Proceeding of the IEEE
International Conference on Robotics and Biomimetics (ROBIO)
Shenzhen, China, December 2013

Pipelining Harris Corner Detection with a Tiny FPGA for a Mobile Robot

M. Fatih Aydogdu, M. Fatih Demirci, and Cosku Kasnakoglu

Abstract— **With their parallelizable inner structures, field programmable gate array (FPGA) are increasing their popularity in today's embedded systems. In this paper, we present an implemented, unique and pipelined FPGA architecture designed with Verilog HDL to be used on a mobile robot for detecting corners in colored stereo images using Harris corner detection (HCD) algorithm in real time. The architecture consists of 3 pipelined modules and processes RGB555 formatted images in 640x480 resolution. The design is implemented on Xilinx's ML501 board having a XC5VLX50 FPGA, one of the smallest FPGAs of Virtex-5 series. Raw and processed data are stored into a single DDR2 memory of Micron, MT4HTF3264HY on the board, allowing only a single read or write operation at a time. By using less than 75% of FPGA resources and a 100MHz system clock, we achieved a corner detection rate of 0.33 pixels per clock cycle (ppcc) corresponding to a corner detection frequency of 54Hz for the stereo images.**

I. Introduction

Vision based systems need sensible features in order to identify and classify environments. Corners are one of the distinguishable features used by these systems. Extracted corners help differentiate patterns, detect objects and guide algorithms to make decisions. Corner detection algorithms may be classified into two groups [1]. The first group is contour-based algorithms in which curvature spaces are formed to classify edges and corners in the images. The other group is intensity-based algorithms, computationally less expensive but also less successful than the former ones.

Among the intensity-based algorithms, Harris [2] and SUSAN [3] algorithms are the most common ones. Different studies [1, 4, 5, 6] argue that Harris algorithm has superior performance than the other intensity based algorithms. In vision systems, corner detection is one of the elementary steps and its performance is critical in terms of performance. Therefore even if the intensity-based algorithms require less computational time with respect to the contour-based ones, they still need to be accelerated.

In this paper, we present all the implementation details of a pipelined FPGA architecture to be used on a mobile robot for HCD in colored stereo images. The design is composed of three pipelined modules generated using Verilog HDL. The architecture is implemented on a XC5VLX50 FPGA, one of the smallest FPGAs of Virtex-5 family of Xilinx. Stereo images were captured with Omni Vision 7720 image sensors in RGB555 format and DDR2 of Micron is used to store raw and processed data of the stereo images. This makes the design less platform dependent and applicable with cheaper hardware. Although the system is tested with a modest system clock of 100MHz its performance is sufficient for real time.

In section II of this paper, recent corner detection implementations are discussed. A brief overview of the HCD algorithm is discussed in section III and the designed architecture is presented in section IV with details. In section V and VI, the resource utilization and performance of the designed architecture are discussed respectively. We conclude this paper with section VII.

II. Related Work

In recent years, there have been studies to accelerate corner detection algorithms. Claus et al. [12] presented an FPGA architecture for SUSAN algorithm and with a clock frequency of 100MHz, the authors achieved a corner detection rate of 0.9652ppcc in images with 640x480 resolution by using 30% of resources available in XC2VP30 FPGA of Xilinx's Virtex-II Pro series. There have also been studies to accelerate HCD in different platforms. Teixeira et al. [7] presented an algorithm for non-maximal suppression (NMS) increasing the speed of corner detection algorithms on graphics processing unit. With a 1350MHz system clock, he achieved a processing rate of 0.088ppcc. Hosseini et al. [8] implemented HCD algorithm on specialized processor architecture with 2 DDR2 memories. Dietrich [9] used MATLAB to generate FPGA hardware for HCD algorithm. Even though similar FPGA designs are possible with high level languages one may have problems while combining or optimizing complicated designs.

III. HCD for Colored Images

One of the earliest corner detection algorithms proposed by Moravec [10] was modified by Harris in order to eliminate its shortcomings. These were anisotropic and noisy response of the algorithm and its handicap in differentiating edges and corners. To decide whether or not pixels of a grey-scale image are corners or edges according to Harris's algorithm, firstly, a characteristic M matrix is built for all of the pixels in the image as in equation 1

Manuscript received September 15, 2013.

M. Fatih Aydogdu is with Electrical and Electronics Engineering Department, TOBB University of Economics and Technology, 06560, Ankara, TURKEY (e-mail: mfatihaydogdu@gmail.com).

M. Fatih Demirci is with Computer Engineering Department, TOBB University of Economics and Technology, 06560, Ankara, TURKEY (e-mail: mfdemirci@etu.edu.tr).

Cosku Kasnakoglu is with Electrical and Electronics Engineering Department, TOBB University of Economics and Technology, 06560, Ankara, TURKEY (e-mail: kasnakoglu@etu.edu.tr).

$$M = \begin{bmatrix} \sum_W I_x^2 & \sum_W I_x I_y \\ \sum_W I_x I_y & \sum_W I_y^2 \end{bmatrix} = \begin{bmatrix} A & C \\ C & B \end{bmatrix}, \quad (1)$$

where I_x and I_y are the horizontal and vertical gradients of pixels in W. W is a mask averaging the gradient products of the pixel whose M matrix is built and the pixels surrounding it. The coefficients of M are selected according to a Gaussian distribution and its size is selected by the implementer. Then, for each pixel, a corner response, R is generated using their M matrix as:

$$R = \det(M) - k(Tr(M))^2 = AB - C^2 - k(A+B)^2, \quad (2)$$

where k is a constant whose typical value can be selected between 0.04 and 0.06. The computed R values are the cornerness measures (CM) of the pixels. If CM of a pixel is positive and greater than a specified threshold the pixel corresponds to a corner. Moreover if it is negative and smaller than a specific threshold the pixel corresponds to an edge. Like in many feature detecting algorithms, while implementing HCD, NMS is applied finally. In this way, only the pixel having maximum corner characteristics in the neighborhood of a corner is selected to be a corner.

The HCD algorithm was modified by Montesinos [11] for colored images. In this case, only the calculation of M matrix is changed as:

$$M = \begin{bmatrix} \sum_W (R_x^2 + G_x^2 + B_x^2) & \sum_W (R_x R_y + G_x G_y + B_x B_y) \\ \sum_W (R_x R_y + G_x G_y + B_x B_y) & \sum_W (R_y^2 + G_y^2 + B_y^2) \end{bmatrix}, \quad (3)$$

where R_x, G_x and B_x are the gradients of red, green and blue channels in x direction respectively and R_y, G_y and B_y are the gradients of these channels in y direction respectively. In the rest of this document, the sums in the parenthesizes of (3) will be called sum of gradient product (SOGP) to make the text plainer. Specifically, $R_x^2+G_x^2+B_x^2$ will be called $SOGP_{xx}$, $R_y^2+G_y^2+B_y^2$ will be called $SOGP_{yy}$ and $R_xR_y+ G_xG_y+ B_xB_y$ will be called $SOGP_{xy}$.

IV. ARCHITECTURE

While pipelining the HCD, algorithmic and hardware constraints shape our design. We intend to implement HCD on a board with a single DDR2 memory imposing a hardware constraint to the design. Memory Interface Generator (MIG) tool of Xilinx is used to generate DDR2 interface module with a burst length of 4. For external read and write commands, the generated module has FIFO buffers allowing a read or write operation to be issued at each clock cycle (CC). As well as obeying external commands, MIG module, issues pre-charge and auto refresh commands crucial for dynamic RAM operations.

In terms of algorithm, the first constraint of HCD is that before computing the M matrix of any pixel, all of the SOGPs of the surrounding pixels in W window of that pixel have to be computed. The other constraint is that before applying NMS to any part of the image, all of the CMs of the pixels in that part of the image have to be obtained. Thus we decide to divide the algorithm into 3 distinct phases and construct 3 distinct pipelined modules. The first module of the architecture is SOGP module taking the intensity values of images and outputs SOGPs of pixels. The second module is CM module getting SOGP values of pixels and outputs their CM values. The third designed module is the NMS module taking CM values and applies NMS. In order to feed these modules with data, there are 2 possible architecture options of which general structures are shown in Fig. 1. According to the first option shown on the left, designer can use different block RAMs (BR) to feed the pipelined modules separately. In this option, one of the BRs is loaded with raw data by the controller module (CMOD), establishing data flow in the FPGA architecture and the other BRs are loaded with processed data of the pipelined modules. Data loaded into these BRs are fed into the following pipelined modules without any latency which minimizes the total processing time. In the second option, it is also possible to use same BR loaded by the CMOD to feed the pipeline modules consecutively as shown on right on the same figure. The former option reduces the total processing time by 3 times but consumes approximately 3 times more BR resources of FPGA. Since the design is implemented on a small Virtex-5 FPGA, we decide to use the second architecture option not minimizing the total processing time but consuming less FPGA resources.

Interface modules for cameras are designed in order to capture the images simultaneously. It is mandatory to use buffers to store the images at the same time since clock signals of the cameras and DDR2 memory are not synchronized. Therefore while transferring images from cameras BRs are used to buffer the incoming image data. 4 BRs capable of storing one row of intensities of pixels are

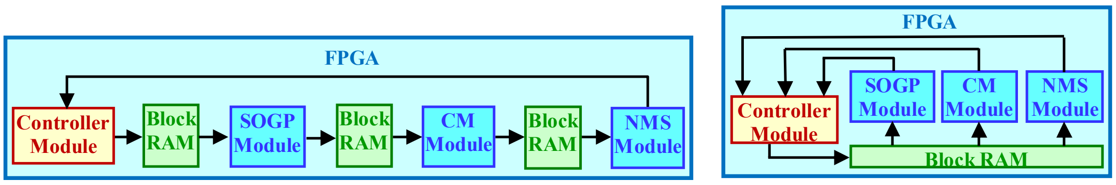

Figure 1. Architecture options to feed the pipeline modules

generated. 2 of them are used to store odd and even rows of the images of left camera and the other 2 BRs are used to store the ones of right camera. The intensity data coming from cameras are written into the odd and even BR buffers consecutively. The content of the BRs filled with intensity data of an image row is transferred into the DDR2 memory by the CMOD. The other details of this interface is not discussed but we just note that the raw data as well as the processed data of the images are stored into the memory in such a way that the time required to read data is minimized. More specifically, the processed and raw data of the pixels in consecutive columns of the same image row are stored into the same row of the memory in column order. Thus while reading data from the memory the number of row access strobe (RAS) latencies are minimized.

The general structure of the implemented design is shown in Fig. 2. Before processing data of stereo images CMOD simultaneously accepts the raw data of the stereo images from the BRs connected to the camera drivers. The raw data of the stereo images are transferred to the MIG module in order to store them in DDR2. After the stereo images are stored i.e. captured in DDR2 CMOD starts to feed the SOGP, CM and NMS modules with data consecutively. At first step, the CMOD reads the raw data of left image from DDR2 with MIG module and transfers the data to the BRs feeding the pipelined modules. The BRs transfer the data to SOGP module and CMOD waits the processed SOGP data from the data buffer in SOGP module. The design includes such data buffers at the end of the pipelined modules to store the processed data temporarily. The CMOD waits until the data buffers are filled with newly processed data ensuring efficient write operations. By efficient write operations we mean that data written into the memory at each write burst contains very high percentage of newly processed data. CMOD writes data taken from the data buffer to DDR2 with MIG module. After SOGP data of all pixels are computed and written into DDR2 CMOD starts to read the SOGP data and fill the BRs feeding the CM module this time. After the processed CM data are written into DDR2 CMOD feeds the NMS module similarly. The pixels of the left image having corner characteristics are determined when the NMS module processes the CM data of the pixels and the coordinates of the detected corners are written to BRs capable of storing the coordinates of 1024 corners for each image. After the coordinates of the corners of the right image are also stored in BRs coordinates of all detected corners are marked in the original images and displayed on a DVI screen with the help of the Chrontel CH7301C driver on the board. Storing the coordinates into BRs will reduce the processing time needed for the following phases of the study.

Having decided the general architecture of the design, the details of the algorithm namely, the mask size and mask coefficients of SOGP and CM modules, *k* constant used in CM module and NMS window size are needed to shape the FPGA hardware accordingly. After empirical tests on the transferred images, we decide to use a 7x7 window in NMS phase and 5x5 masks while computing SOGPs and CMs. In order to minimize resource usage, fixed point data representation more specifically integers are used in the architecture. While computing SOGPs, to decrease the noise, we used masks of which coefficients are modified versions of Gaussian distribution with σ equals to 1.2. In Fig. 3, horizontal and vertical masks are shown on the left and in the middle respectively. The coefficients of the *W* mask used to calculate the CMs are selected to be in Gaussian distribution as Harris proposed. According to the test results, optimum σ of *W* mask is also determined as 1.2. On the right hand side of Fig. 3, the coefficients of the mask whose values are converted into integers are shown. The k constant used to compute corner responses is selected as 0.0625 (1/16) based on tests. Implementing a multiplication with this constant corresponds to a shift operation neither consuming any resources nor adding additional latencies. After determining all the details of the algorithm, 3 pipelined modules are designed.

A. SOGP Module

The first module of the architecture is the SOGP module whose function is to compute $SOGP_{xx}$, $SOGP_{yy}$ *and* $SOGP_{xy}$ of pixels in parallel using pixel intensities of the images with the 5x5 gradient masks. It is designed to be able to get

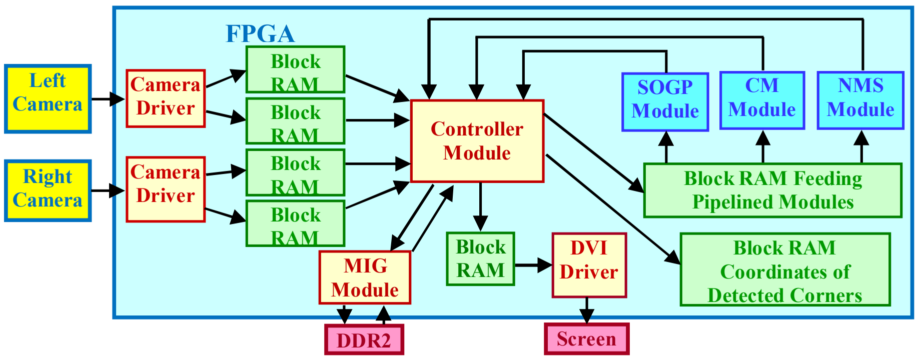

Figure 2. General structure of the implemented design

1	2		2	1
2	6		6	2
3	8		8	3
2	6		6	2
1	2		2	1

1	2	3	2	1
2	6	8	6	2
2	6	8	6	2
1	2	3	2	1

1	2	3	2	1
2	6	8	6	2
3	8	12	8	3
2	6	8	6	2
1	2	3	2	1

Figure 3. The coefficients of the gradient masks and W mask

intensities of 5 pixels located in the same column of 5 consecutive image rows at each CC. To store the intensity data, it has 5 shift registers which we call intensity shift registers and each of these registers has 5 cells to store intensities of pixels in consecutive columns. To maximize the pipeline efficiency of this module, the intensity values of 5 pixels of the same column in consecutive rows of the image should be fed into the each intensity shift register at each CC. To do so internal BRs of FPGA are used as previously mentioned. 6 BRs each of which has the capability of storing one row of intensities composing of 15-bits of pixel data are used. The BRs feeding the pipelined modules are generated in such a way that the number of bits that can be written into them at a CC is more than the number of bits that can be read from them at a CC. Thus it is possible to maximize pipeline occupancy for all of the modules of the pipeline. While processing an image data, firstly, the intensities of the first 5 rows of the image are read from DDR2 memory and written into the BRs by CMOD. After all of the 5 BRs are filled with intensity data, it is started to feed the intensity shift registers in SOGP module with the intensities of the pixels in the consecutive columns in order to calculate the SOGPs of the pixels in the 3rd row of the image. Simultaneously, the unused 6th BR is also filled with the intensities of the 6th row of the image in DDR2 and the outputs of the SOGP module is also written into DDR2. The 6th BR is filled before all of the intensities stored in the first 5 BRs are fed into the SOGP module since data buffers are used to increase the write efficiency and the number of the bits that can be written into the BRs at a CC are greater than the number of bits that can be read from them at a CC. After all the intensities in the first 5 BRs are fed into the SOGP module the intensities of the 6th image row in the 6th BR are fed into the SOGP module with the intensities of 2nd-5th image rows remaining in the 4 BRs in order to calculate the SOGPs of the pixels in the 4th image row. Simultaneously, the unused BR having the intensities of the pixels of the first row is filled with the intensities of the 7th image row and outputs of the SOGP module are written into DDR2. This operation does also finishes before the feeding of SOGP module ends. This routine i.e. simultaneously using 5 of the 6 BRs to feed the pipeline, filling one of them with the intensities of the next row and writing the outputs of the SOGP module into DDR2, continues until all of the intensities are fed into the SOGP module.

In the SOGP module, there is another shift register storing one bit of data in its cells. This shift register, we call reference shift register is used to signal to the CMOD that the output of the SOGP module contains processed and meaningful data. The number of the cells of the reference shift register is equal to 10, CCs needed for a valid input to be processed in the SOGP module. The first cell of the reference shift register is connected to an input of SOGP module and its last cell is connected to one of the outputs of SOGP module. This input is set by the CMOD when the intensity shift registers of the SOGP module are full with meaningful data. The bit set by the CMOD is shifted and the data in the intensity shift registers are processed at each CC. The processed data and the set bit reach to the output simultaneously allowing the CMOD to capture the processed data by checking the output of the SOGP module connected to the last cell of the reference shift register.

The designed SOGP module consists of 10 stages (ST) each of which lasts a single CC. In the first 5 stages, there are 6 parallel sub modules (SM) to calculate the gradients, R_x, R_y, G_x, G_y, B_x and B_y as shown in Fig. 4. Using these gradients, $SOGP_{xx}$, $SOGP_{yy}$ *and* $SOGP_{xy}$ are computed in another SM in the last 5 stages.

The SM responsible for computing R_x is shown in Fig. 5. When the input of the reference shift register is set the intensities of 24 of 25 pixels stored in the intensity shift registers are transferred into the parallel subtraction elements in the first stage of SOGP module. The non-transferred pixel is the center pixel whose SOGPs are being calculated. Its intensity is not required in its own gradient calculations but its content is important since it will be used in SOGP calculation of the next pixel in the next CC.

In Fig. 5, I_R stands for the intensity of the red channel of the transferred pixels. The subsequent numbers respectively show the index of shift registers and the cell of the shift register from where the intensities are transferred. In the first ST, intensity differences are calculated with parallel subtractions. With respect to the horizontal gradient mask, some of the differences are multiplied by constants 2, 3, 6 and 8. Since multiplication with constants 2 and 8 means simple shift operations, these shifts are applied to the relevant differences in the first ST. In the second ST, multiplication operations are performed to the differences which are multiplied by 3 and 6. These multipliers have zero latency and like all the multipliers in the design, they are generated using Xilinx core generator. In the second ST, the differences multiplied with the relevant coefficients in the first ST are started to be summed up by pairs also. No operation (NOP) is performed to the difference not having any pair to be summed. The

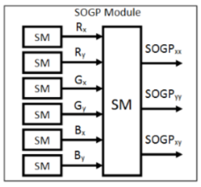

Figure 4. The SMs of the SOGP module

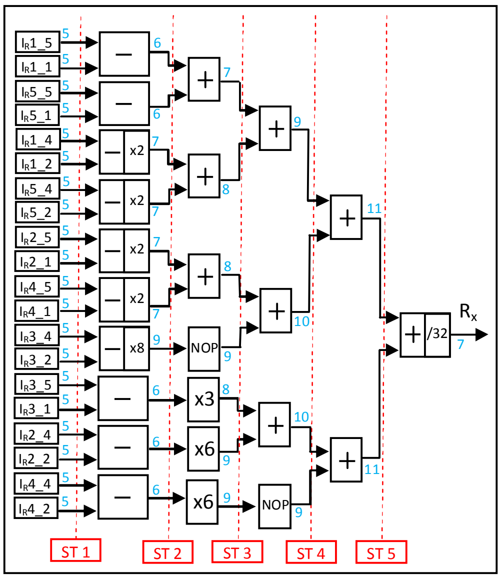
Figure 5. The SM of the SOGP module computing R_x

data without any pair is just stored in another register in the second ST in order to maintain the pipelined structure. The summation operations end in 5 STs. Normally, to compute R_x of a pixel, this sum is divided by 33, the sum of the coefficients of the gradient masks. A divisor element is not embedded since it will require more FPGA resources and add additional stages to the designed module. Instead the least significant 5 bits of the sum is just ignored actually corresponding to a division operation by 32. The shift operation decreases the number of bits to be processed and does not affect the performance of the algorithm. With R_x, all the other gradients R_y, G_x, G_y, B_x and B_y are computed in 5 STs in other parallel SMs.

In the 6th ST of the SOGP module, the computed gradients are transferred into the SM computing SOGPs of pixels as shown in Fig. 6. To compute SOGPs, all gradients are fed into the parallel multipliers with latency of 3 CCs. The products produced by the multipliers are summed by pairs in 9th and 10th STs and $SOGP_{xx}$, $SOGP_{yy}$ and $SOGP_{xy}$ are generated. These 15-bit SOGPs are forwarded to the CMOD which is informed by the output of reference shift register that meaningful SOGPs are available at the output of SOGP module. In the CMOD, SOGPs are stored in 256 bit registers. When the register becomes full all of its content is written to DDR2 maximizing the efficiency of the write operation as discussed before. After all SOGPs of first image are generated and written into DDR2 the intensities of the pixels of the second image are fed into the SOGP module and SOGPs are written into DDR2.

B. CM Module

The function of the CM module is to compute the CMs of the pixels using the SOGPs computed in the previous phase

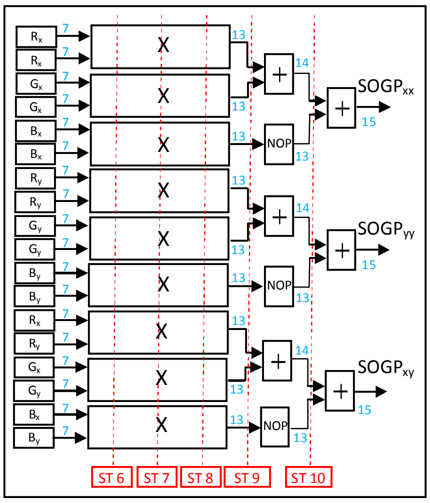
Figure 6. The SM of the SOGP module computing $SOGP_{xx}$, $SOGP_{yy}$ and $SOGP_{xy}$ of pixels

The structure of the CM module is similar to the structure of the SOGP module. It is designed in such a way that at each CC, it is capable of receiving SOGPs of 5 pixels located in the same column of 5 consecutive image rows. In CM computation, 5x5 Gaussian mask is used. Therefore 5 shift registers with 5 cells to store and shift SOGPs are used in the module. These shift registers we call SOGP shift registers are similar to the intensity shift registers used in SOGP module but each of its cell has a capacity of 45-bit this time. In order to feed these registers BRs are used as in the feeding structure of SOGP module. 6 of the BRs used in the previous phase are also used in this phase with 12 new BRs. While computing CMs of the pixels in a row, 15 of these BRs are used to feed the CM module and 3 of them are filled with the new SOGPs. Thus it becomes possible to feed CM module with 15x15=225-bits of SOGP data at each CC. Moreover another reference shift register consisting of 11 STs is used in CM module.

In the first 6 stages of the CM module, A, B and C values of pixels are computed in 3 SMs in parallel as shown in Fig. 7. In the next 5 stages, CMs of the pixels are computed using these values. The SM computing A of the pixels is shown in Fig. 8. In its first stage, $SOGP_{xx}$ values of the pixels stored in SOGP shift registers are taken. The values to be multiplied by 3, 6 and 12 according to the Gaussian W mask are multiplied by these constants in 0 latency multipliers. The other values to be multiplied by the multiples of 2 are summed by pairs and multiplications are performed with shift operations without any latency. From

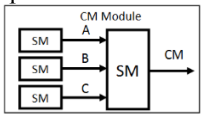

Figure 7. The SMs of the CM module

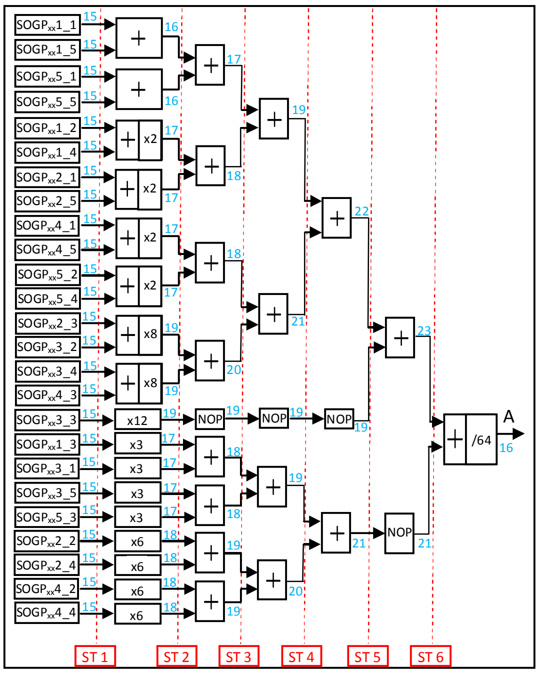

Figure 8. The SMs of the CM module

2nd ST to the end of 6th ST all the multiples of $SOGP_{xx}$ values are summed by pairs to generate the A value of the pixels. In the 6th ST, 24 bit A value is generated. However, only its least significant 21 bits and sign bit are meaningful because it is not possible to obtain quantities represented more than 22 signed bits by the multiplication of signed 15-bit of SOGPs by unsigned decimal 100, the sum of the coefficients of the W mask. Since we do not want to embed division elements the designed module calculates A value by shifting the 22 meaningful bits 6 times corresponding to a division by 64. Thus 16-bit A value of pixels is generated in 6 STs. There are 2 more parallel mirrors of this module computing B and C of the pixels in 6 STs using $SOGP_{yy}$ and $SOGP_{xy}$ of pixels.

After A, B and C values of the pixels are computed SMs of *CM* module they are fed into the other SM of it responsible to compute *CM* of the pixels using (2). To do so A, B and C values are transferred into the SM shown in Fig. 9. In the ST 7, the multiplication of the transferred values starts to compute A*B and C2 values of the pixels. Furthermore the sum of A and B values is generated to be used in the multiplication elements computing $(A+B)^2$

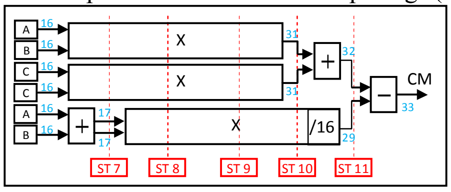

Figure 9. The SM of the CM module computing *CM* of pixels

starting in the 8th ST. All the multiplication elements used in this SM have 3 CC of latencies. In the ST 10, the computation of the A*B and C^2 values of the pixels finishes and they are summed up. The multiplication of (A+B) finishes before ST 11. Since the k constant used in HCD algorithm is selected as 1/16 no hardware resources are used to implement the multiplication with k. Instead this multiplication is performed by ignoring the least significant 4 bits of the value $(A+B)^2$. In the ST 11, the subtraction of $(A+B)^2$ of the pixels from $A*B+C^2$ is applied. Since the pixels with positive CM values greater than a specified threshold are considered as corners according to the HCD algorithm the generated CMs are checked before they are written to the data buffers located in the CMOD. If the generated CMs are negative or smaller than the specified threshold 32-bit decimal 0 is written into the data buffers. Otherwise 33-bit CM is written to the data buffers by ignoring the sign bit. After the data buffers are filled their content is written into the DDR2 memory. After all SOGPs of the stereo images are fed into the CM module and CMs are written back to the DDR2 memory second phase of the algorithm is completed.

C. NMS Module

The function of the NMS module is to apply NMS to the generated CMs in order to select the pixels having maximum corner characteristics in an image window of 7x7. To increase pipeline efficiency, shift registers are used in NMS module as in the other modules. 7 shift registers each of which have 7 cells to store 32-bit CMs are used in NMS module. The BRs feeding the other pipelined modules are also used to feed these shift registers in NMS module, we call NMS shift registers. 16 of these 18 previously generated BRs are used to feed NMS module. When NMS module is active 14 of the 16 BRs feed the module and 2 of them are used to store the CMs of the consecutive rows.

In the NMS module, 48 parallel comparators are embedded. At each CC, the module is capable of accepting 7 CMs of the pixels in the same column and in consecutive image rows to store them in its NMS shift registers. While detecting the corners of each image row, the CM of the pixel in the center of the NMS window is compared with the CMs of the other pixels in the NMS window. The row and column number of the detected corners in left and right images are stored in separate BRs each of which are capable of storing the data of 1024 corners. The capability of these BRs is quite sufficient in practice since the maximum number of the corners detectable in a 640x480 image with a 7x7 NMS window is approximately 19,200. Even if the number of the detected corners exceeds the capacity of the BRs the system will not collapse. Only the first 1024 corners will be stored into the BRs and the others will be ignored.

V. Resource Utilization

As well as the resource utilization of the pipelined SOGP, CM and NMS modules presented in detail, the resource utilization of the other modules in the design is presented in Table 1. According to the table the pipelined modules and their feeding structures occupies approximately 17% of the total registers, 14% of total LUTs, 25% of the DSP units and 38% of the BRs of the XC5VLX50 FPGA. On the other hand the CMOD managing the operations in the design occupies approximately 32% of the total registers and 49% of total LUTs of the FPGA. All of the modules in the design consumes more than the half of the BRs on the FPGA. Therefore it is not possible to implement a feeding structure in which the entire pipeline modules are fed by distinct BRs as in the first architecture option discussed before. On the contrary 37.50% of the BR resources are reutilized to feed the pipeline modules one after another while processing the stereo images as indicated in the second architecture option. According to the utilization results, only 25% of the DSP48E hardware resources are used in the pipeline modules. Since DSP48E resources can be used instead of the LUT resources used in the pipelined modules it is possible to design the architecture having similar performance characteristics with less LUT resources.

VI. Performance

The performance of the designed pipelined architecture processing stereo images is presented in Table 2. According to the implementation results, the designed SOGP, CM and NMS modules have maximum operating frequencies of 445MHz, 387MHz and 233MHz respectively. By increasing the number of the pipeline stages of the modules, it is possible to achieve higher operating frequencies. However, this will increase the resource utilization and number of CCs needed for execution. The increase in CCs needed for execution will not reduce the total execution time since the operating frequency will also increase. However, the increase in resource utilization will decrease the available resources for the following parts of the study. Therefore we decide that the operating frequencies of the modules are high enough for our purpose.

The maximum operating frequency of the whole design is 123MHz. Therefore in the implementation, we select a 100MHz of system clock, less than the maximum operating frequency of the whole design and all of the operating frequencies of the pipelined modules individually. In the architecture, execution of the stereo images in 640x480 resolution takes 1,856,550CCs on average. Therefore the designed architecture is capable of processing 1,856,315/(2x640x480) ≈ 0.33ppcc. This corresponds to an execution time of 18.57ms with the 100MHz system clock used. The execution time of a single image is equal to 9.29ms, half of the time needed to process stereo images. If a 123MHz clock signal is used instead of 100MHz signal used in the implementation the execution time of the stereo images will be reduced to 15.10ms and the execution time of a single image will be reduced to 7.05ms.

While the images are being processed in the architecture the pipelined modules are only active i.e. occupied with data when they are fed by the CMOD. The pipeline occupancies of the modules when they are active are shown in Table 2. If the time required for pre-charge and auto refresh commands is ignored it is possible to read or write 128 bits of data with MIG core. 15, 45 and 32 bits of data are read from DDR2 in SOGP, CM and NMS phases and 45 and 32 bits of data are written into DDR2 in SOGP and CM phases respectively. Therefore a total of 60, 77 and 32 bits of data transfer capabilities are needed in SOGP, CM and NMS phases respectively. These values are smaller than the available 128 bits of data transfer capability provided by MIG module. Thus when the modules are active it is possible to achieve high pipeline occupancies over 97% stated in Table 2. Although such high pipeline occupancies are achieved pipeline

TABLE I. Resource Utilization

	Register	Register Utilization	LUT	LUT Utilization	DSP48E	DSP48E Utilization	Block RAM	Block RAM Utilization
SOGP	1306	4.53%	1283	4.45%	9	18.75%	18	37.50%
CM	2343	8.14%	1966	6.83%	3	6.25%		
NMS	1346	4.67%	777	2.70%	0	0.00%		
AA7720 Drivers x2	204	0.71%	837	2.91%	0	0.00%	4	8.33%
MIG	2129	7.39%	1590	5.52%	0	0.00%	3	6.25%
DVI Driver	107	0.37%	421	1.46%	0	0.00%	0	0.00%
Block RAMs Coordinates of Detected Corners	0	0.00%	0	0.00%	0	0.00%	2	4.17%
Controller Module	9294	32.27%	14182	49.24%	0	0.00%	0	0.00%
Total	16729	58.09%	21056	73.11%	12	25.00%	27	56.25%
Available in XC5VLX50	28800	-	28800	-	48	-	48	-

TABLE II. PERFORMANCE

	Operating Frequency wrt Synthesis Results	Execution Time with 100MHz Clock Signal	Pipeline Occupancy When the Module is Active	Pipeline Occupancy Along the Whole Process
SOGP	445MHz	6.24ms	97.98%	32.92%
CM	387MHz	6.16ms	99.24%	32.91%
NMS	233MHz	6.17ms	98.64%	32.77%
Total	-	18.57ms	-	-

occupancies of the modules along the whole process are less. The pipeline occupancies along the whole process are approximately one-third of the pipeline occupancies when they are active since pipelined modules are not fed with distinct BRs. As stated before the limited resources of XC5VLX50 do not permit to construct such kind of architecture having maximum pipeline characteristics.

According to the implementation results, we show that the designed architecture compares favorably to similar architectures. To illustrate, while the architecture of Claus et al. [12] pipelining SUSAN corner detection algorithm processes each image in 640x480 resolution with a clock signal of 100MHz in 3.142ms, with the same clock frequency, our architecture is capable of implementing a more successful corner detection algorithm in 9.29ms to the images in the same resolution. Moreover it is possible to use higher clock frequencies to achieve shorter processing time with our architecture. Our architecture is capable of processing 0.33ppcc, which is bigger than the processed 0.088ppcc by Teixeira's et al. [7] GPU implementation of HCD.

VII. CONCLUSION

In this paper, we present an optimized and completely pipelined FPGA architecture to implement HCD to stereo colored images. While designing the architecture, we plan to achieve the maximum performance using minimum internal resources of FPGA and minimum external memories which will make the design suitable to be used in mobile robots of which cost are low. The designed architecture needs only a single DDR2 memory and uses 100MHz of system clock in order to achieve real time corner detection performance for RGB555 colored images with 640x480 resolution. To implement the architecture composing of 3 pipelined modules, 25% of the resources of the small XC5VLX50 FPGA of Xilinx is sufficient. The architecture is capable of processing 0.33ppcc. With the 100MHz clock signal used in the tests, we achieved a total processing time of 18.57ms for stereo images. Moreover with the designed architecture, it is possible to achieve a total processing time of 7.05ms for a single image using a system clock of 123MHz, the maximum clock frequency of the system with respect to implementation results.

For a future work of this study, we plan to construct pipelined stereo matching and 3D distance measurement modules to determine the position of the corners with respect to the stereo vision system. 3D distance measurement property is planned to be used on a mobile robot carrying out simultaneous localization and mapping in an indoor environment. In stereo matching, we plan to use the feature based stereo matching algorithm of Barnard [13]. Since the row and column numbers of the detected corners are written into the BRs in row order it will be possible to design an efficient pipelined architectures for stereo matching.

REFERENCES

[1] F. Mokhtarian and F. Mohanna, “A performance evaluation of corner detectors using consistency and accuracy measures,” Computer Vision and Image Understanding, vol.102, 2006, pp. 81-94.

[2] C. Harris and M. Stephens, “A Combined Corner and Edge Detector,” Alvey Vision Conf., 1988, pp. 147–151.

[3] S. M. Smith and J. M. Brady, “Susan - a new approach to low level image processing,” International Journal of Computer Vision, vol. 23, no. 1, pp. 45–78, May 1997.

[4] W. Wang and R. Dony, “Evaluation of image corner detectors for hardware implementation,” Electrical and Computer Engineering, 2004. Canadian Conference on, vol. 3, pp. 1285–1288, May 2004.

[5] L.-h. Zou, J. Chen, J. Zhang and L.-h. Dou, “The comparison of two typical corner detection algorithms,” Second International Symposium on Intelligent Information Technology Application, 2008, pp. 211–215.

[6] P. Tissainayagam and D. Suter, “Assessing the performance of corner detectors for point feature tracking applications,” Image and Vision Computing, vol. 22, 2004, pp. 663–679.

[7] L. Teixeira, W. Celes and M. Gattass, “Accelerated corner-detector algorithms,” 19th British Machine Vision Conference, 2008, pp. 625–634.

[8] F. Hosseini, A. Fijany, and J.-G. Fontaine, “Highly parallel implementation of Harris corner detector on CSX SIMD architecture,” 4th Workshop on Highly Parallel Processing on a Chip, 2010, pp. 58-67

[9] B. Dietrich, “Design and implementation of an FPGA-based stereo vision system for the EyeBot M6,” University of Western Australia, 2009.

[10] H. Moravec, “Obstacle avoidance and navigation in the real world by a seeing robot rover,” Tech Report CMU-RI-TR-3, Carnegie-Mellon University, Robotics Institute, September 1980.

[11] P. Montesinos, V. Gouet, and R. Deriche, “Differential invariants for color images,” 14th International Conference on Pattern Recognition, 1998, pp. 838–840.

[12] C. Claus, R. Huitl, J. Rausch and W. Stechele, “Optimizing the SUSAN corner detection algorithm for a high speed FPGA implementation,” International Conference on Field Programmable Logic and Applications, 2009, pp. 138-145.

[13] S. T. Barnard and W. B. Thompson, “Disparity analysis of images,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-2, no. 4, pp. 333-340, July 1980.