International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:1, 2015

High Level Synthesis of Canny Edge Detection Algorithm on Zynq Platform

Hanaa M. Abdelgawad, Mona Safar, Ayman M. Wahba

***Abstract*—**Real time image and video processing is a demand in many computer vision applications, e.g. video surveillance, traffic management and medical imaging. The processing of those video applications requires high computational power. Thus, the optimal solution is the collaboration of CPU and hardware accelerators. In this paper, a Canny edge detection hardware accelerator is proposed. Edge detection is one of the basic building blocks of video and image processing applications. It is a common block in the pre-processing phase of image and video processing pipeline. Our presented approach targets offloading the Canny edge detection algorithm from processing system (PS) to programmable logic (PL) taking the advantage of High Level Synthesis (HLS) tool flow to accelerate the implementation on Zynq platform. The resulting implementation enables up to a 100x performance improvement through hardware acceleration. The CPU utilization drops down and the frame rate jumps to 60 fps of 1080p full HD input video stream.

***Keywords*—**High Level Synthesis, Canny edge detection, Hardware accelerators, and Computer Vision.

I. Introduction

REAL time processing of image and video applications is essential for functions such as image segmentation, object recognition, and feature tracking. Those functions help in making critical decisions in many applications e.g. video surveillance, machine vision and medical imaging. The preparation phase in many video and image processing is to extract the edges from the image. The edge detection is applied to an unexpected enormous number of edges, thus the processing time used for this operation cannot be expected and become a challenge. Besides, the processing itself in real time is complex and usually demands high computational power. Therefore, processing of real time video and image algorithms is not capable of running on devices with low computing power at a reasonable frame rate.

The optimal solution is the collaboration of CPU and hardware accelerator to offload the computational intensive tasks to the hardware. This solution provides significant advantages in run-time speed and energy [1]. In this way, pixel based image processing and feature extraction are processed in hardware domain as hardware accelerators, while frame based feature processing and decisions are done in software domain.

In this paper, we study Canny edge detection, then apply the HLS constraints and optimization techniques on this algorithm. HLS is an automated design process that deals with the generation of behavioral hardware descriptions from high-level algorithmic specifications. This approach enables the automatic synthesis of high-level, untimed or partially timed specifications (such as in C or SystemC) to a low-level cycle-accurate register-transfer level (RTL) specifications for efficient implementation in ASICs or FPGAs. HLS raises the level of abstraction for hardware design nearer to that of software design. It enables the validation of the functional correctness of the design more quickly than with traditional hardware description languages.

The computation of the Canny algorithm is implemented as software core and hardware core. The algorithm is applied on live videos or Test Pattern Generator (TPG) videos using the Zynq-based Targeted Reference Design (TRD) provided by Xilinx. The TRD Qt GUI provides the option to switch between the two implemented cores: software and hardware. The implementation is synthesized and tested on Zynq-7000 AP SoC ZC702 board.

The rest of the paper is organized as follows. Section II discusses the previous approaches of implementing image and video processing algorithms on FPGA and the related work using HLS. Section III briefly illustrates Canny edge detection algorithm explored in this work. Section IV shows our implementation of Canny on ZYNQ platform using Vivado HLS. The achieved results are discussed in Section V. Conclusions are offered in Section VI.

II. Related Work

Developments in HLS attract many software and hardware designers to enhance the implementation of different solutions. Evolutionary hardware design implementation on Zynq is discussed in [2]. Traffic management is one of the tracks that are delivered in [3] and [4] to proof the concept of implementing real time video solutions on hardware. High level synthesis implementation of the Sobel filter is clarified in [5] with the optimization techniques used. Besides, power evaluation of Sobel filter on Xilinx platform is introduced in [6]. Feature extraction is implemented in [7] on FPGA using Xilinx system generator. The implementation of high-performance, low-power FPGA-based optical flow accelerators in C is shown in [8].

Hanaa M. Abdelgawad is a M.Sc. degree student at Computer and Systems Department, Faculty of Engineering, Ain Shams University, Cairo, Egypt (e-mail: hanaa.mabdelgawad@gmail.com).

Mona Safar is Lecturer and Researcher at Computer and Systems Department, Faculty of Engineering, Ain shams university, Cairo, Egypt (e-mail: mona.safar@eng.asu.edu.eg).

Ayman M. Wahba is Vice Dean for Education and Student Affairs. Professor at Computer and Systems Engineering Department, Ain Shams University, Cairo, Egypt (e-mail: ayman.wahba@eng.asu.edu.eg).

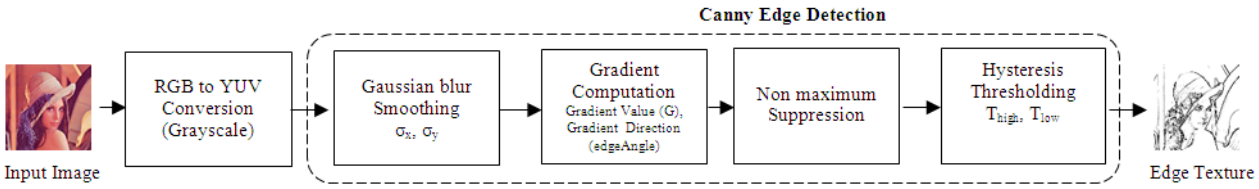

Fig. 1 Canny Edge Detection Flow

Sadri in [1] provides a clear knowledge about the AXI interfaces in Zynq platform with more details about Accelerator Coherency Port (ACP) to improve energy and performance of applications. Canny implementation was synthesized on three different Xilinx FPGAs: Spartan-3E, Spartan6, and Virtex 5 FPGA in [9] by using the Xilinx ISE 12.1 toolchain. The implementation of a hardware accelerator for edge detection based on the Canny edge filter is described in [10]. In this paper, Canny edge detection hardware accelerator is implemented using HLS as IP core on Zynq platform.

III. Canny Edge Detection

Canny edge detection is one of the most widely-used edge detection algorithms due to its reliable performance with noisy images. The computationally intensive nature of this algorithm imposes high clock frequencies and significant power consumption on general microprocessor architectures, especially when it is necessary to meet real-time constraints.

Canny edge detector is a multi-step detector [10]. Canny edge detector stages are shown in Fig. 1 to determine the edge texture. Canny edge detector performs smoothing and filtering from noise by Gaussian blur mask. It is followed by convolving the image with partial derivatives of a 2D Gaussian function Gx and Gy to get the edge strength matrix G and the edge direction edge Angle matrix. G and edge Angle matrices are used in non-maxima suppression operation.

The local maximum is found by comparing the pixel with its neighbors along the direction of the gradient. The pixel that has no local maximum gradient magnitude is eliminated. The comparison is made between the actual pixel and its neighbors, along the direction of the gradient. It is applied to remove any unwanted pixels which may not part of the edge. Hysteresis thresholding is the last stage where two threshold, high and low, are used. Edge pixels stronger than the high threshold are marked as strong; edge pixels weaker than the low threshold are suppressed and edge pixels between the two thresholds are marked as weak. Those operations will give a thin edge in the output image.

IV. Implemented Canny Edge Detector on ZYNQ Platform

The HLS implementation of Canny on Zynq platform requires the knowledge of the algorithm, the challenges of implementing embedded Canny edge detection using HLS tool and the knowledge of the target Zynq platform [11].

A. HLS Implementation Challenges

HLS coding differs from the normal C coding to target the embedded domain. Available tools require coding in a specific style, with additional constraints inside files or within the code as pragmas. In the implementation of Canny hardware accelerator using Vivado HLS, different HLS constraints and optimizations are used targeting ZYNQ platform. Those stages are illustrated as follow:

1. Grayscale Conversion

Since the Canny algorithm is processed on Grayscale images, the input frame image in a RGB or BGR color format is converted to the Y'UV color space (RGB → Y'UV). It is done using HLS video libraries: cvtcolor function and two other supported functions chroma downsampling and chroma upsampling. The processed Y'UV image applies a filter to convert it back to RGB. The luminance intensity conversion is a sum of different weight of each color component of R, G, and B which results in same gray scale level. Y'UV has an advantage that some of the information can be discarded in order to reduce bandwidth. The accuracy of the brightness information of the luminance channel has higher impact on human eye than the spatial sensitivity to color.

2. Memory Structures

The biggest bottleneck, in the computational intensive image and video applications, is the memory accesses and memory buffers. Canny edge detector is considered as multiple video pipeline in which live video streams are written into memory (input), then processing is done through different stages and memory is read to send out live video streams (output). Through Canny stages, the convolution filters operate on a local neighborhood of pixels, called memory window. Those filters require accessing a pixel more than once.

Thus, handling data stream on an FPGA requires a memory architecture to retain data for multiple accesses. An efficient memory architecture for streaming data uses a combination of line buffers for storage of complete image lines and memory windows for the actual processing of local neighborhoods. Fig. 2 shows window-buffer memory architecture used on the input frame image. The line buffer stores the new coming image data, which is then used to update the memory window. The window moves to the right and all of its contents are shifted to the left to update it with the new data. The pixels of the rightmost column are obtained from the current column of the line buffer as well as the new input element.

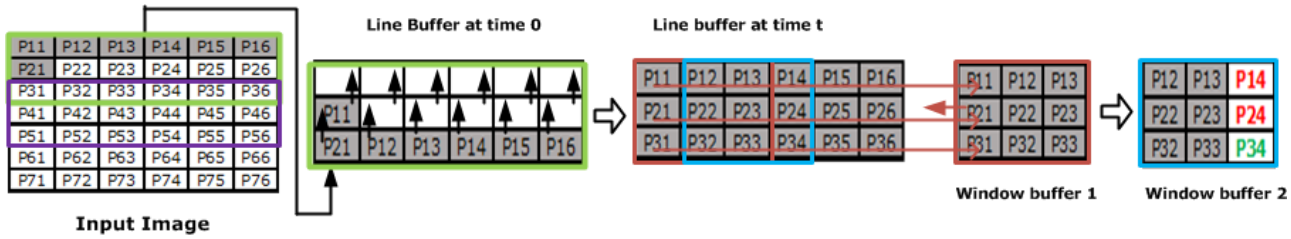

Fig. 2 Line buffer and window buffer used in the convolution operation

3. Loops Optimization Using Directives

In order to apply the computational intensive pixel tasks like Gaussian blur, Gradient filter and non-maxima suppression, loops are used to process the pixel with its neighborhood pixels. There is a HLS constraint of writing loops: only perfect and semi-perfect loops [12] are allowed to be automatically synthesized in HLS. To enhance the performance of those implemented operations, the loop optimization directives are used to direct the generated RTL in a certain way.

For the Gaussian blur operation, a built-in HLS video library is used in the hardware implementation core (hls::Gaussianblur <5,5> (img,dst)). While in software implementation, the 5x5 filter is implemented.

For the Gradient operation and the non-maxima suppression stage, input frame image is processed on the whole window buffer using loops. The Gradient strength and the Gradient direction are saved as struct data type called edgeValues. edgeValues fields are edge angle and edge strength, which are used in the non-maxima suppression. Listing 1 shows an example of the optimized nested for loop used in the algorithm.

LISTING 1 OPTIMIZED HLS LOOP

```
for(int row = 0; row < rows+1; row++){
    for(int col = 0; col < cols+1; col++){
    #pragma HLS loop_flatten off
    #pragma HLS dependence variable = &buff_A false
    #pragma HLS PIPELINE II = 1
    //Image processing Code
    }
}
```

Without optimization directives on the loop, each loop iteration will run in the same hardware state and each loop iteration will run on the same hardware resources. Thus, loops enforce a minimum execution latency depending on the loop condition which is, in our case, (Width x Height) of the input image frame. Incrementing the loop counter always consumes 1 clock cycle, therefore the nested for-loop in the gradient filter will always take at least (Width x Height) clock cycles. Different optimization directives are used to enhance the performance of the algorithm.

- **Pipeline Directive** Loop pipeline causes inputs to be passed to the function or loop more frequently. This decreases latency for the entire loop and increases throughput. Pipeline directive uses Loops Initiation Interval (II) factor, it is used to define the number of clock cycles between start of new loop body.
  #pragma HLS PIPELINE II = 1

In our implementation, the gradient filter and non-maxima suppression loop operation uses II=1, which means that one loop body per clock cycle. Thus, the filter is a 'fully pipelined' datapath.

- **Flatten Directive** Perfect and semi-perfect loops are automatically flattened, flattening eliminates state transitions between loop hierarchy levels and a loop state transition (counter increment) takes 1 clock cycle. In our case, we use the option to turn off the automatic flattening.
- **Unroll Directive** Unroll for-loops to create multiple independent operations rather than a single collection of operations. In the nested for loop, It is used in the inner loop. It is used in loop iterations like this

#pragma HLS UNROLL

- **Dataflow Directive** is used at the top-level function in the algorithm to allow sequential loops to operate concurrently.

All those directives in turn decreases latency and improves the throughput of the RTL design, but at the cost of additional hardware. Xilinx Vivado HLS provides optimized video HLS library that includes embedded OpenCV library and Video interfaces [13]. Not all functions in OpenCV library are provided in the embedded form, the available functions provide the designer with a memory infrastructure and specific algorithm implementations. The generated solution is only as good as the provided C code. Therefore, the C code has to be optimized for hardware use, to improve the quality of the hardware solution [14].

B. Zynq Architecture

The hardware platform used is Zynq 7000 AP SoC image and video kit. XC7Z020 board consists of dual 1GHz core ARM Cortex-A9 processor and a programmable logic FPGA region (PL). PS comes with a high performance memory system of 512Mb RAM, which is large enough for the video applications. PL includes 140 on chip static RAM modules, called Extensible Block RAM (BRAM). Each has a capacity of 36 Kb, this gives a total of 560KB of on chip static RAM available in the FPGA.

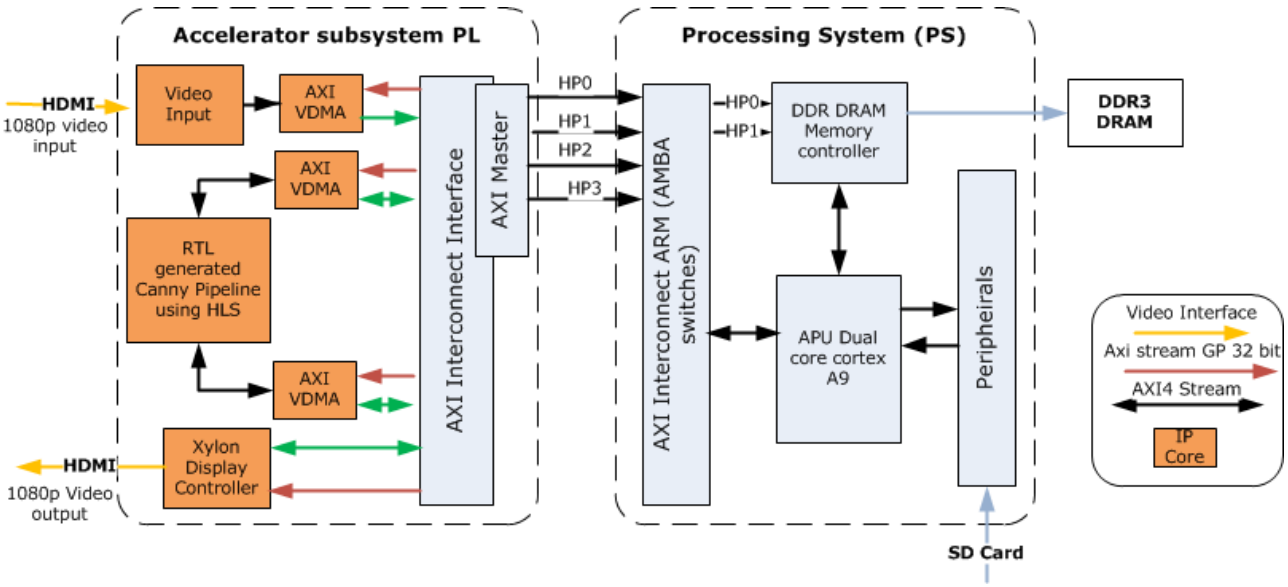

Fig. 3 The block diagram of the system using Zynq-7000 operations

The BRAM cells are designed as dual-port RAM and thus can be read and written simultaneously. DDR3 DRAM is configured to run at 533 MHz, and the AXI interface is running at 355 MHz. This provides a good hardware solution for the memory problem.

In the system block shown in Fig. 3, the input image frame received from the live video through AXI VDMA, then it is filled in openCV matrix of size equal to the frame rows and columns. AXI VDMA implements a high-performance, video-optimized DMA engine with frame buffering. AXI VDMA transfers video data streams to and from memory. The video pipeline is running at 1080p60 (1920 x 1080 frames at 60 frames/sec). Each frame consists of 4 bytes per pixel to represent a high-quality video stream such as RGBA or YUV 4:4:4. ZYNQ communicates using AXI interface which is part of Advanced Microcontroller Bus Architecture (AMBA) Open Standard Interconnect. AMBA is a high bandwidth interconnect between PS and PL.

There will be interface conversion to convert the input matrix to AXI video stream where the image frame inside HLS generated Canny edge detection IP core can be processed. HLS video libraries provide different interfaces. In this work, cvMat2AXIvideo, AXIvideo2cvMat, hls::AXIvideo2Mat and hls::Mat2AXIvideo are used.

To provide high-speed AXI master interfaces in the PL with lower latency and direct access to the Zynq-7000 AP SoC memory interfaces, connections to the high performance (HP) interfaces are required. The Zynq-7000 AP SoC contains four HP interfaces that are 64-bit slave interfaces designed for high throughput. This architecture allows the basis of video systems capable of handling multiple video streams and multiple video frame buffers sharing a common DDR3 SDRAM memory.

Data read by each AXI VDMA is sent to Xylon Display controller capable of multiplexing or overlaying multiple video streams to a single output video stream. The output drives the HDMI video display interface on the board. All AXI VDMAs are connected to separate HP interfaces by means of the AXI Interconnect and are controlled by the Cortex-A9 processor.

V. Results and Evaluation

Vivado HLS facilitates the implementation of the embedded Canny edge detector algorithm as the focus is on the embedded Canny detector algorithm away from too much details of the hardware. Vivado HLS accelerates system verification and implementation times by up to a 100x compared to RTL design entry flows.

Zynq-based TRD [15] helps in testing the implemented core on live videos. The TRD QT GUI [16] interface shows the implemented Canny edge detector, the software core (implementation on PS) and the hardware core (RTL generated Canny pipeline using HLS), on live video or TPG video. The CPU utilization chart in Fig 4 assures that the performance is improved up to 100x through using hardware accelerator. One of CPU core is dedicated to handle the edge detector SW core and the other CPU handles the GUI interface. Table I shows the utilization estimates of Canny hardware accelerator using Vivado HLS 2014.2 on ZC702 board.

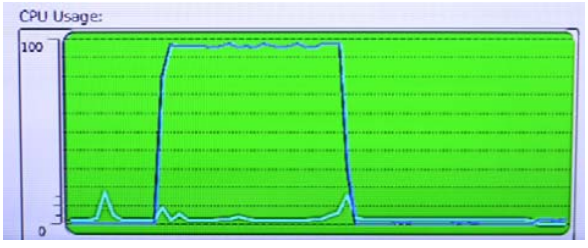

Fig. 4 CPU Usage for Canny hardware accelerator

TABLE I
UTILIZATION ESTIMATES OF CANNY HARDWARE ACCELERATOR

	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	6
FIFO	0	-	165	735
Instance	21	79	26393	34129
Register	-	-	23	-
Utilization (%)	7	35	24	65

VI. CONCLUSION AND FUTURE WORK

In this paper, we show how HLS constraints and optimization directives were applied for timing and area optimization. The implementation of stream-based Canny edge detector processing using C-based HLS is presented. The results show that hardware accelerators enhance the complex computation of the processing functions. The hardware accelerators on FPGA enhance the computational performance: the CPU utilization drops down and the frame rate increases, in ZYNQ platform it is up to 60 fps for a resolution of 1280 x 1024.

There are many computer visions application which can take advantage of hardware accelerators to enhance performance of real-time highly computational applications. When targeting HLS design flow, the implementation of C/C++ code is rapidly developed for hardware accelerator.

In the Future work, besides improving the quality of edge texture map result by distributed Canny edge detection algorithm [17]. Implementation of augmented reality (AR) pipeline [18] is considered to make use of the cooperation between CPU and FPGA. Highly computational video and image processing operations of augmented reality will be as hardware accelerators. This will enhance the real time performance of AR applications.

REFERENCES

[1] Mohammadsadegh Sadri, Christian Weisy, Norbert Wehny, and Luca Benini: "Energy and Performance Exploration of Accelerator Coherency Port Using Xilinx ZYNQ", FPGAWorld '13, September 10-12, Copenhagen, and Stockholm, ACM
[2] Dobai, R.; Sekanina, L.: "Image filter evolution on the Xilinx Zynq Platform," Adaptive Hardware and Systems (AHS), 2013 NASA/ESA Conference on , vol., no., pp.164,171, 24-27 June 2013
[3] Russell, M.; Fischaber, S., "OpenCV based road sign recognition on Zynq," Industrial Informatics (INDIN), 2013 11th IEEE International Conference on , vol., no., pp.596,601, 29-31 July 2013
[4] Yan Han; Oruklu, E., "Real-time traffic sign recognition based on Zynq FPGA and ARM SoCs," Electro/Information Technology (EIT), 2014 IEEE International Conference, pp.373,376, 5-7 June 2014
[5] Josh Monson, Mike Wirthlin, Brad L Hutchings: "Optimization Techniques for a High Level Synthesis Implementation of the Sobel Filter"
[6] Hong. Nguyen. T. K, Cecile. Belleudy1 and Tuan. V. Pham2: "Power Evaluation of Sobel Filter on Xilinx Platform".
[7] Swapnil G. Kavitkar,Prashant L. Paikrao: "FPGA based Image Feature Extraction Using Xilinx System Generator", (IJCSIT) International Journal of Computer Science and Information Technologies,2014
[8] Monson, J.; Wirthlin, M.; Hutchings, B.L., "Implementing high-performance, low-power FPGA-based optical flow accelerators in C," Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference, 5-7 June 2013
[9] Christos Gentsos, Calliope-Louisa Sotiropoulou and Spiridon Nikolaidis:"Real- Time Canny Edge Detection Parallel Implementation for FPGAs"
[10] Chaithra.N.M., K.V. Ramana Reddy,"Implementation of Canny Edge Detection Algorithm on FPGA and displaying Image through VGA Interface", International Journal of Engineering and Advanced Technology (IJEAT), ISSN: 2249 8958, Volume-2, Issue-6, August 2013
[11] Fernando Martinez Vallina, Christian Kohn, and Pallav Joshi, "Zynq All Programmable SoC Sobel Filter Implementation Using the Vivado HLS Tool", XAPP890 (v1.0) September 25, 2012.
[12] Louise H Crockett, Ross A Elliot, Martin A Enderwitz, Robert W Stewart The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc Paperback, July 14, 2014
[13] Kester Aernoudt, "OpenCV, Zynq All Programmable SoC, and Vivado HLS",Xilinx,June, 2013
[14] Xilinx,"How to Accelerate OpenCV Applications with the Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries",August 28, 2013
[15] Xilinx: Zynq Base TRD Wiki http://www.wiki.xilinx.com/Technical +Articles#TRD
[16] UG925 (v7.0) Zynq-7000 All Programmable SoC ZC702 Base Targeted Reference Design (Vivado Design Suite 2014.2) User Guide, August 27,2014
[17] Qian Xu, Srenivas Varadarajan, Chaitali Chakrabarti, and Lina J. Karam,"A Distributed Canny Edge Detector: Algorithm and FPGA Implementation", IEEE Transactions on Image Processing, Vol. 23, No. 7, July 2014
[18] Bernardo Reis, Paulo Borges, Luis Arthur Vasconcelos, Jo˜ao Marcelo Teixeira,Veronica Teichrieb, Judith Kelner," MarkerMatch: an Embedded Augmented Reality case study", XII Symposium on Virtual and Augmented Reality, Natal, RN, Brazil - May 2010