High-Performance FPGA Embedded System for Deep learning-based Thermal Object Tracking

Luu Nguyen , Tung Phan , Chien Thai , Huong Ninh , Hai Tran
Department of Computer Vision, Viettel Aerospace Institute, Viettel Group, Vietnam

2022 International Conference on Advanced Mechatronic Systems (ICAMechS) | 978-1-6654-6380-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICAMechS57222.2022.10003223

***Abstract*—Recently, Field programmable gate array (FPGA) has been adopted for accelerating the implementation of deep learning networks due to their ability to maximize parallelism as well as their energy efficiency. In this paper, we address the usage of convolution neural networks for single tracking in thermal infrared images (TIR) on FPGA embedded systems. Based on the previous tracking network named SiamFC++, we build a tracker for thermal infrared single object tracking problem by utilizing several TIR public datasets. Then we deploy and optimize SiamFC++-TIR tracker models on Xilinx FPGA platform with Vitis AI development environment. Experimental results show that our TIR tracker achieves outstanding performance in two public TIR tracking benchmarks. In addition, we construct a video dataset collected from our thermal infrared device, named QDT-TIR, with annotation of various thermal object classes, scenarios, and challenges. To the best of our knowledge, our tracker is the first TIR tracker that achieves a high operating speed of 102 FPS and low power consumption of 9.744 Watt on FPGA with a precision score of 0.681 on LSOTB-TIR benchmark.**

***Index Terms*—Thermal infrared images, deep learning, tracking, FPGA, Xilinx VitisAI**

I. Introduction

Object tracking is a challenging task in various computer vision applications such as camera surveillance, autonomous driving, and patrols. In general, tracking for RGB images is the most widely studied task in the community. RGB data is supposed to have more perceptual and clear texture information. Besides visual tracking, thermal object tracking also receives more attention recently because of the wide applications of thermal images. Several advantages of infrared sensor cameras are the ability to surveil in darkness, robustness to illumination changes, and shadow effects. Thermal infrared image typically has a lower resolution and a larger percentage of dead pixels compared to visual images. In addition, the noise characteristic of thermal sensors significantly degrades the quality of output images. These drawbacks make more challenges for tracking problems.

Object tracking has recently achieved a significant breakthrough using Convolution Neural Networks (CNN). Deep networks have resolved almost challenging of tracking problems and reported promising results in lots of benchmarks. The intensive representational ability of deep features is suitable for tracking tasks on thermal infrared video. Besides models-based CNN is constructed using a lightweight architecture which able to deploy on embedded or edge devices. One of the major disadvantages of deep learning-based applications is that it takes greater processing time. As a consequence, there are various high computation hardware architecture platforms, such as Field programmable gate array (FPGA) and Graphic Processing Unit (GPU) which have capabilities to implement CNN models. Compared to GPU, the main advantage of FPGA is that the whole system can be implemented on the same silicon, including the connections to the sensors which reduces latency. In addition, FPGA has significant low energy and memory resource. Thus, FPGA technology is an emerging hardware for embedded devices in recent years. There have been multiple tools or development environment which provides FPGA accelerator as the embodiment of a high level of hardware and software operation. As consequence, Xilinx provided Vitis AI and Vitis HLS is a development environment for image processing and AI applications on Xilinx FGPA platforms. Vitis HLS tools synthesize a C or C++ function for acceleration in the programmable logic device. Vitis AI supports frameworks such as Caffe, PyTorch, Tensorflow and the latest model capable of diverse deep learning tasks. Our contribution can be summarized in four-fold:

- Based on SiamFC++ network, we build a tracker for thermal infrared single object tracking task.
- We implement and optimize SiamFC++-TIR tracker models on Xilinx FPGA platform with Vitis AI.
- We construct a video dataset with annotation for a single TIR object tracking evaluation, named QDT-TIR, which has various object classes, scenarios and challenges.
- Our tracker achieves leading performance in two challenging TIR tracking benchmarks. To the best of our knowledge, our tracker is the first tracker that achieves a high operating speed of 102 FPS and low power consumption of 9.744 Watt on FPGA with a precision score of 0.681 on LSOTB-TIR benchmark.

This paper is organized as follows. Section II outlines related work on thermal object tracking methods and FPGA-based accelerator CNN deployment. Section III describes the overview of the background of our suggested strategy for thermal object tracking in FPGA hardware. Experimental results to demonstrate the benefits of our approach are presented in Section IV. Finally, we conclude with key remarks in Section V.

Luu Nguyen is with the Department of Computer Vision, Optoelectronics Center, Viettel Aerospace Institute, Viettel Group, Vietnam (corresponding author, email: luunp2@viettel.com.vn).
Tung Phan, Chien Thai, Huong Ninh and Hai Tran are with the Department of Computer Vision, Optoelectronics Center, Viettel Aerospace Institute, Viettel Group, Vietnam (email: tungpt37@viettel.com.vn, chientv13@viettel.com.vn, huongnt382@viettel.com.vn, haitt27@viettel.com.vn).

II. Related Work

In the following, we present an overview of the various research on thermal object tracking. Some thermal object tracking algorithms have been proposed for handling various challenges. Among them, Correlation Filter (CF) tracking method has become the most common because of its remarkable run-time performance and high accuracy. For example, David S Bolme et al [1] propose the first correlation filter-based tracking method named MOSSE tracker for visual object tracking. Several following kinds of research maintain the limited ability of MOSSE filter for robust visual tracking [2]–[5]. Influenced by the success of CF tracking, several studies are developed for thermal object tracking. For example, Gundogdu et al [6] investigated multiple correlation filters as base trackers and a novel ensemble method turned to TIR data (TBOOST). This method produces tracking decisions among the ensemble correlators trackers. Yu JieHe et al [7] designed an effective thermal tracking method as a detection framework with fused multiple features and a weighted correlation filter. Recently, CNN-based object tracking has achieved a significant performance. Inspired by the success of Convolution Neural Networks (CNNs) in visual tracking, there are several attempts to adapt neural networks to improve the performance of TIR trackers. DSST-TIR [8] investigates the impact of the convolution feature with CF for TIR tracking and shows that the deep features achieve better performance than the hand-crafted features. Liu et al [9]proposed an ensemble tracker with multi-layer convolution features to boost the infrared tracking accuracy (MCFTS). Gao et al [10] employ spatial regularization and interpolation to obtain continuous deep feature maps combination of deep appearance and motion features to represent thermal targets. Li et al [11] transfers automatically RGB data to synthetic TIR data to train an end-to-end neural network to archive significant results for thermal object tracking.

CNN represents the state-of-art ability in computer vision tasks [12], [13]. However, recent deep network models need extremely computationally expensive. The common solution is to accelerate the process such as GPU , and FPGA. The main drawback of GPU architecture is high energy consumption which is not suitable to design for embedded, edge devices or real-time applications [14]. FPGA has high parallel processing ability, portability, and low power consumption which supports well AI edge devices. Therefore, research on the optimization and acceleration of CNN on FPGA has grown increasingly prominent. For instance, Jin Wang et al [15] proposed YOLOv3 FPGA hardware accelerator based on the AXI bus ARM+FPGA architecture to detect the small target. Compared with GPU under the standard model, their results obtained lower throughput and energy consumption when the accuracy was reduced insignificantly. Akihiko Ushiroyama et al [16]utilized Vitis AI development environment of FPGA-Xilinx for target recognition. Their results confirmed that the Vitis AI-based-FPGA platform power consumption is 4.96 times less than that of a GPU. Michal Machura et al [17] evaluated three different CNN hardware accelerator implementations for object detection including LittleNet, FINN, and Vitis AI CNN. The experimental results indicated that Vitis AI provides consistent resource usage that is independent of the network depth. The obtained bitstream can also be utilized with a different network architecture.

III. Background

A. SiamFC++: Fully Convolutional Siamese Tracker

Siamese is the most famous method following a tracking-by-matching strategy. Bertinetto et al. [18] proposed a fully convolution Siamese network to estimate the feature similarity between the template from the first frame and a more prominent search image from the subsequent frames in the tracking sequence. After the SiamFC, many researchers yield a significant improvement in accuracy based on SiamNet such as SiamRPN [19] and SiamRPN++ [20], DaSiamRPN [21], SiamFC++ [22]. In this part, the detail of the SiamFC++ tracker network is described. As shown in Fig. 1, the SiamFC++ framework consists of two parts: the Siamese subnetwork and the Region proposal subnetwork (RPN). The Siamese network is used for feature extraction. It consists of two branches: the template branch and the search branch. Both branches use the same backbone network to perform the same transformation on their corresponding inputs. The outputs of the Siamese subnetwork are denoted as ψ_z and ψ_x. In the region proposal subnetwork, two convolutional layers are added for both ψ_z and ψ_x to adjust common features after performing a cross-correlation between the template patch and the search patch. The classification head and regression head are used after the cross-correlation. At the end of the classification head, a quality assessment branch is added to estimate the prior spatial score (PSS). The score used to choose the final box is computed by multiplying PSS with the classification score. The regression head outputs an extra offsets regression which is used to refine the locations of the predicted bounding box.

B. Xilinx Vitis AI

Xilinx Vitis AI is a development environment that is used to implement and accelerate AI applications on Xilinx FPGA platforms. It consists of optimized IP cores, tools, libraries, models, and example designs. Key components of Vitis AI used in this work are listed below:

a) Deep-Learning Processor Unit (DPU [23]): An optimized configurable IP core is used to accelerate DNN models on Xilinx FPGA platforms.

b) AI Quantizer: A quantizer that converts the 32-bit floating point weights and activations of AI models to the 8-bit fixed point format. This results in faster speed and higher power efficiency when implementing these networks on FPGA platforms.

c) AI Compiler: A compiler that maps the quantized model into a sequence of instructions based on the model structure and DPU architecture.

d) AI Runtime: A unified high-level runtime API is used for reading DPU instructions, loading the weight of the model, and exchanging data between DPU and CPU.

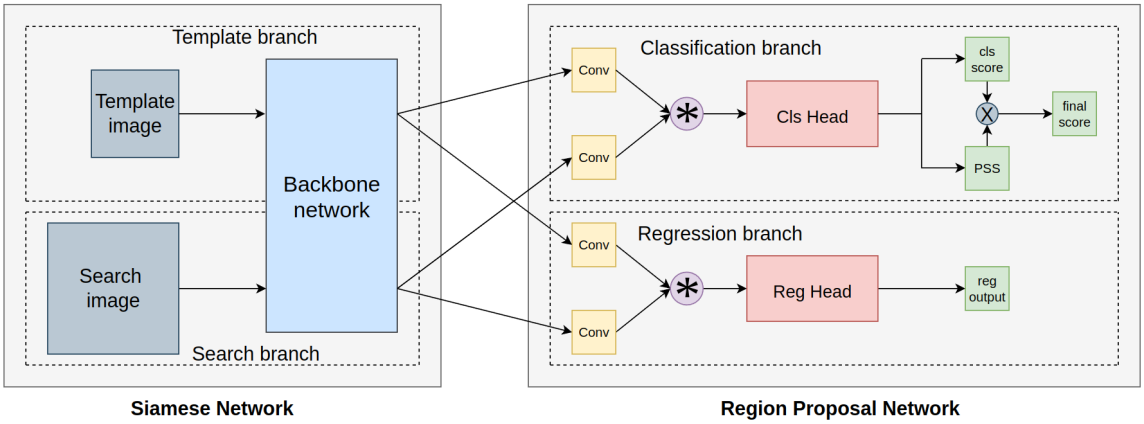

Fig. 1. SiamFC++ network architecture. In the figure, $*$ denotes cross-correlation and $\times$ denotes element-wise production.

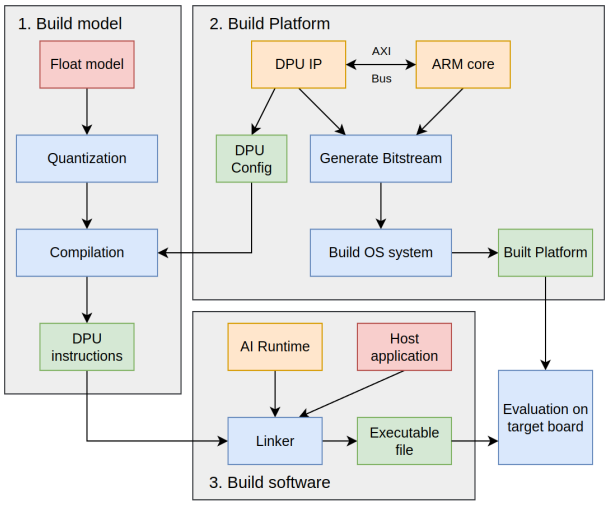

Fig. 2. Vitis AI design flow.

C. Vitis AI design flow

The design flow of implementing AI models on Xilinx FPGA devices is illustrated in Fig. 2. In the first step, the 32-bit floating point model after being trained is converted to an 8-bit fixed point format by AI Quantizer. The quantized model then is tested on the evaluation dataset to check the accuracy loss. Next step, the model and a DPU configuration file are used to generate the DPU instruction sequence by AI Compiler. In the second step, a custom hardware platform with an integrated DPU IP core is built. The DPU configurations can be changed appropriately according to the model structure and hardware resources on the target FPGA board. The ARM core that performs data transferring and task scheduling connects with the DPU through AXI buses. The design is converted into a bitstream using Vivado software [24]. An embedded Linux operating system is built on this customized platform by Petalinux [25] tool. The third step is building an application that can run on the built hardware platform. Vitis AI Runtime is used in this step to load and run the compiled model in the previous step. Finally, the accuracy and the speed of the application running on the target board are checked again.

TABLE I
STRUCTURE OF SIAMFC++-TIR WITH ALEXNET BACKBONE

Model part	Layer	Filter	Kernel/ Stride	Output	
				Template (127×127)	Search (287×287)
Back-bone	bb-conv1	48	11×11/2	59×59	139×139
	bb-pool1	-	3×3/2	29×29	69×69
	bb-conv2	128	5×5/1	25×25	65×65
	bb-pool2	-	3×3/2	12×12	32×32
	bb-conv3	192	3×3/1	10×10	30×30
	bb-conv4	192	3×3/1	8×8	28×28
	bb-conv5	128	3×3/1	6×6	26×26
Classi-fication branch	adj-conv1	256	3×3/1	4×4	
	adj-conv2	128	3×3/1	24×24	
	cross-corr1	-	-	21×21	
	cls-conv1	128	3×3/1	19×19	
	cls-conv2	128	3×3/1	17×17	
	cls-conv3	128	3×3/1	15×15	
	cls-out1	1	1×1/1	15×15	
	cls-out2	1	1×1/1	15×15	
Regre-ssion branch	adj-conv1	128	3×3/1	4×4	
	adj-conv	128	3×3/1	24×24	
	cross-corr2	-	-	21×21	
	reg-conv1	128	3×3/1	19×19	
	reg-conv2	128	3×3/1	17×17	
	reg-conv3	128	3×3/1	15×15	
	reg-out	4	1×1/1	15×15	

IV. EXPERIMENTS

A. Training model and Datasets

a) Model setting: We implement our tracker with two versions of the Alexnet backbone based on the previous work of Krizhevsky et al [26]. The first one which has the same size as the original backbone model is denoted as SiamFC++-TIR. The structure and the detail of each layer in this network are shown in Table I. In the second model, we reduce all the filter

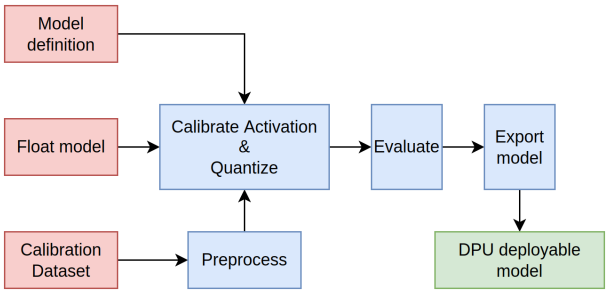

Fig. 3. AI Quantizer workflow.

numbers of each convolution layer by 2 (except for output layers). This smaller version is denoted as SiamFC++-TIR-S.

b) Training data: LSOTB-TIR [27] is used as our main thermal infrared training set. We also adopt RGB datasets such as ILSVRC-VID/DET [28] , COCO [29], YoutubeBB [30], LaSOT [31] and GOT-10k [32]. All RGB samples are converted to grayscale before training. We follow [22] to choose training samples in these datasets. Training with the negative pairs strategy [21] is also applied. To increase training data size, We perform data augmentation methods such as random shifting, scaling, pixel noise, brightness, and contrast on the search image.

c) Evaluation dataset and criteria: We test our tracker on 3 TIR evaluation sets: LSOTB-TIR [27], PTB-TIR [33] and QDT-TIR [1]. QDT-TIR is our evaluation dataset captured from our TIR camera. The dataset contains 47 sequences with 5941 frames which are annotated semi-automatically by using a pre-trained SiamFCpp Tracker. The dataset focus on 3 challenges: fast scaling, fast camera motion and object occlusion. To measure the overall performance of the tracker, we use One Pass Evaluation (OPE) method which computes the success rate and precision based on two widely used criteria in visual tracking: Center Location Error and Overlap Ratio metrics [34].

d) Training phase: We train our models from scratch with 40 epochs with 400k image pairs for each epoch. Stochastic gradient descent (SGD) with a momentum of 0.9 is chosen as our optimizer. We also adopt weight initialization with Gaussian distribution and warming up epoch training techniques of SiamFC++. Our models are trained on an NVIDIA TESLA V100 GPU.

B. Build Model

a) Model Quantization: The floating point model after being trained needs to be quantized into an 8-bit fixed-point format by AI Quantizer before implementing on FPGA platforms. In this work, Post training quantization [35] (PTQ) method is applied to our model. The detail of this quantization process is described in Fig. 3. The AI Quantizer requires 3 inputs: the trained model, the model definition script, and a calibration set. In our work, we use 1000 random image pairs from all training datasets to create the calibration set. All these images are preprocessed and sent to the quantizer. The quantizer then runs several iterations of inference to calibrate and analyze the distribution of activations before quantizing the weight, biases, and activations. The quantization process can lead to some accuracy drop, so we evaluate the quantized model on all the evaluation datasets. Finally, the model with little degradation in accuracy is exported to a DPU deployable format.

b) Model Compilation: After the quantization step, the model is compiled by AI Compiler to run on DPU integrated FPGA flatforms. The compiler takes the quantized model as input and transforms it into the Xilinx Intermediate Representation (XIR) format which is a computational graph-based format used for the compilation and deployment of AI algorithms on DPU. It then breaks up the graph into multiple subgraphs which are divided into two types. The first subgraph type contains layers supported by DPU and the second type is unsupported layers. In our model, only the cross-correlation layers can not be executed on DPU. These special layers perform convolutional operations on two output feature maps and are implemented on the core ARM of the FPGA chip. All the subgraphs then pass through several optimization steps such as node fusion, exploiting data reuse, and instruction scheduling. This whole process is an architecture-aware process and requires DPU information for compilation. The final product is an optimized graph that contains all necessary model parameters and DPU instruction sequences.

C. Build hardware platform

a) DPU configuration: DPU is a configurable computation engine. Its parameters can be changed and optimized according to the hardware resources capacity of the target board, model architecture, and our application. The most important DPU configurable parameter is the convolution architecture which decides the performance of the computing unit based on the peak number of operations per clock cycle. In this work, a single core DPU with the highest performance architecture is used. The DPU core runs at the clock frequency of 300 MHz and is integrated into the programable logic part of the FPGA chip.

b) DPU integrated system: The DPU integrated system on the target FPGA device is shown in Fig. 4. The system is divided into 2 parts: a processing system and programmable logic. The processing system part consists of the ARM core and the external memory. The programmable part contains the whole DPU core. After start-up, the input image is preprocessed on the CPU and put into the external memory. Input data, DPU instructions, and model parameters are stored on the external memory and are managed by the memory controller. These data then are sent to the DPU instruction scheduler and DPU on-chip memory through AXI buses. The DPU instruction unit performs instruction fetching, decoding, and dispatching. The on-chip memory is used to buffer input data, model parameters, instructions, and intermediate output data. The computation engine accesses necessary data from the internal memory through a memory controller and processes

[1]QDT-TIR evaluation dataset is available at https://drive.google.com/file/d/1uLbRogUmYqRrZ5A3gOPzgjDmqq384vHa/view?usp=sharing

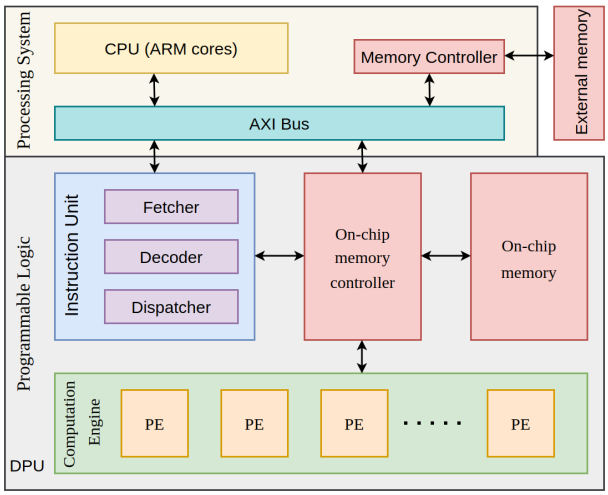

Fig. 4. DPU integrated system

on multiple processing elements (PE). The output data is sent back to the external memory for further post-processing tasks on the CPU.

TABLE II
HARDWARE RESOURCES AND POWER CONSUMPTION

	LUT	FFs	BRAM	DSP
Total Resources	230.4K	460.8K	312	1728
Design Resources	45.9K	95.7K	257	690
Utilization	19.93%	20.76%	82.37%	39.93%
Total Power	9.744W			

c) Build platform and Linux system: The process of building a DPU integrated hardware platform is performed on Vivado software and consists of 3 steps: Synthesis, Implementation, and Generating bitstream. In the synthesis step, the design is transformed into a netlist which is a gate-level representation of the design. This netlist then is placed and routed onto the FPGA chip in the implementation step. After this step, we check the hardware resource utilization and power consumption of the design. The detailed information of these parameters is shown in Table II. If all these parameters can meet our predefined requirements, the implemented design is transformed into a binary bitstream. This bitstream file carries the information on which logical elements on the FPGA chip are configured to implement the target design. A DPU configuration file which is used for the model compilation is generated automatically in this process. The next process is building a Linux operating system on the platform using the Petalinux tool. The DPU driver, necessary packages, and libraries for the application are installed in this step. The final products are a Linux boot image and a root file system that can be copied to an SD card and plugged into the target board.

D. Build Software

A Linux system runs in the PS part. The AI models are implemented in the DPU core built in the PL part, with an application on the Linux system for controlling the DPU, and processing the other algorithms in the PS part. Fig. 5 presents the operation flow of the application. The image input goes into preprocessing implemented in the PS part. Then part of the model is processed in the PL part. In the middle of the model, as we mentioned above, the cross-correlation layer is a special layer that is not supported by Vitis AI, meaning it can not run in the DPU core. We take the output of the previous layers from the PL part, and then execute this layer in the CPU. This layer is convolutional operations, so we apply multi-thread in order to reduce processing time in the PS part. If there are 256 filters, that means there are 256 convolutions, which corresponds to 256 threads. Subsequently, the rest of the model is performed in PL. Lastly, post-processing is done in the PS part, thereby outputting a result.

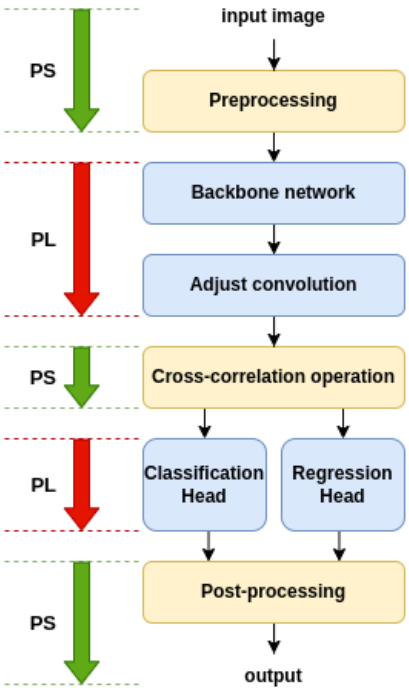

Fig. 5. Build software workflow.

E. Experimental Results

a) Experimental settings: Our tracker models SiamFC++-TIR and SiamFC++-TIR-S are evaluated on two hardware platforms. The floating-point models are tested on an intel core i7-9700 3.0GHZ CPU and the coresponding 8-bits compressed models run on ZCU104 FPGA board. We compare our tracker with other TIR Siamese-based tracker: SiamFC-TIR, CFNet-TIR and HSSNet-TIR from previous work [27]. All these trackers are test on an i7 4.0GHZ CPU and a GTX-1080 GPU.

b) Result on TIR tracking benchmarks: The results of all the trackers on 3 benchmarks LSOTB-TIR (evaluation set), PTB-TIR and our QDT-TIR are shown in III. Our model SiamFC++-TIR (CPU version) ranks 1st in most benchmark scores and outperforms other trackers which even run on a much more powerful GPU platform. The corresponding

Fig. 6. Experimental result snapshots of SiamFC++-TIR tracker on ZCU104. In each image, the number at the top left corner denotes the frame index

TABLE III
QUANTITATIVE RESULTS OF TRACKING MODELS ON LSOTB-TIR, PTB-TIR, AND QDT-TIR DATASETS. THE EVALUATION METRICS ARE PRECISION (PRE.), SUCCESS RATE (SUC.), AND FPS

Model	Plat-form	LSOTB-TIR		PTB-TIR		QDT-TIR		Speed
		Pre.	Suc.	Pre.	Suc.	Pre.	Suc.	(FPS)
SiamFC++-TIR (Ours)	i7 3GHZ CPU	0.709	0.618	0.790	0.627	0.935	0.778	24
	ZCU104	0.699	0.610	0.787	0.628	0.945	0.783	67
SiamFC++-TIR-S (Ours)	i7 3GHZ CPU	0.699	0.597	0.748	0.605	0.939	0.785	62
	ZCU104	0.681	0.582	0.753	0.605	0.925	0.771	102
SiamFC-TIR [27]	i7 4.0GHZ CPU + GTX 1080 GPU	0.700	0.554	0.758	0.566	-	-	45
CFNet-TIR [27]		0.580	0.478	0.726	0.530	-	-	24
HSSNet-TIR [27]		0.566	0.435	0.723	0.490	-	-	15

Note: Red, blue and green, represent 1st, 2nd and 3rd respectively.

compressed model deployed on ZCU104 shows an impressive result with little degradation in accuracy in all benchmarks while achieving real-time processing speed at 67 FPS (2.8x speedup compared to the CPU version). The small version of our tracker SiamFC++-TIR-S (CPU platform) ranks 3rd in LSOTB-TIR with only 0.001 lower precision than SiamFC-TIR at 2nd place. The compressed version of this model keeps a comparable accuracy and surpasses CFNet-TIR and HSSNet-TIR in all scores while reaching the leading processing speed at 102 FPS on a low power consumption FPGA platform. Fig. 6 visualizes some snapshots of SiamFC++-TIR tracking results on FPGA of these three benchmarks in three sequences: airplane_H_001 (LSOTB-TIR evaluation set), crossing (PTB-TIR), DaNang_ship4 (QDT-TIR). The first sequence focuses on the fast scaling challenge when an airplane takes off and comes close to the camera. The second sequence belongs to the occlusion challenge when two persons cross each other. The third sequence shows a deformation challenge when a boat rotates around itself. In all the sequences, the red bounding box of the target in the first frame is used for initialization and the tracker is never re-initialized in the tracking process.

V. CONCLUSION

In this paper, we design trackers for thermal infrared single object tracking task based on SiamFC++ network. Our trackers are compressed and optimized using Vitis AI development environment before implemented on a DNN accelerator integrated system on a FPGA platform. We perform experiments on two trackers models: SiamFC++-TIR and SiamFC++-TIR-S (small version) on three challenging thermal infrared track-

ing benchmarks. The experimental results show that the first model outperforms others trackers in accuracy while keeping a real-time speed at 67 FPS. The smaller model achieves the leading performance on processing speed at 102 FPS while keeping a comparable accuracy and low power consumption.

REFERENCES

[1] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE, 2010, pp. 2544–2550.

[2] M. Danelljan, G. Bhat, F. Shahbaz Khan, and M. Felsberg, "Eco: Efficient convolution operators for tracking," in *CVPR*, 2017.

[3] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," in *TPAMI*, 2014.

[4] D. Do, G. Vu, M. Bui, H. Ninh, and H. Tien, "Real-time long-term tracking with adaptive online searching model," 04 2020, pp. 62–68.

[5] Y. Li and J. Zhu, "A scale adaptive kernel correlation filter tracker with feature integration," in *European conference on computer vision*. Springer, 2014, pp. 254–265.

[6] E. Gundogdu, H. Ozkan, H. Seckin Demir, H. Ergezer, E. Akagunduz, and S. Kubilay Pakin, "Comparison of infrared and visible imagery for object tracking: Toward trackers with superior ir performance," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2015, pp. 1–9.

[7] Y.-J. He, M. Li, J. Zhang, and J.-P. Yao, "Infrared target tracking via weighted correlation filter," *Infrared Physics & Technology*, vol. 73, pp. 103–114, 2015.

[8] E. Gundogdu, A. Koc, B. Solmaz, R. I. Hammoud, and A. Aydin Alatan, "Evaluation of feature channels for correlation-filter-based visual object tracking in infrared spectrum," in *Proceedings of the IEEE Conference on Computer Vision and Pattern recognition Workshops*, 2016, pp. 24–32.

[9] Q. Liu, X. Lu, Z. He, C. Zhang, and W.-S. Chen, "Deep convolutional neural networks for thermal infrared object tracking," *Knowledge-Based Systems*, vol. 134, pp. 189–198, 2017.

[10] P. Gao, Y. Ma, K. Song, C. Li, F. Wang, and L. Xiao, "Large margin structured convolution operator for thermal infrared object tracking," in *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, 2018, pp. 2380–2385.

[11] L. Zhang, A. Gonzalez-Garcia, J. van de Weijer, M. Danelljan, and F. S. Khan, "Synthetic data generation for end-to-end thermal infrared tracking," *IEEE Transactions on Image Processing*, vol. 28, no. 4, pp. 1837–1850, 2019.

[12] X. Zhang, L. Zhang, and D. Li, "Transmission line abnormal target detection based on machine learning yolo v3," in *2019 International Conference on Advanced Mechatronic Systems (ICAMechS)*. IEEE, 2019, pp. 344–348.

[13] K. Han, M. Sun, X. Zhou, G. Zhang, H. Dang, and Z. Liu, "A new method in wheel hub surface defect detection: Object detection algorithm based on deep learning," in *2017 International Conference on Advanced Mechatronic Systems (ICAMechS)*. IEEE, 2017, pp. 335–338.

[14] P. Aguiar, S. Varrier, J. Lozoya, M. López, D. Koenig, and J. C. Tudon-Martinez, "Solving the stereo matching problem using an embedded gpu for a real-time application," in *2017 International Conference on Advanced Mechatronic Systems (ICAMechS)*, 2017, pp. 519–524.

[15] J. Wang and S. Gu, "Fpga implementation of object detection accelerator based on vitis-ai," in *2021 11th International Conference on Information Science and Technology (ICIST)*. IEEE, 2021, pp. 571–577.

[16] A. Ushiroyama, M. Watanabe, N. Watanabe, and A. Nagoya, "Convolutional neural network implementations using vitis ai," in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2022, pp. 0365–0371.

[17] M. Machura, M. Danilowicz, and T. Kryjak, "Embedded object detection with custom littlenet, finn and vitis ai dcnn accelerators," *Journal of Low Power Electronics and Applications*, vol. 12, no. 2, p. 30, 2022.

[18] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *European conference on computer vision*. Springer, 2016, pp. 850–865.

[19] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[20] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, and J. Yan, "Siamrpn++: Evolution of siamese visual tracking with very deep networks," *CoRR*, vol. abs/1812.11703, 2018. [Online]. Available: http://arxiv.org/abs/1812.11703

[21] Z. Zhu, Q. Wang, B. Li, W. Wu, J. Yan, and W. Hu, "Distractor-aware siamese networks for visual object tracking," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 101–117.

[22] Y. Xu, Z. Wang, Z. Li, Y. Yuan, and G. Yu, "Siamfc++: Towards robust and accurate visual tracking with target estimation guidelines," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020, pp. 12 549–12 556.

[23] M. LLC. (2021) Zynq dpu v3.3 product guide. [Online]. Available: https://docs.xilinx.com/r/3.3-English/pg338-dpu/Introduction?tocId=s1fbdzgCvidtCcYf54iW2g

[24] ——. (2021) Vivado design suite user guide. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2021_2/ug910-vivado-getting-started.pdf

[25] ——. (2021) Vivado design suite user guide. [Online]. Available: https://docs.xilinx.com/r/2021.2-English/ug1144-petalinux-tools-reference-guide/Revision-History

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks."

[27] Q. Liu, X. Li, Z. He, C. Li, J. Li, Z. Zhou, D. Yuan, J. Li, K. Yang, N. Fan *et al.*, "Lsotb-tir: A large-scale high-diversity thermal infrared object tracking benchmark," in *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, pp. 3847–3856.

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[29] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[30] E. Real, J. Shlens, S. Mazzocchi, X. Pan, and V. Vanhoucke, "Youtube-boundingboxes: A large high-precision human-annotated data set for object detection in video," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5296–5305.

[31] H. Fan, L. Lin, F. Yang, P. Chu, G. Deng, S. Yu, H. Bai, Y. Xu, C. Liao, and H. Ling, "Lasot: A high-quality benchmark for large-scale single object tracking," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 5374–5383.

[32] L. Huang, X. Zhao, and K. Huang, "Got-10k: A large high-diversity benchmark for generic object tracking in the wild," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, pp. 1562–1577, 2019.

[33] Q. Liu, Z. He, X. Li, and Y. Zheng, "Ptb-tir: A thermal infrared pedestrian tracking benchmark," *IEEE Transactions on Multimedia*, vol. 22, no. 3, pp. 666–675, 2019.

[34] Y. Wu, J. Lim, and M.-H. Yang, "Online object tracking: A benchmark," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 2411–2418.

[35] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1325–1334.