FIELD PROGRAMMABLE GATE ARRAY IMPLEMENTATION OF MINIMUM OUTPUT SUM OF SQUARED ERROR TRACKER ALGORITHM

A THESIS SUBMITTED TO
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
KOCAELİ UNIVERSITY

BY

JOSPHAT CHEGE NJUGUNA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRONICS AND COMMUNICATIONS ENGINEERING

KOCAELİ 2021

FIELD PROGRAMMABLE GATE ARRAY IMPLEMENTATION OF MINIMUM OUTPET SUM OF SQUARED ERROR TRACKER ALGORITHM

A THESIS SUBMITTED TO
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
KOCAELİ UNIVERSITY

BY

JOSPHAT CHEGE NJUGUNA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRONICS AND COMMUNICATIONS ENGINEERING

Assist.Prof. Anıl ÇELEBİ		
Supervisor,	**Kocaeli Univ.**	**.....**
Prof.Dr. Ali TANGEL		
Jury member,	**Kocaeli Univ.**	**.....**
Prof.Dr. Mehmet Kemal GÜLLÜ		
Jury member,	**Bakırçay Univ.**	**.....**

Thesis Defense Date: 22.06.2021

PREFACE AND ACKNOWLEDGEMENT

In the scope of this thesis, the MOSSE tracker algorithm is implemented in FPGA using C++ language. The target tracking technology will be useful in intelligent video surviallance systems.

I want to thank God for good health during this research period. I am also grateful for the support I received from my family. Furthermore, I want to thank my supervisor, Dr. Anil Çelebi, who allowed me to work on this thesis, guided and assisted me at every stage of the work. Additionally, I want to thank Kuantek Electronics Company for providing the laboratory and research materials to perform the research.

Besides, I would like to express my sincere gratitude to Turkey's Presidency for Turks Abroad and Related Communities (YTB) for providing me educational opportunities.

July-2021 Josphat Chege NJUGUNA

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SYMBOLS AND ABBREVIATIONS

e : Euler's number
ε : Regularization parameter
η : Learning rate
σ : Radius of the peak region, (m)
$\prod$: Pi
Θ : Dot product of two elements
$\sum$: Summation
$*$: Complex conjugate

Abbreviations

2D : Two Dimension
ASEF : Average Synthetic Exact Filter
BRAM : Block Random Access Memory
CNN : Convolution Neural Networks
CSRT : Channel and Spatial Reliability Tracker
DCF : Discriminative Correlation Filter
DDR : Double Data Rate
DFT : Discrete Fourier Transform
DSP : Digital Signal Processing
FFT : Fast Fourier Transform
FPGA : Field Programmable Gate Array
GOTURN : Generic Object Tracking Using Regression Networks
HLS : High-level Synthesis
HoG : Histogram of Gradient
HW : Hardware
IFFT : Inverse Fast Fourier Transform
IP : Intellectual Property
IRFPA : Infrared Focal Plane Arrays
KCF : Kernelized Correlation Filters
LUTRAM : Look-up Tables Based Random Access Memory
MIL : Multiple Instance Learning
MOSSE : Minimum Output Sum of Squared Error
ROI : Region of Interest
RTL : Register Transfer Level
SW : Software
TLD : Tracking Learning Detection

ALANDA PROGRAMLANABİLİR KAPI DİZİLERİ İLE EN KÜÇÜK TOPLAM KARESEL HATA SÜZGECİ TEMELLİ İZLEYİCİ ALGORİTMASININ GERÇEKLENMESİ

ÖZET

Bu yazıda, kızılötesi odak düzlemi dizileri (IRFPA) için en küçük toplam karesel hata (Minimum Output Sum of Squared Error (MOSSE) temelli hedef takip algoritmasının alanda programlanabilir kapı dizileri (FPGA) üzerindeki donanım mimarisi sunulmuştur. Mimari, yüksek seviyeli sentez yaklaşımı (High-Level Synthesis (HLS)) ile C++ dili kullanılarak modellenmştir. Geliştirilen donanım mimarisi daha sonra 16nm teknolojide üretilen bir FPGA yongasında gerçeklenmiş ve test edilmiştir. Çalışmalar 640×480 çözünürlüğündeki video çerçeveleri üzerinde gerçekleştrilmiştir. Geliştirme süreçlerinde Xilinx firmasının Vivado HLS yazılımı kullanılmıştır. Deneysel sonuçlara göre, 25 BRAM kullanan ve maksimum 300 MHz çalışma frekansına sahip bir mimari elde edilmiştr. Yazılım ve donanım benzetimleri karşılaştırılarak algoritmayı yüksek doğrulukla gerçekleyen bir donanım mimarsi geliştrilmiştir.

Anahtar Kelimeler: FPGA, MOSSE Takipçisi,Yüksek Seviye Sentez (HLS).

FIELD PROGRAMMABLE GATE ARRAY IMPLEMENTATION OF MINIMUM OUTPUT SUM OF SQUARED ERROR TRACKER ALGORITHM

ABSTRACT

In this thesis, a novel hardware architecture for the minimum output sum of squared error (MOSSE) tracker algorithm is presented. The proposed architecture is implemented on 16 nm field programmable gate array (FPGA). The hardware architecture is modeled with high level synthesis approach by using C++ by using Xilinx Vivado HLS tool. The proposed hardware architecture is implemented on an FPGA device fabricated at a 16nm technology node. Thermal camera development kit is used for experiments in which ULIS PICO Gen2 microbolometer is installed. The Pico Gen2 thermal imaging sensor has a video resolution of 640×480 pixels with 30 frames/second. According to experimental results, the implemented design uses 25 BRAMs and a maximum frequency of 300MHz. According to the the RTL simulation and MATLAB simulation results there is negligible difference between HW and SW implementation.

Keywords: FPGA, MOSSE Tracker, High-Level Synthesis (HLS).

INTRODUCTION

The process of detecting the position of an object in each frame of a video is called visual tracking [1]. The technology is widely used in the military, medicine, traffic control, and aerospace. Tracking an object helps to extract details that are used for further processing depending on the field of application [1].

The appearance of an object and the background information of an image make it difficult for visual tracking. Lighting, natural variations, and non-rigid transformation cause object appearance to change [2]. When objects move out of the frame or are occluded by other objects, it is not easy to continue tracking. When an object moves out of the tracking video, the tracker might start tracking another object or continue tracking the object that caused occlusion [2].

Examples of robust techniques used for tracking include boosting tracker [3], multiple instance learning (MIL) tracker [4], the median flow tracker [5], minimum output sum of squared error (MOSSE) [6], tracking learning detection (TLD) [7], kernelized correlation filters (KCF) based tracking [8], generic object tracking using regression networks (GOTURN) [2], channel and spatial reliability tracker (CSRT) [9]

The thesis is subdivided into 5 chapters. In Chapter 1, an overview of object tracking algorithm is given. An introductory study of microstrip of MOSSE tracker algorithm is discussed in Chapter 2. In Chapter 3, the implementation of MOSSE tracker in hardware is explained in details. In Chapter 4, the simulation and real-time test results are presented and discussed. Chapter 7 concludes this thesis and gives some recommendations.

1. OBJECT TRACKING ALGORITHMS

1.1. Boosting Tracker

Boosting tracker method is proposed in [3]. This method is an improved version of the AdaBoost feature selection algorithm. The boosting tracker is an online version and maintains a global classifier pool with multiple selectors with weak classifiers [3]. Each weak classifier in the pool is updated using the new training samples. The system contains numerous cascaded selectors in which the first selector initializes the current sample's significance, chooses the weak classifier with the slightest error The estimated importance is passed to the subsequent selector until all the selectors have been updated. Finally, the weak classifier is replaced with the robust classifier chosen from the best weak classifier. The initial target in the current frame is utilized as a positive example, and the exact size is used to exploit other areas surrounding the target as a negative example. The trained classifier searches the potential targets from the neighbourhood in the next frame. The boosting tracker is fit to handle complex backgrounds and occlusions.

1.2. Multiple Instance Learning (MIL) Tracker

Multiple instance learning tracker that advances boosting algorithm through a bag that is a set of image patches instead of using one sample for training is proposed in [4]. A bag containing only negative samples is called a negative bag. If it has at least one positive bag, it is called a positive bag. The best positive example is chosen when the MIL tracker collects lots of small bags centred at the tracking object as the potential positive bags. The method prevents the MIL tracker from losing necessary details and mislabelling problems.

1.3. MedianFlow Tracker

Medianflow tracker is an approach that involves forward-backward tracking [5]. The consistency in forward-backward tracking is analyzed as a quality measure in tracking. The corresponding error between forward-backward trajectories constructed by the

medianflow tracker is estimated at each instant time. The trajectory forward-backward error is selected as the sample for the subsequent tracking. In conclusion, the medianflow Tracker is more reliable when tracking consistent movement.

1.4. Tracking Learning Detection (TLD)

Tracking learning detection (TLD) is composed of three parts: a tracker, a learner, and a detector [7]. The tracker follows the object through subsequent frames. The learner depends on a P-expert and N-expert to detect false alarms and misdetection and then updates the detector. According to an appearance structure, the detector identifies the possible targets, sends the output to the learner, and corrects the learner if needed. The TLD has the capability of failure recovery at the expense of instability. Compared to other trackers with the problem of accumulating errors, tracking and detection make the TLD tracker more robust for long-term tracking.

1.5. Kernelized Correlation Filters (KCF) Based Tracking

Kernelized correlation filters based tracking takes advantage of overlapping regions in multiple positive samples. The abundant data is computed in the frequency domain using the Fourier series computational techniques to increase the learning speed. Moreover, the technique depicts the usefulness of negative samples and uses more samples for improved training. A cyclic shift is applied to each critical sample to generate more samples. The features of circulant matrices are utilized to increase the computation. Furthermore, known kernel techniques are used to deal with nonlinear regression problems. The KCF tracker extracts the Histogram of Gradient (HoG) characteristics to improve the tracking accuracy instead of scanning through the raw pixels.

1.6. The Generic Object Tracking Using Regression Networks Tracker

Generic object tracking using regression networks (GOTURN) based on convolution neural networks (CNN) is proposed in [2]. GOTURN tracker adopts an offline dataset used to train CNN. The online tracking uses the generated pre-trained model, and the pretraining process uses the available information in offline datasets to learn both the motion and target appearance relationship. The GOTURN tracker achieves maximum

speed during the tracking process since CNN updates are offline. Although it is not a prerequisite to adding specific training targets in the dataset for pretraining, the GOTURN tracker tends to track the training set compared to the dataset, not in the training set. The main problem with GOTURN can result from the quality of the pre-trained model that can affect the performance of the online tracking process.

1.7. The Channel and Spatial Reliability Tracker (CSRT)

Channel and spatial reliability tracker (CSRT) is a discriminative correlation filter (DCF) based algorithm [9]. It extends the DCF tracker by adding spatial and channel reliability. The optimal filter size is calculated using the spatial reliability map. The CSRT can adjust the filter size that makes it better than the traditional DCF algorithm, excluding the unwanted samples. Furthermore, a spatial reliability map is capable of handling nonrectangular targets. The channel reliability is calculated to weigh the significance of each channel filter, then combine all to find the final response map. The CSRT tracker can achieve accurate results with real-time speed using the HoGs and Colorname standard characteristics sets.

1.8. Minimum Output Sum of Squared Error (MOSSE) Tracker

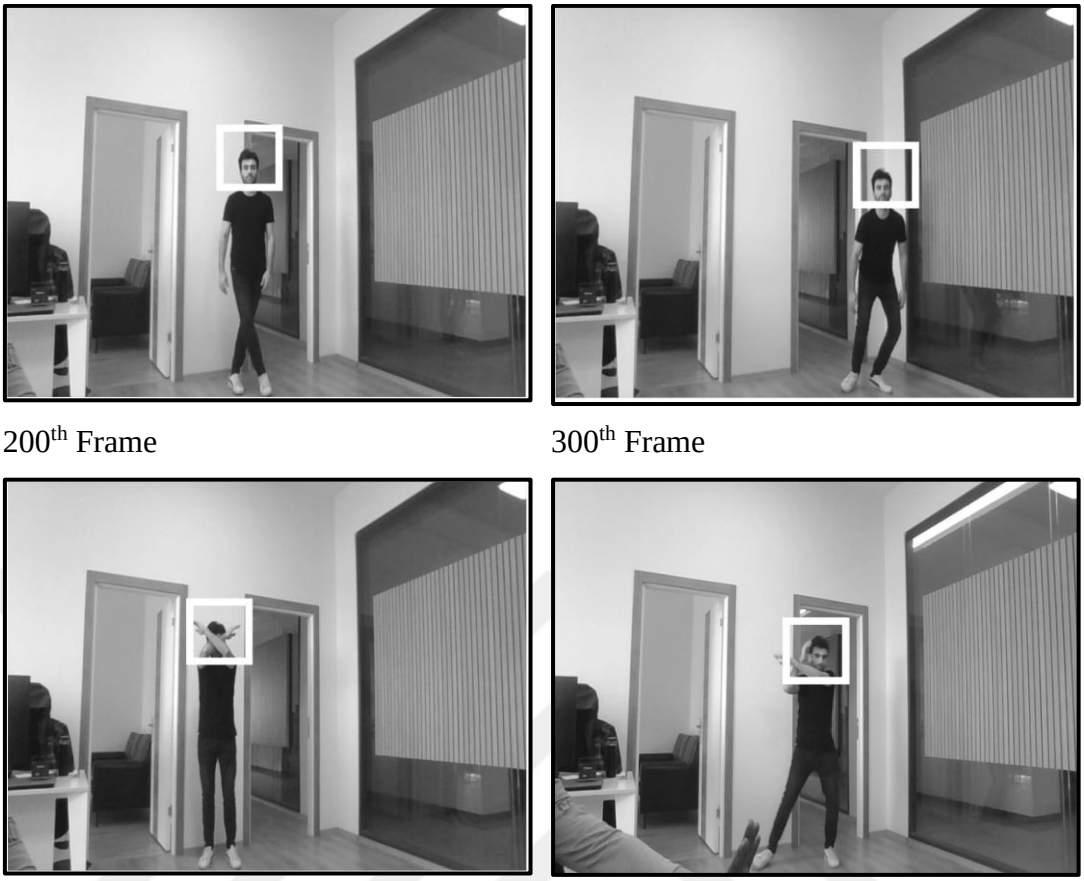
The paper focuses on MOSSE and its application in visual tracking [6]. The filter creates correlation filters that perform better than simple templates and map the images to their ideal outputs [2]. These correlation filters reduce the background interference and achieve better performance. The filters apply techniques to overcome the changes in lighting, scale pose, and shape of an object [2]. The MOSSE advances the ASEF [10] filter that has the problem of overfitting [2]. The filter's flexibility allows the target initialization at any position instead of other filters that require the target to be at the center of the image.

Initialization and tracking are the main components of the MOSSE tracking algorithm [6]. The first frame is used for initialization, and a later update of the kernel is done for the other frames. However, few frames can be used for initialization with no update filter for a simple tracker. First, the region of interest is identified (ROI) and cropped to initialize the filter. The filter is then correlated with a tracking window to find the new location of the object [6]. The process continues as the filter continues updating.

An object is selected by placing it at the center of the window during the real-time test. However, in Matlab testing, the object is selected by drawing a window around the object. Before initializing the filter, a pre-processing stage is undertaken on each frame of the cropped region. The ROI is converted to the Fourier domain, and a synthetic output is generated to initialize and update the filter [11]. The synthetic output is also in the Fourier domain, and the correlation is done in the Fourier domain. The advantage of using the Fourier domain is to make the computation faster [12]. The correlated output is converted to the time domain to identify the new location of the object.

The images used for test contains a person making different movements to test the robustness of the tracker. Figure. 1.1 shows the initialization of the filter at the 7^{th} frame. The person stays for one position for almost 100 frames to test if the filter is initialized correctly. The white bounding box indicates the tracking window. Figure. 1.2 illustrates the tracking process of the person in the 200^{th} , 300^{th} , 400^{th} , and 1000^{th} , respectively.

Figure 1.1. Initialization of the target



200th Frame 300th Frame

400th Frame 1000th Frame

Figure 1.2. Frames tracked in a video

This thesis presents the maths and steps for building the MOSSE filter. Furthermore, it shows the hardware architecture implementation modeled in C++ using the High-Level Synthesis Platform (HLS). Xilinx ZYNQ Ultrascale+ FPGA development board is used for field tests [13].

1.9. Definition of Terms

The terms presented in this section have been repeatedly used and helps to understand the underlying meaning of the context.

Object/ Target: An object is a person or a thing with unique features that identify it from the other objects. It is also referred to as the target in this paper [14].

Template: A template is a section of the image that cropped from the larger image and is used as the input image to perform the required calculations for the tracking process [6].

Correlation: Correlation is a measure that shows the similarity between two images. Template matching uses the correlation of a template and an image to find the location of the object. Correlation is achieved after moving the template at different locations in the image in which the best match is achieved [11].

Filter: A filter is a sub-image with a given resolution less than the original image. The act of filtering is to use the filter and perform a pairwise sum of the filter and the corresponding image pixels [8]. Applying a filter on an image changes the output image depending on the function dedicated by the filter.

Tracking window: Section of the image in which the tracker looks for the object's new location. The tracking window is obtained from the incoming video, and it correlates with the filter to give the new location of the object on track [6].

Synthetic target: It is generated with a Gaussian peak at the object's location to be tracked. The synthetic target is mapped to the input image to generate a filter [6].

Occlusion: An occlusion is where the target is blocked by another object in a video [2].

Tracking: The algorithm finds the object's new location throughout the video in a process called tracking [15].

Initialization: Process of training a filter used to identify the tracking object [6].

Updating: It is the process of changing the filter information because of template change. The changes can be object rotation, scaling among others [6].

2. MOSSE TARGET TRACKING ALGORITHM

The Minimum Output Sum of Squared Error (MOSSE) [6] is an extension of the Average Synthetic Exact Filter (ASEF) [16]. Therefore the section gives a detailed information of ASEF that is an advancement of the Exact filter [16]. The following steps are required during filter initialization: (1) obtain a template and process it, (2) create a synthetic target, (3) convert both synthetic target and preprocessed template from the time domain to frequency domain using Fourier transform equations [17].

2.1. Comparing an Image to a Template.

The template matching technique is used to find the similarity between a template and an image by comparing the pixels. This technique uses the dot product method that helps to compare between the image and a template [8]. The dot product in our case is computed with the following equation.

$$F(x, y) = x.y = \sum_i x_i y_i \tag{2.1}$$

In Equation (2.1) [6], y represents, the image and x represents the template. The dot product of different images is low as compared to two similar images. Therefore, a threshold value can be determined to find the target in an image. The template slides over an image to find every possible similarity of the template over the image in a process called template matching. The pixels with the possibility of similarity are compared through the dot product method to find the similarity score. Fig. 2.1 shows a thermal camera image with a resolution of 640×480, and Fig. 2.2 shows a template of 64×64 pixels. Fig 2.3 shows how the correlation process is done over an image for every possible pixel combination. The correlation happens after every pixel from the left to the right and top to bottom. The peak obtained after correlation is used to find the new location of the target.

Correlation involves calculating the mean of the image of the cropped region $N \times N$. Every pixel is subtracted from the mean value, and later the cropped region is

normalized to make it a unit vector. Therefore, correlation is a y-unit vector sphere that resembles taking the cosine of the angles between the vectors. The technique works well only when matching the same type of objects with similar appearances instead of matching the different objects with similar features. In conclusion, the matching does not work well for variations in lighting, nonrigid transformations, or natural variation [6].

2.2. Preprocessing

The MOSSE filter-tracking algorithm has two main stages that include initializing the filter and tracking the object. Preprocessing step is an important step to be carried out before initializing the filter or tracking. Furthermore, the step is performed to every frame of the video before they are used for tracking [6]. The following steps are the essential steps that are required in the preprocessing stage.

1. A template of 64×64 centered on the object is cropped from an image of resolution 640×480.

2. This step involves the conversion of the template to a grayscale image although in our case the image from the thermal camera is grayscale hence the conversation will not be required.

3. The following equation is used to perform log transformation on the template. The log transformation enhances contrast reduces lighting effects, making high contrast features available for the filter to initialize [6].

$$x = \ln(y + 1) \tag{2.2}$$

In Equation (2.2) [6], x represents the input image, and y the output image after the transformation.

4. After log transformation, the pixels are normalized to get a mean of zero and a normal of one. The process helps in maintaining consistency in illumination and reducing the effects of change in illumination between the different frames.

5. The results are converted to the frequency domain using the Fourier transform equation. The following equation is a 2D DFT that is applied to the image.

$$F(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-2\pi i\left(\frac{xu}{M}+\frac{yv}{N}\right)}...u=0,...,M-1;v=0,...,N-1 \quad (2.3)$$

In Equation (2.3) [6], *f(x,y)* represents the image in the spatial domain and *F(u,v)* represents the image in frequency domain [12]. As discussed earlier to identify the new location the image has to be converted to the time domain using inverse Fourier transform given by the following equation:

$$f(x,y)=\frac{1}{NM}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{-2\pi i\left(\frac{xu}{M}+\frac{yv}{N}\right)}...x=0,...,M-1;y=0,...,N-1 \quad (2.4)$$

In Equation (2.4), *f(x,y)* represents the image in the time domain while *F(u,v)* represents the image in the Fourier domain [18].

Fig 2.1 shows the preprocessing stages in a flow diagram. A video of 640×480 resolution is the input and the template is the output.

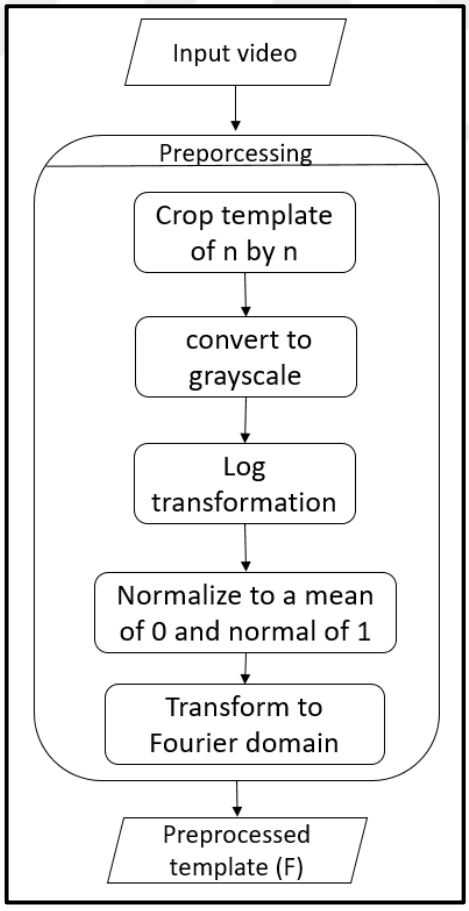


Figure 2.1. Preprocessing flow chart

2.3. Synthetic Target

A synthetic target is a target that contains a Gaussian peak centered on the object synthetically generated. The synthetic target has the role of initializing the target and updating the filter. The following equation illustrates how the synthetic target is generated.

$$gi = \sum e^{-\frac{(x-x_j)^2+(y-y_j)^2}{\sigma^2}} \tag{2.5}$$

In Equation (2.5) [19], y and x represent the location of the pixels. gi represents the synthetically generated target. y_i and x_i represent the center of the object. σ is the radius of the peak region. A value of 9 is used in this paper as it depicted the best performance. Figure. 2.2 shows the image of a synthetic targetshowing sigma with different values.

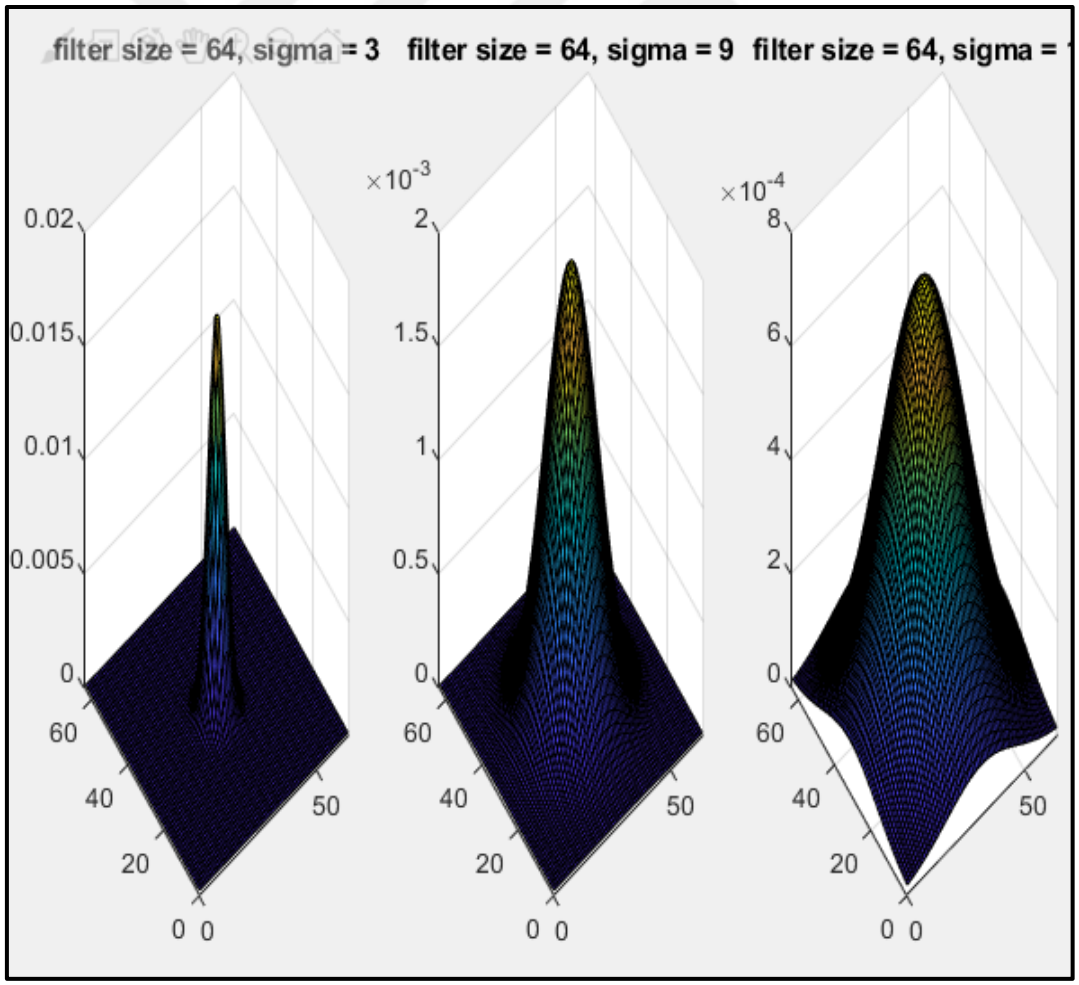


Figure 2.2. Synthetic target

2.4. Exact Filter

Correlating the exact filter with a template on which the filter was initialized produces a strong peak at the location of the object and the surrounding is padded with zeros. The name exact filter comes because of the ability to transform the input to its target output image [16]. Steps followed to initialize the exact filter include the following.

1. The region of interest is cropped of dimension 64*x*64.

2. The template undergoes the preprocessing as discussed in section 2.2.

3. Synthetic target is generated using Equation (2 5).

4. Using the Fourier Transform equations, the synthetic target is converted to the frequency domain.

5. The exact filter is obtained by the following equation.

$$H^* = \frac{G_i \odot F_i^*}{F_i \odot F_i^* + \varepsilon} \tag{2.6}$$

In Equation (2.6) [16], F represents the preprocessed template in the Frequency domain. H^* is a complex conjugate of the filter in the Fourier domain. $F^*{}_i$ is the complex conjugate of F_i and G_i is the synthetic target image in the Frequency domain. Furthermore, ε eliminates division by zero.

2.5. ASEF

ASEF comprises an average of the multiple exact filters. The ASEF is considered to create a more general filter. Taking the average removes the features that are not consistent [16]. The ASEF is initialized using the following steps.

1. The region of interest is cropped of dimension 64×64.

2. The template undergoes the preprocessing as discussed in section 2.2.

3. Synthetic target is generated using Equation (2.5).

4. Using the Fourier Transform equations, the synthetic target is converted to the frequency domain.

5. The ASEF filter is obtained by the following equation.

$$H^* = \frac{1}{N}\sum_{i=1}^{N} H_i^* \tag{2.7}$$

Equation (2.7) [16], H^* represents the complex conjugate of the filter in the frequency domain. N represents the number of the initialized input images. H_i^* is the i^{th} exact filter.

2.6. MOSSE Filter

The MOSSE filter minimizes the sum of squared error between the real output of the correlation and the expected output of the correlation [6]. The initialization of the MOSSE filter resembles the initialization of ASEF although ASEF requires a large set of initialization input images. The process starts with acquiring the template by cropping the image, then preprocessing and Fourier transform takes place. After that, a synthetic target is generated and converted to the Fourier domain. The filter is initialized exactly as the exact filter in Equation (2.6).

2.7. MOSSE Tracker

This section discusses the detailed information of the MOSSE tracker. The mathematic behind the implementation is presented. Stepwise development of the tracker is discussed beginning with basic MOSSE and later how the filter is updated in case the shape of the target keeps on changing.

2.7.1. MOSSE tracker mathematics.

A filter requires a certain number of the template for initialization to achieve accurate tracking. Therefore, a set of frames are required to initialize the filter. In this thesis, 7 frames were set for initialization. The template is preprocessed as discussed in section 2.2 to get *F*. As discussed in section 2.3, a synthetic target *G* is obtained. Having obtained *F* and *G*, they are used to initialize the filter.

$$N_i = \eta(G_i \odot F_i^*) + (1-\eta)N_{i-1} \tag{2.8}$$

$$D_i = \eta(F_i \odot F_i^* + \varepsilon) + (1-\eta)D_{i-1} \tag{2.9}$$

In Equation (2.8) [6] and (2.9) [6], *Fi* represents the preprocessed template in the i^{th} frame. *Gi* represents the synthetic target for the i^{th} frame in the video. The learning rate is represented by η, which ranges between 0 and 1. For a basic MOSSE tracker where the update is not required, the value is equivalent to 0. The symbol * and Θ represent complex conjugate and dot product multiplication respectively. Element ε ensures there is no division by zero and in this case 0.0001 is used for regularization. *Ni* and *Di* represent the numerator and denominator results of Equation (2.6). The overall equation changes to the following:

$$H_i^* = \frac{N_i}{D_i} \tag{2.10}$$

Equation (2.10) [6] H_i^* represents the complex conjugate of the MOSSE filter. After the filter is initialized the second frame becomes the present frame of the tracker. The current frame is preprocessed as discussed in preprocessing steps. The resultant template is multiplied with the filter in the frequency domain to get the new location of the target using the following equation [6].

$$G = H^* \odot F \tag{2.11}$$

In Equation (2.11) [6] *G* represents the result after dot product multiplication of *F* the preprocessed template and H^* the complex conjugate of the MOSSE filter. *G* contains the peak that indicates the new location of the target.

2.7.2. Tracking with filter update

If an occlusion occurs, changes in the object's size, or light illumination, the tracker may lose the object. Therefore a filter update is required that updates with the modification of the object [6]. Sometimes the object may rotate, and the shape may change, leading to incoherent information with the filter. For that reason, a filter update will be able to adapt to the changes of the object. According to Equations (2.8) and

(2.9), η is used as the learning rate. The value ranges between 0 and 1, and during the test, we used 0.075 that gave the best results. Figure. 2.3 illustrates the flow of the MOSSE tracker algorithm [6]. With the learning rate set to 0.075, the tracker tracks more frames than the learning rate at 0.

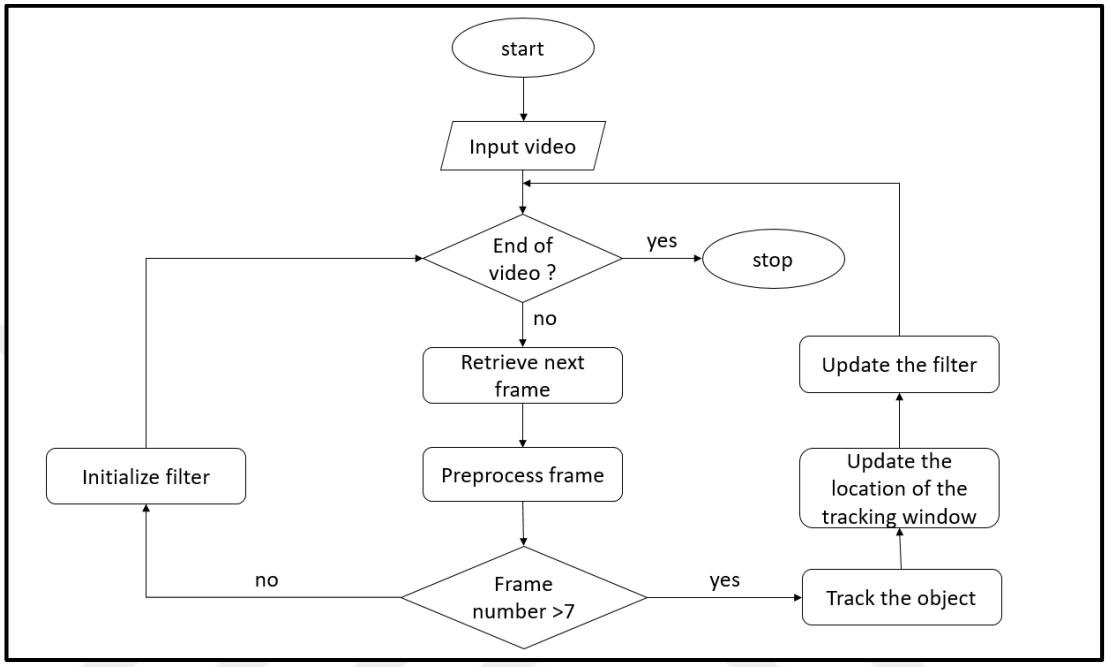


Figure 2.3. Flow chart for the MOSSE tracking algorithm

3. HARDWARE ARCHITECTURE

3.1. High-Level Synthesis (HLS)

All the design is modeled in C++ using the Vivado HLS software The tool converts the C++ code to the traditional Register Transfer Level RTL, for example, VHDL or Verilog [20]. The advantage of using HLS it reduces time to market, especially when implementing a complex algorithm. The process involves writing the code, verifying the results with the MATLAB results, Performing Co-simulation that can generate simulation results for RTL. If the results are satisfactory, one can optimize the code through available pragmas or rewrite the functions in a more optimized format. Figure 3.1 shows a flow diagram of the process to generate and RTL code modeled from C++ in the HLS platform [21].

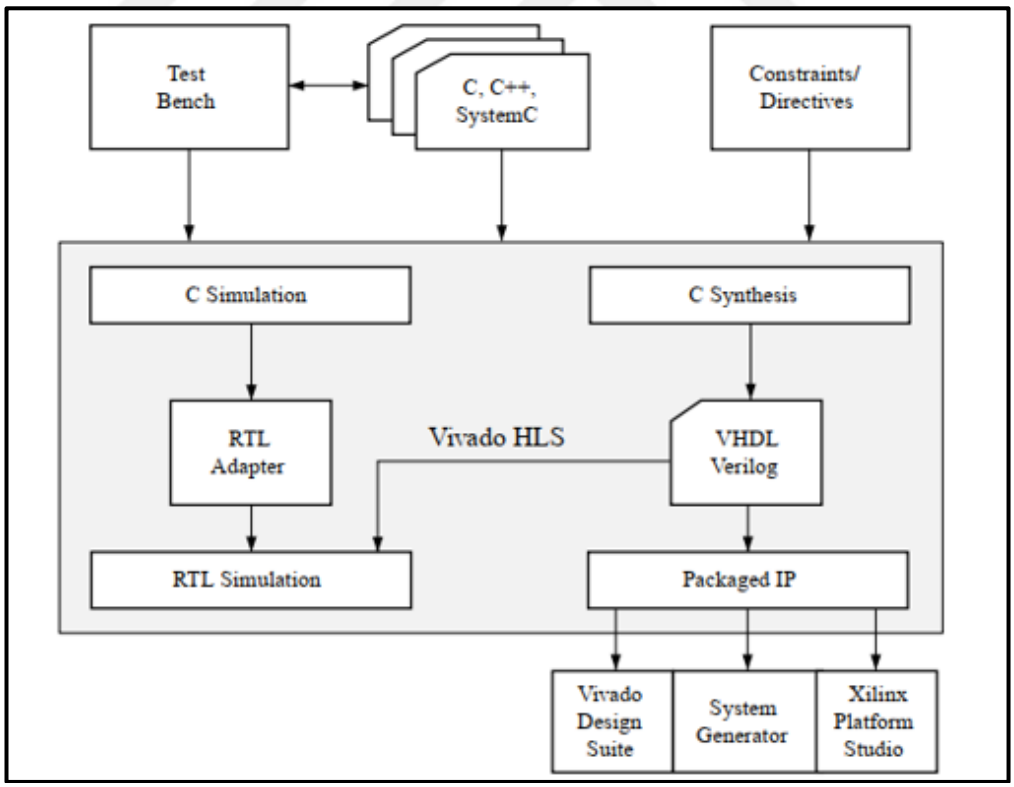


Figure 3.1. HLS flow process

3.2. FFT Hardware Architecture.

The design requires the implementation of DFT in hardware. The FFT technique is adopted in this design because it requires less computation and resource utilization [18]. Several steps need to be followed to perform 2D FFT. The steps include the following:

3.2.1. Bitreverse

The reverse bits for a 1D array image with 8 pixels are as shown in Table 3.1.

Table 3.1. Bitreverse

Index	Binary	Bit-Reversed Binary	Bit reversed Index
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

In this project, the length of a 1D array is 64. A LUTRAM is used to hold the address in reverse order. When the array is copied from the array, it is stored directly to its reverse position in the line buffer.

3.2.2. 2D forward FFT

Once the line buffer is filled with reversed samples, the line buffers are transformed to frequency domain line by line using the FFT technique. [22]. FFT technique requires the calculation of the twiddle factors [22]. For this case, twiddle factors for both FFT and IFFT are calculated and stored in LUTRAM. Again, this is a technique to save on computational resources. Every line is stored in transpose format in the DDR memory after FFT calculation. The transpose format is a requirement for 2D FFT calculations. After that process is completed, the process repeats itself, but this time, the input being

the transposed 1D FFT results. The results of 2D are transposed to the DDR memory line by line. Figure 3.3 illustrates the theoretical aspect of FFT [12]. Figure 3.4 illustrates the hardware architecture of 2D FFT.

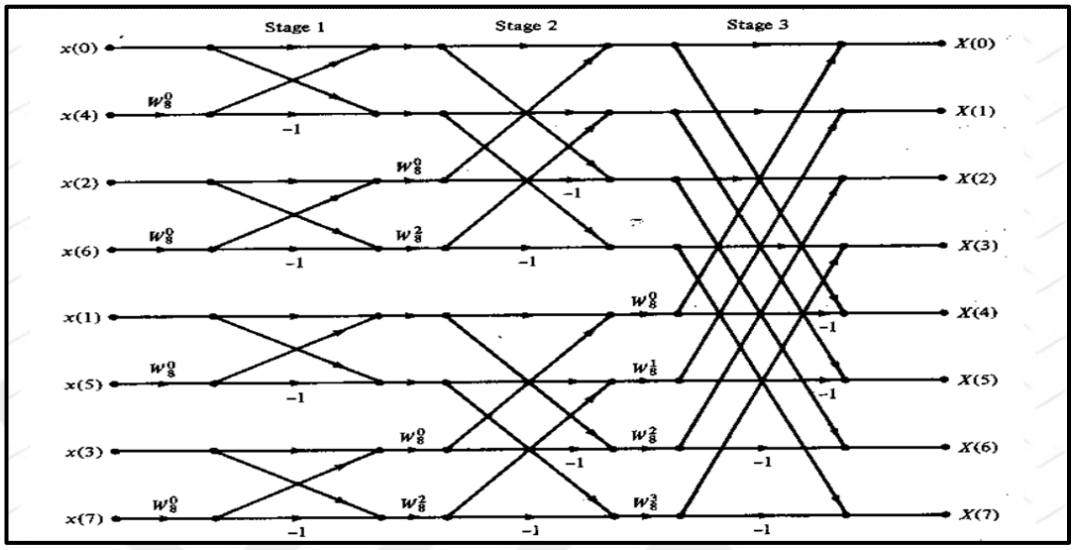


Figure 3.2. Theoretical aspect of 1D FFT for 8 pixels

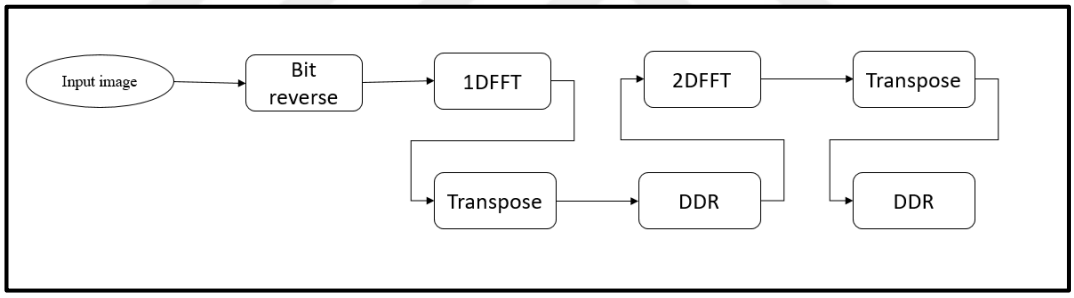


Figure 3.3. 2DFFT hardware architecture

3.2.3. 2D IFFT

According to research, inverse FFT can be achieved by the same method of forward FFT [12]. The only difference between the two complex conjugates of the FFT image is considered. Figure 3.5 illustrates the IFFT by complex conjugation [22]. In a hardware implementation, twiddle factors of the complex conjugate are stored in LUTRAM. The result is divided by 64x64, the size of the image used. Figure 3.6 shows the hardware architecture. of IFFT.

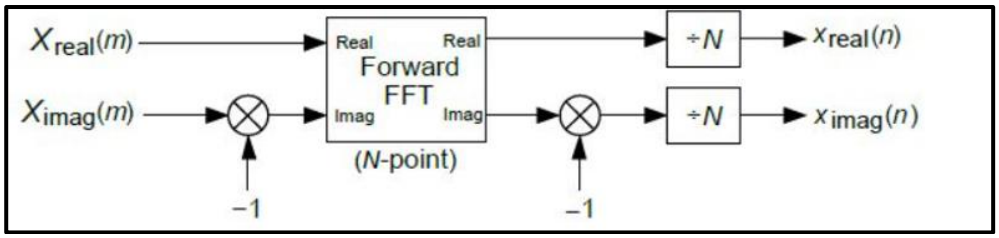


Figure 3.4. Illustration of IFFT by complex conjugation

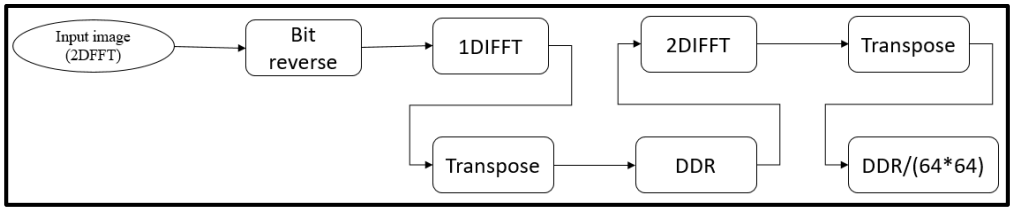


Figure 3.5. Hardware architecture for IFFT

3.3. Preprocessing Hardware Architecture

The input video has a resolution of 480p. A template of 64×64 with the target at the center is cropped. Equation (2.2) is applied in which the mean value of x is calculated along. The output is stored in DDR. Normalization does not happen immediately because the standard deviation has to be calculated that depends on o the mean value. Therefore, since the processing is done line by line due to limited resources in FPGA, it helps to use only one BRAM 64 in length. After the storage, the frame is extracted line by line from DDR for normalization and send back to DDR. When normalization has been completed, the line-by-line process repeats itself, allowing a dot product of the normalized template and Hanning filter. Storing Hanning filter data in LUTRAM removes the online calculation process. The advantage of calculating the Hanning filter before saves DSPs and reduces latency. The dot product results are then stored in DDR, ready for 1D FFT. The process continues as shown in Figure 3.7 until 2D FFT is obtained as the required template to initialize the filter.

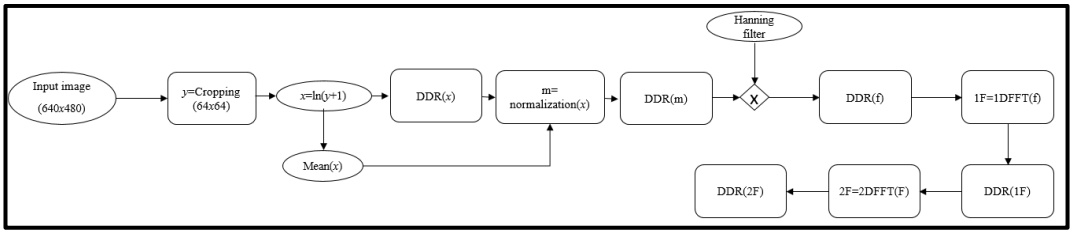


Figure 3.6. Preprocessing hardware architecture block design

3.4. Hardware Architecture for Initializing MOSSE Filter

Following Equations (2.8) and (2.9), the learning rate is taken as 0 during initialization. The template stored in DDR after preprocessing is used as the input in this stage. Furthermore, synthetic target *G* is calculated in MATLAB and stored as a constant in LUTRAM. Accessing the DDR is done line by line in which the dot product of synthetic target and the template is calculated. As shown in Figure. 3.8. *N* stands for the numerator parameter and *D* denominator parameter of Equations (2.8) and (2.9), respectively. F^* and H^* represent the complex conjugate of the respective parameters. Every line result in *N* and *D* parameters stored in DDR. Using Equation (2.10). H^* is calculated and stored in DDR as the initialized filter.

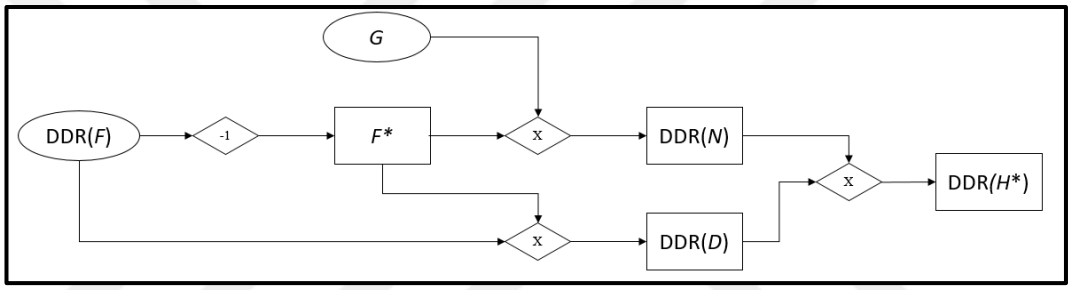


Figure 3.7. MOSSE filter initialization block design

3.5. Hardware Architecture for Finding the Location of the Target

A new template is preprocessed using the incoming frame. In Figure 3.9 both the inputs are loaded from DDR to calculate the dot product *R* as in Equation (2.11). The template is multiplied with the initialized filter H^* using the dot product method. The result *R* is stored in DDR. The results need to be converted from frequency domain to spatial domain to obtain the maximum response. IFFT of *R* is taken to achieve the condition. The IFFT of *R* indicted by *gi* is stored in the DDR The resultant image *gi* in the spatial domain has a peak response considered the new center of the target. Finally, the *gi* image is scanned pixel by pixel extracted from the DDR row by row to find the maximum response. If there is more than 1 maximum response pixel, the average is calculated, and the new location denoted by *dx* and *dy* is identified. The location is used to crop the new location of the target object as the whole process repeats itself.

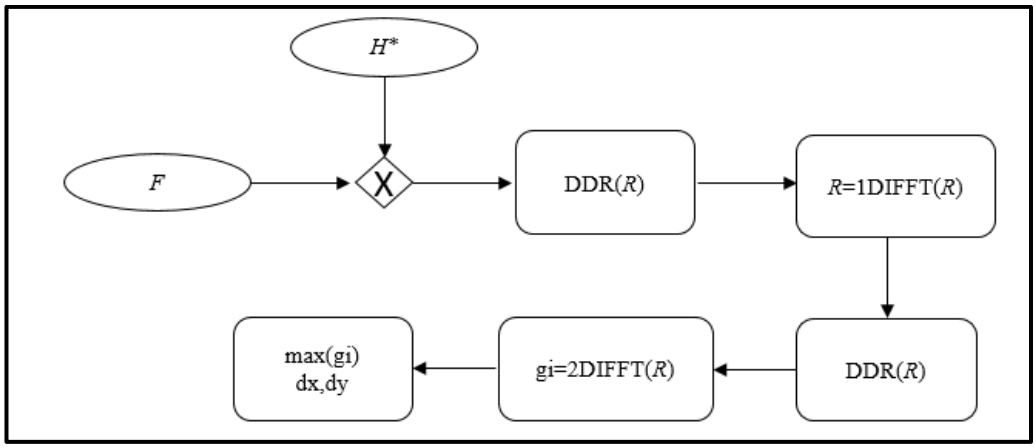


Figure 3.8. Finding the location of the target block design

3.6. Hardware Architecture for Filter Update

At this point, Equations (2.8) and (2.9) are used. $N_{\_prev}$ and $D_{\_prev}$ represent the previous numerator and denominator parameters stored in DDR. New N and D are calculated as in section 3.4. The learning rate represented by the η symbol is selected randomly between 0 and 1 to achieve the best tracking. In this case, 0.075 is used for the test. The new parameters obtained are used to generate the updated filter as explained in Equation (2.10). Furthermore, the updated values of N and D are stored in DDR to be used in the next frame as the previous parameters. Figure 3.10 provides detailed information on the hardware architecture.

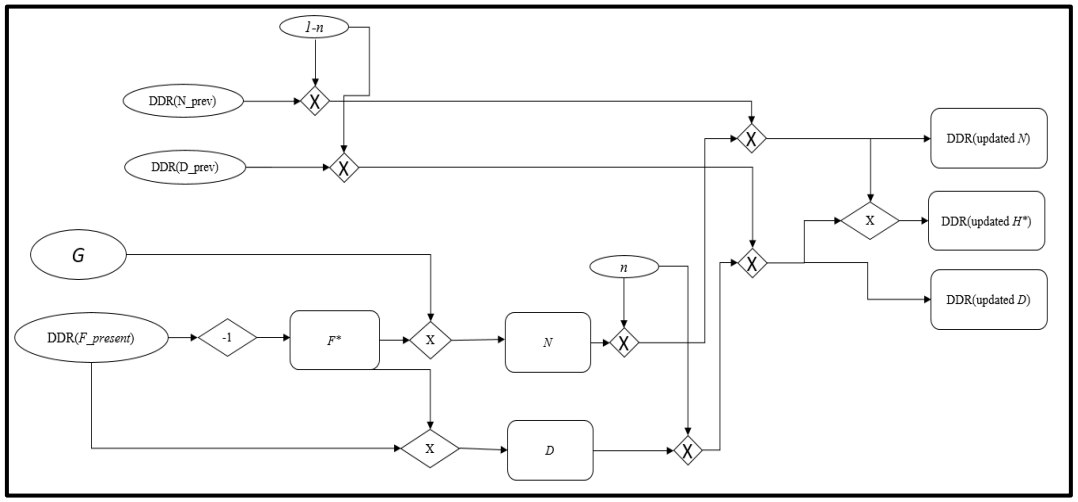


Figure 3.9. Hardware architecture for filter update

3.7. Alternative Hardware Architecture for Filter Update

The following equation is modified and tested both on MATLAB and FPGA. The results compared with the previous design had a close similarity. The advantage of the alternative method leads to a reduction in DSP utilization and access to the DDR. According to the modified solution, the learning rate is applied to the MOSSE filter represented by $H*$ instead of N and D parameters. The modified equation is as follows:

$$H^*_{\_new} = (H^*_{\_new} \times \eta) + (1-\eta) \times H^*_{\_old} \tag{3.1}$$

$H^*_{\_new}$ represents the current MOSSE filter calculated in Equation (2.10) and $H^*_{\_old}$ is the previous MOSSE filter stored in DDR. Figure 3.11 shows the hardware architecture block design.

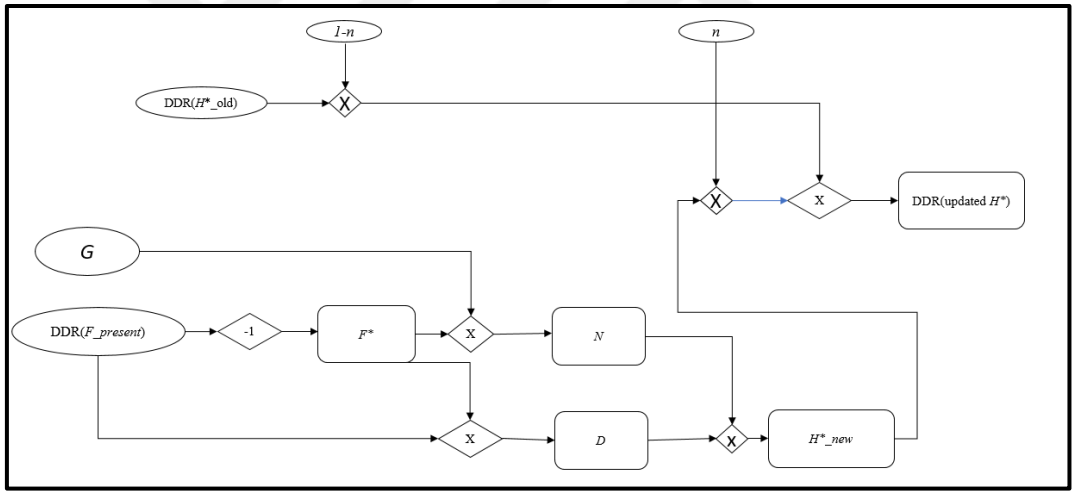


Figure 3.10. Alternative filter update hardware architecture block design

4. EXPERIMENTAL RESULTS

MATLAB results are compared with real-time results. The real-time test is conducted using a Fullscale Thermal Camera and Utrascale Zynq Board. The design achieved a maximum frequency of 300MHz and has a latency of 261630 clock cycles. According to Table 4.1, the number of DSPs is significantly high. This number can be attributed to the algorithm that requires more DSPs for computation. Using the DDR significantly reduced the number of BRAMs required

Figure 4.1 shows the ISP generated in VIVADO as the top module function of the module. The design has both input and output. The tracker receives the activation signal from the crop module for the input section once the cropped data has been stored in DDR memory and is ready for use. There are a mean value and learning rate that is set externally. The mean value is generated while in the cropping process. The learning rate is an external parameter that can be varied between 0 and 1. On the output section, there is a memory map stream that reads and writes to DDR memory. Every location has a unique address that helps identify where to write and where to read the data. The tracker generates the new location and sends it to Figure 4.2 to crop the new location for processing.

Therefore, Figure 4.2 has the input section that consists of the axi4 stream for video streaming and the location of the region of interest. The output side has the memory map stream that writes the cropped region to DDR at the specified address. It also sends the mean value and starts signal to start and stop the tracker. The tracker has to start and stop because its latency is lower than the input image, and therefore, it will complete the process and repeat the same data leading to loss of synchronization. Additionally, the output section has the output stream that sends the video for display.

Table 4.1. Resources Utilization and Timing

clock	frequency	latency	LUT	BRAMs	DSPs	Flipflops	SRL
2.89 ns	300 MHz	261630	66596	25	254	48858	1168

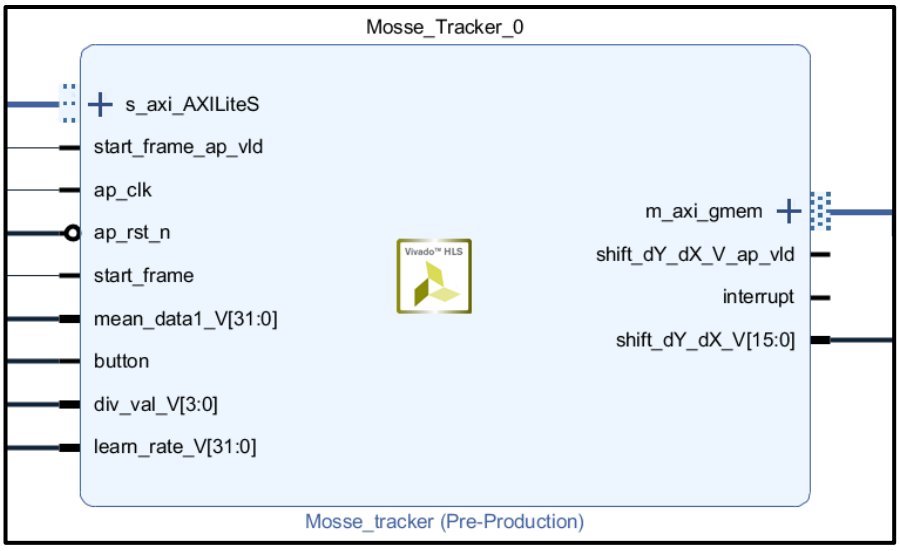


Figure 4.1. MOSSE tracker IP

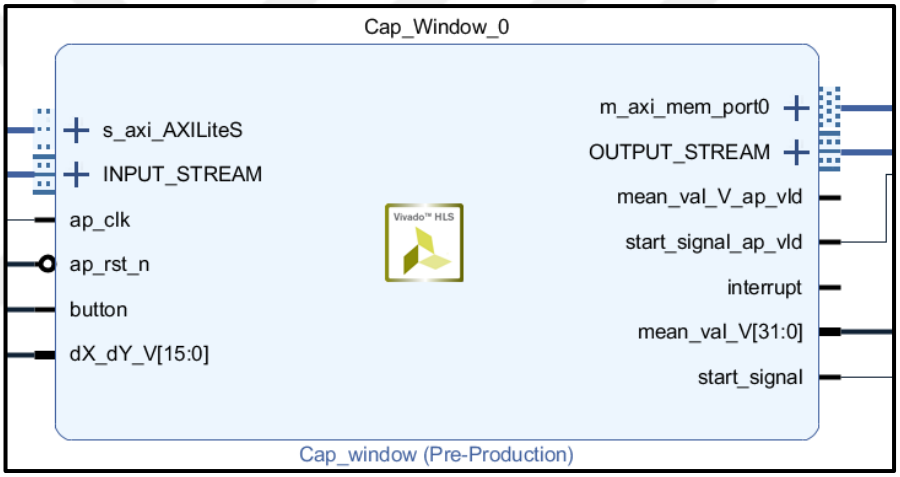
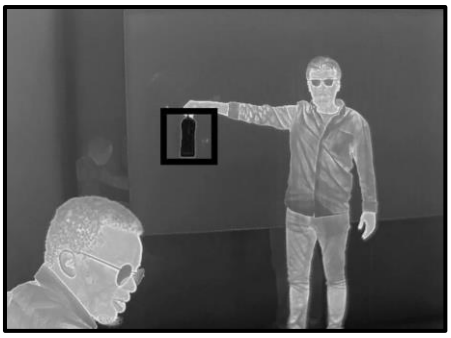


Figure 4.2. Cropping window IP

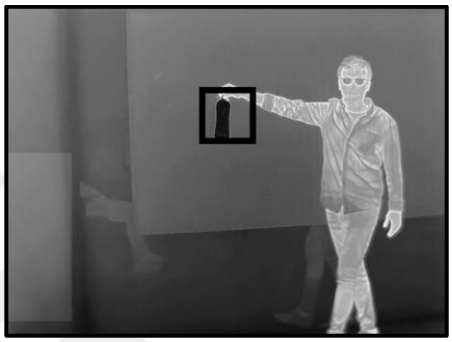
Figure 4.3 shows the output results from MATLAB. The images include 100th, 200th, 300th, 400th, 600th, 1000th, 1300th, and 1500th. The same motion was tested in real-time the images recorded. Since we could not set the module and collect the initial images, the images were recorded from 1000th 1100th, 1200th, 1300th, 1400th, 1600th, 1800th, 2000th. Figure 4.4 shows the FPGA output results. The results were very promising even when the body was rotating. Although when the body rotates at very high motion, the tracker could lose the object.



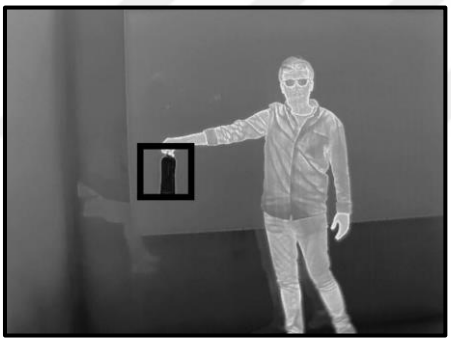
100th frame

200th frame

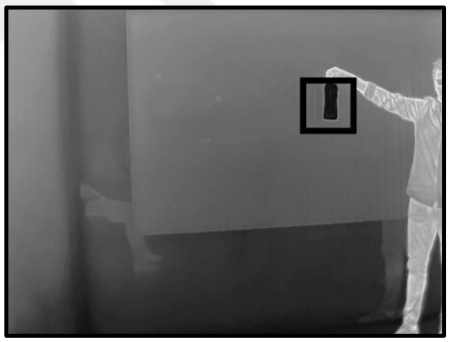
300th frame



400th frame



600th frame



1000th frame

1300th frame

1500th frame

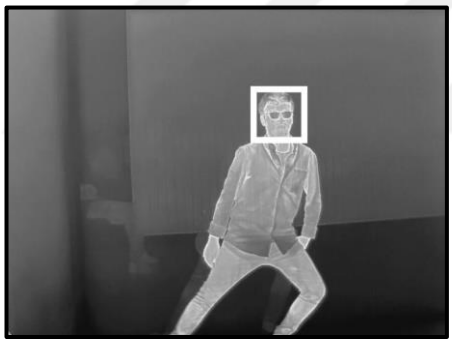
Figure 4.3. MATLAB results

1000th frame

1100th frame

1200th frame

1300th frame



1400th frame

1600th frame

1800th frame

2000th frame

Figure 4.4. FPGA results

5. CONCLUSIONS AND FUTURE WORK

The algorithms for object tracking discussed include the MOSSE tracker, a robust tracking algorithm that finds application in many fields. The MOSSE tracker is selected for implementation in this thesis due to the ease in implementation. The thesis provides a mathematical model behind the implementation of the algorithm. The challenges and the advantages of the MOSSE tracker are presented in this thesis. The main goal was to implement the MOSSE tracker in FPGA. Hardware architecture for FFT block module, preprocessing module and finding the object's current location architecture are presented. The thesis discusses an alternative method used to update the filter. Adopting the alternative method led to a reduced number of DSPs utilization. The hardware architecture is modeled in C++ language using the HLS tool. Finally, MATLAB results compared with the real-time results tested on an FPGA board showed close similarities. The design achieved a maximum frequency of 300MHz and 25 BRAMs utilization.

For future work, the Hanning filter, Gaussian filter, and twiddle factor can be stored in DDR memory to reduce LUTRAM utilization. Furthermore, the design handles a constant template of 64×64 which can lead to the target losing track if the object moves closer or away from the camera, leading to a change in size. Scaling can be applied in the future to accommodate the challenge.

REFERENCES

[1] Lu Y., Zhou Z., Zhao J., Visual Object Tracking Using PCA Correlation Filters, 2018, 2544–2550, DOI: 10.2991/caai-18.2018.10.

[2] Mi T., Yang M., Comparison of tracking techniques on 360-degree videos, *Appl. Sci.*, 2019, **9**(16), DOI: 10.3390/app9163336.

[3] Grabner H., Grabner M., H. Bischof, Real-time tracking via on-line boosting, *BMVC 2006 - Proc. Br. Mach. Vis. Conf.*, January 2006, 47–56, DOI: 10.5244/c.20.6.

[4] B. Babenko, M.-H. Yang, and S. Belongie, Visual tracking with online Multiple Instance Learning, 2010, 983–990, DOI: 10.1109/cvpr.2009.5206737.

[5] Kalal Z., Mikolajczyk K., Matas J., Forward-backward error: Automatic detection of tracking failures, *Proc. - Int. Conf. Pattern Recognit.*, 2010, 2756–2759, DOI: 10.1109/ICPR.2010.675.

[6] Bolme D., Beveridge J., Draper B., Lui Y., Visual object tracking using adaptive correlation filters, *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2010, 2544–2550, DOI: 10.1109/CVPR.2010.5539960.

[7] Kalal Z., Mikolajczyk K., Matas J., Tracking-learning-detection, *IEEE Trans. Pattern Anal. Mach. Intell.*, 2012, **34**(7), 1409–1422, DOI: 10.1109/TPAMI.2011.239.

[8] Henriques J., Rui C., Pedro M., Horge B., Kernelized Correlation Filters, *IEEE Trans. Pattern Anal. Mach. Intell.*, 2015, **37**(3) 583–596.

[9] Lukežič A., Vojíř T., Čehovin Zajc L., Matas J., Kristan M., Discriminative Correlation Filter Tracker with Channel and Spatial Reliability, *Int. J. Comput. Vis.*,2018, **126**(7), 671–688, DOI: 10.1007/s11263-017-1061-3.

[10] Bolme D., Lui Y., Draper B., Beveridge J., Simple real-time human detection using a single correlation filter, *Proc. 12th IEEE Int. Work. Perform. Eval. Track. Surveillance, PETS-Winter 2009*, 2009, DOI: 10.1109/PETS-WINTER.2009.5399555.

[11] Wang C., Zhang L., Xie L., Yuan J., Kernel cross-correlator, *32nd AAAI Conf. Artif. Intell. AAAI 2018*, 2018, 4179–4186.

[12] Oberst U., The fast Fourier transform, *SIAM J. Control Optim.*, 2007, **46**(2), 496–540, DOI: 10.1137/060658242.

[13] Cong J., Liu B., Neuendorffer S., Noguera J., Vissers K., Zhang Z., High-level synthesis for FPGAs: From prototyping to deployment, *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, 2011, **30**(4), 473–491, DOI: 10.1109/TCAD.2011.2110592.

[14] Adam A., Rivlin E., Shimshoni I., Robust fragments-based tracking using the integral histogram, *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2006, **1**, 798–805, DOI: 10.1109/CVPR.2006.256.

[15] Bao C., Wu Y., Ling H., Ji H., Real time robust L1 tracker using accelerated proximal gradient approach, *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2012, 1830–1837, DOI: 10.1109/CVPR.2012.6247881.

[16] Bolme D., Draper B., Beveridge J., Average of Synthetic Exact Filters, 2010, 2105–2112, DOI: 10.1109/cvpr.2009.5206701.

[17] Mejia-Parra D., Arbelaiz A., Ruiz-Salguero O., Lalinde-Pulido J., Moreno A., Posada J., Fast simulation of laser heating processes on thin metal plates with FFT using CPU/GPU hardware, *Appl. Sci.*, 2020, **10**(9), DOI: 10.3390/app10093281.

[18] Cooley J., Tukey J., An Algorithm for the Machine Calculation of Complex Fourier Series, *Math. Comput.*, 1965, **19**(90), 297, 19, DOI: 10.2307/2003354.

[19] Zhang X., Hu W., Maybank S., Li X., Graph based discriminative learning for robust and efficient object tracking, *Proc. IEEE Int. Conf. Comput. Vis.*, 2007, DOI: 10.1109/ICCV.2007.4409034.

[20] Lahti S., Sjovall P., Vanne J., Hamalainen T., Are We There Yet? A Study on the State of High-Level Synthesis, *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, 2019, **38**(5), 898–911, DOI: 10.1109/TCAD.2018.2834439.

[21] Xilinx Inc., Vivado Design Suite User Guide, *Ug903*, 2015, **4**, 1–173, [Online].Available:http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug903-vivado-using-constraints.pdf.

[22] Salaskar a., FFT / IFFT implementation using Vivado TM HLS.

PERSONAL PUBLICATIONS AND WORKS

[1] **Njuguna J.**, Alabay E., Çelebi A., Field Programmable Logic Arrays Implementation of Scene-Based Nonuniformity Correction Algorithm, *2021 8th International Conference on Electrical and Electronics Engineering (ICEEE)*, 2021, 6-9, DOI: 10.1109/ICEEE52452.2021.9415961.

BIOGRAPHY

He completed his primary and secondary school in Nakuru Kenya in 2008 and 2011 respectively. He joined Kenyatta University to persue degree in Electrical and Electronics Engineering in 2012. He graduated with a first class honors in 2017 with a bachelors degree in Electrical and Electronics Engineering. Between 2017-2018 He worked at Unisource Energy Company as an intern. In 2020, he joined Kocaeli University to pursue masters degree in Electronics and Communication Engineering.