Lowering Dynamic Power in Stream-based Harris Corner Detection Architecture

Siew-Kei Lam
School of Computer Science and Engineering
Nanyang Technological University,
Singapore
siewkei_lam@pmail.ntu.edu.sg

Rakesh Kumar Bijarniya
Department of Electrical Engineering
Indian Institute of Technology, Patna
Bihar, India
rkbiitp@gmail.com

Meiqing Wu
School of Computer Science and Engineering
Nanyang Technological University,
Singapore
meiqingwu@ntu.edu.sg

***Abstract*—Stream-based image processing architectures are extremely attractive as they can achieve high throughput and do not require external memories for storing input video frames. However, a major challenge in designing stream-based architectures lies in lowering the dynamic power consumption since all the processing elements are typically in continuous operation to keep up with the rate of incoming pixel streams. In this work, we show that the dynamic power of the stream-based Harris corner detector (HCD) can be reduced by inhibiting redundant signal activity in the complex calculations of the corner scores. Specifically, we perform simple approximations to detect non-likely corners at the early stages of the pipeline for putting the subsequent pipeline stages into a dormant state. Synthesis results on the Altera Cyclone V FPGA show that the proposed strategy leads to an average dynamic power reduction of over 10% compared to the conventional implementation with similar performance and negligible increase in resources. In addition, we performed repeatability tests to show that the proposed stream-based HCD architecture achieves comparable accuracy with the conventional implementation.**

Keywords—FPGA, corner detection, low power, hardware acceleration, embedded vision

I. INTRODUCTION

The proliferation of small, cheap and low power cameras in embedded systems has given rise to the need for custom hardware implementation of computer vision algorithms to meet the real-time demands as well as satisfy the energy and cost constraints of these systems. Real-time computer vision algorithms will be extensively used in a wide range of embedded applications such as navigation of unmanned vehicles [1] and robots [2], video tracking [3] and visual SLAM [4]. In this regard, stream processing for computer vision architectures has tremendous benefits in area and energy savings as well as eliminating the memory access bottleneck for reading the image frames from memory for processing, and writing the intermediate results to memory.

Stream processing avoids the need for storing and accessing input video frames from external memories by processing the incoming pixel streams on the fly [5][6]. Many computer vision operations (e.g. gradient computation, filtering, morphological operations, etc.) necessitate processing on small local neighborhood of pixels. Under the assumption that the image is read sequentially using a raster scan mode at a rate of one pixel per clock cycle, the incoming pixels need to be cached locally using a set of row buffers. Fig. 1 shows an example of a stream processing architecture that concatenates four row buffers in the form of FIFO (First-In, First-Out) delay buffers to cache the incoming pixels. The size of each row buffer is equivalent to the horizontal resolution of the image, and hence each row buffer effectively delays the input by one row. The n-bit (n = 8 for grayscale images) pixels at the tail end of each row buffer are shifted into a set of registers (often called a window or convolution buffer), which will be processed by the pipeline architecture. The operations in the architecture are pipelined to maintain the required throughput. In stream processing, a new output is typically produced at the rate of an incoming pixel, and hence the response time of stream processing is dominated by the camera frame rate.

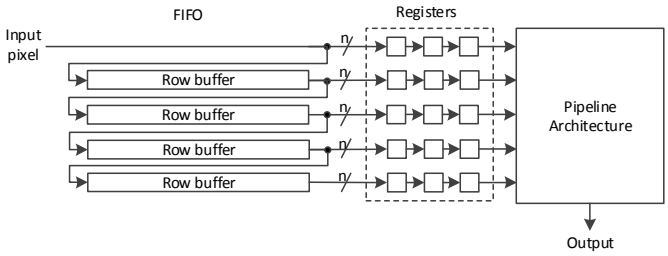

Fig. 1. Stream processing architecture

A major challenge in designing stream processing architectures lie in lowering the dynamic power consumption. This challenge arises due to the following reasons. Firstly, the row buffers typically contribute to a significant portion of the total power consumption. Known power reduction techniques such as clock gating or retiming [7] cannot be applied to the row buffers as pixels are continuously streamed in at each clock cycle. Secondly, the operators in the pipelined architecture for most computer vision applications are typically kept in continuous operation to maintain the desired throughput. This presents a difficulty in switching the operators to 'idle' state through clock gating, which has been previously shown to provide an effective means for reducing power. An application which exhibits this characteristic is the Harris Corner Detector (HCD) [8], where all the pipeline operations contribute to processing the output at each pixel location.

This research was supported by National Research Foundation Singapore

978-1-5090-4825-0/17/$31.00 ©2017 IEEE

Corner detection is a fundamental step in many computer vision applications as corners represent identifiable anchor points in the image. These corners are used for visual odometry, stereo matching, optical flow computation, object tracking and as robust image representation when combined with feature descriptors for object recognition. The HCD is a widely used feature detection algorithm due to its robustness in detecting corners in noisy images [9]. However, the computationally intensive operations in the algorithm incurs a performance bottleneck and significant power consumption on general purpose microprocessors [9].

In this paper, we proposed a hardware implementation to reduce the dynamic power of the HCD by inhibiting redundant signal activity in the complex calculations of the corner scores. This is achieved by performing simple approximations at the early stages of the pipeline to detect non-likely corners for switching the subsequent pipeline stages into a dormant state, hence lowering the overall power consumption. Despite the dominant contribution of power consumption from the row buffers, we achieved notable dynamic power reduction compared to the conventional implementation with similar performance and negligible increase in resources. In addition, we show that the proposed HCD architecture did not compromise on accuracy to achieve the power savings.

The rest of the paper is organized as follows. Section II discusses previous work on hardware implementation of the HCD algorithm. Section III provides a brief overview of the HCD baseline hardware implementation. Section IV describes the proposed low power implementation of HCD that limits redundant signal transitions in the HCD pipeline stages. In Section V, we evaluate the accuracy of the proposed hardware implementations using repeatability tests and provide the FPGA synthesis results to demonstrate the power gains of the proposed strategy. We conclude the paper in Section VI.

II. Related Work

Hardware implementations of HCD have been proposed on ASIC [10], FPGA [11][12][13], cell processor [14], and SIMD architecture [15]. The HCD algorithm computes an auto-correlation matrix for each pixel using the first-order derivatives of the intensity values and this matrix represents the degree of intensity variations in different directions around the corresponding pixel. A complex corner measure computation is then performed for every pixel in the image. This step is highly compute-intensive, and becomes a bottleneck for real-time vision tasks. In [16], a modified number representation for the corner response is used in custom instructions on the NIOS II processor. In [17], a hardware implementation that performs HCD on a rank transform image instead of the original image is presented.

In [18] a frame buffer based approach is described in which every individual step of the HCD algorithm is sequentially carried out for the entire image frame. Multiple image regions can be processed in parallel which allows for high operating frequency at the cost of large area and high power consumption. In [9][11][19][20], stream-based processing of the HCD is employed. A small scanning window is utilized to determine whether the center image pixel is a corner or not. Since all the sequential steps of HCD algorithm are performed locally rather than over the complete image, the intermediate data can be stored in local registers which results in low storage requirements. The work in [19] describes two architectures with different scanning window sizes and number of row buffers.

The above-mentioned techniques for stream processing often exploit the inherent parallelism in the corner detectors and find a reasonable trade-off between the number of row buffers and the computational resources to manage the resource utilization while ensuring high throughput and acceptable result quality. However, there has been little effort undertaken to investigate methods for achieving low power. Reducing power and energy consumption is particularly important given the increasing use of complex computer vision applications on battery-operated embedded devices [21][22][23].

III. Baseline Architecture for HCD

The HCD algorithm performs corner detection based on a local auto-correlation function that is approximated by matrix M within a small window W of each pixel $p(x,y)$ as shown in Eq. (1). I_x and I_y are the horizontal and vertical gradients, and $w(x)$ is the Gaussian weight function. The two eigenvalues of M, i.e. λ_1 and λ_2, indicate the intensity change in the window W centered on $p(x,y)$. Specifically, $p(x,y)$ is a corner if both the eigenvalues are large.

$$M = \begin{bmatrix} \sum_w w(x) I_x^2 & \sum_w w(x) I_x I_y \\ \sum_w w(x) I_x I_y & \sum_w w(x) I_y^2 \end{bmatrix} = \begin{bmatrix} a & b \\ b & c \end{bmatrix} \quad (1)$$

To avoid explicit computations of the eigenvalues, the HCD combines the eigenvalues into a single corner measure R as shown in Eq. (2), where k is an empirical constant and is usually set between 0.04 and 0.06. A threshold T is applied on the corner response to discard the obvious non-corners. The pixels with the highest corner response are then selected as corners after applying non-maximal suppression.

$$\begin{aligned} R &= \lambda_1 \cdot \lambda_2 - k \cdot (\lambda_1 + \lambda_2)^2 \\ &= \det(M) - k \cdot trace^2(M) \\ &= (ac - b^2) - k \cdot (a + c)^2 \end{aligned} \quad (2)$$

Fig. 2 shows the conventional HCD architecture, which is similar to the one proposed in [19]. The HCD architecture consists of the following five pipeline stages: 1) Gradient Computation, 2) Product of Gradients, 3) Gaussian Smoothing, 4) Corner Response, and 5) Non-Maximal Suppression (NMS).

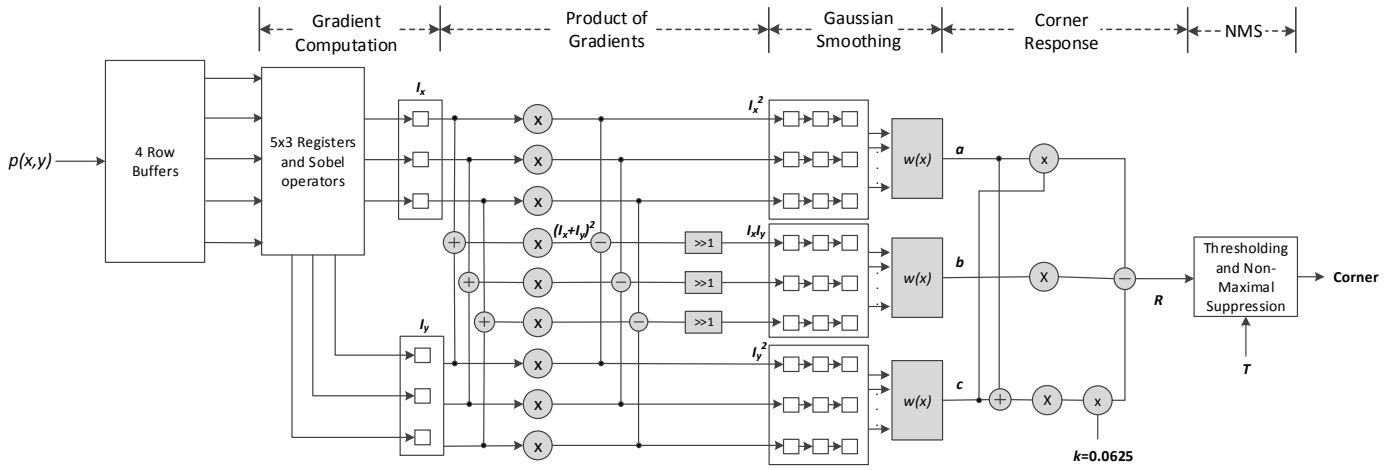

Fig. 2. Conventional HCD architecture (each square box represents a register)

- *Gradient Computation*: The first pipeline stage computes the horizontal and vertical gradients I_x and I_y of the incoming pixels from the row buffers. The gradient computation is performed on a 5x3 neighborhood of pixels (see Fig. 1). This allows for simultaneous computations of three sets of I_x and I_y values at every clock cycle.
- *Product of Gradients*: The second pipeline stage computes the product of gradients to generate three sets of I_x^2, I_y^2 and I_xI_y in each clock cycle.
- *Gaussian Smoothing*: Gaussian smoothing is applied to the product of gradients in this stage to produce a, b and c as shown in Eq. (1). This is achieved by caching the product of gradients using three sets of 3x3 registers and applying the Gaussian weight function $w(x)$ to compute a, b and c in parallel.
- *Corner measure*: The fourth pipeline stage computes the Harris corner measure R as shown in Eq. (2). Similar to the implementation in [18], we have approximated the value of k to be 0.0625 (1/16) so that constant shift operation can be used in place of a multiplier.
- *Non-Maximal Suppression*: In the final pipeline stage, the corner response R is first compared with a threshold T. If the corner measure is less than T, R is set to 0 (this avoids unnecessarily storing negative corner responses which are non-corners) otherwise the original value of R is retained. The corner responses are then cached using two row buffers as the non-maximal suppression (NMS) operation requires a 3x3 pixel region whereas only a single output is produced at the Corner Response stage. The NMS operation determines whether the center pixel of the 3x3 region is a corner or not by comparing its corner response to the corner responses of its 8 adjacent pixels. Fig. 3 shows the architecture of the NMS pipeline stage.

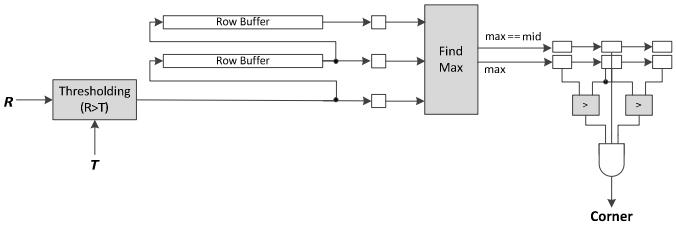

Fig. 3. Non-Maximal Suppression

IV. Proposed Implementation

The row buffers at the input and NMS stage in Fig. 2 do not offer much opportunities for power reduction since they must be in active state during each clock cycle to maintain the desired throughput of the system. In this section, we will discuss our strategy to lower the power at the Product of Gradients, Gaussian Smoothing and Corner Response stages.

The proposed strategy is based on the observation that in the conventional architecture, the above-mentioned pipeline stages are perpetually in active states even though only a reduced set of pixels will be eventually regarded as corners (determined by the thresholding step in the NMS stage). Hence, if it is possible to detect the likelihood of a pixel being a non-corner in the early stages, we can put the subsequent pipeline stages to a dormant state (either using clock gating or propagating '0's) to reduce the overall switching activity. With regards to this, there are some recent attempts to reduce the computational complexity of corner detection algorithms on embedded processors [24][25][26] by pruning the search space for corners using simple approximations before applying a more complex corner measure step to evaluate the candidate set. For example, the work in [24] approximates the values of I_x^2 and I_y^2 (or a and c terms) as follows:

$$a' = \sum |I_x|, c' = \sum |I_y| \qquad (3)$$

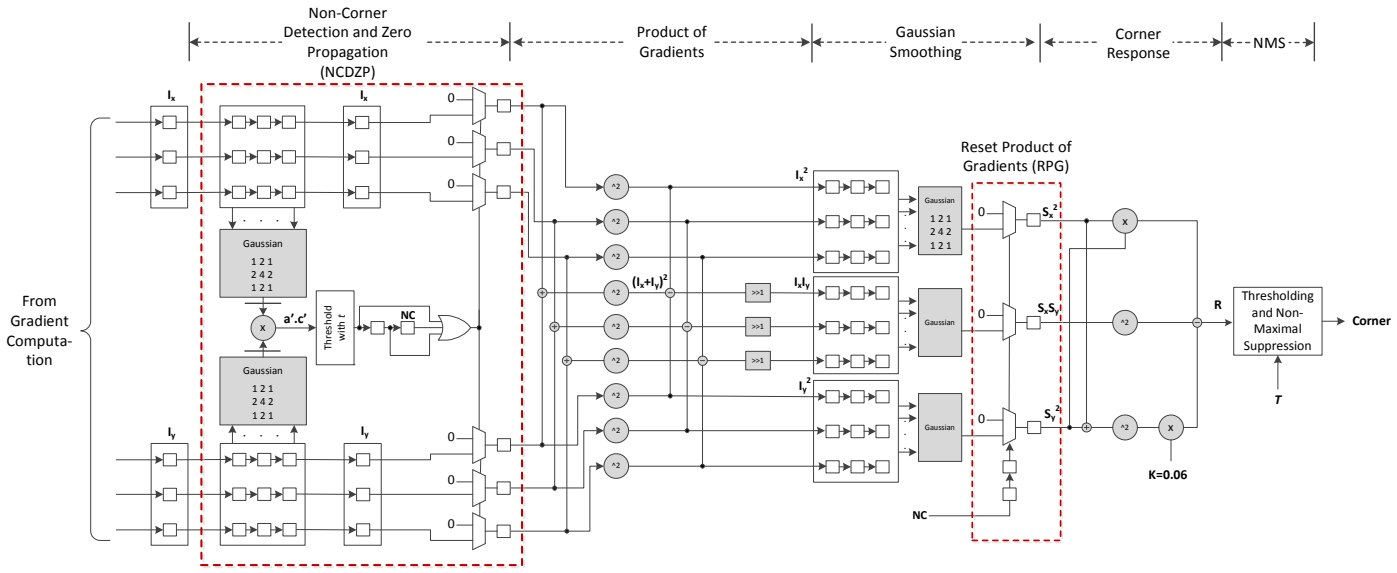

Fig. 4. Proposed HCD architecture (each square box represents a register)

In [24], each pixel $p(x, y)$ is evaluated by computing the corresponding *a'c'* value, and comparing it with a threshold $t = 0.05 \cdot \max(a'c')$, where $\max(a'c')$ is the maximum *a'c'* value in the current image frame. If the *a'c'* value exceeds *t*, the corresponding pixel is stored in a candidate corner set as a potential corner. After all the corner candidates of the image frame has been identified, the corner candidates are subjected to the HCD algorithm to determine if they are valid corners.

While the method in [24] has been shown to achieve substantial speedup over the conventional algorithm on an embedded microprocessor, it does not lend itself well towards efficient stream processing due to the following reasons. Firstly, the candidate corner set is computed for the full image frame and stored, before being subjected to more stringent evaluations. This would require a large storage, which in the worst case, is equivalent to the size of an image frame. Secondly, the threshold *t* must be determined by evaluating all the pixels in the image frame before it can be used for pruning. Again, this implies that a large image frame buffer is required to store the image frame since all the pixels must be revisited after *t* has been computed.

Our proposed stream-based architecture adopts the approximation in Eq. (3) to reduce the overall switching activity by propagating '0's to the Product of Gradients, Gaussian Smoothing and Corner Response stages, when the *a'c'* value does not meet the threshold. Alternatively, we can use clock gating to switch these stages to 'idle' state when the non-corners are detected. In order to facilitate stream processing without the need of intermediate frame buffers, we compute the $\max(a'c')$ term in the threshold from the previous image frame with the assumption that it will not differ much from the current frame. Similar assumptions have been adopted in other computer vision methods such as the brightness constancy for optical flow computation [27] and background update [28] with high frame rates. While processing the current frame, we can simultaneously update the $\max(a'c')$ value in the present frame using negligible hardware resources (only a comparator and register is required). This value will be used as the threshold for the next frame.

Fig. 4 shows the proposed architecture, where the dotted regions indicate the additional resources that are included to facilitate 1) early non-corner detection and zero propagation logic to switch the subsequent pipeline stages to 'idle state (NCDZP stage), and 2) resetting the smoothed product of gradients for non-corners (RPG). Note that the input row buffers and Gradient Computation stage are not shown Fig. 4.

In the proposed architecture, the new pipeline stage NCDZP is responsible for computing *a'c'* and comparing it with *t*. If *a'c'* does not meet the threshold, the corresponding pixel is considered a non-corner (NC), and zeros are propagated to the subsequent pipeline stages (via the multiplexers) instead of the gradients I_x and I_y. An additional check is performed using the OR gate to evaluate if the neighbor pixel of NC is also a non-corner before propagating zeros. This check must be performed because even if the current pixel (NC) is a non-corner, its corresponding gradient values are still required in the Gaussian Smoothing stage if its neighbor is a corner. As such, the additional check ensures that zeros are propagated only when the current pixel and its neighbor pixel are non-corners. In cases where the gradients of a non-corner $p(x, y)$ are passed to the subsequent stages, RPG resets the corresponding smoothed product of gradients I_x^2, I_y^2 and I_xI_y (since there is no need for computing the corner response).

V. Experimental Results

In this section, we will provide experimental results for the proposed implementation in terms of accuracy and hardware synthesis results.

A. Accuracy Evaluation

We used the repeatability criteria [29] to compare the accuracy of the conventional (Conv.) and proposed implementations. The repeatability criterion is based on the notion that detection of corners should be invariant of imaging conditions e.g. blurring, zooming, and rotation of the scene. An accurate feature detector should be robust to the changes in imaging conditions and hence, should be able to detect features at close proximity between images with changes in viewpoint. The repeatability rate is defined as the ratio of the number of repeated features between two images within certain pixel allowance, to the minimum number of features that are in common region of the two images of the same scene but with changes in imaging condition(s). We have used the image dataset from [30] for the accuracy evaluation. In our experiments, we have used four 640x480 image sets (Bikes, Leuven, Trees, and UBC) with various image transformation sequences such as changes in viewpoint, zoom, rotation and illumination. Fig. 5 shows the original images (before transformation) of each image set.

Fig. 5. Image set used in the experiments. From Top left: Bikes, Leuven, Trees, UBC

The difference in the repeatability rate (a pixel allowance of 2.5 pixels is used in the evaluations to account for the transformations) of the proposed and the conventional architectures is computed using Eq. (4). Table 1 reports the number of final corners detected by the conventional and proposed implementation, and the maximum, minimum and average absolute difference in repeatability rate for the four image sets. It is evident that the overall difference in the number of detected corners and repeatability between the proposed and conventional architectures is marginal for the image sets considered. In particular, only 0.00485 average difference in repeatability rate is observed. These results demonstrate that the accuracy of corners generated by the proposed architecture and the conventional architecture are very similar.

$$\Delta r = repeatability_{proposed} - repeatability_{conv} \quad (4)$$

Table II shows the percentage of likely non-corners that are detected in the Non-Corner Detection stage of the proposed implementation. It is evident, that for the images considered (here we only use the original images in each set), a large number of pixels out of the 640x480 pixels in an image frame are detected as non-corners. For these non-corners, the Product of Gradients, Gaussian Smoothing and Corner Response stages are switched to 'idle' state (by propagating '0's) to reduce dynamic power. Table 1 and Table II shows that a high potential for dynamic power reduction in the proposed implementation is possible without compromising on accuracy.

TABLE I. NUMBER OF CORNERS AND DIFFERENCE IN REPEATABILITY

	Number of corners (Conv.)	***Number of corners (Proposed)***	***Max Abs Δr***	***Min Abs Δr***	***Average Abs Δr***
Bikes	672	668	0.01592	0.00000	0.00934
Leuven	477	478	0.00651	0.00219	0.00409
Trees	623	624	0.01015	0.00000	0.00221
UBC	798	798	0.00874	0.00061	0.00376

TABLE II. PERCENTAGE OF NON-CORNERS DETECTED AT THE NCDZP STAGE (OVER 307,200 PIXELS)

Bikes	***Leuven***	***Trees***	***UBC***
94.46%	92.01%	68.38%	88.21%

B. Hardware Synthesis

The conventional and proposed architecture was implemented in Verilog, synthesized using Quartus II Version 15.0.0 and targeted the Altera FPGA Cyclone V (5CGXFC7C7F23C8) device. Table III shows the maximum clock operating frequency and area utilization of both implementations. It can be observed that both the conventional and proposed architectures have similar maximum clock frequency. The proposed implementation has slightly higher area utilization (less than 0.6%) due to the introduction of additional resources (i.e. NZDZP and RPG).

TABLE III. HARDWARE AREA-TIME RESULTS

Method	***Max Frequency (MHz)***	***Area (ALM)***
Conv	67.97	33,122
Proposed	70	33,318

TABLE IV. DYNAMIC POWER ANALYSIS

Image	***Conventional (mW)***	***Pruning (mW)***	***Percentage Power Reduction (%)***
Bikes	144.39	125.29	13.23
Leuven	146.03	130.79	10.44
Trees	160.30	148.20	7.55
UBC	156.80	141.01	10.07

To obtain accurate power analysis, we perform timing simulation using ModelSim-Altera 10.3d to obtain the actual switching activity statistics of the architecture for the original images in the four image sets. The switching activity statistics are then used for power analysis using Altera PowerPlay. For a fair comparison, we have used the same clock operating frequency of 65MHz in both implementations. Since the static power of both implementations are similar, we only report the dynamic power in Table IV. The dynamic power of both implementations is dominated by the row buffers for caching the inputs and those required by NMS. Despite this, the results show that an average of over 10% dynamic power can still be achieved using the proposed strategy which inhibits redundant switching activity in the pipeline stages when non-corners are detected in NCDZP stage. The trend in percentage dynamic power reduction is consistent with the percentage trend of detected non-corners in Table II. It is worth mentioning that the dynamic power reduction in the proposed implementation is achieved at the same performance as the conventional architecture, with negligible increase in resources.

VI. Conclusion

In this paper, we investigated dynamic power reduction for stream-based HCD architecture. This is particularly challenging as the row buffers, which are used to facilitate local neighborhood processing contribute to significant power consumption. Despite this, we showed that notable dynamic power savings can still be achieved in the pipeline stages by performing simple approximations at the early stages of the pipeline to detect non-corners. This enables us to switch the subsequent pipeline stages into a dormant state when non-corners are detected. FPGA synthesis results show that the proposed strategy leads to average dynamic power reduction of over 10% compared to the conventional implementation with similar performance and negligible increase in resources. In addition, the average difference in repeatability between the proposed and conventional implementation is only about 0.4%.

References

[1] S. Ehsan, and K.D. McDonald-Maier, "On-board vision processing for small UAVs: time to rethink strategy", Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems, 2009

[2] A. Schmidt, M., Kraft, and A. Kasinski, "An evaluation of image feature detectors and descriptors for robot navigation", Computer Vision and Graphics, Vol. 6375, pp. 251-259, 2010

[3] S. Gauglitz, T. Hollerer, and M. Turk, "Evaluation of interest point detectors and feature descriptors for visual tracking", International Journal of Computer Vision, Vol. 94, pp. 335-360, 2011

[4] A. Gil, O.Mozos, M.Ballesta, and O.Reinoso, "A comparative evaluation of interest point detectors and local descriptors for visual SLAM", Machine Vision and Applications, Vol. 21, pp. 905-920, 2010

[5] D.G. Bailey, “Design for Embedded Image Processing on FPGAs”, John Wiley & Sons, 2011

[6] K. Dohi, K. Negi, Y. Shibata and K. Oguri, “FPGA Implementation of Human Detection by HOG Features with AdaBoost”, IEICE Transactions on Information and Systems, Vol. E96.D, No. 8 pp. 1676-1684, 2013

[7] H. Blasinski, F. Amiel, and T. Ea, “Impact of Different Power Reduction Techniques at Architectural Level on Modern FPGAs”, IEEE Latin American Symposium on Circuits and Systems, February 2010

[8] C. Harris, M. Stephens, "A combined corner and edge detection", Proceedings of The Fourth Alvey Vision Conference, pp. 147-151, 1988

[9] P.R. Possa, S.A. Mahmoudi, N. Harb, C. Valderrama, and P. Manneback, "A Multi-Resolution FPGA-based Architecture for Real-Time Edge and Corner Detection", IEEE Transactions on Computers, Vol. 63, No. 10, pp. 2376-2388, October 2014

[10] C. Chih-Chi, L. Chia-Hua, L. Chung-Te, S.C. Chang, C. Liang-Gee, "iVisual: an intelligent visual sensor SoC with 2790fps CMOS image sensor and 205GOPS/W vision processor", Design Automation Conference, pp. 90-95, 2008

[11] T.L. Chao and K. H. Wong, "An efficient FPGA implementation of the Harris corner feature detector", 14th IAPR International Conference on Machine Vision Applications (MVA), 2015, pp. 89-93.

[12] H. Orabi, N. Shaikh-Husin and U. Ullah Sheikh, "Low cost pipelined FPGA architecture of Harris Corner Detector for real-time applications", International Conference on Digital Information Management, 2015

[13] A. Hernandez-Lopez, C. Torres-Huitzil and J.J. Garcia-Hernandez, "FPGA-based flexible hardware architecture for image interest point detection", International Journal of Advanced Robotic Systems, Vol. 12, No. 93, 2015

[14] T. Saidani, L. Lacassagne, S. Bouaziz, T. Khan, "Parallelization strategies for the points of interests algorithm on the cell processor", Parallel and Distributed Processing and Applications, Vol. 4742, pp. 104-112, 2007

[15] F. Hosseini, A Fijany, J.-G Fontaine, "Highly parallel implementation of Harris Corner detector on CSX SIMD architecture", Conference on Parallel Processing, 2011

[16] S. Piskorski, L. Lacassagne, S. Bouaziz and D. Etiemble, "Customizing CPU instructions for embedded vision systems", IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2007

[17] B. Tippetts, D.-J. Lee and J. Archibald, "An on-board vision sensor system for small unmanned vehicle applications", Machine Vision Applications, 23(2), pp. 1-13, 2012

[18] M.F. Aydogdu, M. F. Demirci, C. Kasnakoglu, "Pipelining Harris corner detection with a tiny FPGA for a mobile robot", Proc. IEEE International Conference on Robotics and Biomimetics, ROBIO, 2013

[19] A. Amaricai, C.-E. Gavriliu and O. Boncalo, "An FPGA sliding window-based architecture Harris corner detector", Field Programmable Logic and Applications (FPL), 2014 24th International Conference, pp. 1-4

[20] P. Y. Hsiao, C.L. Lu, L.C. Fu "Multilayered Image Processing for Multiscale Harris Corner Detection in Digital Realization", IEEE Trans. On Industrial Electronics, Vol. 57, Issue 5, 2010.

[21] A. Petrovai, A. Costea, F. Oniga, and S. Nedevschi, "Obstacle detection using stereovision for Android based mobile devices", IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 2014

[22] J.M. S´aez, F. Escolano, and M.A. Lozano, "Aerial obstacle detection with 3D mobile devices", IEEE Journal of Biomedical Health Information, Vol. 19, No. 1, 74-80, 2015

[23] T. Schops, J. Engel, and D. Cremers, "Semi-dense visual odometry for AR on a smartphone", IEEE International Symposium on Mixed and Augmented Reality, 2014

[24] M. Wu, N.Ramakrishnan, S.-K. Lam and T. Srikanthan, "Low-Complexity Pruning for Accelerating Corner Detection", IEEE International Symposium on Circuits and Systems (ISCAS), May 2012, pp. 1684-1687

[25] N. Ramakrishnan, M. Wu, S.-K. Lam, and T. Srikanthan, "Enhanced Low-Complexity Pruning for Corner Detection", Journal of Real-Time Image Processing, 2014

[26] N. Ramakrishnan, M. Wu, S.-K. Lam, T. Srikanthan, "Automated Thresholding for Low Complexity Corner Detection", NASA/ESA Conference on Adaptive Hardware and Systems, 2014, pp. 97-103

[27] D. Rosenbaum and Y. Weiss, “Beyond Brightness Constancy: Learning Noise Models for Optical Flow”, Available: https://arxiv.org/abs/1604.02815

[28] K. Garg, S.-K. Lam, T. Srikanthan, and V. Agarwal, “Real-Time Road Traffic Density Estimation using Block Variance”, IEEE Winter Conference on Applications of Computer Vision (WACV), March 2016

[29] C. Schmid, R. Mohr, and C. Bauckhage, “Evaluation of Interest Point Detectors”, International Journal of Computer Vision, June 2000, pp. 151-172

[30] Affine Covariant Features. Available: http://www.robots.ox.ac.uk/~vgg/research/affine/