Received: 17 November 2023 | *Revised:* 13 February 2024 | *Accepted:* 23 February 2024
DOI: 10.37917/ijeee.21.2.12 *Vol. 21 | Issue 2 | December 2025*

Iraqi Journal for Electrical and Electronic Engineering
Original Article

Open Access
IJEEE University of Basrah College of Engineering

FPGA-Based Implementation of a Basic Background Subtraction Algorithm for Real-Time Application

Marwan Abdulkhaleq Al-Yoonus*[1], Dr. Saad Ahmed Al-Kazzaz[2]
[1]Electrical Engineering Department, College of Engineering, University of Mosul, Iraq
[2]Mechatronics Engineering Department, College of Engineering, University of Mosul, Iraq

Correspondance
*Marwan Abdulkhaleq Al-Yoonus
Electrical Engineering Department1, College of Engineering,
University of Mosul, Iraq
Email: marwanathy1972@uomosul.edu.iq

Abstract
An intelligent video system's basic function is the detection of moving objects. Moreover, real-time systems frequently pose limitations for applications involving video processing. Practically, to increase the frame rate or in the case of limited hardware resources, the real-time processing is done on an interested image segment called the region of interest (ROI). Applying the background subtraction (BGS) algorithm to this region is considered the main preprocessing operation. This paper presents a practical study for video processing based on FPGA to detect moving objects using the BGS technique. The proposed algorithm was developed using Verilog hardware description language (HDL), synthesized, and implemented in the programmable logic (PL) part of the ZYBO-7Z010CLG400-1 platform. Finite State Machine (FSM) controller method was used to design the Intellectual Property (IP) module that controls data transfer with BRAM (loading and reading) of the input and reference image. The simulation results of the timing signal sequences of the proposed IP module with the practical test of the BGS to detect several traffic signs of image size (90×90) pixels demonstrate that the module functions as intended. The system that is being presented has a latency of 13.468 nanoseconds, making it appropriate for real-time applications.

Keywords
Background Subtraction, BRAM, Finite State Machine, Real-time, Timing Signals, Verilog HDL.

I. Introduction

The process of determining an object's physical movement within a specific area or region is known as motion detection. Using the presumption that the current frame differs from the background image only in that it contains moving objects, background subtraction looks for this motion (foreground). Consequently, all that is needed to obtain the movement regions is to threshold the difference (subtraction) between the background and the current frame [1].

An appropriate method for investigating the intrinsically parallel nature of many video processing algorithms is to use an FPGA-based approach [2]. One reason for this is that it is becoming increasingly challenging to ship designs error-free due to growing system complexity. Therefore, the ability to correct errors after fabrication has become crucial, and devices that are customizable, like FPGAs, greatly simplify this process [3]. FPGA designs offer greater reconfigurability to accommodate any functionality when compared to ASIC designs. FPGAs have a high degree of customizability and are a very suitable technology for high-speed parallel data processing due to their parallel processing capability [4–6].

The BGS method is an essential step in many computer applications. This algorithm is frequently used to detect moving objects. The fundamental idea behind BGS is that it is a method that creates a background model and then compares it with the current frame to identify areas where a notable difference is present [7–9]. BGS is designed to identify only the unusual events at the vision sensor level by removing all

circumstances in which nothing abnormal is occurring in the scene [10]. In order to achieve better timing performance, the design must include finite state machine (FSM) controllers, which is one of the key components in achieving the intended performance of the overall design. Arbitrary counters and sequence detectors are also implemented using FSMs, in addition to the controllers [11, 12].

Though many research papers are published in journals and conference proceedings, only a small number of them provide the design processes and timing illustration of video signals for real-time applications. This paper presents a hardware design and implementation of a BGS module for fixed camera and static object detection. The present work seeks to describe some basic principles that can add to the research field a proposed hardware design of such an algorithm in its basic operation. The application that has been utilized in this work to confirm the suggested system's functionality in a perfect setting is traffic signs detection.

The rest of the paper is organized as follows: Section II provides an overview of several related research papers. In Section III a brief description is presented of the basics of the BGS algorithm. The hardware implementation steps for the proposed system are found in Section IV. Real-time explanation is discussed in Section V. The experimental results are presented in Section VI. Finally, the conclusion of this work is given in Section VII.

II. Literature Survey

The development of imaging and computer vision technologies is being significantly influenced by the rising popularity of smart video systems [13]. The literature is replete with studies that highlight the benefits of using FPGAs for embedded vision systems. A summary of some of the current research in this area is given in this section.

An alternative version of the BGS method based on independent component analysis (ICA) was proposed by F. Carrizosa et al. [1]. Four image sequences were examined for motion detection. The results from the implementation that used both the FPGA and the embedded processor of the SoC showed a decrease in runtime from 4326.60 to 125.81 milliseconds compared with others that used only the embedded processor. The background estimation and picture capturing were not taken into account during the measurements in the mentioned paper.

Using color invariant and grayscale information, G. Cocorullo et al. [7] proposed the MBSCIG algorithm, a multimodal BGS. Four historical frames, a limited number, were used to compute the background model in the proposed approach. Complex operations were avoided to minimize computational time and the need for logic and memory resources.

An offline digital image processing fundamentals presented in [14] by Dharmavaram et al on an image file of size (768×512) pixels. A processing system (PS) was used for prototype verification. Reconfigurable logic devices (FPGAs) and video separation were covered by I. N. Rodrigues et al. [15] with no video timing signals illustration. The object detection method employed in the first stage was evaluated using three different threshold types.

The fundamentals of moving object detection, BGS, and FPGA were covered by Xin Ren et al. [16]. The work was notable for using non-floating point arithmetic to simplify hardware implementation, lower resource consumption, and pipeline processing to increase system throughput. Although the authors described the process steps in detail, there is no information about resource utilization.

Giuseppe Conti et al. [17] described the testing of two hardware implementations that use RGB and grayscale color spaces, respectively, and an analysis of the system reliability. The developed system had the capability to process data in real-time, ranging from (192×192) base resolution to (640×480), from a commercial Zenith camera at 15 frames per second.

The survey of pertinent past and present works that came before it makes it clear that there is a gap in the literature when it comes to the explanation of video timing signals for hardware design for real-time systems. In our work, we used a video signal from an HDMI laptop port with (1280×720) size which is different from [14] which used an image file of (768×512) pixels. The timing illustration of the video signal is also included in our work which was not mentioned in [7] and [15]. In [17] no hardware design was illustrated for the proposed system or timing description. Therefore, the objective of this work is to present the hardware implementation of a basic BGS algorithm using Verilog HDL, along with the design of an IP module that uses FSM to control the input frame's load and read processes. In order to examine the system performance during real-time operation, we attempt to present the video timing verification and analysis of the suggested hardware for basic video/image processing using a low-cost FPGA-SoC platform.

III. Background Subtraction Technique

The basic models include creating a background model by taking few initial frames into consideration without any moving object. For each video sequence, an absolute difference is computed between the reference template and the current template known as frame difference [18]:

$$\mathrm{BM}_N(x,y) = \frac{\sum_{m=1}^{N} I_m(x,y)}{N} \tag{1}$$

where, BM_N (x,y) represents the pixel's (x,y) intensity of the background model, I_m (x,y) represents the pixel's (x,y) intensity of the m^{th} frame and N refers to the total number of frames used to build background model. BGS can also be modeled using mean, median, average, or histogram analysis over time. However, these models can handle only some specific challenges and are best suited for static backgrounds. The following common equation describes the operation of the BGS algorithm [2, 16, 19]:

$$output(BGS) = \begin{cases} 1 & \text{if } |\text{ I(i,j,t)-Ref(i,j) }| > TH \\ 0 & \text{if } |\text{ I(i,j,t)-Ref(i,j) }| < TH \end{cases} \quad (2)$$

where I(i,j,t) and Ref(i,j) refer to current input and reference image pixels, respectively, the predetermined threshold (TH) can be estimated by training the system or via observation. In this work, a single reference frame was used, (90×90) pixels, that can be updated manually during the operation to implement the BGS algorithm.

IV. Implementation Using Verilog HDL

The Verilog hardware description language is a formal notation that can be used at any stage of the development of electronic systems (HDL). Its machine- and human-readable nature makes it easier to develop, verify, synthesize, and test hardware designs. Because Verilog has so many features, most integrated circuit (IC) designers have chosen it as their language of choice [20, 21].

To load the background image from a video frame sequence in real-time, the BRAM IP was used to save the background image in gray-scale format (8-bit) in order to be the reference image for the BGS algorithm. This background image can be manually updated during the operation. Each memory location in BRAM holds one byte (i.e. width =1 Byte), and the BRAM depth is computed from the selected ROI which is (90×90) as shown in Fig. 1 and it is equal to (90×90=8100) locations.

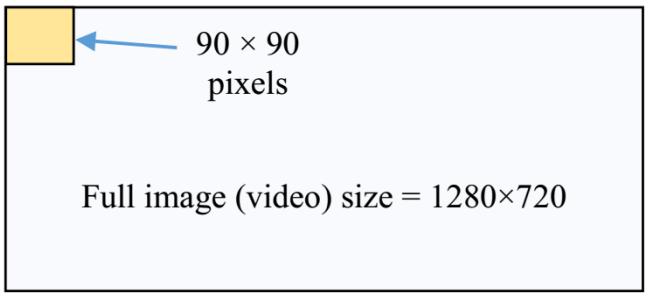


Fig. 1. The full input image/video size and the ROI

Fig. 2 shows the start and last address of each row cell (Row1, Row2, ..., Row89, and Row90) which contains the data of each line of the ROI which is (90 bytes/row). FSM methods have been used to design the IP module that controls loading (orang path) and then reading of the ROI (green path) as shown in Fig.3. As shown, the FSM consists of seven states. The implementation of the FSM was done using Verilog HDL, as will be declared later in Algorithm 1. The seven states that control the operations are; Wait-sw0, W1-R0-Vsync, Write-to-BRAM, Wait-line-end, Check-W, Read-BRAM, and Check-R. The blue square is for user interface to update the BG image.

Rows of the ROI (90×90) pixels	**Address range (A_0-A_{12})Hex.**	**BRAM (memory)**
Row1 start add.	00	Row 1 (90 Byte)
Row1 last add.	59	
Row2 start add.	5A	Row 2 (90 Byte)
Row2 last add.	B3	
-	-	-
-	-	-
-	-	-
-	-	-
Row 89 start add.	1EF0	Row 89 (90 Byte)
Row 89 last add.	1F49	
Row 90 start add.	1F4A	Row 90 (90 Byte)
Row 90 last add.	1FA3	

Fig. 2. Memory map arrangement for loading and reading the ROI of (90×90) pixels

The timing diagram shown in Fig. 4 explains the multi-domain clock signal that has been depended on to implement the proposed subsystem to control the sequence of its real-time operation. These clock signals can be generated from the clocking wizard IP module, the brown triangle on the negative edge of the clk-D clock is used to provide the BRAM address through the address counter and the orang triangle is for input pixel sampling. The green one is the instance of RGB to grayscale converter. The black triangle at the positive edge of clk-D is for reading the BG pixel and the red triangle is for subtraction operation. Finally, the blue triangle indicates the instance of data width back converting from 8-bit to 24-bit. By using multi clock domain the latency is equal to one pixel clock period (13.468nsec).

V. Real Time Module Explanation

For real time operation, an experimental test has been done to measure the pixel clock frequency practically and the frame resolution. The video signals that have been obtained from a laptop HDMI port are shown in Fig. 5. From the video timing signals (Hsync and Vsync) the frame resolution can be calculated practically by using the two relationships:

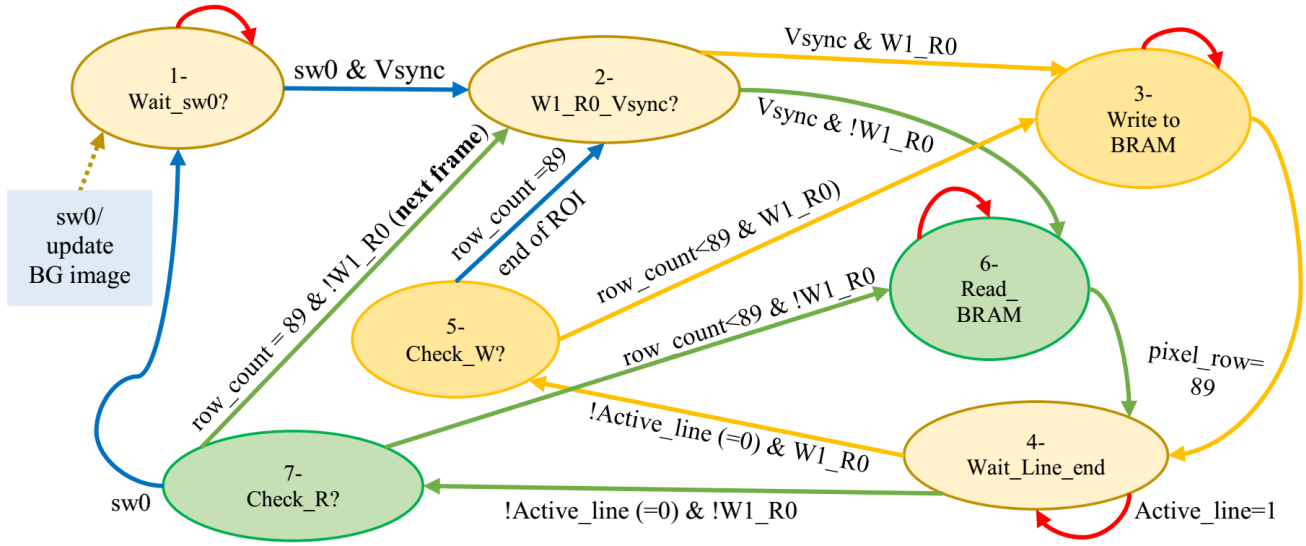


Fig. 3. FSM state diagram to control the reading of ROI, (!) refers to Not logic gate

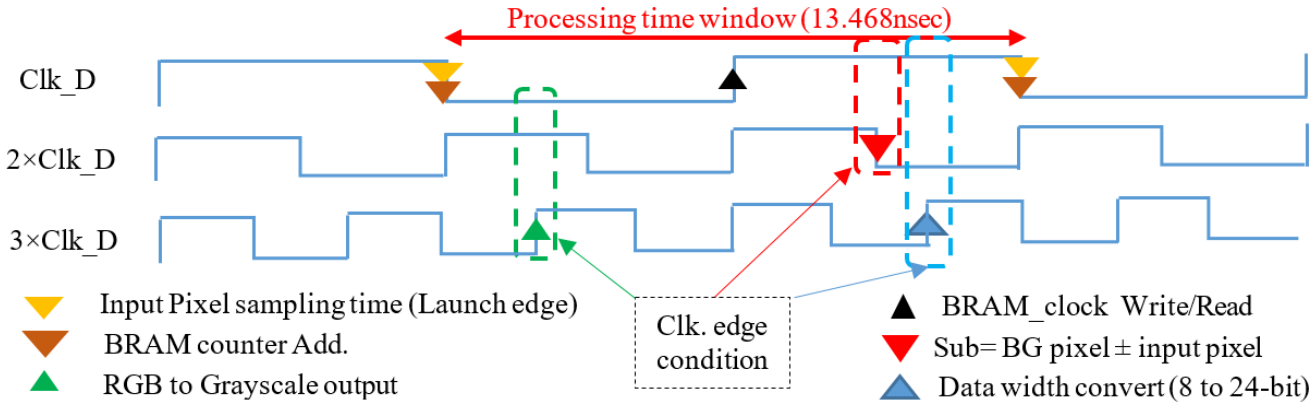


Fig. 4. Timing diagram analysis for basic video processing with load/read instants to/from BRAM

$$\text{line per frame} = \frac{16msec}{22.2usec} \cong 720 \quad lines \tag{3}$$

$$\text{pixels per line} = \frac{17.2usec}{\frac{1}{74.25MHz}} \cong 1280 \quad columns \tag{4}$$

From the results, the signals shown are for a video signal of (1280×720) resolution, and the frame rate is 60 frames per second (fps). So the pixel clock frequency is (74.25MHz). The real-time operation of the proposed BGS algorithm depends on these results. The whole digital system works in multi-clock domain frequencies, 222.75MHz, 148.5MHz, and 74.25MHz, which are three, two, and one-time pixel clock frequency respectively. The latency of the proposed ROI reading subsystem was 13.468ns (one-pixel period), which matched the requirements for real-time applications for similar system specifications.

The complete circuit shown in Fig. 6 of the proposed BGS algorithm with the proposed basic video processing IP modules, was synthesized and implemented using Vivado IP cores. Also, the two IP modules (green blocks) that control the loading/reading of image pixels are designed using Verilog HDL. The binary counter task is to provide the BRAM address during load and read cycles.

The timing diagram details of the FSM (see Fig. 3) are shown in Fig. 7. The operation sequence of the BGS algorithm is as follows; the first signal is the clock (clk=74.25MHz). The Vsync signal represents the vertical synchronous video signal.

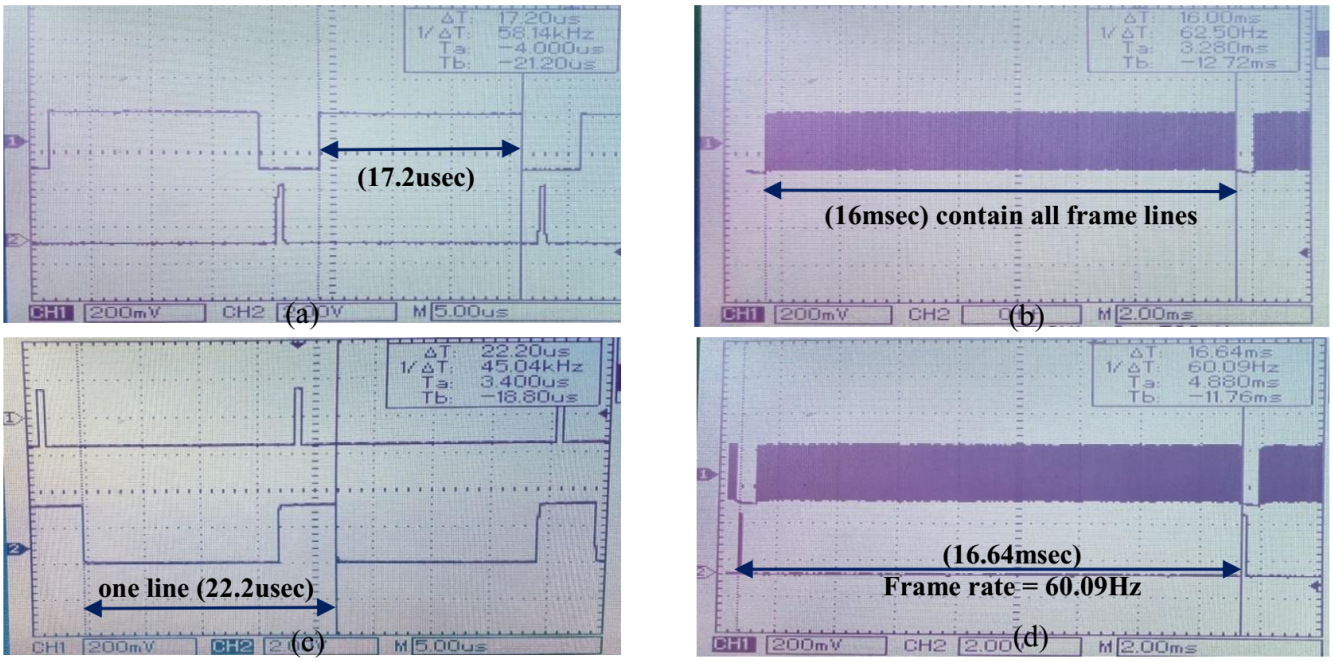


Fig. 5. (a) upper; Active line period and lower; Hsync (b) One active frame period (c) upper; Hsync and lower; tlast (Designates the last valid pixel of each line, and is also known as the end of line (d) upper; one frame (60.09Hz) and lower; Vsync

The active-line signal is an indication that the video data is valid. The fourth signal is the switch signal (sw0), which can be changed manually or by an external condition, assert the operation (yellow circle) so that W1-R0 becomes logic "1" and the operation started at the condition of Vsync and sw0=0, red circle, to begin to save (load) the real-time input pixels with the size of 90×90 pixels (see Fig. 1), into BRAM to be the background image.

The operation of loading/reading to BRAM is enabled by the five signals, EN-ROI, during the period (white circle) on the selected active lines. The BRAM-add signal is the address of BRAM [12:0]. The end of saving the whole pixels (90×90=8100) of the ROI is controlled by the row-count signal. The W1-R0 signal will change to logic "0" in order to initiate the reading process from BRAM as indicated by the blue circle. The read operation started at the next Vsync (red square sign).

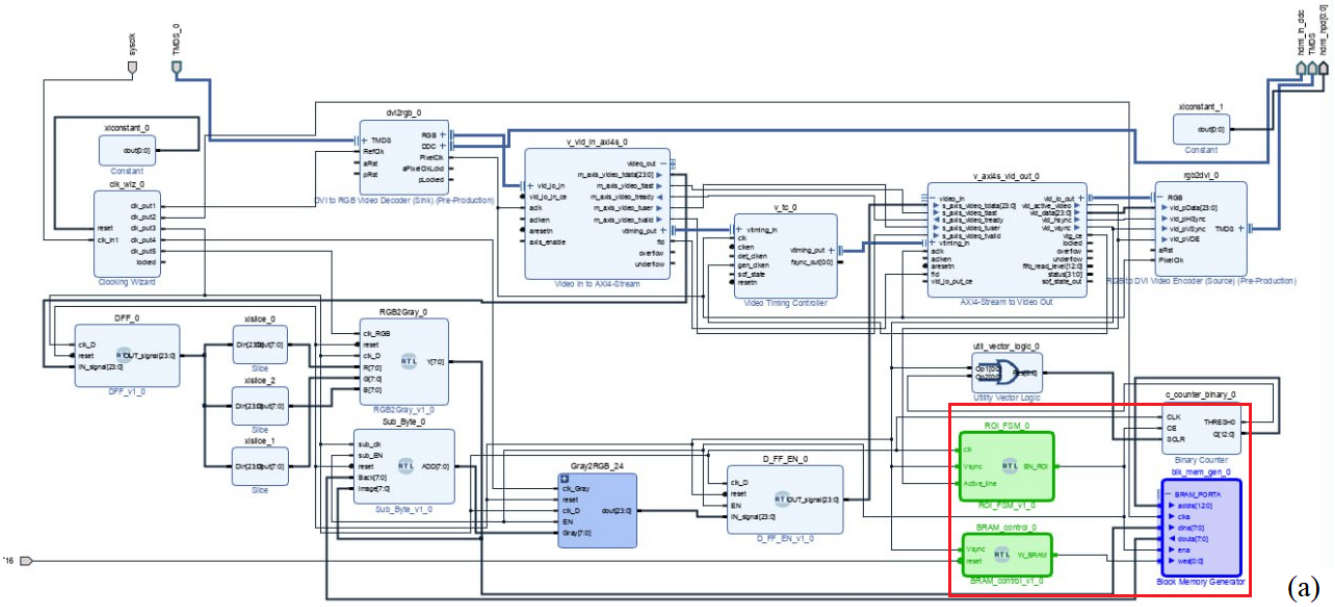
The row-count signal controls the end of saving the whole pixels (90×90=8100) of the ROI. The W1-R0 signal will change to logic "0" in order to initiate the reading process from BRAM as indicated by the blue circle. The read operation started at the next Vsync (red square sign).

VI. EXPERIMENTAL RESULTS

To test the operation, a video source has been applied (60fps) from the Laptop HDMI port to the HDMI input of the ZYBO z7-10. Traffic sign images with the size of (90×90) pixels are chosen to verify the operation in real-time conditions of the proposed BGS. Each subplot shown in Fig.8 consists of two sections; the source video, on the right, and the output video, on the left. The sequence (see Fig. 4) of the real-time operations that are done on each input pixel is:

1- Convert the input pixel from RGB (24-bit) to Grayscale (8-bit).
2- Read the first BG pixel from BRAM.
3- Subtract the BG pixel from the input pixel.
4- Compare the result in step 3 with a predetermined threshold (TH).
5- Back convert the result of step 4 from (8-bit) to (24-bit).

These operations are completed at a one-pixel clock period (13.468nsec) by using the multi-clock domain as mentioned before. Fig. 8a shows a traffic sign (1st input object) on the left and the output (BGS-1) on the right. In this case, there is no reference image in BRAM (BRAM is empty), so the input image is displayed as shown and the BG image was loaded manually in this step. In Fig. 8b, there is no input image and the BG image appeared (BGS-2) as the result of subtraction. Now the traffic sign that is loaded to BRAM appears as shown in Fig. 8c (on the left) and the pixel-by-pixel subtraction is (BGS-3) showing a high percentage of matching. A different traffic sign is displayed now as in Fig. 8d (New input object),



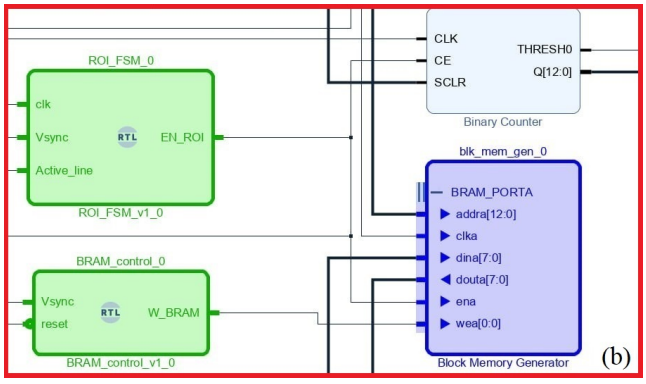


Fig. 6. (a) The complete module for the proposed BGS algorithm, (b) The designed IP cores (Green blocks) for the FSM (Fig. 3) with binary counter and BRAM (dark blue) IPs

the result of the BGS process (BGS-4) shows less amount of matching, and so on. Fig. 8g shows a new object and the result (BGS-4) that indicates a large percentage of mismatching as shown in Fig. 8h.

The resources used for implementing the recommended module on a ZYBO Z7-10 device are listed in Table I. It is clear from the resource utilization mentioned in the table, that flip-flops (FFs) are the most used among other FPGA resources. The other logic cell is the look-up table (LUT) which is configured during the synthesized operation to implement different logical functions. The suggested design has a total on-chip power of 0.589W.

The simulation results, which describe the FSM events sequence depicted in (Fig. 7), and the actual results of the BGS algorithm operation, which was displayed in Figure 8, demonstrate that the suggested BGS module functions as intended under the previously mentioned conditions to detect stationary objects using a fixed camera.

Table II gives an overview of the implementation that is discussed in this paper and lists relevant works that were available in the literature. Since these works focus on different implementation processes, the comparison is restricted to the platform, operating frequency, processed image size, HDL language, and processing method. So the proposed work operates at different clock frequency domains, uses only PL (FPGA), displays only the region of interest, and uses the cheapest FPGA platform compared with others.

TABLE I. Resource Utilization (ZYBO-7z010clg400-1)

FPGA Resources	Slice LUTs	Slice Reg.	F7 Mux	F8 Mux	BRAM
Available	17600	35200	8800	4400	60
used	1254	1883	16	8	8
Utility %	7.13	5.35	0.18	0.18	13.33

VII. Conclusion

This paper presents a hardware implementation of a basic BGS algorithm for video surveillance systems using Verilog HDL, with the goal of detecting static objects. The intended BGS module's ability to detect traffic signs with a window size equal (90×90) has been tested at an input image resolution (1280×720) from a laptop HDMI port. By using a multi-domain clock, specific operations have been carried out during

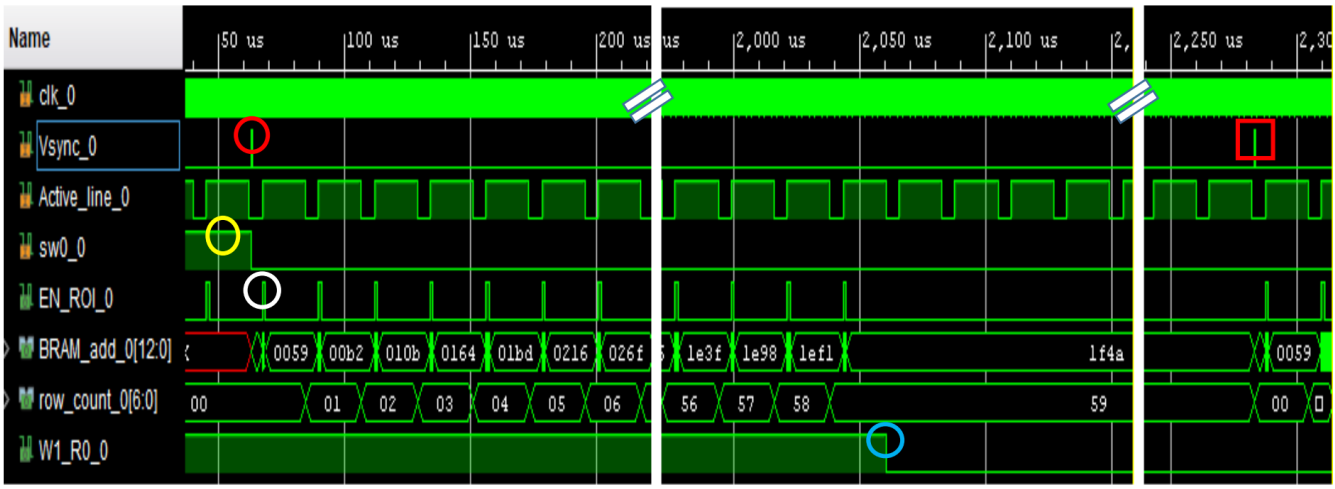


Fig. 7. Details of the timing signals of the FSM that control the loading and reading process of the BG algorithm

TABLE II. Summary of some existed work for the last decade and the presented work, (Pix; pixels)

Comparison factors	Existed work [7] 2016	Existed work [1] 2018	Existed work [17] 2020	Existed work [14] 2023	Proposed Work
Device (Board)	Zed board xc7z020 Zynq	ZedBoard	Zed board 7020 (CPU+FPGA)	Zed board	ZYBO-(7Z010 clg400-1)
Prog. Language	VHDL with IP cores	VHDL	C++, OpenCV 3.1 libraries	C and MATLAB	Verilog with IP cores
Simulation & synthesis tools	—	MATLAB, Python, and Vivado	Vivado HLS	Xilinx Vivado and SDK Soft.	Xilinx Vivado
-I/P image -frame fps	1920*1080 Pix 74 fps	— —	640*480 Pix 15 fps	768*512 Pix from a file	1280*720 Pix 60 fps
Processed image size	1920*1080 Pix	320*200 Pix	640*480 Pix	96*64 Pix	90*90 Pix
Data width	RGB	8-bit	RGB & 8-bit	RGB	8-bit
BG Storage memory	32MB External, 6 Internal BRAM	—	—	—	BRAM
Algorithm	MBSCIG	BGS	BGS	Image Enhancement	ROI & BGS
Clock freq.	FPGA: 129MHz CortexA9:800MHz	100MHz	—	—	-74.25MHz -148.5MHz -222.75MHz

the sampling period (13.468nsec). Whether an object is in the foreground or background depends on the difference between the current and background frames, which can be manually updated during the operation. Results from pixel-by-pixel subtraction can be used for further processing, such as traffic sign identification and recognition.

A comparison has been made to illustrate the different implementation strategies used for such an algorithm. The recommended system can only be applied to detect a stationary object using a fixed camera. The suggested system can also be expanded to differentiate between different kinds of traffic signs depending on the data gathered by employing smart technologies.

Future development and extension of the suggested module could include enlarging the processed image size and including an IP module to update the BG image whenever there are notable changes in daytime illumination. The module can be extended to implement the BGS algorithm with various dimensions of an image frame in the range from hundreds to thousands (e.g., 260×260, 1280×720) by using an

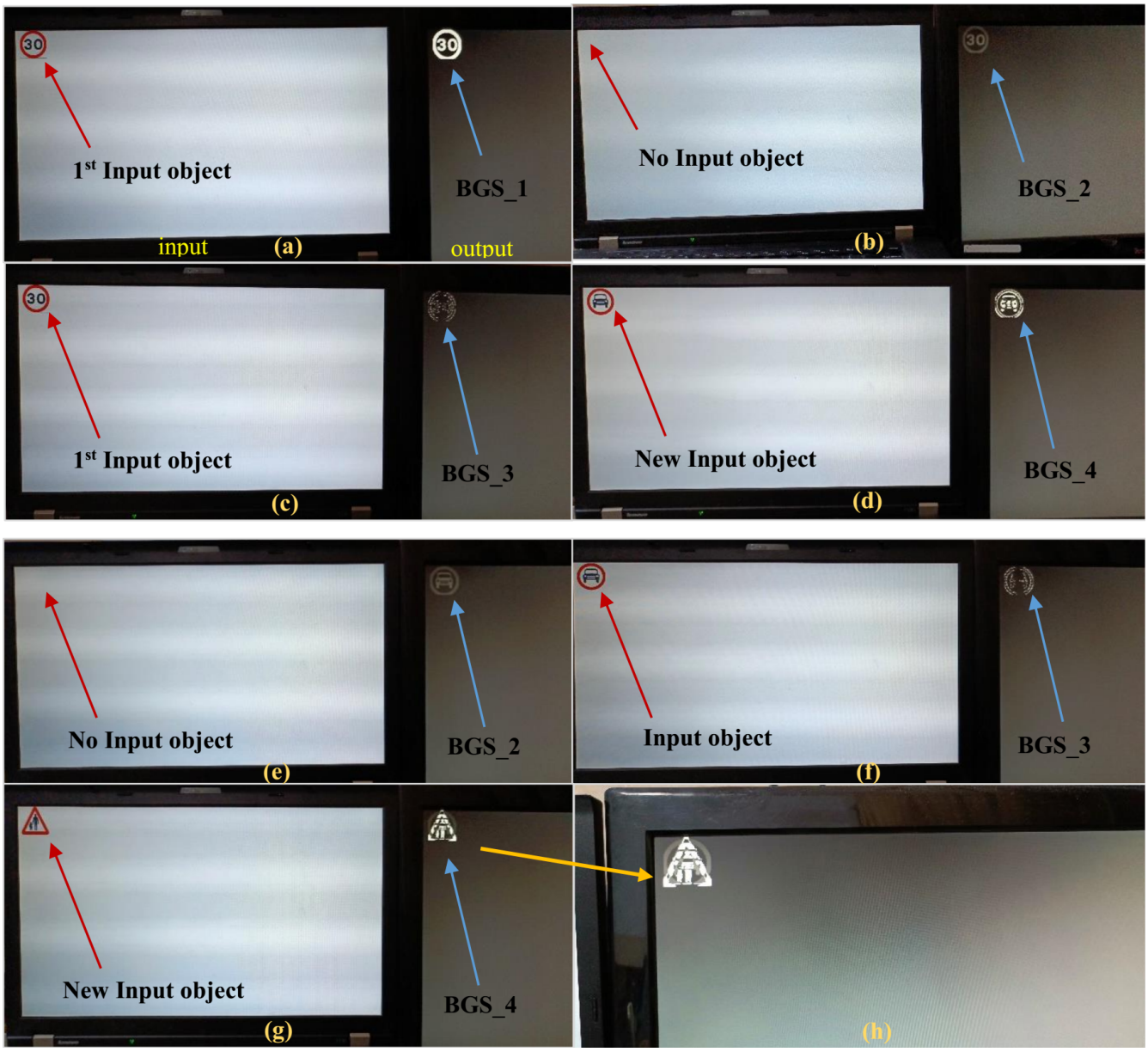


Fig. 8. Results of the implemented BGS algorithm with image size (90×90) pixels on three traffic signs. The red arrows refer to the input image while the blue arrows refer to the BGS result

FPGA platform with more resources. Utilizing both software (achieved by the PS part) and hardware (gained by the PL section), which is also known as hardware/software co-design, is another enhancement that can be made to reduce design time and increase algorithm implementation efficiency. Finally, although hardware implementation is time-consuming and requires a wide knowledge of electronics it is the recommended method to solve the critical time operation.

Acknowledgment

This research paper was completed in the University of Mosul, Electrical Engineering Department laboratory. The authors would like to thank the University of Mosul for their support.

Conflict of Interest

The authors have no conflict of relevant interest to this article.

Algorithm 1 Verilog code for the FSM shown in Fig.3 to control the load/read process in/from BRAM

```
module BRAM_WR (clk, Vsync, sw0,
Active_line, row_count, BRAM_add, W1_R0);
input wire clk, sw0, Vsync, input Active_line;
output reg W1_R0;
output reg [12:0] BRAM_add;
output reg [6:0] row_count;
reg [6:0] pixel_row;
reg [2:0] state_reg;
parameter Start = 3'b000;
parameter Write_EN = 3'b001;
parameter Write_to_BRAM = 3'b010;
parameter Wait_Line_end = 3'b011;
parameter row_count_W = 3'b100;
parameter Read_EN = 3'b101;
parameter Read_BRAM = 3'b110;
parameter row_count_R = 3'b111;
always @( posedge clk or posedge Vsync )
if ( ! sw0 )
state_reg <= Start;
else case ( state_reg )
Start:
if ( sw0 && Vsync )
state_reg <= Write_EN;
W1_R0 < =1'b1;
else if ( ! sw0 && W1_R0==0 )
state_reg < = Read_EN;
Write_EN:
if (Vsync & W1_R0)
row_count < =0;
BRAM_add  < =12'b0;
pixel_row < =0;
state_reg < = Write_to_BRAM;
Write_to_BRAM:
if (Active_line && (pixel_row==7'd90))
state_reg < = Wait_Line_end;
Wait_Line_end:
pixel_row < =0;
if( ! Active_line && W1_R0)
state_reg < =row_count_W;
else if ( ! Active_line && ( ! W1_R0))
state_reg < = row_count_R;
row_count_W:
if ( row_count < 90)
state_reg < = Write_to_BRAM;
else W1_R0 < =1'b0;
state_reg < = Start;
Read_EN:
if (Vsync && ( ! W1_R0))
row_count < =0;
BRAM_add < =12'b0;
```

Algorithm 1 (continued)

```
pixel_row < =0;
state_reg < = Read_BRAM;
Read_BRAM:
if (Active_line && (pixel_row==7'd90))
state_reg < = Wait_Line-end;
row_count_R:
if (row_count < 90)
state_reg < = Read_BRAM;
else state_reg < = Start;
default:
state_reg < = Start;
endcase
endmodule
```

REFERENCES

[1] F. Carrizosa-Corral, A. Vázquez-Cervantes, J.-R. Montes, T. Hernández-Díaz, J. C. Solano Vargas, L. Barriga-Rodríguez, J. A. Soto-Cajiga, and H. Jiménez-Hernández, "Fpga-soc implementation of an ica-based background subtraction method," *International Journal of Circuit Theory and Applications*, vol. 46, no. 9, pp. 1703–1722, 2018.

[2] I. Garcia and E. Guzmán-Ramírez, "A fpga-based experimentation system for designing, implementing, and evaluating real-time video processing and analysis algorithms at undergraduate level," *Computer Applications in Engineering Education*, vol. 27, no. 2, pp. 387–405, 2019.

[3] I. Skliarova and V. Sklyarov, *FPGA-BASED hardware accelerators*. Springer, 2019.

[4] P. Sikka, A. R. Asati, and C. Shekhar, "Real time fpga implementation of a high speed and area optimized harris corner detection algorithm," *Microprocessors and Microsystems*, vol. 80, p. 103514, 2021.

[5] S. Wang, C. Zhang, Y. Shu, and Y. Liu, "Live video analytics with fpga-based smart cameras," in *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pp. 9–14, 2019.

[6] R. Kaibou, M. S. Azzaz, M. Benssalah, D. Teguig, H. Hamil, A. Merah, and M. T. Akrour, "Real-time fpga implementation of a secure chaos-based digital crypto-watermarking system in the dwt domain using co-design approach," *Journal of Real-Time Image Processing*, vol. 18, no. 6, pp. 2009–2025, 2021.

[7] G. Cocorullo, P. Corsonello, F. Frustaci, L.-d.-l.-A. Guachi-Guachi, and S. Perri, "Multimodal background subtraction for high-performance embedded systems," *Journal of Real-Time Image Processing*, vol. 16, pp. 1407–1423, 2019.

[8] V. P. Korakoppa, H. R. Aradhya, *et al.*, "An area efficient fpga implementation of moving object detection and face detection using adaptive threshold method," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pp. 1606–1611, IEEE, 2017.

[9] O. Barnich and M. Van Droogenbroeck, "Vibe: A universal background subtraction algorithm for video sequences," *IEEE Transactions on Image processing*, vol. 20, no. 6, pp. 1709–1724, 2010.

[10] M. Benetti, M. Gottardi, T. Mayr, and R. Passerone, "A low-power vision system with adaptive background subtraction and image segmentation for unusual event detection," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 11, pp. 3842–3853, 2018.

[11] V. Taraate, Taraate, and Meherishi, *Advanced HDL synthesis and SOC prototyping*. Springer, 2019.

[12] J. L. Brock, *Introduction to Logic Circuits & Logic Design with Verilog*. Spinger, 2019.

[13] M. Tomasi, S. Pundlik, and G. Luo, "Fpga–dsp co-processing for feature tracking in smart video sensors," *Journal of Real-Time Image Processing*, vol. 11, pp. 751–767, 2016.

[14] D. A. Devi, N. R. Kathula, G. Kalluri, and L. S. Bondalapati, "Design and implementation of image processing application with zynq soc," *International Journal of Computing and Digital Systems*, vol. 14, no. 1, pp. 377–385, 2023.

[15] I. N. Rodrigues, C. L. S. de Melo, V. M. da Frota Botinelly, and J. P. de Oliveira, "Fpga hardware architecture with parallel data processing to detect moving objects using the background image subtraction technique," in *2015 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pp. 841–846, IEEE, 2015.

[16] X. Ren and Y. Wang, "Design of a fpga hardware architecture to detect real-time moving objects using the background subtraction algorithm," in *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*, pp. 428–433, IEEE, 2016.

[17] G. Conti, M. Quintana, P. Malagón, and D. Jiménez, "An fpga based tracking implementation for parkinson's patients," *Sensors*, vol. 20, no. 11, p. 3189, 2020.

[18] T. Bouwmans, "Traditional and recent approaches in background modeling for foreground detection: An overview," *Computer science review*, vol. 11, pp. 31–66, 2014.

[19] S. Arivazhagan and K. Kiruthika, "Fpga implementation of gmm algorithm for background subtractions in video sequences," in *Proceedings of International Conference on Computer Vision and Image Processing: CVIP 2016, Volume 2*, pp. 365–376, Springer, 2017.

[20] J. Cavanagh, *Verilog HDL design examples*. CRC Press, 2017.

[21] I. Grout, *Digital systems design with FPGAs and CPLDs*. Elsevier, 2011.