A high frame-rate of cell-based histogram-oriented gradients human detector architecture implemented in field programmable gate arrays

Syifaul Fuada[1], Trio Adiono[2], Hans Kasan[3]
[1]Hardware Laboratory, Program Studi Sistem Telekomunikasi, Regional Campus of Purwakarta, Universitas Pendidikan Indonesia, Purwakarta, Indonesia
[2]IC Design Laboratory, Department of Electrical Engineering, School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Bandung, Indonesia
[3]Computer system and Network Laboratory, School of Electrical Engineering and Informatics, Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea

Article Info

Article history:

Received Sep 6, 2021
Revised Aug 30, 2022
Accepted Sep 29, 2022

Keywords:

Field programmable gate arrays
HOG algorithm
Human detection
Parallel architecture
Pipeline architecture

ABSTRACT

In respect of the accuracy, one of the well-known techniques for human detection is the histogram-oriented gradients (HOG) method. Unfortunately, the HOG feature calculation is highly complex and computationally intensive. Thus, in this research, we aim to achieve a resource-efficient and low-power HOG hardware architecture while maintaining its high frame-rate performance for real-time processing. A hardware architecture for human detection in 2D images using simplified HOG algorithm was introduced in this paper. To increase the frame-rate, we simplify the HOG computation while maintaining the detection quality. In the hardware architecture, we design a cell-based processing method instead of a window-based method. Moreover, 64 parallel and pipeline architectures were used to increase the processing speed. Our pipeline architecture can significantly reduce memory bandwidth and avoid any external memory utilization. an altera field programmable gate arrays (FPGA) E2-115 was employed to evaluate the design. The evaluation results show that our design achieves performance up to 86.51 frame rate per second (Fps) with a relatively low operating frequency (27 MHz). It consumes 48,360 logic elements (LEs) and 4,363 registers. The performance test results reveal that the proposed solution exhibits a trade-off between Fps, clock frequency, the use of registers, and Fps-to-clock ratio.

Corresponding Author:

Trio Adiono
Department of Electrical Engineering, School of Electrical Engineering and Informatics, Institut Teknologi Bandung
Gd. Ahmad Bakrie Lt. III, ITB Campus Bandung, Jl. Ganesa No.10, Lb. Siliwangi, Kecamatan Coblong, Kota Bandung (40132), Jawa Barat, Indonesia
Email: tadiono@stei.itb.ac.id

1. INTRODUCTION

Image processing has been widely utilized to detect humans [1], [2]. However, detecting humans in an image is challenging due to their various and wide range of appearance variables [3]. To recognize the existence of humans accurately, extensive computations are required. The system should be highly accurate while still able to maintain low-energy and low resource consumption on real-time processing. In respect of the accuracy, one of the well-known techniques for human detection is the histogram-oriented gradients

(HOG) method; this method is initially proposed by [3] in 2005. In the publication as mentioned earlier, the HOG technique was proven to outperform other existing human detection techniques significantly. Since then, many researchers have modified the HOG technique to increase the detector performance. Moreover, since HOG has a promising accuracy, the idea to combine the method along with other classification algorithms to distinguish humans under challenging conditions such as illumination, rotation, and even deformation is promising [4]. Besides human detection application, HOG has been widely applied in various cases [5]–[7], such as pedestrian detection, classical dance classification [8], vehicle detection [9], traffic sign detection, crowd density estimation, general object detection, object tracking, feature matching, feature descriptors [10], anomaly detection, digit recognition, and so on.

HOG based human detection has been explored in many aspects. For example, a cascade-of-rejector approach, which is usually utilized for face recognition, was combined with HOG features by [11] to get better accuracy. Jia and Zhang [12] combined the HOG method with Viola’s face detection framework to perform real-time human detection processing. Zhang *et al.* [13] reported a computational cost reduction using a multi-resolution framework. Wang *et al.* [14] combined the HOG method with local binary pattern (LBP) as the feature sets in the human detection system. Schwartz *et al.* [15] utilized a partial least square analysis to provide a richer descriptor. It was similar to edge-based features that utilizes additional color and texture information. There was also research that attempted to combine HOG with human’s body ratio estimation technique to distinguish human from nonhuman category [16]. However, most of these works (on the HOG topics) explored and experimented with improving the accuracy performance using combinations of HOG features and other potential techniques. These accuracy improvements tend to have higher computational costs and complexity than the original HOG algorithm as the result of the technique combinations. Reducing the computation resources is necessary since hardware implementation of HOG is very possible [17].

Hardware implementation offers better speed performance and power efficiency to keep up with real-time processing requirements [7]. Hence it is expected to provide better performance than the software implementation. Many works, as in [18]–[24] utilized a field programmable gate array (FPGA) to implement HOG in hardware as it is able to accommodate parallel architectures and suitable for real-time image processing [18], [25]. Moreover, it can maintain the design configurable and shorten the design time-to-market [26]. Unfortunately, the HOG feature calculation is complicated [4]. Although hardware implementation offers a high-speed computation, it could lavishly consume resource and power if not appropriately designed. On the other hand, resource-efficient and low-power systems are currently in high demand. Trends of electronics and applications are going toward green technology, in which case, resource, and energy consumption are important aspects of being considered (*i.e.,* as low as possible). For this reason, in this paper, we designed a simplified HOG algorithm, digital hardware architecture and its FPGA implementation. The proposed design is dedicated to low-power and resource-efficient characteristics.

Section 1 of this paper explains the research background and a glance on the HOG technique. In section 2, the simplified algorithm is presented along with the equations that have been remodeled to avoid large division operations. Later, we describe our hardware architecture that realizes the simplified algorithm presented in section 3; it is then followed by the FPGA implementation and its functionality and verification results. Performance evaluation is also presented in section 3, and it is enriched with benchmark comparisons with other techniques. Finally, we draw a concluding remark to highlight the research and its significant contribution.

2. MODIFIED HOG ALGORITM

To reduce the computation complexity, we simplify the computation of HOG-based human detection algorithm to make it suitable for hardware implementations. Despite the computational reductions, detection quality can still be maintained. The original HOG algorithm has high computational complexity due to its division operations and intensive looping operations in its window-based processing. Thus, it is more suitable to be implemented using the software due to its complex processes. It has to be simplified and modified to suppress its cost and power consumption to be ideal for application-specific integrated circuit (ASIC) implementation, which is commonly referred to as its redundancy and concurrency may be exploited for parallel and pipeline processing (as addressed by [27]). The designed hardware architecture specifications are presented in Table 1.

Based on the basic idea of HOG algorithm, the input image is divided into cells (*C*), blocks (*B*), and windows (*W*) [3], as illustrated in Figure 1. The normalized gradients for these properties, are eventually collected over a Window-based detection for person or non-person classification. The cells consist of 8×8 pixels. Therefore, there are 80×60 cells within a frame. We index the cell in raster scan from 1 to 4,800 (C_1 to $C_{4,800}$). Every 2×2 cells are grouped into a block. There are 50% overlapping Cell data between each Block and its neighbor blocks, both in the horizontal and vertical direction. Therefore, we will have 79×59 Blocks within a frame, indexed as 1 to 4,661 (B_1 to $B_{4,661}$). The window consists of 8x16 cells or equivalent to

64×128 pixels. These numbers of cell size, block size, window size and block overlap give the least miss rate compared to other sizes and overlaps [3]. The window will be moved by one cell column or one cell row after each evaluation. The pixels in the window will be classified by the support vector machine (SVM) for human detection. There are in total 3,285 windows to be analyzed in a 640×480 pixels image. To be fit for hardware implementation, we modify the HOG algorithm by proposing cell-based processing, cell derivatives with neighboring edge anti-aliasing, magnitude calculation using linear approach, fixed-weighted binning, block normalization using newton-raphson algorithm, block-wise SVM classification and fixed-point representation methods. The detail of each technique will be described in the following section.

Table 1. System specification

Parameter	Value
Image size	640×480 pixels
Window (*W*) size	64×128 pixels
Block (*B*) sizes and Cell (*C*) size	2×2 cells and 8×8 pixels
Bin size	9 (0^o–180^o)
HOG feature size	3,780
#Window and #Block row and #Block column	3,285 and 15 and 7
Gradient binning	Linear approach to avoid Euclidean distance (*L2-norm*)
Histogram normalization	Manhattan distance (*L1-norm*) to avoid Euclidean distance (*L2-norm*)
Block overlap	50%
System clock	27 MHz

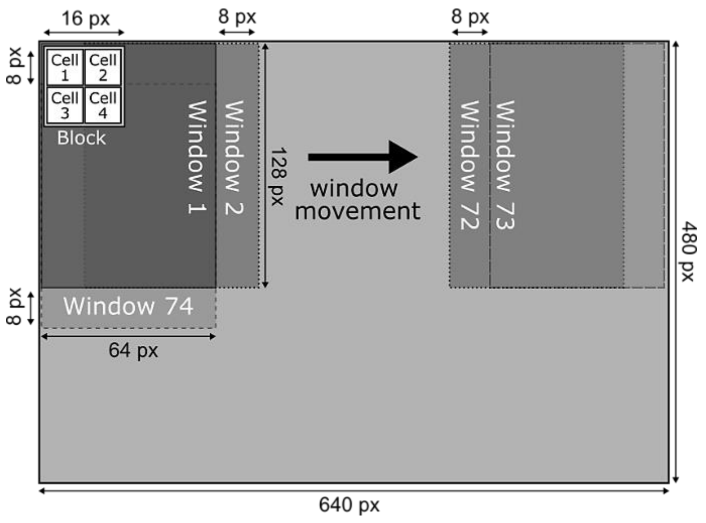


Figure 1. Illustration of HOG image structure, (discalimer: the figure is not drawn to real scale)

2.1. Cell-based processing

Instead of using Window-based, we used Cell-based processing for computing the derivative value in the *x*-direction (*dx*) and in *y*-direction (*dy*). Figure 2(a) and Figure 2(b) illustrate the how the window-based and cell-based processing in raster scan is executed, respectively. Using cell-based processing, we can extremely reduce derivative computation redundancy by skipping overlapped cell data computations in window-based processing.

As shown in Figure 2(a), cell-based processing eliminates large overlapped cell area, which results in low computational complexity as well as low memory bandwidth requirements [28]. However, the derivative value (*dx* and *dy*) results are still identical to the original Window-based HOG algorithm. This method can be applied because we can reuse the computed cell derivative (*dx* and *dy*) data for different Windows instead of recalculating the derivative of all the cells inside a window each time a new window is evaluated. This proposed method is different from [22], where the computation of the overlapped data is done using complex pipeline stage. In this method, we store the calculated cell data in temporary random-access memory (RAM). Each time the system analyzes a new window, it will fetch the respective cell calculation results from the RAM, hence avoiding unnecessary recalculation.

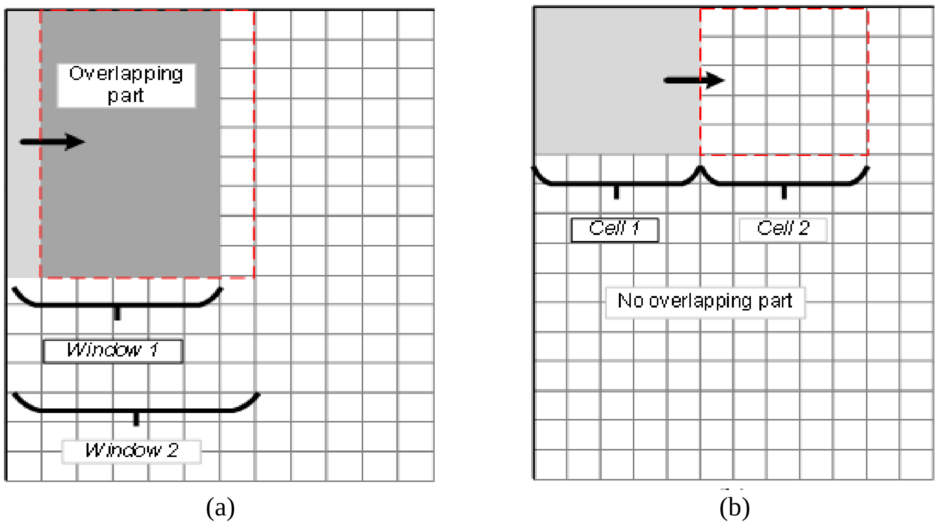


Figure 2. Comparison between two approaches on HOG: (a) Window-based raster scanning and (b) Cell-based raster scanning

2.2. Cell derivates with edge neighboring anti-aliasing

In the HOG algorithm, the derivative values (*dx* and *dy*) are computed for every pixel using convolution kernel as (1). Since we utilize cell-based calculation, there will be many edges within a window [29]. The edges may produce invalid *dx* and *dy* values of pixels located in corner areas as the pixels only possess one adjacent pixel instead of two. In order to combat this problem, we assign the *dx* and *dy* values of pixels in the edge areas to similar values to its neighbor pixels, as shown in Figure 3. We apply this method to pixels located in both horizontal and vertical edges. As illustrated in Figure 3, grey-colored squares represent pixels with distinct derivatives. meanwhile, blue- and yellow-colored squares represent pixels with identical derivatives as the result of duplicating the derivative values of adjacent pixels.

$$h_x = [-1 \quad 0 \quad 1\,],\, h_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{1}$$

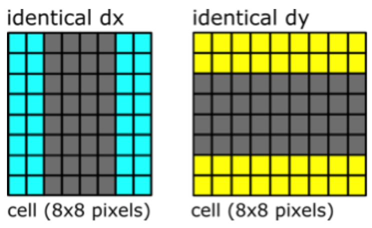


Figure 3. The *dx* and *dy* values of cell (8×8 pixels) in the edges are similar to the derivative values of their neighbor pixels (color version of this images can be distinguished in the online article version)

2.3. Magnitude calculation using linear approach

The original HOG method used euclidean distance (*L2-norm*) of each derivative value (*dx* and *dy*) to get the magnitude of each pixel. However, the equation of *L2-norm* consists of a square-root calculation [30], [31], which is very complex to be implemented in the hardware [32]. Therefore, this complicated computation needs to be avoided by other approaches for estimation purposes. In this work, we use a linear method (2) to calculate the magnitude, instead of *L2-norm*. Figure 4 reveals the comparison between *L2-norm* (*X*: 30, *Y*: 40) and linear methods (*X*: 30, *Y*: 42.43).

$$M(x,y) = \begin{cases} \frac{dy}{3} + dx & \text{if } dx \geq dy \\ \frac{dx}{3} + dy & \text{if } dx < dy \end{cases} \tag{2}$$

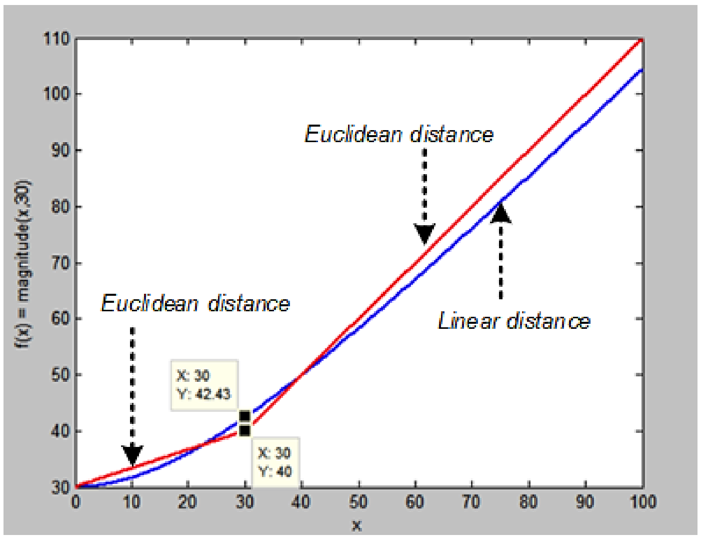


Figure 4. The comparison of magnitude results between *L2-norm* (red-colored track) and the linear (blue-colored track). Color version of this images can be distinguished in the online article version

As formulated in (2), the magnitude is denoted as *M (x, y)*. Our linear method can significantly reduce computational complexity as it merely uses addition and division-by-constant operations. The division by three can be implemented by simple shifting and adding function as described in (3).

$$\frac{a}{3} = (a \gg 2) + (a \gg 4) + (a \gg 6) \tag{3}$$

Based on Figure 4, it can be seen that this simplification is able to deliver a satisfactory estimation of the actual *L2-norm*. Three different approximations –for calculating magnitude, angle and distance– will be used to avoid square root and divisions in the processing of the HOG. In this work, we do not examine the effects of such simplifications as it only to show the proposed linear method compared to *L2-norm* on a graph, as shown in Figure 4. The overall accuracy for the proposed system will be introduced in the future work as we will provide the error rate for a specific set of benchmarks.

2.4. Fixed weighted binning

For Histogram function, pixel angle can be calculated using complex *arctangent* and division operations. However, computing pixel angle with *arctangent* function and division will result in a very complex computation and of course, it is not suitable for hardware implementation. Furthermore, since pixel angles are computed for all pixels, there will be a lot of data to be analyzed. This computation demands very large computation cycles and latency. To cope with the requirements, we use a simplified method by setting a fixed region of bin for every 10º (*i.e.*, *tangent* 10º, 30º, 50º, 70º, 90º, 110º, 130º, 150º, and 170º). Suppose *tangent* 10º = 0.1763269807, then it will similarly equal to $2^{-3} + 2^{-5} + 2^{-6}$, which is 0.171875. Thus, we will have 9 bins with its approximated *tangent* values, as shown in Table 2.

Table 2. Tangent value approximations

Tangent	Approximated value
tan (10º)	$2^{-3} + 2^{-5} + 2^{-6}$
tan (30º)	$2^{-1} + 2^{-4} + 2^{-6}$
tan (…)	…
tan (170º)	$- 2^{-3} - 2^{-5} - 2^{-6}$

The pixel angle of pixel *A (x, y)*, is computed from derivative *dx* and *dy* values using the following pseudocode:

Algorithm 1 Pixel angle pseudocode

```
i =1;
$\theta_i = 10$;
```

```
$\theta_{(i+1)} = 20;$
$\boldsymbol{while}\ (d_x \,.\, tan\,(\theta_i) < d_y \leq d_x \,.\, tan\,(\theta_{(i+1)})\ \{$
$A(x,y) = \theta_{(i+1)};$
$\boldsymbol{exit}();\}$
$\boldsymbol{else}\ \{$
$\theta_i = \theta_{(i+1)};$
$\theta_{(i+1)} = \theta_{(i+1)} + 10;\}$
```

The *tangent* multiplication can be approximated with bit shifting and addition operations in the hardware implementation. By considering computed pixel value *A (x, y)*, we can calculate the Histogram using the rules as described in Table 3. If the pixel angle *A (x, y)* lies on the bin center, then the magnitude value *M (x, y)* value is entirely stored into the respective bin. For example, if *A (4,4)* = 20^o and *M (4,4)* = 1.5, then *bin#1* = 1.5. On the other hands, if the pixel angle *A (x, y)* lies on the bin boundary, then the magnitude value *M (x, y)* is split equally into both neighboring bins. For example, if *A (5,5)* = 30^o and *M (5,5)* = 4, then *bin#1* = 2 and *bin#2* = 2. This scheme has also been used by [33]–[35].

Table 3. Bin value rules

Angle	Bin center?	Target Bin	Weight
0^o	No	*#1* and *#9*	*M (x, y)*/2
10^o	Yes	*#1*	*M (x, y)*
20^o	No	*#1* and *#2*	*M (x, y)*/2
…	…	…	…

2.5. Block normalization using newton-raphson method

There are several normalization methods that can be employed to normalize the Histogram, such as *L2-norm* and Manhattan distance (*L1-norm*) [36]–[38]. In this case, *L1-norm* is more suitable for hardware implementation as it does not use square root operations unlike *L2-norm*, even though further simplification approaches are still required. Vector normalization is obtained using (4), where *L1-sum* is Manhattan distance summer.

$$v_{norm} = \frac{v}{|v|\ L1{-}sum} \tag{4}$$

Since $v_{\text{norm}} = v \times d$, the distance d is stated as (5),

$$d = \frac{v}{|v|\ L1{-}sum} \tag{5}$$

To calculate *d*, newton-raphson approximation is used as in (8). It is derived from (6) and (7) that are formula for x_0 and x_1 on a newton-raphson digital blocks.

$$x_0 = (3 \ll n) - (2 \times sum) \tag{6}$$

$$x_1 = x_0\,[(2 \ll 2n) - (sum \times x_0)] \tag{7}$$

$$[representation]\ d \ll 12 = x_1 \gg (4n - 12) \tag{8}$$

Where *n* is defined as *n* = *MSB (sum)*. For instance, if sum = 13, then *n* = bit 4. The result will be delivered in decimal fraction numbers.

2.5.1. Blockwise SVM classification

The idea of this method is to multiply the SVM coefficient blockwise, instead of per-window. However, it is important to note that *block#1* corresponds only to *window#1*, but *block#2* corresponds to both *window#1* and *window#2*. Thus, *block#2* will be used for SVM classifications of *block#1* and *block#2*. This also applies to other blocks that correspond to multiple windows. Section 3 (results and discussion) will further explain the hardware design, which takes advantage of pipelined architecture to handle these complicated calculations. The SVM coefficients are trained with the simplified algorithm. We used the *libsvm* library to train our SVM with massachusetts institute of technology (MIT) pedestrian dataset. Then, we examined the linear SVM and retrained the false positives. The number of images used for classifier training is amount of 924 and 13,680 for positive and negative images trained, respectively.

2.5.2. Fixed-point representation

Fixed-point is used to represent the fractional data. The data-width of all the modules is depicted in Table 4; it contains input pixel, derivatives, magnitude, and so on. The bit-width is determined by searching for the shortest bit-width in each module that will not cause any bit-overflow or interfere with the calculation results.

Table 4. Functional module bit-width optimization

Module	Sign	Bit-Width	Data Type
Input pixel	Unsigned	8	Integer
Derivates	Signed	9	Integer
Magnitude	Unsigned	9	Integer
Histogram bin	Unsigned	15	Integer
Normalized	Unsigned	12	Fraction, ≪ 12
SVM coefficient	Signed	14	Fraction, ≪ 12
Window score	Signed	32	Integer
Detection	Unsigned	1	Integer

3. RESULTS AND DISCUSSION

3.1. Hardware architecture implementation

Our system block diagram is shown in Figure 5. The input of the system is 640×480 images. The output is a grayscale image on an external display. The output image will be marked in parts of the image that are believed to be human figures. The system is comprised of a control unit, derivative, gradient binning using linear approach, cell grouping, Histogram normalization using *L1-norm*, and sliding window and SVM classification modules.

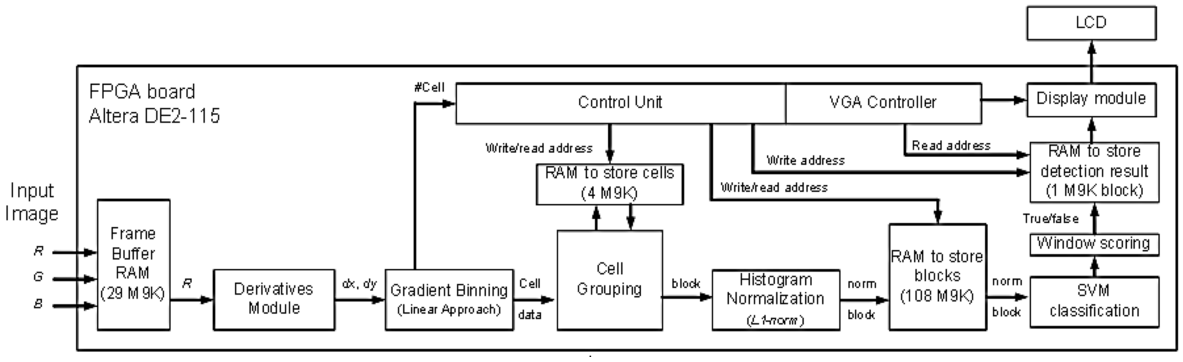


Figure 5. Block diagram of the proposed system with 29 M9K frame buffer RAM

In order to increase the processing speed and enable the system to work in a real-time, we applied pipeline architecture to our system, as reported in Table 5. The M9K is memory block of altera FPGA DE2-115. The pipeline architecture also enables us to reduce the memory bandwidth as it does not require all cell and block values to be stored, but only some blocks that correspond to the windows being processed at that time. Consequently, we are able to use embedded RAM (internal RAM) as the processing storage instead of external RAM, which translates to a significant reduction of pins utilization and power consumption. Table 6 reveals the benefits introduced by a pipeline mode in the proposed hardware, which obtained from syntesis process. The pipeline architecture enables us to reduce the clock cycle latency dramatically. The overall process without pipelining is 13,112; it is obtained from the summation of three digital blocks (*i.e.,* cell grouping, block normalization, and SVM classification proceses). However, as a note, pipeline does not allow any latency improvements for the SVM classification, the block normalization and the cell grouping modules.

Control unit module is used to generate read and write addresses for embedded RAM used in each module. This module plays an important role in our pipeline architecture since the RAM access scheduling should be accurate at all times. Embedded RAMs used and its memory size is Table 6.

The derivative module calculates the cell-based derivative (*dx* and *dy*) of the image; this block contains pixel derivatives as shown in Figure 6 and anti-aliasing filter as shown in Figure 3. The anti-aliasing filter applied to the pixels in the image edges is to overcome the zero-padding convolution problem. We designed highly parallel architecture, as shown in Figure 7. Our architecture can simultaneously calculate the derivatives and gradient bins for 64 pixels by calculating them simultaneously. This will significantly reduce clock latency.

Table 5. Reduced cycle count by pipeline architecture

Blocks	Without pipelining	With pipelining
Cell grouping	4,800	4,800
Histogram normalization	4,719	4,719
SVM Classification	3,593	3,593
Overall process	13,112	4,888

Table 6. Embedded RAMs

Embedded RAM	Size	Data-Width
Frame buffer (input image)	(29 M9K block)	8 Bit
Cell grouping	(4 M9K block)	8 Bit
Sliding window	(108 M9K block)	8 Bit
Store detection result	(1 M9K block)	1 Bit

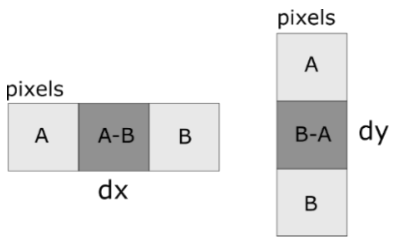


Figure 6. Pixel derivates

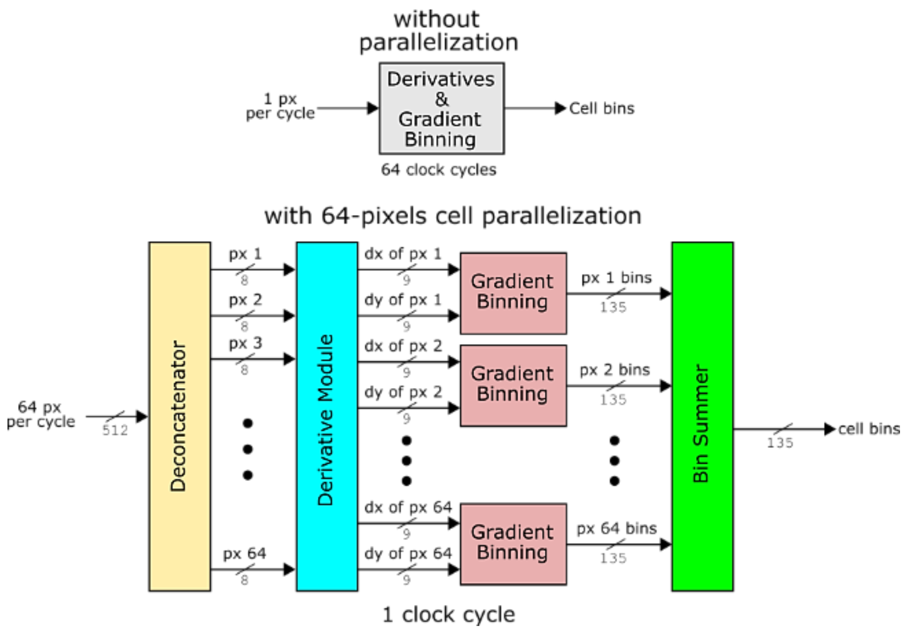


Figure 7. Cell-based parallel architecture

Gradient binning module consists of a rotator, magnitude calculator and binning unit as Figure 8. The rotator unit is comprised of four digital blocks (*i.e.*, inverse, bit extender, comparator and multiplexer units), as shown in Figure 9. The inverse unit is used to negate a number based on two's (2's) complement notation. Bit extender is used to represent a number with larger bits without altering its value. The *dx* and *dy* representation uses 12 bits instead of 9 bits to avoid overflow as the magnitude calculator and binning unit involve shifting and adding the values of *dx* and *dy*. Comparators and multiplexers are used to determine the quadrant of *dx* and *dy*. The magnitude calculator conducts approximation in (3) using four digital blocks (*i.e.*, multiplexer, right shifter, comparator, and adder), as shown in Figure 10. Finally, all the magnitudes are grouped and summed to the respective bins based on the rule specified in (5). The output of this module is a cell histogram data consists of 9 bins × 15 bits that are concatenated into a single line as Figure 11.

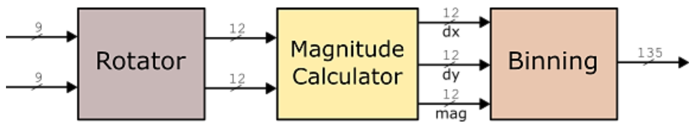


Figure 8. Gradient binning diagram

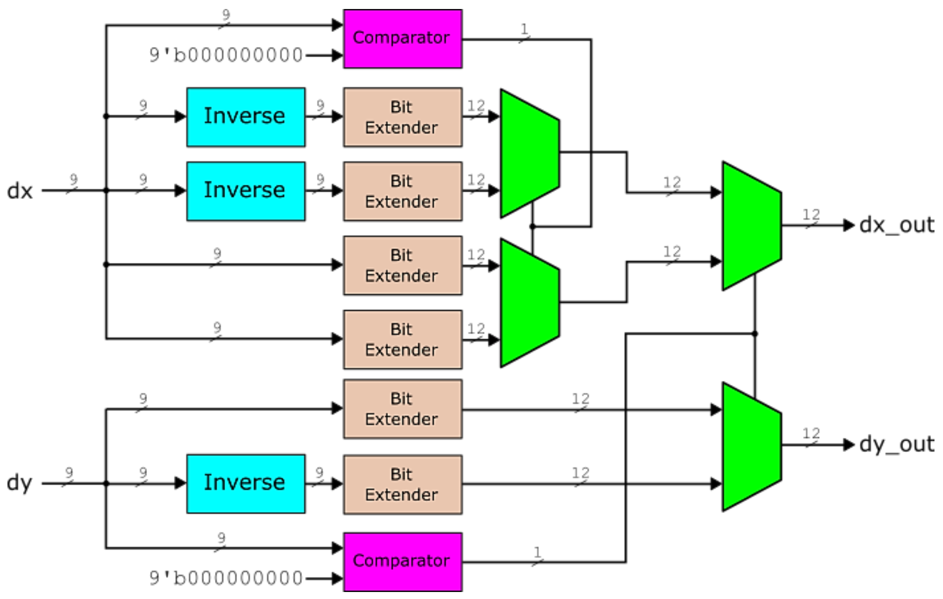


Figure 9. Rotator architecture

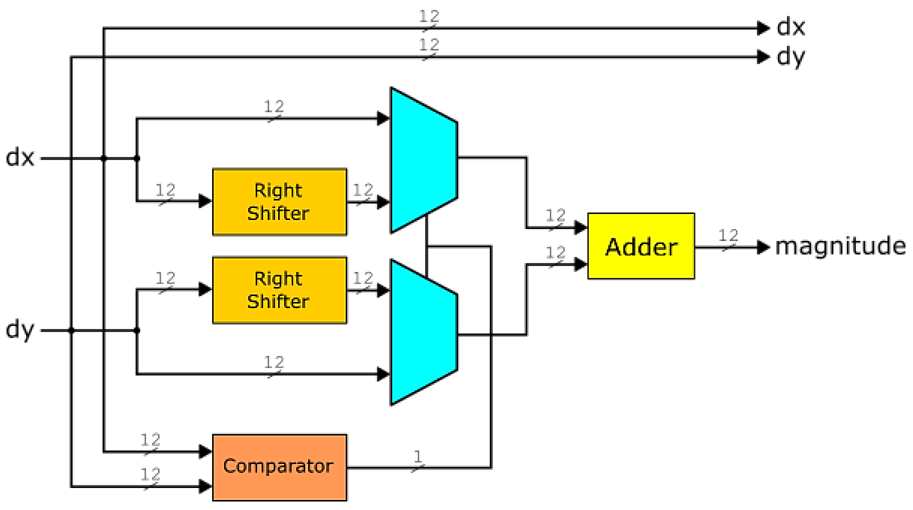


Figure 10. Magnitude calculator architecture

This module is used to group cells into a block and performed block normalization processing. To group it with the raster scan sequence, we use line delay, as shown in Figure 12. It is implemented using RAM to delay 80 cells of data, which is the row size of the input image, as shown in Figure 1. Each Cell consists of 9 bins × 15 bits of data. Using this configuration, we can group four Cells of data that comprise a single Block. For example, when cell *#82* data is fed to the module, *block#1* which consists of *cell #1, #2, #81*, and *#82*, will be constructed. The block data will then be normalized using combinational circuits.

To minimize memory resource usage, we decided to only use line delay RAM with the size of 80 Cells to store all 4800 Cells that will be generated to blocks. The first 80 Cells will be stored initially in the

RAM. However, when *cell#81* is fed into the module, the module reads *cell#1* from RAM and overwrite *cell#81* to *cell#1* in the RAM. *Block#1* will be entirely constructed when *cell#82* is dispensed to the module. There are 82 clock latencies to start the block processing: 1 clock cycle for the register at the line delay input, 1 clock cycle for the register at the output, and 80 clock cycles to fill the line delay RAM initially. After providing the module with the first 82 Cells data, it will consume 3 clock cycles to generate a block. The complete block diagram is depicted in Figure 13.

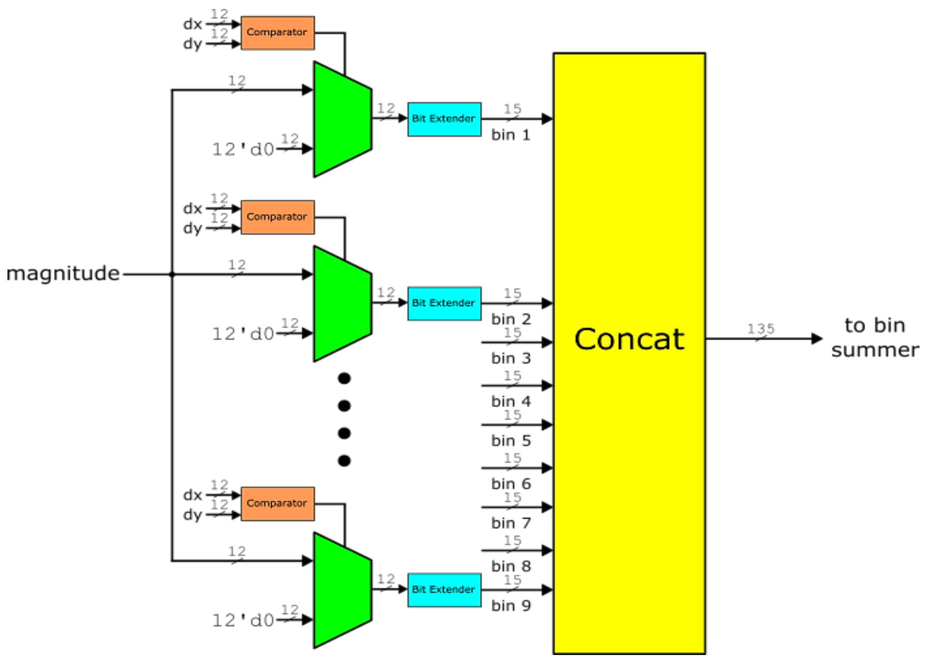


Figure 11. Histogram binning architecture

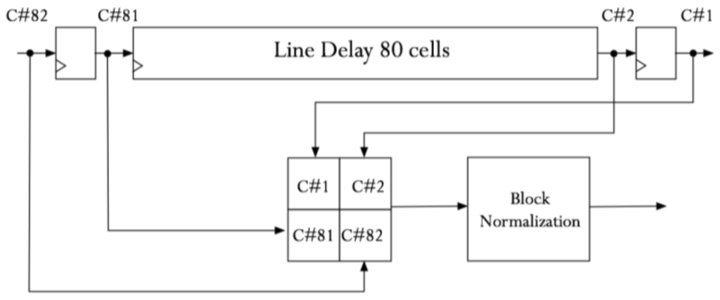


Figure 12. Cell grouping architecture

As stated before, our architecture avoids the usage of any division operation since it is too complex for hardware implementation. Each block should be normalized using *L1-norm* to simplify the calculation. The module contains *L1-sum* to find the Manhattan distance of 9 bin vectors × 4 Cells (36 bin vectors in total). Newton-Raphson algorithm is then used to approximate *d* as in (5). This module uses two multipliers, two subtractors, and bit shifters. Finally, the computed vectors are multiplied by *d*, and then concatenated into a single line data as in Figure 14. This module consumes 1 clock cycle. The results are then stored in the RAM.

The sliding window works in pace with block-wise SVM classification. We designed a highly paralleled and pipelined architecture to be able to calculate 7 windows simultaneously. The SVM classification is done column-wise, because several columns are used to calculate more than one windows. For example, column #1, which consists of block *#1*, *#81*, *#161*, and *#1121*, is used to calculate window *#1*. But column *#2*, which consists of block *#2*, *#82*, *#162*, and *#1122*, is used to calculate both window *#1* and *#2*, and so on. After 7 columns of the respective window consisting of 105 blocks has been calculated, the score will be subtracted by an SVM bias. The comparator will then decide whether there is any person or not using the value of the sign bit. The system will continue to analyze a new window after a window has been calculated until the whole image has been inspected. The hardware architecture is Figure 15.

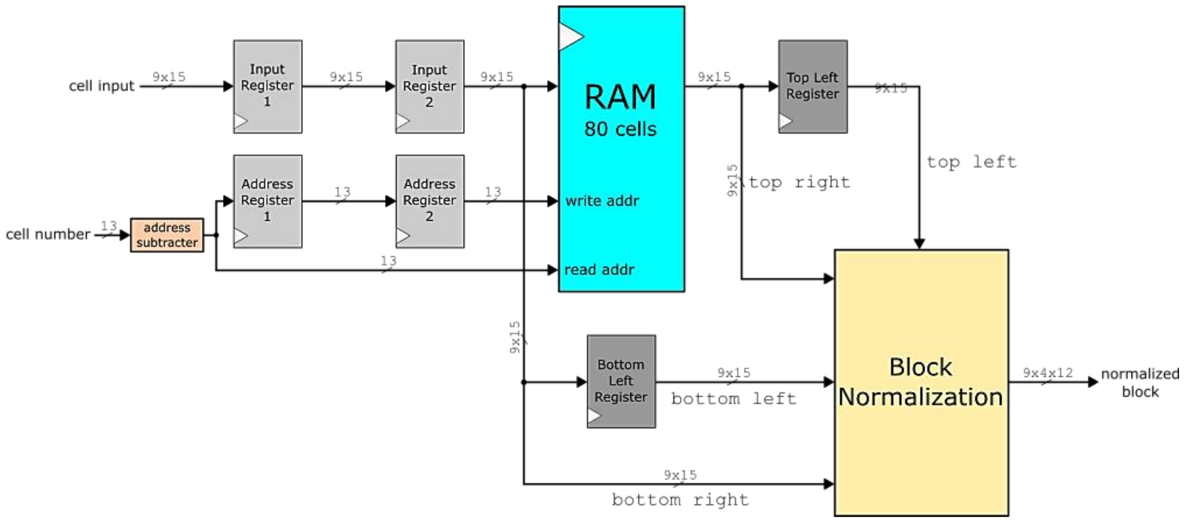


Figure 13. Architecture of Cell grouping hardware design

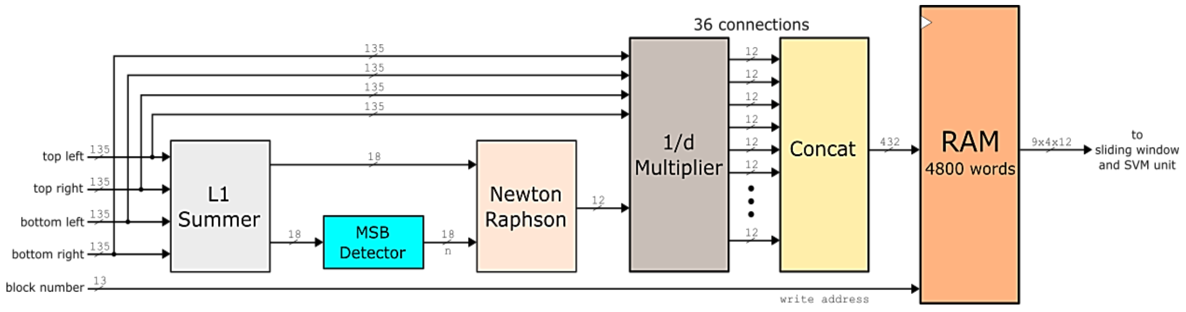


Figure 14. Histogram normalization architecture

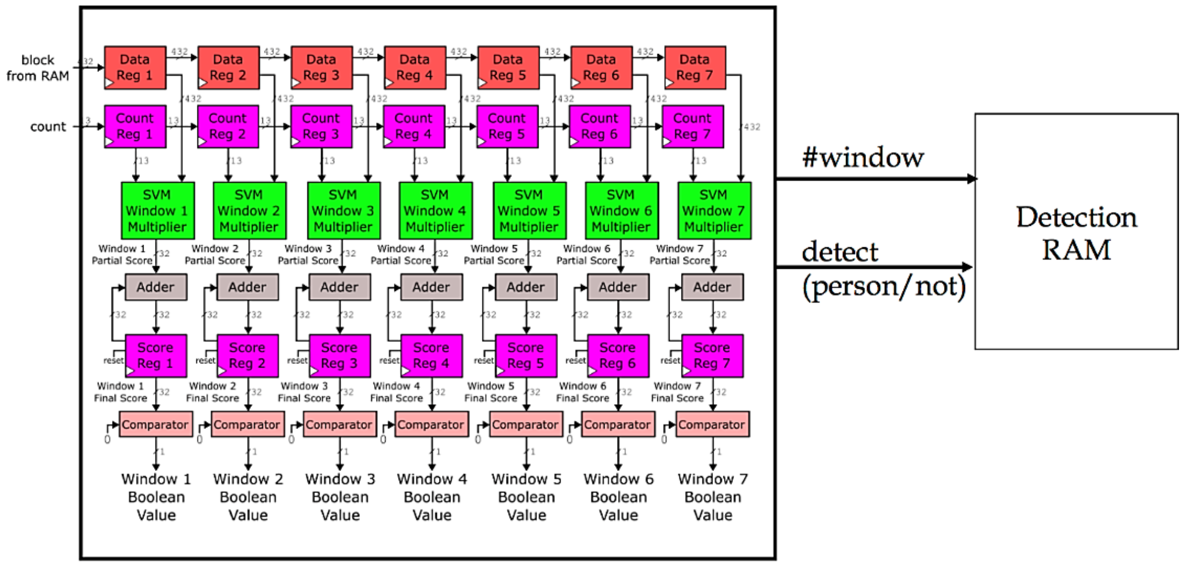


Figure 15. Window sliding and SVM classification

The final module of this design is the display module. In relation with the video graphics array (VGA) controller, this module works by generating read address from frame buffer RAM (input image) and detection RAM (HOG results). As a note, there are no special requirements for the display module because its function is only to drawing a detection box over the object to be detected. It uses 640×480 pixels with 25 MHz VGA clock. Both RAMs are fed by the same clock to read the values. By counting the pixels with front and back porch, the VGA counts to 800×600. Therefore, it needs 800×600 per 25,000,000 second to complete one frame, which is around 52 Fps. In summary, the design must operate at minimum 52 Fps in order to fit the VGA configuration. The display module also generates markings on windows that are considered to contain human figures.

3.1.1. Performance implementation

To evaluate our architecture in terms of effectiveness parameter, we implemented our design in FPGA. We used altera DE2-115 board (Cyclone IV EP4CE115 FPGA chip). The board is connected to an external VGA display to show the resulting image. We tested several 640×480 pixels color images to verify the system functionality. For static detection, we chose the images with relatively small-sized pedestrian images with 128×64 pixels instead of full image 640×480 pixels. The image in Figure 16(a) contains the best pedestrian detection. The pedestrians are quite similar to our positive training dataset. In Figure 16(b), the image has various lighting conditions. However, since HOG uses the gradient feature, our detector can still detect the pedestrians and does not interpret the shadows as humans. On the other hand, in Figure 16(c), the HOG detector may not be able to detect various poses reliably as their gradients will vary. To increase the detector performance, images containing various poses should be used as our training dataset.

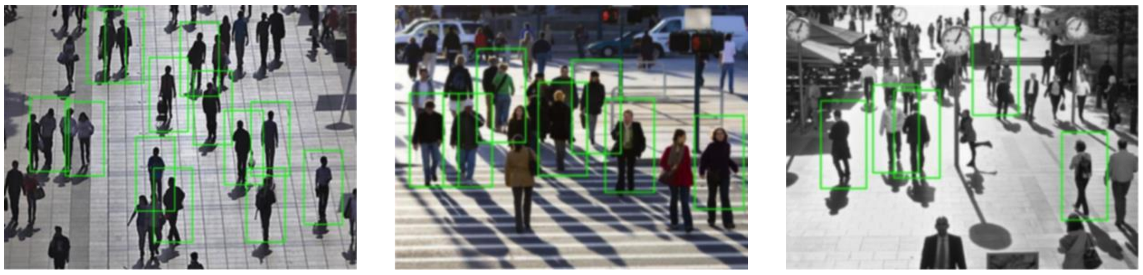
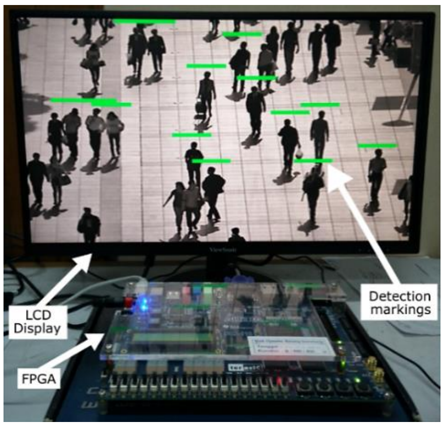

(a) (b) (c)

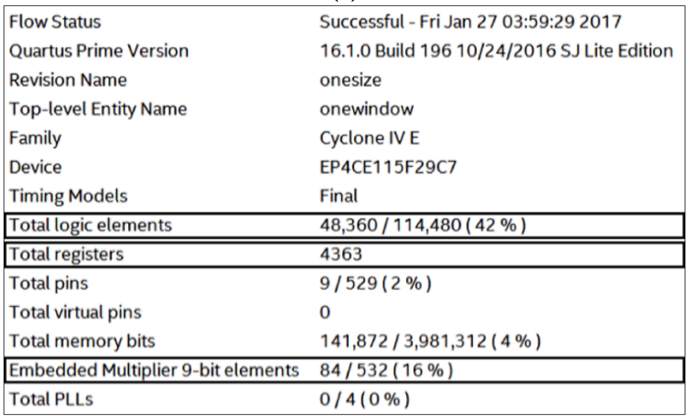
Figure 16. Detection results on pedestrians: (a) under uniform lighting; (b) under various lighting condition and (c) with various poses

Figure 17(a) shows the FPGA displaying the detection result to a monitor. The marks drawn on humans indicate successful detections. Our architecture is coded in Verilog hardware description language (HDL). We use a top-down approach to design the system architecture and a bottom-up approach to code the hardware modules. Each sub-module is designed and tested before being integrated. Figure 17(b) shows the flow summary of the design analysis and synthesis result. The design consumes 48,360 logic elements (LEs), 4,363 registers, and 84 of 9-bit embedded multipliers. It merely consumes 0.141 Mbits of memory. Our architecture requires 4,888 clock cycles to complete one frame detection of image. It also needs 640× 480 = 307,200 clock cycles to receive the image data and store them into the frame buffer. Therefore, the overall system needs 312,088 clock cycles (cacluated from 307,200 + 4,888) to process one frame of image data. It is important to note that the frame buffer embedded RAM is a huge speed bottleneck as it is only able to deliver one pixel every clock cycle.

The TimeQuest Timing Analyzer shows that the maximum operating frequency allowed for the design is 28.62 MHz. By setting the system clock frequency to 27 MHz, we obtained 86.51 Fps (obtained from 27,000,000/312,088). Since the VGA has 52 Fps refresh rate, our design will be suitable to be used with VGA due to the frame output will always be available every time the VGA refreshes. However, the Fps may be improved significantly by reducing the latency of the frame buffer because most of memory resources are consumed by the input buffer. The actual processing unit (without frame buffer) is able to deliver 5,523.732 Fps (obtained from 27,000,000/4,888). This indicates that the frame buffer latency has severely overshadowed the actual capability of the processing unit. It may possible to reduce resources in particular modules and keep the algorithm consistency at the smaller throughput.



(a)



Flow Status	Successful - Fri Jan 27 03:59:29 2017
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	onesize
Top-level Entity Name	onewindow
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	48,360 / 114,480 (42 %)
Total registers	4363
Total pins	9 / 529 (2 %)
Total virtual pins	0
Total memory bits	141,872 / 3,981,312 (4 %)
Embedded Multiplier 9-bit elements	84 / 532 (16 %)
Total PLLs	0 / 4 (0 %)

(b)

Figure 17. Performance implementation of the proposed system: (a) experimental setup using Altera DE2-115 FPGA and an external monitor and (b) screenshot of synthesis summaries

As shown in Figure 15, it is regarded as a first-in-first-out (FIFO) function, a static random-access memory (SRAM) t_0 can be used to replace it. Therefore, gate count and power can be saved. The FIFO always toggles to consume power, but SRAM only activated one cell to access data. Moreover, ping-pong mode can enable a SRAM to perform "read" and "write" concurrently. The detection success of the proposed algorithm will be compared with the software implementation of the original HOG algorithm in the near future work. This becomes an open challenge that should be exploited.

3.1.2. Performance comparison

Table 7 and Table 8 evaluate the performance result with other competitors. The strongest points compare to the others are frame per second (Fps) and Fps-to-clock ratio. All the competitors exploit standard FPGA boards. Compared to the earlier works, our implementation performs significantly well in terms of the ratio of delivered Fps and operating clock frequency. This is made possible with our pipeline architectures and custom-designed hardware modules, instead of using general-purpose processors. Processors may be smaller in size, but dedicated modules are more efficient compared to processors. Moreover, all cells and blocks will not be kept in the RAM concurrently. Unused data will be overwritten to minimize memory usage. Additionally, the low operating clock frequency translates to less power consumption.

Working at a low frequency surely allows power consumption to be reduced. For instance, in Table 7 [23] has a much higher image resolution (1920×1080), higher operational clock (270 MHz), and a lower frame rate (64 Fps). Instead, the proposed work has a lower image resolution (640× 480), ten times lower operational clock (27 MHz), and a high frame rate (86.51 Fps). This becomes the architecture design trade-off. In [39] has the highest Fps (162 MHz) but the operating clock is the highest (150 MHz) resulting in lower Fps-to-clock ratio. Instead, the proposed work the highest Fps-to-clock ratio (3.2041), lower frame rate, lower operating clock (27 MHz), more efficient in memory usage (141,872 bits), and the lowest registers (4,363). With the same image resolution usage (640×480), in [40] has the lowest embedded multipliers (40 DSP block) as well as the operating frequency (25 MHz), which is not to close with our proposed architecture. But our architecture has a higher Fps and Fps-to-clock ratio, also resource-efficient. In summary, it is safe to say that our architecture is considered the best trade off comprared to previous works, in terms of operating clock frequency (MHz), Fps (Hz), Fps-to-clock ratio, and the use of Registers.

Table 7. Performances comparison against other works used Xilinx Virtex FPGA

Parameters	[20]	[21]	[23]	[41]	This work
FPGA board	Virtex-5	Virtex-5	Virtex-5	Virtex-6	Cyclone IV
LUTs**	28,495	17,383	5,188 (↑)	113,359 (↓)	48,360
Registers	5980	2,181 (↑)	5,176	75,071(↓)	4,363
Embedded multipliers / DSP block***	2 (↑)	N/A*	49	72	84 (↓)
Memory usage (bits)	2,196,000	1,327,000	1,188,000	4,284,000 (↓)	141,872 (↑)
Operating clock frequency (MHz)	167	44	270 (↓)	25 (↑)	27
Frame per second (Hz)	38 (↓)	62	64	60	86.51 (↑)
Fps-to-clock ratio	0,2275 (↓)	1.4091	0.237	2,4	3.2041 (↑)
Image resolution	320×240 (↓)		1920×1080 (↑)	640×480	640×480

* N/A: not available.
(↑): The highest value in the comparison table among the equivalent parameters compared
(↓): The lowest value in the comparison table among the equivalent parameters compared
** LUTs in FPGA logic building blocks may not serve as an accurate parameter for design size comparison. LUTs in Cyclone devices have 4 inputs, while Xilinx Virtex-5 has 6 inputs. Moreover, logic building blocks differ among devices, as in Altera's logic element (LE) and Xilinx's logic cell (LC), each with their respective hardware design. These factors may cause different synthesis results in Altera and Xilinx FPGAs.
*** Embedded multiplier refers to 9×9 multipliers in altera cyclone III and IV. DSP block refers to DSP48E slice in Xilinx Virtex-5, which is equipped with a 25×18 multipliers.

Table 8. Performances comparison against other works used ALTERA Cyclone FPGA

Parameters	[39]	[42]	[43]	[40]	[44]	[45]	This work
FPGA board	Cyclone IV	Cyclone IV	Cyclone IV	Cyclone III	Cyclone III	Cyclone V	Cyclone IV
LUTs**	16,060	83,497 (↓)	34,403	17,419	14,895	11,156 (↑)	48,360
Registers	7,220	17,383 (↓)	23,247	11,306	9800	13,191	4,363 (↑)
Embedded multipliers / DSP block***	69	90 (↓)	68	N/A*	40 (↑)	N/A*	84
Memory usage (bits)	334,000	2,800,000 (↓)	348,000	1,046,647	280,000	2,137 (↑)	141,872
Operating clock frequency (MHz)	150 (↓)	50	40	70	25 (↑)	76	27
Frame per second (Hz)	162 (↑)	129	72	20	48	8 (↓)	86.51
Fps-to-clock ratio	1,08	2,58	1.8	0.2857	1.92	0.1053 (↓)	3.2041 (↑)
Image resolution	800×600	1280×1024 (↑)	800×600		640×480 (↓)		

(↑): The highest value in the comparison table among the equivalent parameters compared
(↓): The lowest value in the comparison table among the equivalent parameters compared

4. CONCLUSION AND FUTURE WORKS

This paper presents a hardware architecture design to implement a simplified HOG algorithm. We have designed a cell-based raster scanning computation instead of window-based to reduce computation redundancy. The magnitude calculation using a linear approach provides us with a reasonable approximation of magnitude without using exponentiation and square root operations; due to *L2-norm* approach is too difficult to be implemented in hardware implementation. By using fixed-weighted binning for histogram classification, we can avoid using arctangent and division operations. Furthermore, by using the newton-raphson algorithm, we can execute block normalization without using any division operations. Finally, the overall parallel and pipeline architecture gives accurate detection with less memory usage and maximum Fps-to-clock frequency ratio. This work used MIT pedestrian dataset for training. The primary feature of this work is to simplify HOG for an efficient hardware implementation. This simplification certainly makes some degradation on the performance of original HOG. It is important to examine in detail the performance of original HOG comprated to this work (simplified HOG) further. We will also address several interesting issues, *e.g.*, the impact of computational reduction in term of detection accuracy, measure the throughput achieved of the GPU implementations, and a more objective figure-of-metric (FOM). This will be considered to prove that the proposed hardware architecture is more efficient than the other competitors. Later, the effect of accuracy improvement to computational costs and system complexity will be evaluated further. In the recent years, various other challenging datasets have been introduced by many researchers. Therefore, we will use various dataset provided globally and dataset produced my ourselves to train and evaluate our proposed system comprehensively.

AKNOWLEDGMENTS

The publication fee for this work was sponsored by Integrated Circuit Design Laboratory Pusat Unggulan IPTEK Perguruan Tinggi (PUI-PT) Mikroelektronika ITB and Program Peningkatan *Global Competitiveness* Perguruan Tinggi Indonesia Universitas Pendidikan Indonesia 2021 Batch II with No SK 1370/UN40/PT.01.02/2021.

REFERENCES

[1] T. A. Adiono, K. Shidqi, C. Deo, B. Yuwono, and S. Fuada, "HOG-adaboost implementation for human detection employing FPGA ALTERA DE2-115," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 10, pp. 353–358, 2018, doi: 10.14569/IJACSA.2018.091042.

[2] C. Zaharia, F. Sandu, and A. Balan, "Usage of asymetric small binning to compute histogram of oriented gradients for edge computing image sensors," in *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, Dec. 2020, pp. 1–6. doi: 10.1109/RoEduNet51892.2020.9324883.

[3] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005, vol. 1, pp. 886–893. doi: 10.1109/CVPR.2005.177.

[4] R. Kadota, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto, and Y. Nakamura, "Hardware architecture for HOG feature extraction," in *2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Sep. 2009, pp. 1330–1333. doi: 10.1109/IIH-MSP.2009.216.

[5] S. Ghaffari, P. Soleimani, K. F. Li, and D. W. Capson, "Analysis and comparison of FPGA-based histogram of oriented gradients implementations," *IEEE Access*, vol. 8, pp. 79920–79934, 2020, doi: 10.1109/ACCESS.2020.2989267.

[6] S. Ghaffari, P. Soleimani, K. F. Li, and D. Capson, "FPGA-based implementation of HOG algorithm: techniques and challenges," in *2019 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, Aug. 2019, pp. 1–7. doi: 10.1109/PACRIM47961.2019.8985056.

[7] A. Saidi, S. Ben Othman, M. Dhouibi, and S. Ben Saoud, "FPGA-based implementation of classification techniques: A survey," *Integration*, vol. 81, pp. 280–299, Nov. 2021, doi: 10.1016/j.vlsi.2021.08.004.

[8] K. V. V. Kumar and P. V. V. Kishore, "Indian classical dance mudra classification using HOG features and SVM classifier," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 5, p. 2537, Oct. 2017, doi: 10.11591/ijece.v7i5.pp2537-2546.

[9] M. Anandhalli, V. P. Baligar, P. Baligar, P. Deepsir, and M. Iti, "Vehicle detection and tracking for traffic management," *IAES International Journal of Artificial Intelligence (IJ-AI)*, vol. 10, no. 1, p. 66, Mar. 2021, doi: 10.11591/ijai.v10.i1.pp66-73.

[10] C. Wattanapanich, H. Wei, and W. Petchkit, "Investigation of robust gait recognition for different appearances and camera view angles," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 5, p. 3977, Oct. 2021, doi: 10.11591/ijece.v11i5.pp3977-3987.

[11] Qiang Zhu, Mei-Chen Yeh, Kwang-Ting Cheng, and S. Avidan, "Fast human detection using a cascade of histograms of oriented gradients," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, vol. 2, pp. 1491–1498. doi: 10.1109/CVPR.2006.119.

[12] H.-X. Jia and Y.-J. Zhang, "Fast human detection by boosting histograms of oriented gradients," in *Fourth International Conference on Image and Graphics (ICIG 2007)*, Aug. 2007, pp. 683–688. doi: 10.1109/ICIG.2007.53.

[13] W. Zhang, G. Zelinsky, and D. Samaras, "Real-time accurate object detection using multiple resolutions," in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8. doi: 10.1109/ICCV.2007.4409057.

[14] X. Wang, T. X. Han, and S. Yan, "An HOG-LBP human detector with partial occlusion handling," in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 32–39. doi: 10.1109/ICCV.2009.5459207.

[15] W. R. Schwartz, A. Kembhavi, D. Harwood, and L. S. Davis, "Human detection using partial least squares analysis," in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 24–31. doi: 10.1109/ICCV.2009.5459205.

[16] Kelvin Lee, Che Yon Choo, Hui Qing See, Zhuan Jiang Tan, and Yunli Lee, "Human detection using Histogram of oriented gradients and human body ratio estimation," in *2010 3rd International Conference on Computer Science and Information Technology*, Jul. 2010, pp. 18–22. doi: 10.1109/ICCSIT.2010.5564984.

[17] S. S. Selvi, B. D, A. Qadir, and P. K. R, "FPGA implementation of a face recognition system," in *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, Jul. 2021, pp. 1–5. doi: 10.1109/CONECCT52877.2021.9622348.

[18] T. P. Cao and G. Deng, "Real-time vision-based stop sign detection system on FPGA," in *2008 Digital Image Computing: Techniques and Applications*, 2008, pp. 465–471. doi: 10.1109/DICTA.2008.37.

[19] Tam Phuong Cao, Guang Deng, and D. Mulligan, "Implementation of real-time pedestrian detection on FPGA," in *2008 23rd International Conference Image and Vision Computing New Zealand*, Nov. 2008, pp. 1–6. doi: 10.1109/IVCNZ.2008.4762094.

[20] M. Hiromoto and R. Miyamoto, "Hardware architecture for high-accuracy real-time pedestrian detection with CoHOG features," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, Sep. 2009, pp. 894–899. doi: 10.1109/ICCVW.2009.5457609.

[21] K. Negi, K. Dohi, Y. Shibata, and K. Oguri, "Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm," in *2011 International Conference on Field-Programmable Technology*, Dec. 2011, pp. 1–8. doi: 10.1109/FPT.2011.6132679.

[22] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "Architectural study of HOG Feature extraction processor for real-time object detection," in *2012 IEEE Workshop on Signal Processing Systems*, Oct. 2012, pp. 197–202. doi: 10.1109/SiPS.2012.57.

[23] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, "FPGA-based real-time pedestrian detection on high-resolution images," in *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Jun. 2013, pp. 629–635. doi: 10.1109/CVPRW.2013.95.

[24] M.-S. Wang and Z.-R. Zhang, "FPGA implementation of HOG based multi-scale pedestrian detection," in *2018 IEEE International Conference on Applied System Invention (ICASI)*, Apr. 2018, pp. 1099–1102. doi: 10.1109/ICASI.2018.8394472.

[25] Takashi Saegusa, Tsutomu Maruyama, and Yoshiki Yamaguchi, "How fast is an FPGA in image processing?," in *2008 International Conference on Field Programmable Logic and Applications*, 2008, pp. 77–82. doi: 10.1109/FPL.2008.4629911.

[26] P. Soma, C. Sravanthi, P. Srilakshmi, and R. K. Jatoth, “Implementation of Single Image Histogram Equalization and Contrast Enhancement on Zynq FPGA,” in *in Advances in Communications, Signal Processing, and VLSI*, 2021, pp. 75–82. doi: 10.1007/978-981-33-4058-9_7.

[27] P. Dai, J. Tang, J. Yuan, and Y. Yu, “A hardware-efficient HOG-SVM algorithm and its FPGA implementation,” in *2021 2nd International Symposium on Computer Engineering and Intelligent Communications (ISCEIC)*, Aug. 2021, pp. 145–150. doi: 10.1109/ISCEIC53685.2021.00037.

[28] F. An, P. Xu, Z. Xiao, and C. Wang, “FPGA-based object detection processor with HOG feature and SVM classifier,” in *2019 32nd IEEE International System-on-Chip Conference (SOCC)*, Sep. 2019, pp. 187–190. doi: 10.1109/SOCC46988.2019.1570558044.

[29] F. An, X. Zhang, A. Luo, L. Chen, and H. J. Mattausch, “A hardware architecture for cell-based feature-extraction and classification using dual-feature space,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 10, pp. 3086–3098, Oct. 2018, doi: 10.1109/TCSVT.2017.2726564.

[30] Seonyoung Lee, Haengseon Son, Jong-Chan Choi, and Kyungwon Min, “High-performance HOG feature extractor circuit for driver assistance system,” in *2013 IEEE International Conference on Consumer Electronics (ICCE)*, Jan. 2013, pp. 338–339. doi: 10.1109/ICCE.2013.6486918.

[31] S. Lee, H. Son, J. C. Choi, and K. Min, “HOG feature extractor circuit for real-time human and vehicle detection,” in *TENCON 2012 IEEE Region 10 Conference*, Nov. 2012, pp. 1–5. doi: 10.1109/TENCON.2012.6412287.

[32] N. Sudha, “A pipelined array architecture for Euclidean distance transformation and its FPGA implementation,” *Microprocessors and Microsystems*, vol. 29, no. 8–9, pp. 405–410, Nov. 2005, doi: 10.1016/j.micpro.2004.10.003.

[33] S. Bauer, S. Kohler, K. Doll, and U. Brunsmann, “FPGA-GPU architecture for kernel SVM pedestrian detection,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, Jun. 2010, pp. 61–68. doi: 10.1109/CVPRW.2010.5543772.

[34] M. Bilal, A. Khan, M. U. K. Khan, and C.-M. Kyung, “A low-complexity pedestrian detection framework for smart video surveillance systems,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 10, pp. 2260–2273, Oct. 2017, doi: 10.1109/TCSVT.2016.2581660.

[35] A. Suleiman and V. Sze, “An energy-efficient hardware implementation of HOG-based object detection at 1080HD 60 fps with multi-scale support,” *Journal of Signal Processing Systems*, vol. 84, no. 3, pp. 325–337, Sep. 2016, doi: 10.1007/s11265-015-1080-7.

[36] M. D. Malkauthekar, “Analysis of euclidean distance and manhattan distance measure in face recognition,” in *Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)*, 2013, pp. 503–507. doi: 10.1049/cp.2013.2636.

[37] L. Greche, M. Jazouli, N. Es-Sbai, A. Majda, and A. Zarghili, “Comparison between Euclidean and Manhattan distance measure for facial expressions classification,” in *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, Apr. 2017, pp. 1–4. doi: 10.1109/WITS.2017.7934618.

[38] G. Khosla, N. Rajpal, and J. Singh, “Evaluation of euclidean and manhanttan metrics in content based image retrieval system,” in *in 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2015, pp. 12–18.

[39] J. Luo and C. Lin, “Pure FPGA implementation of an HOG based real-time pedestrian detection system,” *Sensors*, vol. 18, no. 4, p. 1174, Apr. 2018, doi: 10.3390/s18041174.

[40] Y. YAZAWA *et al.*, “FPGA hardware with target-reconfigurable object detector,” *IEICE Transactions on Information and Systems*, vol. E98.D, no. 9, pp. 1637–1645, 2015, doi: 10.1587/transinf.2014OPP0008.

[41] M. Komorkiewicz, M. Kluczewski, and M. Gorgon, “Floating point HOG implementation for real-time multiple object detection,” in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2012, pp. 711–714. doi: 10.1109/FPL.2012.6339159.

[42] T. Adiono, K. S. Prakoso, C. Deo Putratama, B. Yuwono, and S. Fuada, “Practical implementation of a real-time human detection with HOG-adaboost in FPGA,” in *TENCON 2018 - 2018 IEEE Region 10 Conference*, Oct. 2018, pp. 0211–0214. doi: 10.1109/TENCON.2018.8650453.

[43] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, “An FPGA implementation of a HOG-based object detection processor,” *IPSJ Transactions on System LSI Design Methodology*, vol. 6, pp. 42–51, 2013, doi: 10.2197/ipsjtsldm.6.42.

[44] A.-Y. Guo, M.-H. Xu, F. Ran, and A. Li, “FPGA implementation of a real-time pedestrian detection processor aided by E-HOG IP,” *Journal of Computers*, vol. 28, no. 2, pp. 87–103, 2017.

[45] V. Ngo, A. Casadevall, M. Codina, D. Castells-Rufas, and J. Carrabina, “A pipeline hog feature extraction for real-time pedestrian detection on FPGA,” in *2017 IEEE East-West Design & Test Symposium (EWDTS)*, Sep. 2017, pp. 1–6. doi: 10.1109/EWDTS.2017.8110057.

BIOGRAPHIES OF AUTHORS

Syifaul Fuada is with the the Study Program of Telecommunications, Universitas Pendidikan Indonesia as a young lecturer. Mr. Fuada has several has several achievements, such as the most student outstanding award of Universitas Negeri Malang in 2013, a top of 10–student travel grant to the IEEE Asia Pacific Conference and Systems (APCCAS 2016) that was held in Jeju (South Korea), receiving one of 108 Indonesia Innovations by BIC-LIPI awards (2016) for Smart Home Product, student winner nominee of NOLTA conference (2017), receiving Best Paper Award from IEEE IGBSG 2018 that was held in Yi-Lan, Taiwan, receiving Best Paper Award from IEEE ICTRuDev 2018 that was held in Bali, Indonesia, receiving the Best paper award from a Scopus-indexed journal, i.e., i-JOE in 2019, receiving Best Paper Award from IEEE IGBSG 2019 that was held in Yichang, China, receiving the 111 Indonesia Innovations by BIC awards (2019) for E-Nelayan and LI-FI products, receiving the 112 Indonesia Innovations by BIC awards (2020) for three innovations: Bidirectional DC/DC Converter for Electric ATV, Smart Home, and Contact/Contactless-based Payment Device, the 3rd place of UPI's most productive researcher in the SCOPUS-based publication 2020 (awarded on 2021), the 3rd place of UPI's inventor awarded on 2021. His study interests include analog circuit design and instrumentation, engineering education, IoT, image processing, System-on-Chip, and Visible Light Communication. He can be contacted at email: syifaulfuada@upi.edu.

Trio Adiono received a B.Eng. in electrical engineering and an M.Eng. in microelectronics from Institut Teknologi Bandung (ITB), Indonesia, in 1994 and 1996, respectively. He obtained his Ph.D. in VLSI Design from Tokyo Institute of Technology in 2002, Japan. In 2005, he was a visiting scholar at MESA+, Twente University, Netherlands. He is a full-professor at Sekolah Teknik Elektro dan Informatika ITB. He was also Adjunct Assoc. Prof. NTUST-Taiwan. He has developed several microchips, such as Binary Template Matching Processor, WiMax Baseband Chipset, Smart Card, and IoT. He also has job experience working with Chip Design House in Fukuoka Japan, handling chip design for several multinational companies. He received the "Second Japan Intellectual Property (IP) Award" in 2000 from Nikkei BP. He received Award Karya Lencana Wira Karya from President of Republic Indonesia for his innovation in 4G chip developments. He also holds a Japanese Patent on "High Quality Video Compression System". He was also chair of IEEE Solid State Circuits Society. His research interests include VLSI design, signal and image processing, VLC, smart cards, and electronics solution design and integration. He can be contacted at email: tadiono@stei.itb.ac.id.

Hans Kasan received his B.Eng. degree in Electrical Engineering from Bandung Institute of Technology (ITB), Indonesia in 2017. He joined Computer System and Network Lab (CSNL) in the Department of Electrical Engineering at Korea Advanced Institute of Science and Technology (KAIST) in September 2018 and now he is pursuing Ph.D. degree with the same department under supervision of Professor John Kim. He has several achievements, on of them is the first runnerup (Electronic Device Industry News award) of the 2017 LSI Design Contest in Okinawa. He has published papers about RFID-based warehouse management system and Montgomery multiplier IC design. His research interests are in RF, VLSI and custom digital IC designs. His current research topic is adaptive routing in interconnection networks. He can be contacted at email: hanskasan@live.com.