A Hardware Architecture Design for Area Constrained Integral Image Generation for Face Detection Application

Nidhi Panda, Supratim Gupta

Department of Electrical Engineering
National Institute of Technology, Rourkela –769008, INDIA
nidhipanda15@gmail.com, sgupta.iitkgp@gmail.com

ABSTRACT

Human face detection finds an important role in various Human-Computer Interaction (HCI) and computer vision applications. The seminal work of Viola Jones for automatic face detection found it's popularity in many such applications. The inherent parallelism in the algorithm makes it more applicable for hardware implementation. It utilizes integral image computation as a preprocessing step to reduce the overall computation burden of Haar-like features. Although the calculation of the integral image consists of simple addition operations, the total number of operations increases with increase in image resolution. Therefore, for resource-constrained real-time embedded applications, the computation and storage of integral values present several design challenges. This paper proposes an optimized hardware architecture of integral image computation for a resource constraint low-cost system. The proposed architecture utilizes the advantage of overlapping area in the sliding window used to find face features in the Viola-Jones face detector. The architecture is simulated using VIVADO® Design Suite 2018.2 for the ZYNQ (ZC702) board. It is found that the implemented architecture achieves a significant reduction in the hardware resource utilization compared to the state-of-the-art integral image computation implementations.

Keywords: Face detection, integral image, pipelined architecture, parallel processing.

I. INTRODUCTION

Face Detection is the first and essential step in many application areas that includes security, biometrics, law enforcement, entertainment, personal safety, etc. Despite the relative ease with which humans can detect faces, it is always a challenging task to detect the complex face features at the algorithm level. The uncertainty and the time constrain in real-time scenarios elevate the complexity in face detection. Many face detection methods have been proposed in the literature, which may be broadly classified into four categories: Knowledge-based, Feature-based, Template matching-based, and Appearance-based approaches [1]. Among these, the face detector proposed by Viola and Jones (Feature-based approach) attained much popularity in real-time applications as it considers both data diversity and data computation in the dual-direction [2]. Viola-Jones utilized the integral image as a look-up table to speed up the Haar feature calculation. An integral image facilitates us to calculate summation over image sub-regions in constant time– by mirroring the use of cumulative distribution function. Owing to this property, an integral image is widely used as an intermediate image representation technique for applications such as multi-scale local feature detection, speech detection, block matching, human activity measure, etc. [1]. Mostly, sequential processors have been adopted for various face detector implementations using the OpenCV library. However, the slow processing speed of the sequential processor makes them inadequate for real-time high-speed applications. A more adequate approach to accelerate the face detection algorithm is to utilize the advantages of GPU, FPGA or ZYNQ by exploiting the maximum parallelism possible in the hardware units. In this paper, we have used ZYNQ SoC (ZC702) board to achieve better computational complexity using parallel processing. ZYNQ SoC comprises of two main parts: a Processing System (PS) formed around a dual-core ARM Cortex-A9 processor, and Programmable Logic (PL), which is equivalent to that of an FPGA (Artix-7 for ZC702 board). The integral accelerator is designed for the FPGA part of the ZC702 board.

The calculation of the integral image only consists of simple addition operations. However, the higher number of addition operations and the storage of output integral image for further processing are the two major bottlenecks in the hardware implementation of the integral computational module. In this work, we have proposed to use the overlapping property of the sliding window method to reduce the number of addition operations. The storage problem of integral image is reduced by assigning a fixed location to a small portion of the integral value, instead of wasting the memory space by saving the complete integral image.

The rest of the paper is organized as follows: a brief description of the integral image computation technique with different representations of the integral image computation is presented in Section II. An overview of the previous integral image module implementations on hardware is also illustrated in Section II. The proposed hardware architecture is presented in Section III, whereas Section IV provides the analysis of the proposed architecture and its comparison with previous hardware implementations of the integral module. Finally, Section V concludes this paper.

II. BACKGROUND AND RELATED WORK

Integral transformation for any image value $i(x, y)$ finds the 2D discrete antiderivative $ii(x, y)$ by summing up the values above and to the left of the location (x, y) including $i(x, y)$ pixel value (see Eq.(1)) [2]. An input image and its corresponding integral image is presented in Fig. 1(a) & (b) respectively. Moreover, the property of the integral image –to calculate the rectangular sum of any image sub-region in constant time– is presented in Fig. 1(c).

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y') \quad (1)$$

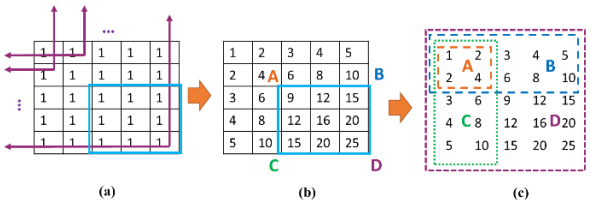

Fig. 1. (a) Input image, (b) Integral image of input, (c) rectangular sum computation for rectangle present in (a) using $(A + D) - (B + C)$.

To compute the integral image for an image size of $M \times N$ in sequential processing using Eq.1 it requires $\frac{1}{4}M^2N^2$ number of addition operation with $(M \times N) - 1$ number of adders at $M \times N$ clock cycles [4]. This computation will be more tedious and will demand more clock cycles for images with high spatial resolution. To mitigate the problem in integral image computation using Eq. 1, in [2] recursive equations (presented in Eq.2 & Eq.3) is used.

$$S(x,y) = i(x,y) + S(x,y-1) \quad (2)$$

$$ii(x,y) = ii(x-1,y) + S(x,y) \quad (3)$$

Here, $S(x,y)$ is the cumulative row sum value at the image location(x,y). This recursive equation reduces the number of addition operation drastically to an order of $2MN$ with only 2 adders at the cost of extra memory to store the past integral values and cumulative row sum values. However, as presented in the data-flow diagram in Fig. 2, we get only one integral value at each instance, which slowdowns the calculation speed and generates a timing overhead and therefore, not appropriate for real-time implementation.

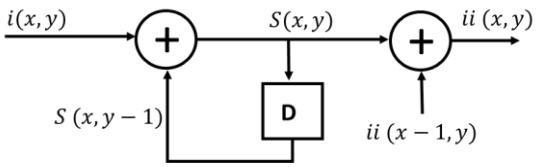

Fig. 2. Data flow diagram of integral image computation using recursive equation.

$$ii(x,y) = i(x,y) + ii(x,y-1) + ii(x-1,y) - ii(x-1,y-1) \quad (4)$$

To analyze the pixel dependency in integral image calculation, Eq.1 can be reformulated as Eq.4 [3]. This representation will take two adders, one subtractor and $M \times N$ clock cycles to compute the integral values for an image of $M \times N$ resolution. It is evident from Eq.4 that the integral value calculation at any location (x,y) is dependent on the past integral values and input image pixel value at (x,y). However, as presented in Fig.3 (using blue color) the maximum integral values which can be computed independently (in parallel) are $min(M,N)$. This parallelization reduces the number of clock cycle requirement to $(M+N)-1$ cycles at the cost of $min(M,N)$ number of adders.

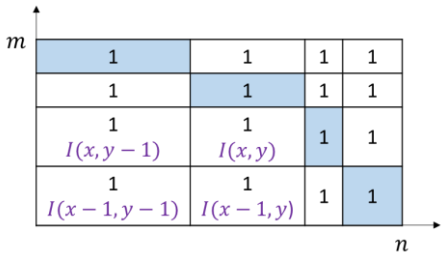

Fig. 3. Data dependency analysis for integral image calculation.

In literature few hardware implementations of the integral images are available [3]-[8]. Kyrkou and Theocharides [6] proposed a hardware architecture for Haar feature-based face detector. They have used a systolic array for integral image computations. The computation is performed using addition with vertical and horizontal shift operation in $(M + (M-1) + (N-1))$ cycles. Though this remodeling increases the computational speed, it does not fully exploit the parallelism in integral image computation [3]. Ouyang *et al.* [3] proposed to use dual-direction data-oriented computation of the integral image. This method reduced the time complexity to *O*(*N*) by using the pipelining technique. However, this method consumes high resources as the image is divided into multiple strips and these strips are executed in a pipelined way. Other approaches for resource-constrained integral image computing method are developed in [4] and [8]. In both the implementation, the decomposition of Eq.2 & Eq.3 is used to establish a trade-off between resource utilization and processing speed. In [4] for an image of spatial dimension $M \times N$, the algorithm achieves $MN + {}^{MN}/_{2}$ number of additions by adhering four row parallel method, whereas [8] reduces the usage of adders by using DSP slices in SIMD mode. In all the above-discussed scenarios, either the processing time or the resource requirement is very high. In most of these works, to speed up the integral image generation process, the whole image is divided into small parts, and for each part, integral values are computed in either parallel or pipelined way. This tenchnique increases resource utilization. Also, as the integral image is used as a preprocessing step, saving the complete integral image for the next computation module generates a huge memoryrequirement(depending upon the image size). Therefore, in this work, we aim to design an architecture for an area-efficient integral image accelerator.

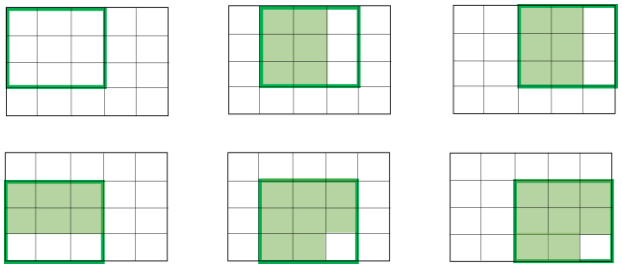

Fig. 4. Integral image calculation on moving window based application.

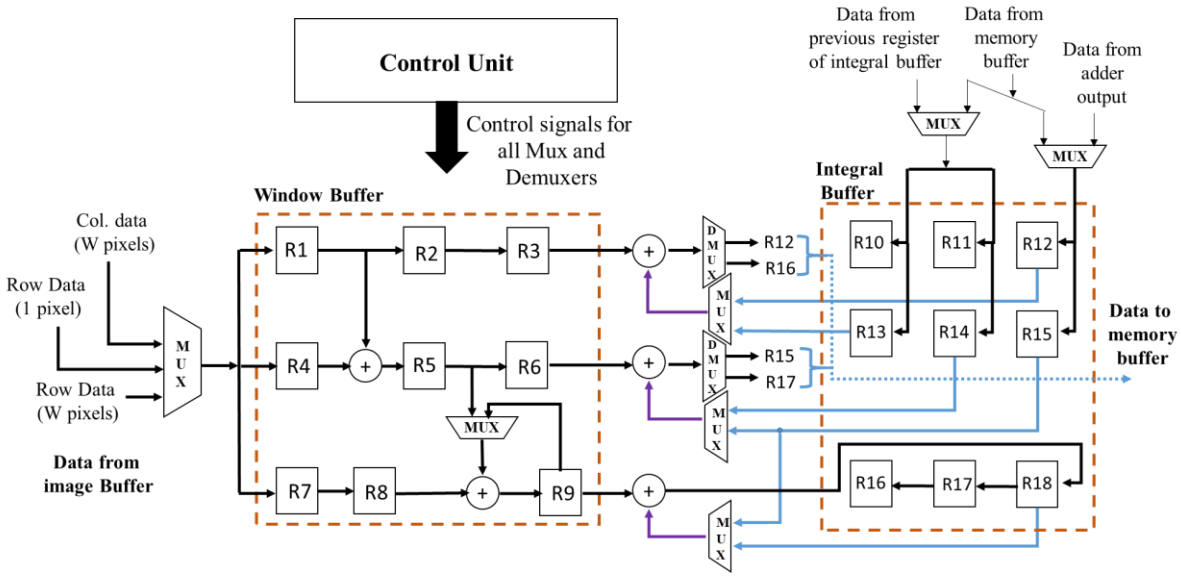

Fig. 5. Complete system overview for integral image computation module.

Mathematically, the problem can be formulated as:

$$minimize\{A \; \forall \; ii = f(i) | \; (t_L)_{min}, (t_{th})_{max}\}$$

Here, i denotes the input image frame. ii, A, f denotes the integral image, total resource utilization for the integral image engine and integral image circuit mapping function respectively. Similarly t_L , t_{th} represents latency and throughput value respectively.

III. PROPOSED HARDWARE ARCHITECTURE

Viola-Jones face detector employs the sliding window technique to search for the face region in the image. On analyzing the sliding window technique, we found that there is an overlapping area between the two consecutive windows as shown in Fig.4 (the overlapped region is represented in green color). It is evident from the Fig.4 that after the integral values are calculated for the first window, to create a new window only integral values of either one column/row or one (x, y) position is needed. The use of the overlapping property of the sliding window approach reduces the number of addition operations drastically.

The top-level architecture of the proposed integral image computation module is presented in Fig. 5. It includes three memory buffer units named as window buffer($W \times W$, W is the window size) integral buffer ($W \times W$), and intermediate memory buffer ($W \times N$). Generally, for the face detection application the window size is 20×20 or 24×24 [2], [5]. However, in this paper for representation simplicity, we have shown all the figures with window size W of 3×3. Registers are used for the entire buffer unit implementation so that each value of the buffers can be accessed simultaneously. Further, to reduce memory usage, the window and integral buffer units are designed using the time multiplexing of the resources. Also, instead of saving the integral values for the full image, an intermediate memory buffer (of size $W \times N$) is used. A controller is designed to facilitate the time multiplexing of resources for the data and adder units. Integral value calculation for the entire image is completed by combining three techniques as presented in Fig. 6, Fig. 7 and, Fig.8 respectively. These three steps differ in the method of data fetching from input image and the computation in integral buffer unit. The proposed steps for the integral value calculation are presented in Fig. 9. Here, the yellow color represents the diagonal adders placed in the window buffer whereas the light orange color represents the addition operation in the integral buffer unit. The input of the window buffer can access data in either row or column-wise from image buffer by using the control signal for the MUX unit. Data values from the image buffer are copied in the first column of the window buffer. At each clock cycle all data values are shifted towards right and addition operation is performed on diagonal positions. In the first approach as presented in Fig. 6 window buffer unit fetches column-wise data. The last column values of the integral buffer unit are loaded with the summation of the last

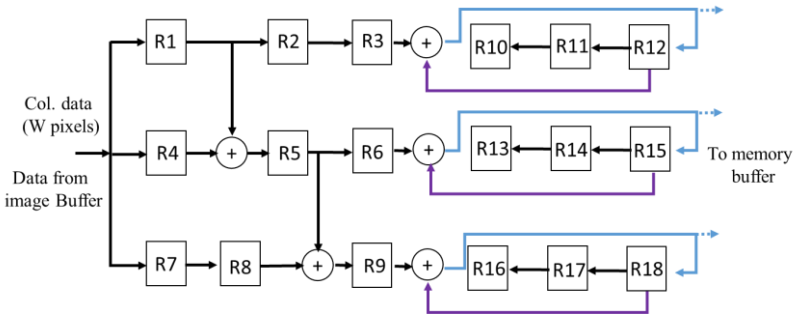

Fig. 6. Architecture for column wise calculation of integral values

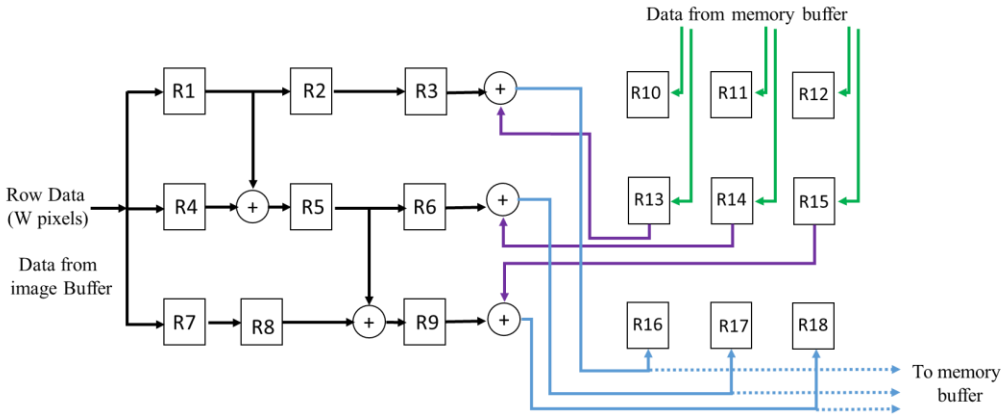

Fig. 7. Architecture for row wise calculation of integral values

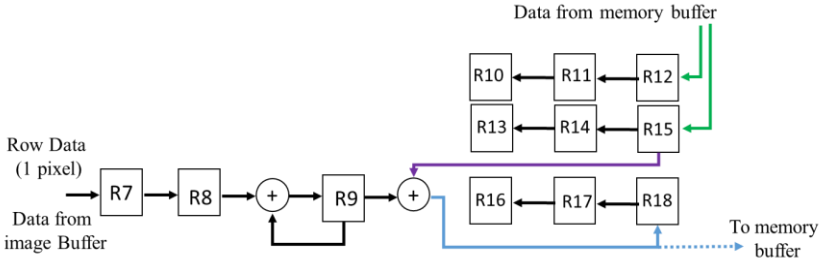

Fig. 8. Architecture for integral value calculation of one position $i(x, y)$.

column of the window buffer, and the previous column of the integral buffer values. Integral buffer values are shifted towards left with each clock cycle. Once the search window reaches the maximum column of the image, the window buffer starts fetching row-wise data (second approach) from the input image. The integral buffer fetches integral values from memory buffer and calculates the integral value only for the last row of the search window as sin Fig. 7. In the next clock cycle as presented in Fig. 4, integral value for only one (x, y) position is needed to form a new window. The architecture for the third and last approach is presented in Fig. 8. Whenever an integral window reaches the maximum number of columns, the memory buffer values are shifted upwards to make room for the next upcoming integral values. The classifier engine directly uses the integral image buffer for further computations, which reduces the memory requirement to save the complete integral image values.

IV. FUNCATIONALITY TEST AND SYSTEM PERFORMANCE ANALYSIS

The proposed approach utilizes the advantages of pipelining technique along with the multiplexing of the resources. The latency for the proposed architecture is W clock cycles for W integral value computation and $2W$ clock cycles for the complete $W \times W$ window. However, after the initial latency, at each clock cycle, we got one complete window of integral values. The critical path timing of the architecture is T_A, where T_A presents the computation time of the adder unit. Therefore, the sampling frequency of the module is $1/T_A$ i.e. after every $1/T_A$ clock cycle we can fetch a new input pixel. The resource requirement for the proposed architecture includes an image buffer, an integral buffer and one intermediate memory buffer. The resources requirement for window size of W and image size of $M \times N$, in terms of registers and adders are illustrated in TABLE I

TABLE I

RESOURCE REQUIREMENT FOR THE COMPLETE INTEGRAL MODULE

Buffer units	Register	Adder
Window Buffer	$W \times W$	W - 1
Integral Buffer	$W \times W$	W
Memory Buffer	$W \times N$	0
Overall integral module	$2(W \times W) + (W \times N)$	2 W - 1

It is evident from the TABLE 1 that except for the intermediate memory buffer size which dependes on the image width, the proposed architecture's resource requirement does not vary with the image resolution. Also, considering the image resolution as 320 × 240, and window size as 20×20, the total number of the sliding window are 66,220. The proposed architecture utilizes 39 adders for 522 windows for the rest of the windows (i.e. 66,220-522 = 65,698 windows) it only requires 2 adders. The run-time

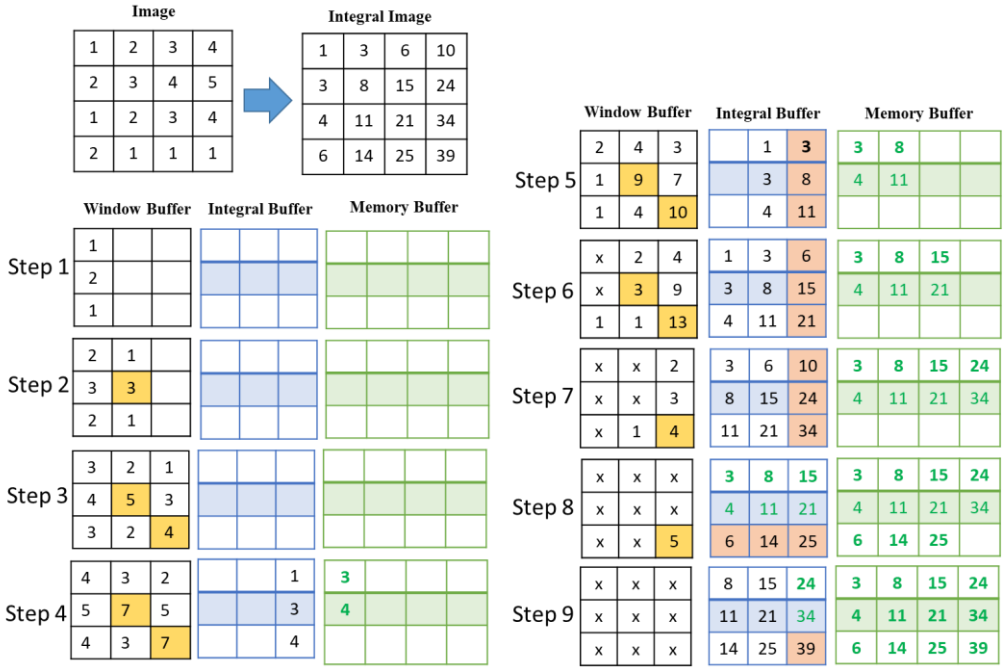

Fig. 9. Proposed integral image calculation steps

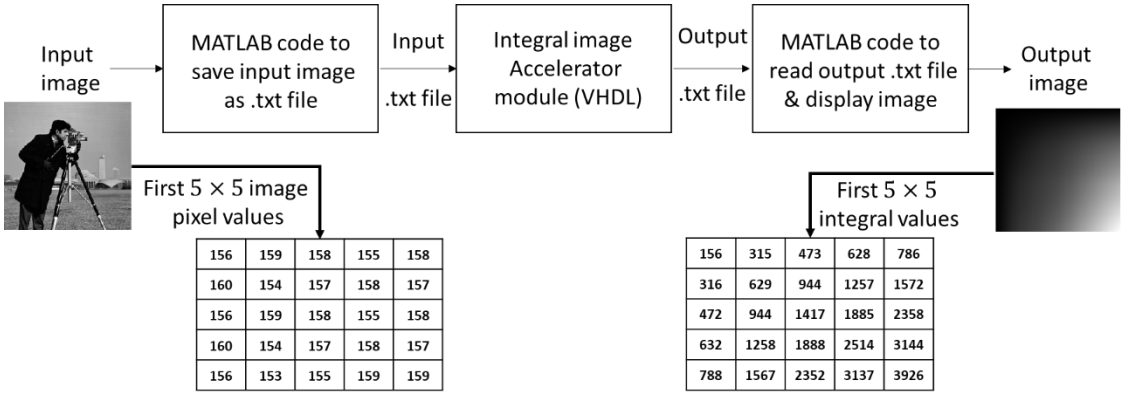

Fig. 10. Behavioural simulation process and result of the proposed integral image module

analysis for the proposed architecture is presented in TABLE II. It is evident that the complete processing speed 0.0005 seconds (2000 FPS) and 0.002 seconds (500 FPS) for 320 × 240 and 640 × 480 image resolution respectively. The number of sliding windows required to scan the complete image puts little more burden on the proposed implementation. Thus, the run-time complexity of our architecture is comparatively high with respect to [3]. TABLE II also illustrates that the initial latency to generate the first window of integral values is only time $2WT_A$ unit and just one clock cycle is needed to get a new integral window.

The differences in computing technologies (multi-core processors, GPU, FPGA (architecture, part and speed grade)), applications, image size and performance measures, fair and meaningful comparisons of our proposed algorithm with other implementations is a difficult task. Therefore, at the first level of analysis, we have directly compared the architecture itself. In this

TABLE II
TIMING CALCULATION FOR THE INTEGRAL IMAGE MODULE, WHERE W = 20, T_A = 8.103ns.

Image size	Total number of $W \times W$ windows	Total time required for integral image module (Total number of windows -1)T_A + Latency for the first window	Total time (s)
320 × 240	66,220	(66,220 -1) T_A + (2 × $W \times T_A$)	0.0005
640 × 480	2,85,660	(2,85,660-1) T_A+ (2 × $W \times T_A$)	0.002

paper, we have compared our proposed architecture with the work proposed by Ouyang *et al.* [3] and Srivastava *et al.* [9] (presented in TABLE III). The integral module designed in [9] usage same number of adder unit i.e. $2W-1$ with $W\times N$ BRAMs and $(W\times 2W)+ W\times W$ register units. In [3] the authors have used the pipelining based approach to exploit the maximum parallelism. The complete image is divided into the number of strips (strip width W) and each strip is executed in pipelined fashion. For $W = 32$ optimum result with 95 numbers of adders for $W \times W$ structure of the calculation unit is presented. However, the actual number of adders required in [3] for complete image is $\left(\frac{M}{W}\ strips \times (W + 2M - 1)\right)$ adder units. In our approach with window size equal to 32 it only takes $2W-1$ i.e. 63 adder modules. Also, these 63 adders are for complete image computation, which are very less compared to the method proposed in [3]. Moreover, as we are storing only $W \times N$ integral values on cache memory buffer instead of the integral values, it also reduces the

TABLE III
HARDWARE RESOURCE COMPARISON FOR AN IMAGE SIZE OF 640 × 480 AND WINDOW SIZE OF 32.

Resources	P. Ouyang *et al.* [3]	Srivastava *et al.* [9]	Proposed work
Registers	423040	3072	10400
Adder units	26220	63	63
BRAM	0	32 BRAM with depth of 640	0

memory requirement drastically. Further, we have also simulated our architecture using VHDL on VIVADO® Design Suite 2018.2 for functionality check. The simulation steps along with the input image and corresponding output integral image is presented in Fig. 10. Matlab 2018.2 is used to generate the input text file from the input image and to convert the ouput text file into the resultant integral image. The simulation result validates the functionality of the proposed architecture.

V. CONCLUSION

In the quest for optimum speed yet efficient hardware implementation of the integral image module (for face detection application), a pipelined hardware architecture is proposed in this paper. The overlapping property of the sliding window is used to reduce the resource utilization. The designing tactics such as time-multiplexed usage of the hardware resources and saving only a small part of the integral image also facilitates to generate the integral image module with minimum resources. The simulation of the proposed architecture using VIVADO® Design Suite 2018.2 for the ZC702 board validates the functionality of the architecture. The authors are engaged in the testing and analysis of the proposed architecture on the hardware platform.

ACKNOWLEDGEMENT

This work was partially supported by the Board of Research in Nuclear Sciences (BRNS), Government of India, under grant number 34/14/08/2016.

REFERENCES

[1] M.-H. Yang, D. J. Kriegman, and N. Ahuja, "Detecting faces in images: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 1, pp. 34–58, 2002.

[2] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, 2001, vol. 1, pp. I–I.

[3] P. Ouyang, S. Yin, Y. Zhang, L. Liu, and S. Wei, "A fast integral image computing hardware architecture with high power and area efficiency," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 62, no. 1, pp. 75–79, 2014.

[4] S. Ehsan, A. F. Clark, N. U. Rehman, and K. D. McDonald-Maier, "Integral images: efficient algorithms for their computation and storage in resource-constrained embedded vision systems," *Sensors*, vol. 15, no. 7, pp. 16804–16830, 2015.

[5] S. Yin, P. Ouyang, X. Dai, L. Liu, and S. Wei, "An adaboost-based face detection system using parallel configurable architecture with optimized computation," *IEEE Syst. J.*, vol. 11, no. 1, pp. 260–271, 2015.

[6] C. Kyrkou and T. Theocharides, "A flexible parallel hardware architecture for AdaBoost-based real-time object detection," *IEEE Trans. very large scale Integr. Syst.*, vol. 19, no. 6, pp. 1034–1047, 2010.

[7] B. Kisacanin, "Integral image optimizations for embedded vision applications," in *2008 IEEE Southwest Symposium on Image Analysis and Interpretation*, 2008, pp. 181–184.

[8] F. Spagnolo, P. Corsonello, and S. Perri, "Efficient architecture for integral image computation on heterogeneous FPGAs," in *2019 15th Conference on Ph. D Research in Microelectronics and Electronics (PRIME)*, 2019, pp. 229–232.

[9] Srivastava, Nitish Kumar, et al.,"Accelerating face detection on programmable SoC using C-based synthesis. ," *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2017.