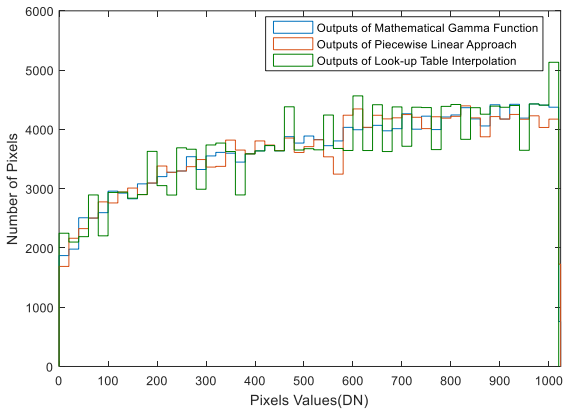


Figure 7. Comparison of image in terms of histogram mathematical gamma function, look-up table interpolation and piecewise linear gamma function.

### *Picture quality*

To analyze the picture quality we should consider that the piecewise linear gamma solution has been implemented in a Spartan 6 XC6SLX150T and transmitted through a USB3 connection. Although the original data is 10 bit based, it is later processed by a viewer with an 8-bit structure, so the 2 LSB are truncated. This truncating is necessary due to image and video representation in true color with 24-bit for RGB colors.

The 8 bit results are then compared resulting in MAE of 0.9227, a lower value compared to what was obtained before for 10 bit. This is consistent with the fact analyzed before that the error was mostly on the LSB.

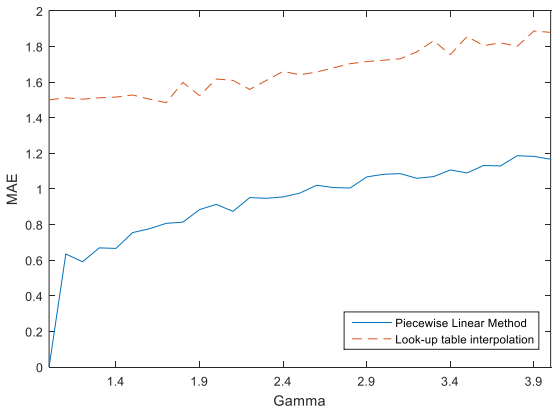


Figure 8. Comparison between look-up table interpolation which stores every 4th sample and Piecewise Linear Method in 8bits.

In Figs. 9 and 10 it is possible to see the same image using first (Fig. 9) the correction with 1.2 value for the correct gamma function and (Fig. 10) with the piecewise linear method, both for 8 bit representation and very difficult to see the difference.

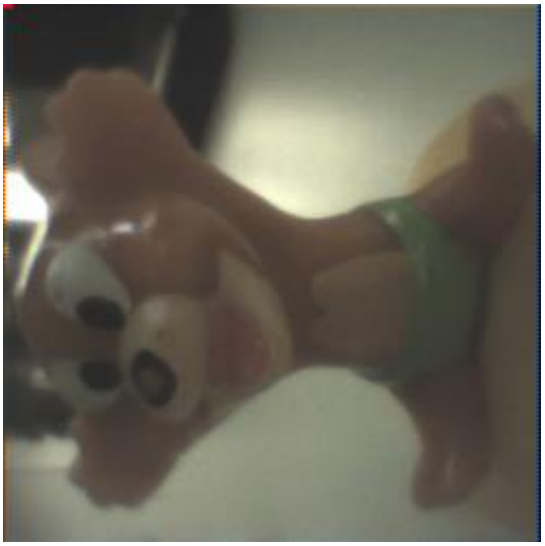

Figure 9. Image captured with the NanEye camera and corrected using the true mathematical gamma function with a parameter of 1.2

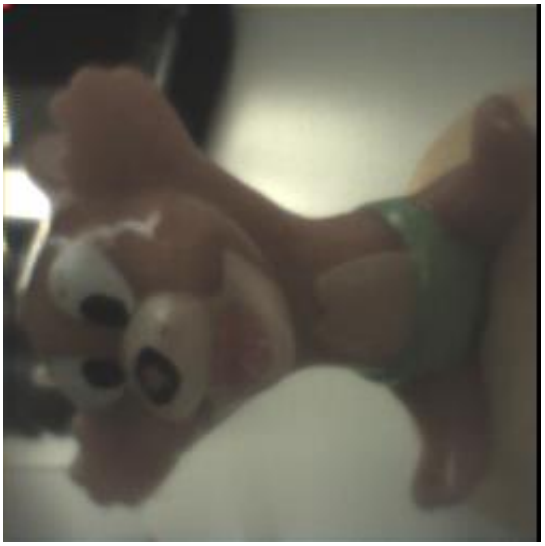

Figure 10. Image captured with the NanEye camera and corrected using the true mathematical gamma function with a parameter of 1.2

Capture image of the NanEye camera using piecewise linear gamma of 1.2 implemented in FPGA.

Figures 9 and 10 are now compared with the histogram of Fig. 11. The purpose is to understand the differences between the original function and the hardware implementation. As it can be seen, the hardware implementation presents some different values but follows quite closely the original function.

The bigger differences can be seen at the beginning of the histogram and the hardware implementation's histogram appears left shifted. This may result in a slightly darker image.

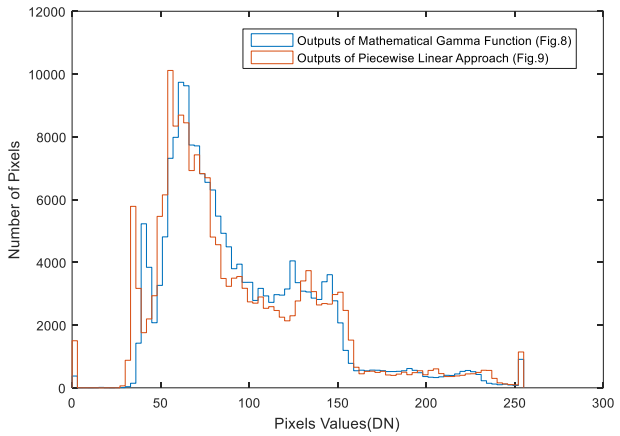


Figure 11. Histogram comparison of images of the actual gamma function and the piecewise linear gamma function with NanEye camera.

### *Performance Comparison and Resource Utilization*

This section contains the resource comparison between the implementation developed in this work and the LogiCORE IP Gamma Correction v3.0 [13], considering as platform the Spartan 6. The summary can be seen in Table 2.

It should be noted that there are some common resources to both solutions that are not implemented in the same way. The resources used by the LogiCORE IP were retrieved from [13], where N/A indicates Not Available.

From the results in Table 2 it can be seen that:

- The piecewise linear gamma function implementation does not use any BRAM16 blocks and LUT6-FF pairs
- Considering the information available it is assumed that the LogiCORE IP does not use DSP48 and BUFG, while piecewise linear solution uses a very small amount of these elements.
- The piecewise linear implementation uses almost 14 times more the number of FFs.

Finally in terms of maximum clock frequency, both solutions can achieve 250 MHz, but due to the NanEye camera to capture the images, the piecewise linear solution is working at 100 MHz clocking frequency.

TABLE II. RESOURCE UTILIZATION AND TARGET SPEED FOR SPARTAN-6 - XC6SLX150, FOR THE LOGICORE IP GAMMA CORRECTION V3.0 [13] AND THE PIECEWISE LINEAR GAMMA FUNCTION IMPLEMENTATION.

| **Resource** | **LogiCORE IP** | **piecewise linear gamma** |
|---|---|---|
| FFs | 72 | 952(fully used 449) |

| | | |
|---|---|---|
| LUT6-FF Pairs | 59 | 0 |
| BRAM16 | 3 | 0 |
| BUFGs | N/A | 1 |
| DSP48A1s | N/A | 3 |
| Clock Frequency (MHz) | 256 | 100 (max 250) |

## Conclusion

This work presents a hardware implementation for the gamma function correction using a FPGA. The solution proposed is based on a piecewise linear method to solve nonlinearity and presents a 2.1747 MAE (in 10 bits) after implemented in the hardware. This value decreases to 0.9227 when using 8 bits that are used for true color RGB. A potential improvement to this solution could be obtained from adding more segments especially in lower part of the gamma function.

This implementation shows that the piecewise linear method is a good alternative to save resources compared to a ROM/LUT solution. This solution also doesn't require external memory, obtaining faster response and consuming less power.

The analysis of the resources used, although not totally clear since the IP core information does not disclosure everything, seems to point for lower resource use on the IP core. Nevertheless, it should be pointed out that on a solution fully developed there is a greater control of all the detail than what can be achieved with an IP core.

## ACKNOWLEDGMENT

The authors would like to acknowledge the Awaiba company for their support during the development of this work and the Portuguese Foundation for Science and Technology for their support through project PEst-OE/EEI/LA0009/2011. Also acknowledged is the Funding Program + Conhecimento II: Incentive System to Research and Technological Development and Innovation of Madeira Region II, through the project Vision 3D – MADFDR – 01 – 0190 – FEDER – 000014.

Acknowledgments to ARDITI - Agência Regional para o Desenvolvimento da Investigação Tecnologia e Inovação through the support provided by the FSE - Madeira 14-20.

## REFERENCES

[1] A. F. Peery, E. S. Dellon, J. Lund, S. D. Crockett, C. E. McGowan, W. J. Bulsiewicz, N. J. Shaheen, "Burden of Gastrointestinal Disease in the United States: 2012 Update." Gastroenterology, vol. 143, Issue. 5, pp. 1179–1187.e3, 2012.

[2] D.-U. Lee, R.C.C Cheung, J.D. Villasenor, "A Flexible Architecture for Precise Gamma Correction", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol: 15, Issue: 4, Pages: 474 - 478, 2007.

[3] S. Kang, H. Do, B. Cho, S. Chien, and H. Tae, "Improvement of low gray-level linearity using perceived luminance of human visual system in PDP-TV," IEEE Trans. Consum. Electron., vol. 51, no. 1,pp. 204–209, Feb. 2005.

[4] S. Hecht, "A theory of visual intensity discrimination," J. General Physiol., vol. 18, no. 5, pp. 767–789, 1935.

[5] *Specification NanEye 2D Web v4.4*, AWAIBA, Lda Madeira, October 19, 2015. http://www.cmosis.com/products/product_detail/naneye

[6] *Product Sheet NanEye Area Scan Sensors*, AWAIBA, Lda Madeira, http://www.cmosis.com/products/product_detail/naneye

[7] *EFM-02 Hardware reference (Hardware Revision 1.2)*, CESYS GmbH, Herzogenaurach, Germany, UG102 (v1.7) April 07, 2014 www.cesys.com

[8] B. Lucas, "Method and apparatus for converting floating-point pixelvalues to byte pixel values by table lookup," U.S. Patent 5 528 741, Jun.18, 1996.

[9] T. Lin, H. Cheng, and C. Kung, "Adaptive piece-wise approximation method for gamma correction," U.S. Patent 6 292 165, Sep. 18, 2001.

[10] E. Kim, S. Jang, S. Lee, T. Jung, and K. Sohng, "Optimal Piece Linear Segments of Gamma Correction for CMOS Image Sensors." IEICE Transactions vol. 88-C, no.11, pp. 2090-2093, 2005.

[11] *LogiCORE IP Multiplier v11.2*, Xilinx, Inc. San Jose, California, (DS255), March 1, 2011.

[12] *LogiCORE IP Adder/Subtracter v11.0*, Xilinx, Inc. San Jose, California, (DS214), March 1, 2011.

[13] *LogiCORE IP Gamma Correction v3.0*, Xilinx, Inc. San Jose, California, (DS719), September 21, 2010