SAUCIS

SAKARYA UNIVERSITY JOURNAL OF COMPUTER AND INFORMATION SCIENCES

http://saucis.sakarya.edu.tr/

e-ISSN: 2636-8129
Publisher: Sakarya University

Vol. 7, No. 1, 2024, 11-21
DOI: 10.35377/saucis...1402561

SAKARYA UNIVERSITY

RESEARCH ARTICLE

Zynq FPGA-Based Acceleration of Kernelized Correlation Filters via High-Level Synthesis of a Custom DFT Block

Mustafa Yetiş [1], Enver Çavuş [2]

[1] Electric and Electronics Engineering Department, Faculty of Engineering and Natural Sciences, Yildirim Beyazit University, Ankara, Türkiye

[2] Electric and Electronics Engineering Department, Faculty of Engineering and Natural Sciences, Yildirim Beyazit University, Ankara, Türkiye

Corresponding author:
Mustafa Yetiş, Electric and Electronics Engineering Department, Faculty of Engineering and Natural Sciences, Yildirim Beyazit University, Ankara, Türkiye
yetis1990mustafa@gmail.com

Article History:
Received: 09.12.2023
Accepted: 15.01.2024
Published Online: 15.01.2024

ABSTRACT

This study presents a hardware-software co-design implementation of an accelerator for the Kernelized Correlation Filter (KCF) tracking algorithm. Leveraging High-Level Synthesis (HLS) and the Zynq heterogeneous platform, the KCF algorithm's performance is enhanced by using a custom hardware implementation for the computationally intensive Discrete Fourier Transform (DFT) operation. Within this framework, a custom combined DFT and inverse DFT IP, named CDFT, is developed and optimized on the Programmable Logic (PL) side of the Xilinx ZCU102 FPGA, whereas the rest of the KCF algorithm is run with customized Petalinux build on the (Processing System) side. To assess real-world performance, a driver for the CDFT IP and a user application were created to measure metrics like Center Location Error (CLE), Intersection over Union (IoU), and Frame per Second (FPS). The designed DFT accelerator achieves a remarkable speedup of 21x compared to a software DFT implementation. At the algorithm level, the KCF accelerator obtains a 6x speed up with negligible precision loss. In comparison to prior studies employing exclusively hardware implementations, the proposed approach demonstrates a high accuracy at a moderate speed, while there exists potential for further optimizations to enhance its performance even further.

Keywords: Tracking, FPGA, KCF, HLS, DFT

1. Introduction

In the modern era of computer vision and artificial intelligence, the demand for real-time and high-performance processing has grown exponentially. One attractive solution to meet this demand is the hardware accelerators, where specific functions are offloaded from the main processor to specialized hardware components to achieve faster execution and reduced power consumption. One such complex algorithm that requires a hardware accelerator for real-time applications is the Kernelized Correlation Filter (KCF) algorithm. The KCF algorithm enables efficient object tracking via kernelized correlation calculations using the Discrete Fourier Transform (DFT) [1]. Among the numerous tracking algorithms in the literature, the KCF algorithm stands out for its exceptional computational efficiency [2]. It achieves this through the use of cyclic shift and the representation of data matrices as circulant matrices.

In the literature of KCF algorithm implementations for real-time object tracking, a series of progressive developments have emerged over the years. Yang et al. [3] presented an accelerator built around the Histogram of Oriented Gradients (HOG) feature extraction and correlation calculations. This architecture was deployed on Xilinx's ZCU102 Ultrascale MPSoC (Multi Processor System on Chip) platform using High-Level Synthesis (HLS). Within this framework, resource-intensive processes like HLS::sqrtf, HLS::DFT, HLS::2DFilter, HLS::exp, matrix division, and element-wise multiplication were implemented on the Programmable Logic (PL). Notably, their results exhibited performance achievements at a resolution of 960x540 and 30 Frame per Second (FPS). It was highlighted that the performance was notably constrained by input image resolution. However, the absence of detailed synthesis reports impedes comprehensive comparison or evaluation of their work. Liu et al. [4] employed a scale pyramid similar to the Discriminative Scale Space Tracking (DSST) algorithm for mitigating target scale variations. Their adaptation of HOG features into a 6-dimensional format incurred a 4% accuracy loss as opposed to the original 31-dimensional format. Leveraging radix-2 Fast Fourier Transform (FFT), the work reported a potential performance of 25 fps. However, accounting for practical considerations which are elaborated in the Experimental Results section of our study, the estimated performance was around 16 fps. Cong et al. [5] proposed a distinctive strategy encompassing multi-feature fusion and the KCF algorithm, all realized through HLS techniques. Their enhancements to the

Cite as: M. Yetiş and E. Çavuş, "Zynq FPGA-based acceleration of Kernelized correlation filters via high-level synthesis of a custom DFT block", *Sakarya University Journal of Computer and Information Sciences*, vol. 7, no. 1, pp. 11-21, 2024. https://doi.org/10.35377/saucis...1402561

KCF algorithm were manifest in incorporating Local Binary Pattern (LBP) and HOG features, yielding an enriched target feature representation. An innovative dimensionality reduction method for LBP was introduced to amplify real-time performance while preserving feature extraction efficacy. Executed on FPGA hardware, their algorithm achieved a frame rate of 35 fps with 320x240 resolution while preserving the precision.

Unlike the previous approaches that sought to fully implement the KCF algorithm on the PL section, in this work a software-hardware co-design methodology is adopted. Instead of porting the entire KCF algorithm, we exclusively focus on the most computationally intensive component: the DFT operation. Specifically, we've developed an HLS-based accelerator tailored to optimize the 2D DFT and IDFT (Inverse DFT) functions. A custom 2D Discrete Fourier Transform (DFT) IP is developed in HLS, which achieves over a 24x reduction in latency while maintaining precision compared to the 2D FFT IP based on Xilinx FFT IP Core. The combined 2D DFT and IDFT IP block, namely CDFT, is implemented on the PL fabric of Xilinx's ZCU102 platform, whereas the remaining functionality is realized with a customized Petalinux build on the PS (Processing System) side. Also, on the PS side, a custom device driver is written for the CDFT IP, and a user application is designed to measure the performance using metrics such as FPS, Intersection over Union (IoU), and Center Location Error (CLE). The implemented accelerator system achieves a more than 6x speedup over a software implementation, obtaining an 18.5 fps performance at a resolution of 640x360 with preserving precision. When the resolution is upscaled to 1280x720, the speed reduces to 16 fps, which shows our design is not affected greatly by the resolution change. These results suggest that when designing a high-accuracy HLS-based accelerator for the KCF algorithm on SoC platforms, the dominant portion of the performance gain can be obtained by carefully designing DFT and IDFT blocks. Additional speed-up gains can be obtained by further optimizations.

The rest of this paper is organized as follows. Section 2 reviews the details of the KCF algorithm. Software optimizations and hardware implementation of the KCF algorithm are discussed in Section 3. Then, Section 4 presents the experimental results of the HLS-based accelerator design of the KCF algorithm. Finally, Section 5 concludes the paper.

2. KCF Algorithm

The Kernelized Correlation Filter (KCF) algorithm was introduced by J. F. Henriques et al. in their influential 2015 paper titled "High-Speed Tracking with Kernelized Correlation Filters" [1]. KCF's primary objective is to achieve precise and efficient object tracking across consecutive frames in a video sequence. Unlike traditional methods relying on simple pixel intensities, KCF operates in a higher-dimensional feature space facilitated by kernelization, enabling it to capture intricate relationships between the object and its surroundings.

At its core, KCF involves several key processes that collectively contribute to its robust tracking capabilities. These include feature extraction with consideration for object translations (object movements), kernelization, and learning correlation filters. The feature extraction step not only captures the object of interest and its immediate context but also takes into account potential changes in the object's position across frames. Common techniques like the HOG and Scale-Invariant Feature Transform (SIFT) are employed to extract informative feature vectors. Following feature extraction, KCF takes a crucial step in enhancing its tracking capability. By generating cyclically shifted samples from the extracted features, KCF constructs a more comprehensive training dataset. These samples, created through cyclic shifts, provide a range of variations that account for potential translations in the object's location.

The Gaussian kernel function comes into play to compute the correlation between the target sample, which represents the appearance of the object of interest, and the sample being tested, which represents a potential movement location in the current frame. The coordinates of the point with the highest response value indicate the most recent location of the target. Kernels play a pivotal role in KCF by enabling the algorithm to implicitly operate in a higher-dimensional feature space without the need for explicit transformation computation. The Gaussian kernel, often used in KCF, has shown effectiveness in capturing complex non-linear relationships between data points, enhancing its ability to handle intricate tracking scenarios.

The last key process, which is the central pillar underlying KCF's efficacy, is its approach to learning correlation filters. These filters play a critical role in establishing a robust connection between the appearance of the object and its corresponding feature representation. Notably, KCF executes this learning process within the frequency domain, adjusting filter coefficients to amplify responses for positive samples (object-related features) while attenuating responses for negative samples (background-related features). A distinct advantage of the KCF algorithm arises from its strategic use of the diagonalization property inherent to cyclic matrices in Fourier space. This property contributes to the efficiency of operations within the algorithm's frequency domain computations. By focusing on correlation filters, KCF exploits both spatial and frequency information to refine its tracking precision.

The flowchart of the KCF algorithm is shown in Figure [1]. KCF's execution comprises two primary phases: training and detection. The execution begins by setting up the algorithm parameters and extracting the next frame from the video sequence. In the first frame, a Fourier-domain Gaussian regression label for ridge regression is generated and a Hanning window-based sampling window is calculated. The sampling window is saved for later feature extraction to avoid redundant calculations. Also, HOG features are extracted to be used in the training phase. If it is not the first frame, the tracker first moves to the detection and then the training phases. During detection, assuming the target's motion range is small, the target and surrounding sub-window images are acquired based on the target's position in the previous probe image. The previous window is used to extract features from the sub-window image. After extracting features, they are transformed using DFT. Kernelized Gaussian cross-correlation and transform dot product results are calculated using IDFT to find the new target position. After the detection phase, the training model is updated. In the training phase, autocorrelation, and update regression coefficients (alpha) are calculated to finalize the model update. For the current target, a sub-window image is taken and transformed using DFT to extract the features in the frequency domain. Rapid Ridge regression is then applied using autocorrelation results. Ultimately, features for the target's final position are extracted, and the classifier is updated. The algorithm continues by tracking the next frame.

The detection operation is repeated for the different scales until the multiscale operation is completed to obtain the best result during the detection phase. The multiscale mode enables tracking to continue even when the target undergoes scale changes and allows tracking to be performed with higher accuracy. If the multiscale mode is enabled, detection can be repeated at half and twice the scales separately in the detection phase, which increases the number of DFT and IDFT operation calls. Both kernelized auto-correlation and cross-correlation operations consist of a 2-norm squared sum, conjugate dot product, Inverse DFT, Gaussian kernel correlation, and DFT calculations. The FHOG operation consists of a Gradient Histogram, Unit Energy Distribution, and 31-dimension FHOG feature calculations.

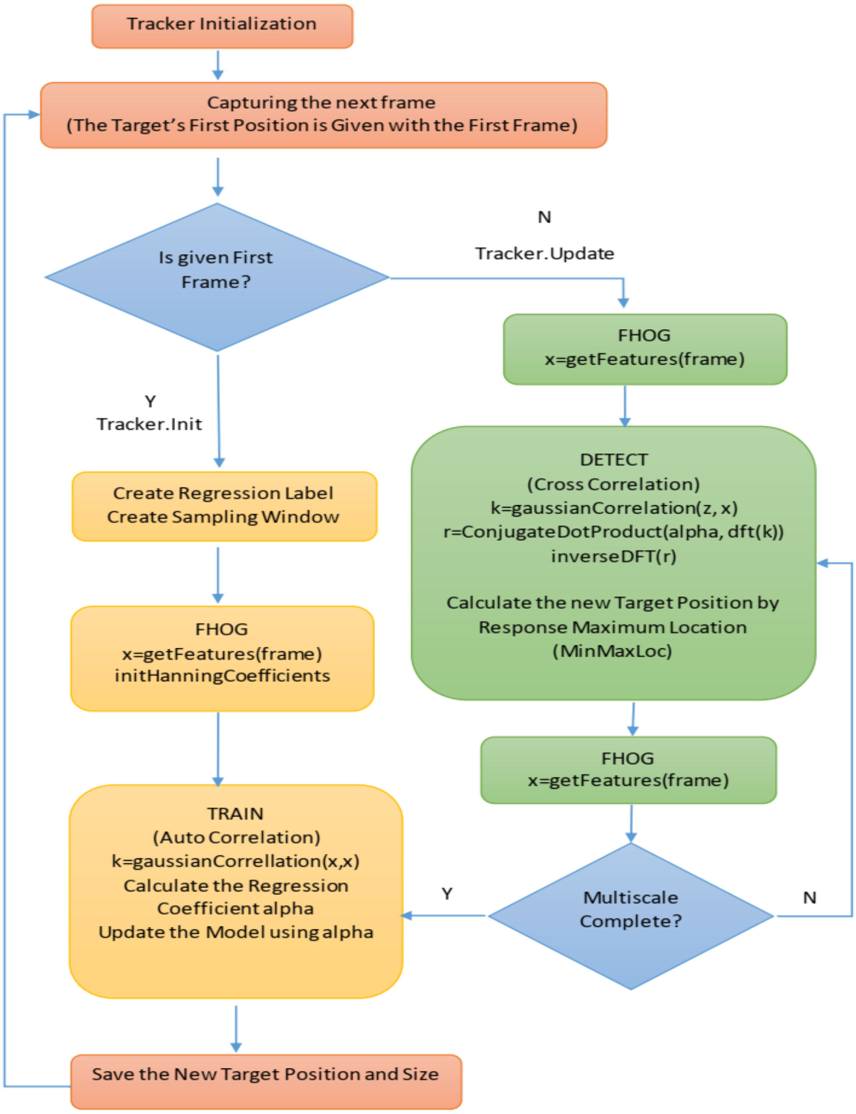


Figure 1 Flowchart of the KCF Algorithm (Multiscale Enabled)

3. Implementation

This section unfolds the implementation process for the KCF tracker accelerator using High-Level Synthesis techniques, which is divided into subsections, each focusing on a specific implementation aspect. Section 3.1 performs an analysis to identify components of the system that can be optimized and parallelized and investigates the impact of the FFT implementation on the accelerator's efficiency and discusses the target platform selection. Subsequently, Section 3.2 delves into the design of DFT and IDFT functions with software optimizations. Section 3.3 covers the basics of HW optimizations using HLS Directives. The structure and design considerations of the accelerator are outlined in Section 3.4.

3.1 Hardware/Software Profiling and Target Platform Selection

Hardware-software co-design in image processing offers a holistic approach to system optimization, combining the strengths of both hardware and software to achieve superior performance, energy efficiency, scalability, reduced latency, and efficient resource utilization. In the literature, several advancements and benefits are highlighted in hardware-software co-design approaches in computer vision. In [10], a co-design framework for video stabilization on FPGA processes real-time video streams at 28 fps, which is twice as fast as software-only approaches. Compared to full-software implementations, a hardware-software co-design significantly lowers overall computational time and hardware resource usage for a feature extraction and image matching algorithm [11]. In [12], a co-design implementation preserves the operating speed of neural networks while allowing flexibility in changing parameters without impacting the FPGA part. An embedded vision services framework in [13] uses heterogeneous hardware accelerators for rapid and efficient integration in applications like image stabilization and moving target indication. A co-design approach for intelligent camera applications significantly reduces development time and boosts performance in FPGA-based services [14]. In [15], a ZYNQ platform-based co-design approach enables high real-time binocular stereo vision, enhancing driving safety.

Based on the significant benefits of the hardware-software co-design approach in FPGA implementations, this paper explores a hardware-software co-design implementation of the KCF tracking algorithm, focusing on optimizing the discrete Fourier Transform (DFT) operation through custom hardware acceleration. The bottleneck of the Kernelized Correlation Filter (KCF) algorithm often lies in the computation of the Discrete Fourier Transform (DFT) and the Inverse DFT (IDFT) operations. These operations are integral to KCF's frequency domain correlation calculations and are performed repeatedly during both the training and detection phases. These operations have a complexity of $O(N^2)$, which can become a performance bottleneck when processing large input data. The co-design approach efficiently balances performance and resource utilization, offloading the most computationally intensive parts to hardware while handling other algorithm aspects in software. The co-design not only achieves significant speed improvements but also maintains accuracy with negligible precision loss, a critical factor for tracking applications. Furthermore, this methodology provides the flexibility needed for future optimizations and enhancements, demonstrating its adaptability to evolving requirements and potential improvements in real-time tracking technology.

In this work, an open-source software implementation of the KCF algorithm is used [6]. To assess the effect of DFT operations on the KCF performance, the latency of the OpenCV DFT function is quantified, along with an assessment of the frequency of calls and the proportion of total DFT latency per frame. Notably, in the case of multiscale mode-enabled KCF, the overhead associated with DFT and IDFT operations accounted for 75% of the total latency. These observations underscore the substantial impact of the DFT function on the KCF algorithm's performance. To mitigate this, a combined accelerator Intellectual Property (IP) block for DFT and IDFT functions, called CDFT, is used within the PL fabric of the FPGA. Consequently, the residual code operates on the PS side of the FPGA. In the context of multiscale KCF, each frame necessitates a total of 379 calls to the CDFT. The initial thought might be that managing 379 input-output transitions is a significant task. However, it is important to recognize that the small array sizes and inherent computational overhead characteristic of the DFT process help to achieve significant acceleration for the KCF algorithm. It is also worth noting that the DFT process primarily encompasses multiply-accumulate operations, rendering it conducive to parallelization and optimization.

FPGAs are known for their high computational speed, abundant resources, and portability. However, implementing intensive computations on FPGAs using hardware language is challenging. In this study, algorithms are automatically converted from the algorithmic layer to the register transfer layer using high-level synthesis, thereby reducing the disadvantages of the FPGA, and leveraging its advantages. The KCF algorithm, with its simple operational structure and high calculation repeatability, is suitable for FPGA implementation. However, certain complex structures requiring flexibility are better suited to be operated on the SoC. Therefore, Xilinx’s ZCU102 ZYNQ UltraScale + MPSoC multi-core heterogeneous system, which combines ARM and FPGA capabilities, is selected to meet real-time processing and image processing flexibility requirements. The computing tasks are divided between the PS, representing the software portion, and the PL, representing the hardware portion.

3.2 Custom Design of DFT and IDFT Functions Using Software Optimizations

The 2D DFT and the 2D Inverse DFT functions are seen in Equations 1 and 2, respectively.

$$\boldsymbol{F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}} \tag{1}$$

$$\boldsymbol{f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})}} \tag{2}$$

Where M x N is the image size, (x, y) gives the image pixel position, and (u, v) represents the spatial frequency - the rate of change of intensity values in the image. First, DFT and IDFT functions are implemented separately based on Equations 1 and 2 respectively. Then they are combined in the CDFT IP to achieve a broader optimization. In this work, two software optimization methods are utilized, namely the usage of lookup tables and a combination of DFT and IDFT operations. These optimizations are explained in the following sub-sections.

3.2.1 Usage of Lookup Tables

After evaluating the initial implementation, it is seen that direct implementations of Equation 1 and Equation 2 have high computational complexity. To reduce the computational complexity, Equation 1 and Equation 2 can be transformed from the polar form to the cartesian form, where sine and cosine functions can be replaced with lookup operations. Equation 3 is used to transform the exponential terms of DFT and IDFT functions to Cartesian form, where in this case θ equals to $2\pi(\frac{ux}{M}+\frac{vy}{N})$.

$$\boldsymbol{e^{j\theta} = cos\theta + jsin\theta} \tag{3}$$

After the transformation to the Cartesian form, sine and cosine estimations can be replaced by the lookup table operations. This is possible because the dimensions of images are fixed in the KCF algorithm and the possible values of x, y, u, and v in the 2D DFT and IDFT functions are also fixed. Therefore, pre-calculated values of sine and cosine functions can be placed in lookup tables and then these values are used in each iteration of the DFT and IDFT loops. Now the real and imaginary parts of the 2D DFT operations can be reduced to Equation 4 and Equation 5, respectively. The same optimization can also be applied to the 2D IDFT function.

$$\boldsymbol{F(u,v)_{re} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)_{re} * lut\mathbf{sin}(\mathrm{u,v,x,y}) + f(x,y)_{im} * lut\mathbf{cos}(\mathrm{u,v,x,y})} \tag{4}$$

$$\boldsymbol{F(u,v)_{im} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)_{re} * lut\mathbf{cos}(\mathrm{u,v,x,y}) + f(x,y)_{im} * lut\mathbf{sin}(\mathrm{u,v,x,y})} \tag{5}$$

In Equation 4 and Equation 5, the real part, cosθ, and the imaginary part, sinθ, are calculated for each x, y, u, and v value and they are stored in the lookup tables, lutcos[u][v][x][y] and lutsin[u][v][x][y], as const float data types. With this optimization, the calculations of sinθ and cosθ, which require 6 operations for θ and 2 operations each for Re{F(u,v)} and Im{F(u,v)}, are eliminated as these values are now retrieved from the lookup table. These operations are performed in nested loops with a total of u*v*x*y iterations during DFT calculation, so even the slightest simplification translates into significant computational savings. A minimum of 15 times speedup is obtained from the test results of DFT acceleration. Furthermore, functions like sine and cosine consume excessive resources, which causes synthesis failure in HLS. This optimization provides resource savings, enabling a successful synthesis. Thus, with this initial software optimization, both energy and logic resource consumption are reduced leading to an improved performance.

3.2.2 Combining the DFT and IDFT Functions

To save on resources and achieve further optimizations 2D DFT and IDFT functions are combined in the CDFT IP block and a parameter is added to select the operation type as DFT or IDFT. As the input image of the DFT function has only real terms, the terms containing the imaginary part of the input can be dropped from the calculations inside the loop. Dropping the redundant terms in calculations makes the algorithm much more efficient. If DFT has been directly implemented using readymade functions in HLS, as in other studies, this simplification would not be possible leading to inferior performance. Now, the real and imaginary portions of the 2D DFT function are simplified to Equation 6 and Equation 7, respectively.

$$F(u,v)_{re} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)_{re} * \mathbf{lutsin}(\mathrm{u},\mathrm{v},\mathrm{x},\mathrm{y}) \tag{6}$$

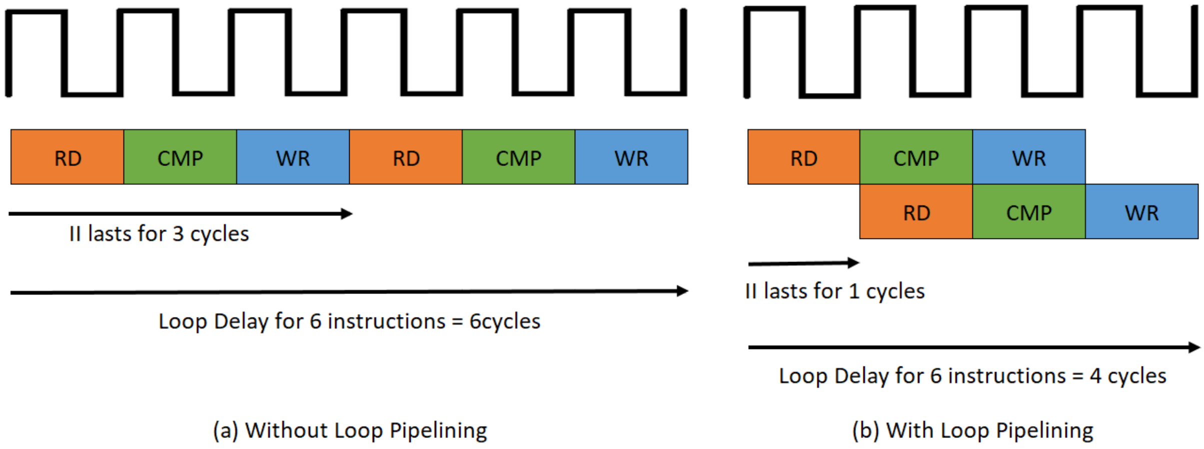
$$F(u,v)_{im} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)_{re} * \mathbf{lutcos}(\mathrm{u},\mathrm{v},\mathrm{x},\mathrm{y}) \tag{7}$$

As the input of the IDFT function is complex, the terms that contain imaginary parts of input values cannot be dropped in the calculation of the IDFT function. Although using a lookup table doubles the speedup value in DFT and IDFT operations, the effect of dropping the imaginary part in the DFT function reduces the number of calculations by 33.3% as an IDFT is called after every two consecutive DFT calculations in the KCF algorithm for the correlation calculation. It should be noted that these improvements are not at the level of the KCF algorithm, but rather obtained at the level of the DFT function.

3.3 HW Optimizations using HLS Directives

High-level synthesis (HLS) tools play a crucial role in converting C specifications into Register Transfer Level (RTL) designs. This integration of hardware and software disciplines offers several fundamental advantages. HLS tools enable hardware engineers to operate at an elevated level of abstraction, facilitating the development of efficient hardware. Simultaneously, they offer software engineers a new avenue to enhance the performance of their algorithms by targeting Field-Programmable Gate Arrays (FPGAs) for computational acceleration. However, HLS tools alone may not guarantee optimal task scheduling and resource allocation. Consequently, it is vital for designers to incorporate their expertise through specific optimization directives. By applying these directives, developers can effectively implement various optimization strategies, tailoring the HLS process to meet specific performance and efficiency goals.

In this work, the automatic utilization of ARRAY MAP and ARRAY PARTITION are utilized to enhance the efficiency of BRAM access and generate more opportunities for PIPELINE optimization. ARRAY PARTITION directive is employed to divide extensive arrays into several smaller arrays or separate registers. This aims to enhance data access and eliminate BRAM bottlenecks. ARRAY MAP is utilized to consolidate multiple smaller arrays into a larger one, which is subsequently directed to a single extensive memory resource such as RAM or FIFO. By default, HLS stores certain small arrays in the distributed RAM, which can lead to an overuse of the distributed RAM resources. In this particular design, the pre-calculated sine and cosine values are stored in a lookup table, which is partitioned into N-port ROMs, where the number of ports in the ROMS is determined based on the need for simultaneous read operations. All the other optimized storage structures are partitioned into true dual-port RAMs. This storage approach facilitates the simultaneous reading of multiple elements during subsequent optimization, thereby reducing the II (Initiation Interval) and minimizing latency. Note that the initiation interval measures the number of clock cycles required for a specific operation to begin execution once its inputs are available. PIPELINE directive is used to improve latency and maximize kernel throughput and performance. An illustration of II and pipelining optimization is given in Figure 2.


RD
CMP
WR
RD
CMP
WR
II lasts for 3 cycles
Loop Delay for 6 instructions = 6cycles
RD
CMP
WR
RD
CMP
WR
II lasts for 1 cycles
Loop Delay for 6 instructions = 4 cycles

(a) Without Loop Pipelining

(b) With Loop Pipelining

Figure 2 Pipeline Optimization

In addition to the above optimization directives, the CDFT block is realized using a fixed point to obtain a more efficient implementation. Initially, a floating point data type was used, which resulted in an initiation interval (II) of 6 with a latency of 3984 cycles. When the floating point data types are converted to the fixed-point, an II of 1 and a latency of 858 cycles is obtained. This corresponds to 4.6 times latency improvement compared to the floating point results at the IP block level. Table 1 summarizes the implementation results for floating and fixed-point implementation. One apparent conclusion is that fixed point implementation can utilize approximately 3.5 times more DSP blocks, which leads to lesser latency and reduced logic usage as given in FF and LUT columns.

Table 1 Synthesis Reports Results Before and After Optimizations

Module	II	Latency(cycles)	BRAM	DSP	FF	LUT
Floating	6	3983	485	803	187819	125724
Fixed	1	857	488	2852	72615	15460

Another optimization trial involved using dual-port BRAMs instead of single-port BRAMs. However, the synthesis process failed due to exceeding the maximum BRAM, FF, and LUT resource usage limit as a result of the increased complexity of implementing Dual Port BRAM usage. If another FPGA with more BRAM, LUT and FF resources are used, an additional 1.8 times improvement in latency could have been achieved, making the total latency improvement approximately 8.4 times compared to the floating point implementation at the block IP level.

3.4 Accelerator Structure

In this implementation, the KCF accelerator is composed of a 2D DFT and IDFT module located in the PL portion of the Zynq-based FPGA. This module includes dedicated Block RAM (BRAM) units for storing the input and output data for the required DFT and IDFT computations. An AXI interconnect is employed to facilitate seamless communication between these BRAM blocks and the PS. The system operates as follows: When the KCF algorithm calls for the execution of DFT or IDFT operations, instead of utilizing a pre-built GPU-accelerated function as found in OpenCV, the input data is routed to the custom accelerator. The processing operation is initiated by sending a start signal to the accelerator, which then begins its computations. The system subsequently enters a waiting state, awaiting the completion of the computation. Upon completion of the DFT or IDFT computations, the resulting outputs are read from the BRAM units. Writing to BRAM and reading from BRAM operations are conducted by the memcpy function on the PS side. The function of memcpy is only transferring data regions on the memory map. If it is possible, the memcpy function uses the CPU’s hardware features to optimize and accelerate the data transfer rates. These outputs are subsequently returned from the function, providing the desired results. In summary, the majority of the code execution occurs within the Processing System (PS) portion of the system.

Figure 3 illustrates the overall system architecture, highlighting various key components. One important point to note is that in this specific design, the GPU remains unused. The rationale behind this decision is to accomplish the same computational tasks with comparable performance and accuracy while minimizing power consumption. Nonetheless, the GPU is retained and made available for potential future use, such as for hybrid optimization methods or other computationally intensive operations required by the application.

4. Implementation Results and Performance Comparisons

In this section, the implementation results of the HLS-based accelerator design of the KCF algorithm using Xilinx's ZCU102 Ultrascale MPSoC platform are presented. First, the resource usage and latency of CDFT are compared with Xilinx FFT IP. Then, the performance comparison of the CDFT-based accelerated KCF algorithm is provided against a pure software implementation and a GPU-accelerated KCF implementation. Finally, the CDFT-based accelerated KCF algorithm is compared with various KCF implementations in the literature. To provide a comprehensive evaluation and comparison, several metrics to assess the performance and accuracy of the implementation are utilized. These metrics, including Center Location Error (CLE), Intersection over Union (IoU), and Frames Per Second (FPS), offer insights into the precision, overlap accuracy, and computational efficiency of the tracker, respectively. Each of these metrics is crucial in understanding both the strengths and potential areas of improvement for the KCF algorithm on the specified platform.

CLE metric evaluates the accuracy of the predicted bounding box center concerning the ground truth center. It is often reported as the Euclidean distance between the predicted center and the ground truth center. The smaller the value means the more accurate the result. Intersection over Union (IoU), also known as the Jaccard Index, measures the spatial overlap between the predicted bounding box and the ground truth bounding box. It is calculated as the ratio of the intersection area to the union area of the two bounding boxes. IoU values range from 0 to 1, where higher values indicate better tracking accuracy. It is formulated in Equation 8.

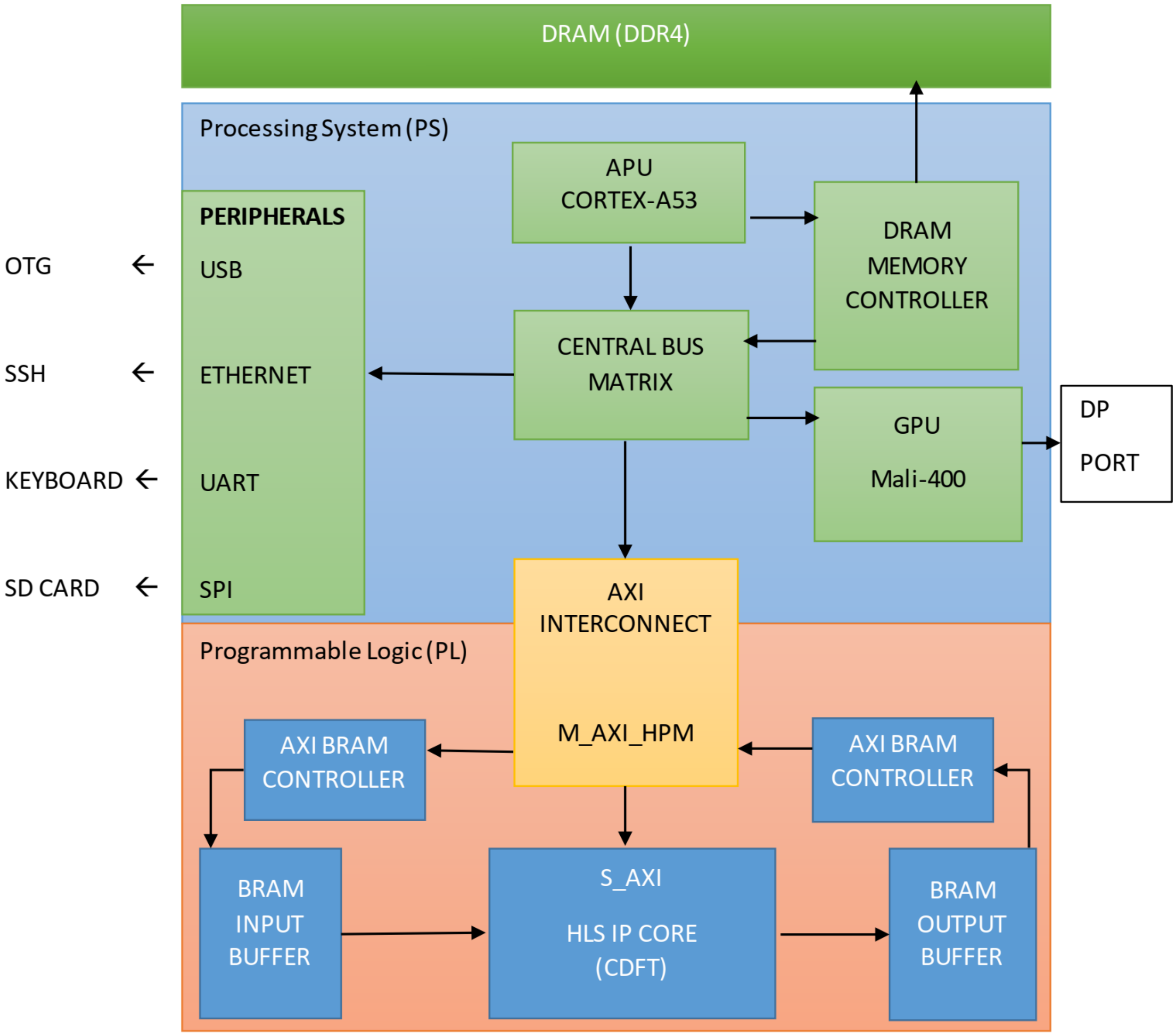


Figure 3 KCF Accelerator Design Architecture

$$\boldsymbol{IoU} = \frac{\boldsymbol{Intersection\ Area}}{\boldsymbol{Union\ Area}} \tag{8}$$

where the Intersection Area is the area of the overlapping region between the predicted bounding box and the ground truth bounding box. Union Area is the total area encompassed by both the predicted bounding box and the ground truth bounding box. The FPS speedup factor shows how much the KCF algorithm is accelerated. While not directly a tracking-specific metric, FPS represents the processing speed of the tracking algorithm. Faster algorithms are generally preferred as they can handle real-time applications. The Latency improvement factor shows how PL implementation of the accelerator IP finishes the processing earlier than its software version. This improvement will be evaluated by the speedup factor calculated using the FPS metric.

In this paper, the speedup comparisons are calculated according to Amdahl's Law [7]. To monitor the real performance of the designed accelerator, petalinux is customized by reserving specific memory and enabling necessary packages for a user application, which measures the aforementioned metrics. This makes it possible to evaluate the performance of the accelerator design in a real-world scenario.

First, the efficiency of the designed CDFT block is compared with the Xilinx FFT IP Core provided within the HLS library. In the Xilinx FFT core, a 2D Fast Fourier Transform (FFT) is implemented using the row-column algorithm and the 1D FFT

IP core. The first step involves performing a one-dimensional FFT in each row, while the second step requires performing a one-dimensional FFT in each column. For the second step, the results of the first step are transposed, and then a one-dimensional FFT is computed for each row again. To minimize resource usage, fixed-point data types are employed in FFT IP Core, where the 'radix_2_io' architecture is used with 64-bit complex data types for input and output. In CDFT IP, the same data width is used, but real and imaginary components are implemented as separate parameters. Zero-padding is applied to adapt the input size to the FFT length (32 samples), and spectral leakage prevention is addressed using Hanning window coefficients, which are stored in a constant lookup table. The implementation based on Xilinx FFT IP resulted in a frame rate of approximately 3 fps and significant accuracy loss, making tracking unfeasible. Table 2 compares the resource consumption and latency values obtained from synthesis reports. The custom 2D DFT IP in HLS achieves over a 24x reduction in latency while maintaining precision compared to the Xilinx FFT IP Core-based 2D FFT algorithm.

Table 2 Synthesis Reports Results Comparison Between Xilinx FFT IP Core and Our custom DFT IP

Module	Optimization	Latency(cycles)	BRAM	DSP	FF	LUT
Xilinx FFT IP core	Dataflow	21198	38	12	15844	15564
CDFT IP	Pipeline II=1	857	488	2852	72615	15460

Next, the performance of the designed KCF accelerator is evaluated against software-based and GPU-based KCF algorithms using the Bolt from the VOT2014 dataset [8]. The comparison is presented in Table 3.

Table 3 Performances from our tests using the Bolt dataset from VOT2014

	KCF-SW	KCF-GPU_accelerated	KCF-CDFT_floating_point	KCF-CDFT_fixed_point
FPS	3	22	10	18.5
IoU	0.961184	0.964186	0.949865	0.965927
CLE	4.6116	4.6105	4.6558	4.8077
Speedup	-	7.33x	3.33x	6.17x
Power Consumption	12W	15W	7W	10W

The measurements show that the fixed-point CDFT-based accelerator uses less power than both the single-core software (SW) version and the GPU-accelerated version. In contrast, it achieves 18.5 fps compared to 22 fps of GPU accelerated GPU. The original KCF source code utilizes the DFT function from the OpenCV library, which leverages GPU acceleration. Implementing the CDFT block on the PL section achieves a performance close to the on-chip GPU in the PS part.

Table 4 shows the latency reductions when using the hardware-supported memcpy function for input and output transfers in the fixed-point CDFT-based accelerator. For each frame in the KCF algorithm, our CDFT is called 252 times as a DFT function and 127 times as an IDFT function. The memcpy optimization provides a significant 4 times latency improvement, increasing the frame rate from 12 fps to 18.5 fps.

Table 4 Performances and latencies before and after memcpy function usage

	Before memcpy utilized	After memcpy utilized
DFT BRAM input Transfer Latency(ns)	1630	200
IDFT BRAM input Transfer Latency(ns)	2080	2690
DFT BRAM output Transfer Latency(ns)	115231	28122
KCF Total CDFT Transfer Latency Per Frame(ms)	44.35	11.05
KCF Speedup	4x	6.17x

In Table 5, a comparison with the previous studies in the literature is also presented. Note that except for the CDFT-based accelerator, all other designs use the HLS library to implement KCF-related functions in the PL section. As it is clear from Table 5, one advantage of the hardware-software co-design is the high accuracy. Although our study achieves a similar speed as the compared study [4], the accuracy is superior for several reasons. When a significant portion of the operations in the KCF algorithm is performed in the PL using a fixed-point implementation, a high precision loss occurs. These losses include the use of FFT (possible padding losses), the use of 6-dimensional HOG instead of 31 dimensions resulted in a reported 4% accuracy loss, implementation using fixed point data type instead of float for variable types in functions, and the use of bilinear interpolation for resizing. From this perspective, our work requires less effort, provides more flexibility to the KCF algorithm for further optimizations, and offers higher accuracy at the same speed compared to the study [4].

Table 5 Performance comparison of the CDFT-based accelerator with other implementations in the literature

Study Name	Hardware	Method	Resolution	FPS	Prec. Loss
Our design	CDFT	HLS	640x360	18.5	Low
Liu et al.[4]	Scale pyramid similar to DSST, HOG dimension reduction, Radix-2 FFT, Bilinear Interpolation	HLS*	1280x720	14-22	High
Cong et al.[5]	Fusing LBP with HOG, Dimensionality reduction method for LBP	HLS*	320x240	35	Low

*Almost All functions are from the HLS library.

In the study by Cong et al. [5], The resolution is significantly lower than the 640x360 resolution in our study. The image resolution has a big impact because transferring the entire image to the PL impacts the throughput negatively. Increasing the resolution on the same platform at the same fps is not possible because it would involve excessive resource consumption. To fit it, a significant compromise would have to be made in terms of accuracy and fps values. PIPELINE, radix-8 multiple FFT instances, and similar resource-hungry optimizations were required to achieve 35fps. It is clear that if implemented at the same resolution with the same number of resources, this study would yield lower fps and accuracy compared to ours.

5. Conclusions and Future Works

In contrast to the earlier FPGA implementation studies of the KCF algorithm, which port all the functions to the PL section, a hardware-software co-design approach is utilized on Xilinx’s ZCU-102 board. Only DFT and IDFT operations are performed on the PL side, resulting in a speedup of over 21 times for the IP and over 6 times at the algorithm level for KCF. By maintaining sufficient data width in the fixed-point version and implementing only the DFT function on the PL section, precision loss is kept negligible. Examining the experimental results, it is observed that a performance close to that of a GPU with lower power dissipation is achieved. While KCF is effective for tracking, for a full object tracking solution, it often works in tandem with a detection algorithm. The processing load of KCF is offloaded to the accelerator in the designed PL, freeing the GPU for the detection algorithm and enabling the meeting of real-time requirements if further optimizations are implemented. The factors influencing this speedup are discussed and compared to other works in the literature. In this approach, an advantage emerges from not transferring the entire image. It is found that, while other studies implement multiple functions to achieve the same performance as this IP, which implements only one function in the PL, this work provides higher accuracy at the same speed. In other studies, the implementation of multiple complex functions in the KCF algorithm, due to limited PL resources, prevents sufficient optimization and necessitates transferring the entire image to the PL side.

For future work, several further performance optimizations are possible, especially when using a target platform with more PL resources. The first optimization can be to use dual-port BRAMs instead of single-port BRAMs. Using dual-port BRAMs, PS-PL transfer latency can be halved, leading to better real-time performance. Additionally, if a higher clock frequency is used, real-time performance requirements can be met while preserving precision. Another improvement can be achieved using a parallel implementation of the Kernelized Gaussian Correlation. Each term in the Kernelized Gaussian Correlation estimation, can be calculated in parallel to improve processing time. The correlation calculation involves two separate DFT and one IDFT operation, which can be optimized using the extended version of the proposed implementation in this paper. In light of this data, implementing Gaussian Kernelized Correlation on a platform with higher PL resources can increase the number of DFTs processed per unit time and reduce the input-output transfer count, resulting in higher performance.

References

[1] J.F. Henriques, R. Caseiro, P. Martins, et al., "High-speed tracking with kernelized correlation filters", I*EEE Transactions on Pattern Analysis and Machine Intelligence*, 37,(3), pp. 583-596, 2015.

[2] M. Mueller, N. Smith, B. Ghanem, "A benchmark and a simulator for UAV tracking". *European Conf. Computer Vision*, pp. 445-461, Amsterdam, Netherlands, October 2016.

[3] H. Yang, J. Yu, S. Wang, X. Peng, et al. "Design of airborne target tracking accelerator based on KCF". J. Eng., 2019, Vol. 2019 Iss. 23, pp. 8966-8971. IET Journals doi: 10.1049/joe.2018.9159, 2019

[4] X. Liu, Z. Ma, M. Xie, J. Zhang, T. Feng, et al., "Design and implementation of scale adaptive Kernel correlation filtering algorithm based on HLS", *IEEE ICSPCC*. doi:10.1109/ICSPCC52875.2021.9564815, 2021

[5] P. Cong, M. Xie, K. Yang, X. Zhang, H. Su and X. Fu, "Design and implementation of multi-feature fusion kernel correlation filtering algorithm based on HLS," *IET International Radar Conference (IET IRC 2020),* Online Conference, 2020, pp. 645-649, doi: 10.1049/icp.2021.0761.

[6] J. Faro, "C++ KCF Tracker" 2021. [Online]. Available: https://github.com/joaofaro/KCFcpp ,[Accessed: 30.08.2023]

[7] G.M. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities" . *AFIPS Conference Proceedings* (30): pp. 483–485, doi:10.1145/1465482.1465560, 1967

[8] M. Kristan, J. Matas, A. Leonardis, and et al., "The visual object tracking vot2014 challenge results.", ECCV Workshop, 2014

[9] S.G. Raju, “Accessing BRAM in LINUX” 2021. [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842412/Accessing+BRAM+In+Linux, [Accessed: 01.04.2023]

[10] J. Hassan, M. Bilal, and S. Masud, A Hardware-Software co-design framework for real-time video stabilization. *J. Circuits Syst. Comput.*, 29, 2020. 2050027:1-2050027:18. doi:10.1142/S0218126620500279.

[11] C. Chien, C. Chien, and C. Hsu, “Hardware-software co-design of an image feature extraction and matching algorithm. *2019 2nd International Conference on Intelligent Autonomous Systems (ICoIAS)*, 37-41, 2019. https://doi.org/10.1109/ICoIAS.2019.00013.

[12] T. Dang, and Y. Hoshino, “Hardware/Software co-design for a neural network trained by Particle Swarm optimization algorithm”. *Neural Processing Letters*, 49, 481-505, 2019. doi: 10.1007/s11063-018-9826-4.

[13] E. Gudis, P. Lu, D. Berends, K. Kaighn, G. Wal, G. Buchanan, S. Chai, and M. Piacentino, “An embedded vision services framework for heterogeneous accelerators. *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 598-603, 2013. doi: 10.1109/CVPRW.2013.90.

[14] A. Ruta, R. Brzoza-Woch, and K. Zielinski, “On fast development of FPGA-based SOA services—machine vision case study”. *Design Automation for Embedded Systems*, 16, 45-69, 2012. https://doi.org/10.1007/s10617-012-9084-z.

[15] Y. Pan, M. Zhu, J. Luo, and Y. Qiu, “A Hardware/Software co-design approach for real-time Binocular Stereo Vision based on ZYNQ (Short Paper)”, 719-733, 2018. https://doi.org/10.1007/978-3-030-12981-1_50.

Conflict of Interest Notice

The authors declare that there is no conflict of interest regarding the publication of this paper.

Ethical Approval and Informed Consent

It is declared that during the preparation process of this study, scientific and ethical principles were followed, and all the studies benefited from are stated in the bibliography.

Availability of data and material

Not applicable.

Plagiarism Statement

This article has been scanned by iThenticate ™.