Runtime Hardware Security Verification Using Approximate Computing: A Case Study on Video Motion Detection

Mengmei Ye, Xianglong Feng, and Sheng Wei

Department of Electrical and Computer Engineering
Rutgers University, Piscataway, NJ, USA
Email: {mengmei.ye, xianglong.feng, sheng.wei}@rutgers.edu

***Abstract*—The heterogeneous CPU-FPGA system architecture has been adopted in system-on-chip (SoC), server, and cloud computing platforms to achieve design flexibility and hardware-level performance acceleration. While benefiting the system performance, the newly added FPGA component in the traditional CPU-based computing platforms could result in undetectable system security issues via third-party FPGA IP cores that are produced by untrusted vendors. Traditional hardware and/or software security verification mechanisms do not suffice to address the unique security and runtime performance challenges introduced by the new system architecture. In this paper, we develop a novel approximate computing-based approach to achieve a fast and accurate enough repeated execution for security verification. We implement and evaluate the approximate computing-based security verification framework by conducting a case study on a CPU-FPGA based video motion detection system, in which our experiments on Xilinx Zynq SoC justifies the premium security and low performance overhead obtained by the proposed approach.**

I. Introduction

The heterogeneous CPU-FPGA system architecture has been recently adopted in practice to achieve design flexibility, performance acceleration, and power optimization. For example, Intel Xeon CPU [1] and Xilinx Zynq system on chip (SoC) [2] adopt FPGAs as on-chip accelerators. Also, Amazon Web Services (AWS) [3] and Microsoft Azure [4] have deployed FPGAs to support accelerated cloud computing.

Although CPU-FPGA system is capable of achieving superior performance benefits, there exist many challenges that the developers and industrial practitioners must address to effectively adopt it into practice. From the development perspective, the software (i.e., CPU) part is straightforward to design, debug, and repair, since there are many mature software debugging or repair tools available [5]. However, developing error-free Intellectual Property (IP) cores in FPGA is a challenging task, and there are much fewer tools available to debug the IP cores. Under this circumstance, the system designers would heavily rely on third-party vendors to design and deliver the IP cores for system integration in the CPU-FPGA system.

978-1-7281-3544-1/19/$31.00 ©2019 IEEE

While the third-party IP (3PIP)-based model has been effectively adopted as a means to optimize the hardware production and integration flow, there exist inevitable security concerns given that the 3PIPs may be produced by untrusted vendors. While in the past decades many existing research efforts have been targeting the 3PIP security problem under the context of traditional IC supply chain [6][7][8], we observe that the 3PIP security problem would become even more challenging under the context of CPU-FPGA systems. For example, the fact that FPGAs are more and more widely deployed in the public cloud infrastructure significantly enlarges the traditional attack surfaces for potential hardware Trojan and other physical attacks. Also, FPGAs are designed to be flexible and capable of conducting dynamic partial reconfiguration [9][10], which the attacker could leverage to covertly embed and reconfigure hardware Trojans at runtime [11], pushing the requirement for security verification (e.g., hardware Trojan detection) from offline/static analysis to online/runtime monitoring.

In this paper, we aim to address the 3PIP security issue under the context of the CPU-FPGA system by developing an efficient runtime security verification mechanism. Although there have been several existing solutions aiming to verify the security of IP cores, they either rely on the availability of a golden chip (e.g., [12]), which is not feasible in reality, or conduct only offline/static verification (e.g., [13]) which does not fulfill the runtime security requirement. Our key insight in addressing the runtime hardware security challenge is inspired by the runtime redundancy-based verification [14][15] used in fault-tolerant system designs, where one or multiple repeated executions of the key algorithms or operations are conducted to approve or disapprove the potentially faulty primary execution. While the redundancy-based verification works effectively for fault-tolerant systems, there are two major challenges to apply it to the security domain. (1) *security challenge*: Under the security context the repeated executions must be conducted on a separate trusted component, other than the component under verification, as otherwise the repeated executions would also be contaminated with the attack and eventually result in false negatives in the security verification; and (2) *performance challenge*: The repeated executions may cause significant performance overhead and thus easily compromise the premium runtime performance benefit brought by the IP core.

We address both challenges by developing a CPU-FPGA based security verification framework that employs approximate computing. In particular, we address the *security challenge* by deploying the verification process (i.e., the repeated executions) in the CPU component, which is separate from the FPGA (i.e., where the IP core resides) and considered as trustworthy. Furthermore, to address the *performance challenge*, we employ approximate computing and develop an approximate software version of the IP core on the CPU for the runtime repeated executions. The key rationale behind this design is that the computation accuracy required by the repeated executions (i.e., for security verification) is orders of magnitude lower than that required by the IP core under verification. It is because a malicious security attack would typically dramatically change the results of the system, which can be effectively and efficiently captured by an inaccurate but fast approximate execution.

To this end, we develop and deploy the approximate computing-based verification framework to a CPU-FPGA prototype and conduct a comprehensive case study using a video motion detection application. The approximate computing algorithm employs two types of application-level approximations, namely *spatial approximation* and *temporal approximation*, to achieve the goals of runtime repeated execution and verification. Our empirical hardware evaluation on the Xilinx Zynq CPU-FPGA platform justifies the premium security and performance of the proposed approach.

II. Related Work

A. Security Verification Methods

Currently there are two main categories of techniques for security verification against untrusted software and hardware computations. The first category, namely verifiable computing, examines the correctness of the computation by requiring a proof associated with the computation results [16][17][18]. However, it is challenging to reduce the performance overhead between the prover (i.e., the party that conducts the computation) and the verifier (i.e., the party that verifies the correctness) in the verification protocol. The second category examines the information flow of the computation to ensure that the data and program execution are routed and executed in the expected manner [8][13][19]. The existing information flow tracking techniques are mostly static (i.e., at compilation time instead of runtime), which are not sufficient to defend against the hardware security threats discussed in this research.

B. Approximate Computing

Approximate computing has been leveraged to trade computation accuracy for performance acceleration and energy/resource savings [20]. In addition, there have been research works that employ approximate computing for authentication and information protection in embedded systems [21][22]. The approximation strategies include hardware-level approximation such as critical path reduction [23] and precision control [24], as well as software-level approximation such as loop perforation [20].

To the best of our knowledge, our work is the first use of approximate computing to verify 3PIP security in the CPU-FPGA architecture. Compared to the verifiable computing mechanisms, the IP cores as provers in the proposed verification process do not need to provide any additional protocol-based proofs. In addition, our method focusing on runtime results-based verification is orthogonal to the recent CPU-FPGA security work aiming to prevent the attacks using hardware and system level techniques [25][26].

III. Threat Model

Fig. 1 shows the threat model we consider in this work, where a hardware Trojan in the untrusted 3PIP manipulates the system functionality and sends falsified results to the CPU. We consider two types of hardware Trojans, namely the spatial Trojan (or *STrojan*) and the temporal Trojan (or *TTrojan*), which falsify the computation results in the spatial and temporal manners, respectively.

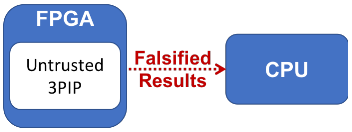


Fig. 1. Threat model considered in this paper: The untrusted 3PIP in the FPGA sends falsified results to the CPU.

Specifically, in the CPU-FPGA system, the CPU requests the FPGA to compute a certain number of jobs, where each job consists of multiple independent iterative tasks. The STrojan compromises a subset of tasks in every requested job, and the TTrojan compromises a subset of jobs completely (i.e., all the tasks in these targeted jobs are compromised). For example, in an encryption IP that encrypts messages, we assume each message is a job and each word in the message is a task. The IP with the STrojan could return the falsified encrypted messages with incorrectly encrypted words in each of them. The IP with the TTrojan could return the encrypted messages where all the words in a subset of the messages are incorrectly encrypted. Note that other possible attacks, such as side channel attacks [27][28], are out of the scope for this paper.

IV. Approximate-Computing Based Security Verification in CPU-FPGA System

To defend against the aforementioned threat model, we design a hardware security verification mechanism by using approximate computing (AC). In the proposed mechanism, we assume that the CPU is trusted and benign, but the FPGA is not trusted. Fig. 2 shows the CPU-FPGA system overview with the proposed framework. The AC verification mechanism residing in the CPU contains two stages, namely approximate computation and security verification. In the approximate computation stage, we propose two approximation strategies, namely *spatial approximation* and *temporal approximation*

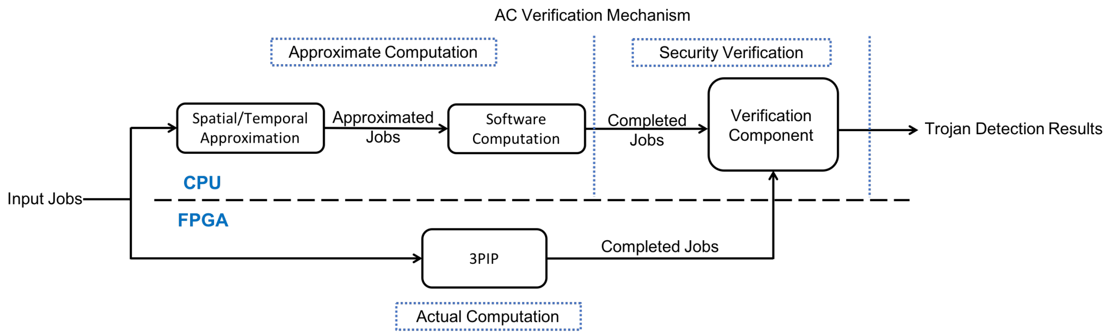


Fig. 2. CPU-FPGA system overview with the AC verification mechanism. The 3PIP in the FPGA conducts the actual computation. The AC verification mechanism in the CPU requires two stages to detect if the 3PIP contains Trojans, namely approximate computation and security verification.

to convert the input jobs to approximated jobs. Then, the software computation processes the approximated jobs and sends the completed jobs to the verification component in the security verification stage. In the meantime, the IP in the FPGA sends the completed jobs by the actual computation to the verification component in the CPU. Furthermore, the verification component compares the approximate results with the actual results and generates the Trojan detection results.

Spatial/Temporal Approximation. As shown in Fig. 3, there are m input jobs, where each job includes n tasks, i.e., $n*m$ tasks in total. The spatial approximation randomly selects x different tasks from each job and generates m approximated jobs, i.e., $x*m$ tasks in total. For the temporal approximation, y out of the m jobs are selected with each containing all the n tasks.

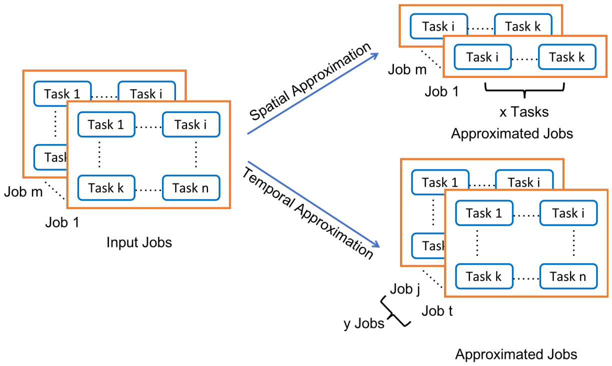


Fig. 3. Demonstration of spatial/temporal approximation. The spatial approximation randomly selects a subset of tasks from every input job. The temporal approximation randomly selects a subset of jobs from all the inputs jobs.

Security Verification. To detect whether the IP contains hardware Trojans, the verification component calculates the factor value f_i for each job as shown in Equation (1), where IP_i indicates the results of the completed ith job generated by the IP, and AC_i indicates the completed ith job generated by the software computation. The verification component checks the difference between the factor value f_i and the ideal factor value I shown in Equation (2). For the spatial approximation, n and x indicate the number of tasks in each input job and the number of tasks in each approximated job, respectively. For the temporal approximation, since the selected jobs have the same tasks in the corresponding input jobs, the ideal factor I is 1. If the difference between f_i and I is larger than a threshold, the verification component will report the IP is malicious. For the special case of $AC_i = 0$, we consider it as an invalid verification.

$$f_i = \frac{IP_i}{AC_i} \tag{1}$$

$$I = \begin{cases} \frac{n}{x}, & \text{spatial approximation} \\ 1, & \text{temporal approximation} \end{cases} \tag{2}$$

V. Case Study on Video Motion Detection CPU-FPGA System

Based on the proposed framework, we conduct a case study on a CPU-FPGA based video motion detection system. As shown in Fig. 4, the system involves three stages, namely video pipeline, video processing, and AC-based security verification. The implementation of the video pipeline stage is based on [30] and [31]. First, the system loads the video input via an HDMI-in module. The VDMA module (aka., video direct memory access) transfers the video into the input block of the DDR memory. In the video processing stage, the third-party video motion detection IP reads the input memory block and detects if the video contains any motions. Specifically, the IP identifies the background and the foreground in the input video frame, and conducts the pixel-by-pixel matching between the foreground and the background to calculate the pixel differences, which represents motions. After the video processing stage, the IP loads the processed video into the output block of the DDR memory. Then, the VDMA transfers the video output via the HDMI-out module.

The output monitor in Fig. 4 shows that the benign IP successfully detects the pedestrian (labelled with red rectangles) as motions (i.e., white pixels). In the STrojan scenario, the malicious IP removes a subset of detected motions to mislead the system into ignoring partial actual motions. In the TTrojan scenario, the malicious IP removes all the motions in certain video frames, equivalent to disabling the motion detection feature during the playback of those frames. To

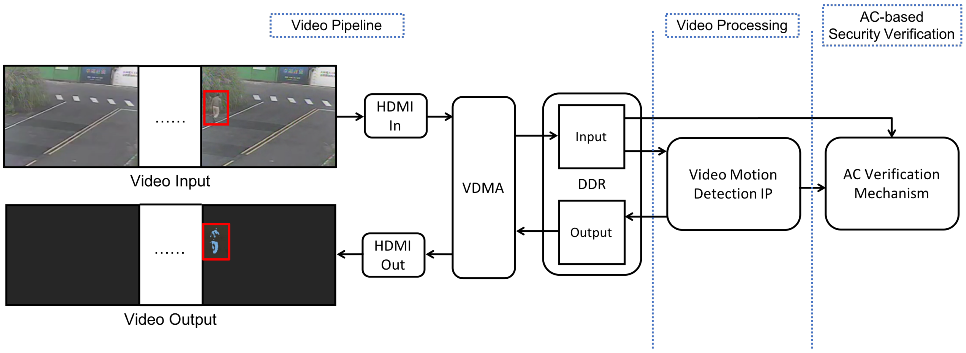


Fig. 4. The workflow of the video motion detection system (the video example is from [29]). The system requires three stages to complete the motion detection process, namely video pipeline, video processing, and AC-based security verification.

detect the Trojans at runtime, the AC verification mechanism is activated when the IP is generating the motion detection results. It obtains the number of motions generated by the IP and compares it with the approximate computing results to decide whether the IP is malicious.

We implement the video motion detection CPU-FPGA system on a Xilinx Zynq-7000 SoC ZC702 board that contains dual ARM processors and FPGA as shown in Fig. 5. In the FPGA, there are HDMI-in/out module (via a FPGA mezzanine card, namely FMC), VDMA, and video motion detection IP. The AC verification mechanism is executed by the ARM processors. The laptop provides the video input, and the PC monitor displays the video output with the detected motions.

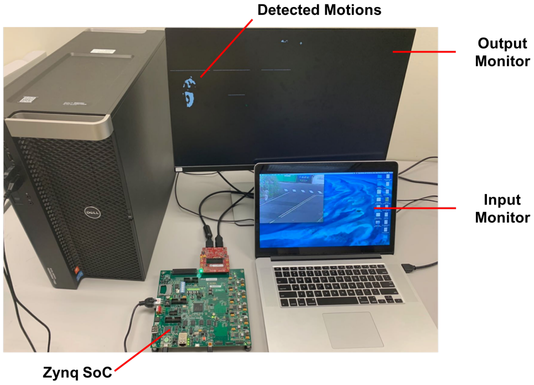


Fig. 5. Video motion detection system setup (the video example is from [29]). The input monitor provides the input video to the Zynq SoC for the motion detection process. Then, the output monitor displays the detected motions.

VI. Experimental Evaluation

We apply the proposed verification mechanism into the CPU component of the video motion detection system to defend against the STrojan and TTrojan embedded in the FPGA component of the system. We evaluate the effectiveness and efficiency of the proposed method on a video input with 400 frames from [29]. Specifically, we aim to answer the following questions.

1) *Effectiveness*: Can our AC verification mechanism distinguish between benign IPs and malicious IPs successfully?
2) *Efficiency*: Can our approximate computation achieve high performance at runtime?

A. Experimental Setup

In the FPGA, we develop the following IPs in the motion detection system. (1) *NoTrojan IP*: The IP is benign and conducts accurate computation. Note that it is only to evaluate if our AC verification mechanism identifies the NoTrojan IP as a benign IP correctly, and we do not rely on this IP to detect Trojans, i.e., no golden Trojan-free model is required in our verification approach. (2) *STrojan IP*: The IP hides a certain number of pixels in the output frames. In the experiments, it removes the upcoming motions after more than 5000 motions in the frame. There are 353 out of 400 frames that are compromised by the STrojan. (3) *TTrojan IP*: The IP hides all the motions in a certain number of frames. In the experiments, there are 142 frames compromised by the TTrojan (i.e., from the 30th frame to the 90th frame, and from the 180th frame to the 260th frame).

In the CPU, the AC verification mechanism treats each frame as a job and each pixel in a frame as a task. There are two approximation strategies. (1) Spatial approximation (*SAC*), which processes one pixel in every x pixels (represented as SAC_x); and (2) temporal approximation (*TAC*), which processes accurate computation for one frame in every y frames (represented as TAC_y). The motion detection algorithm adopted by the software is the same as that implemented in the FPGA IP. For security verification, the verification component calculates the factor value for each frame and decides whether the IP is benign or malicious.

B. Effectiveness Evaluation

We adopt the following four metrics to quantify the effectiveness of the proposed approach. Note that our AC verification mechanism only identifies if the IP is malicious and does not identify which Trojan is embedded in the IP.

- True positive (TP): the AC verification mechanism successfully identifies the malicious IP.
- True negative (TN): the AC verification mechanism successfully identifies the benign IP.

- False positive (FP): the AC verification mechanism misidentifies the benign IP as the malicious IP.
- False negative (FN): the AC verification mechanism misidentifies the malicious IP as the benign IP.

We first evaluate whether the AC verification mechanism detects the NoTrojan IP correctly by calculating its accuracy based on Equation (3), which measures the occurrence rate of all the TN outcomes, since the TP and FN do not exist in the NoTrojan IP. TABLE I indicates the accuracy when we apply SAC_36/TAC_20 with different thresholds. The threshold is a percentage of the ideal factor value to distinguish between the benign IP and the malicious IP. In the experiments, we observe that the AC verification mechanism never misidentifies the NoTrojan IP as the malicious IP unless the threshold is set as 100% for SAC. The threshold 100% indicates that the mechanism does not tolerate any difference generated from the approximate computation and the actual computation, which is why the accuracy decreases under this threshold.

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{3}$$

TABLE I
ACCURACY FOR NOTROJAN IP DETECTION BY SAC_36 AND TAC_20.

Threshold	25%	50%	75%	100%
SAC Accuracy	1	1	1	0.97
TAC Accuracy	1	1	1	1

Furthermore, we evaluate whether the AC verification mechanism is able to detect the Trojans in the IP. Since the TN and FP do not exist in the IPs with Trojans, we calculate the *precision* and *recall* to comprehensively show the two aspects of the accuracy as shown in Equation (4). (1) *precision*: In all of the frames where the AC verification mechanism identifies the Trojan as active, how many of them are truly compromised by the Trojan; and (2) *recall*: In all of the frames that the Trojan is truly active, how many of them are correctly identified by the AC verification mechanism.

$$\begin{aligned} precision &= \frac{TP}{TP+FP} \\ recall &= \frac{TP}{TP+FN} \end{aligned} \tag{4}$$

TABLE II
PRECISION AND RECALL FOR STROJAN IP DETECTION BY SAC_36 AND TAC_20.

Threshold		25%	50%	75%	100%
SAC	Precision	1	1	1	1
	Recall	0.87	0.92	0.94	0.95
TAC	Precision	1	1	1	1
	Recall	0.94	1	1	1

TABLE II shows the precision and recall results for SAC_36/TAC_20 with different thresholds when they defend against STrojan. We observe that, when the AC verification mechanism identifies an IP as malicious, the IP is always truly embedded with Trojans (i.e., the precision is always 1). However, with different thresholds the AC verification mechanism ignores 5% to 13% cases when the Trojan is active. We argue that increasing the threshold is helpful to increase the detection sensitivity and reduce the errors. TABLE III shows the precision and recall for SAC_36/TAC_20 with different thresholds against TTrojan. We observe that the TAC perfectly identifies the IP with TTrojan (i.e., the precision and recall are always 1s). For the SAC, there are 4% error rates for the recall, because the SAC misidentifies the malicious IP as a benign IP when the hidden motions are relatively small. Also, increasing the threshold to 100% compromises the precision of SAC for the same reason as in the NoTrojan experiments.

TABLE III
PRECISION AND RECALL FOR TTROJAN IP DETECTION BY SAC_36 AND TAC_20.

Threshold		25%	50%	75%	100%
SAC	Precision	1	1	1	0.95
	Recall	0.96	0.96	0.96	0.96
TAC	Precision	1	1	1	1
	Recall	1	1	1	1

$$F_1\text{-score} = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{5}$$

After the experiments of SAC_36/TAC_20 with different thresholds, we fix the threshold as 85% that performs well with the high accuracy in all the aforementioned experiments. Then, we change the parameters of SAC/TAC to discuss how the parameters affect the detection results. In the experiments, we notice that the parameters only affect the results for SAC against STrojan/TTrojan. To illustrate the precision and recall at the same time, we calculate the F_1-score [32] as shown in Equation (5). Fig. 6 shows the F_1-score results of SAC with different parameter values against the STrojan and TTrojan. We observe that higher SAC parameter values increase the computation error and reduce the F_1-score. Also, the decreasing trend is approximately linear.

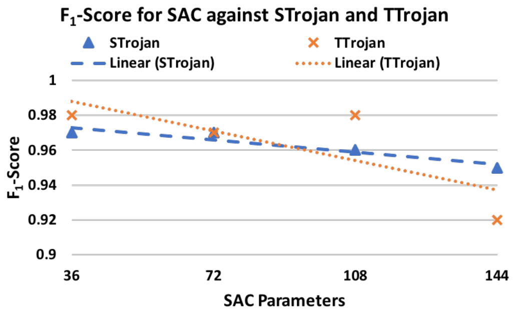
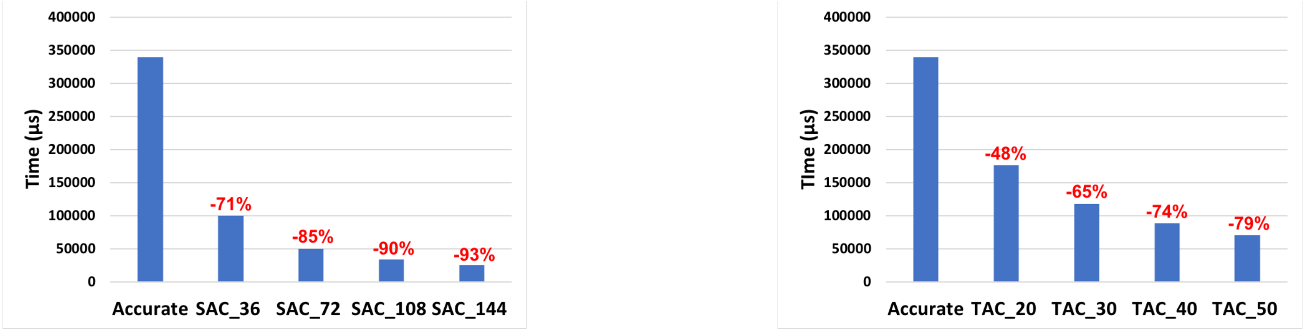


Fig. 6. F_1-score comparison for SAC with different parameters (i.e., SAC_36, SAC_72, SAC_108, and SAC_144).



(a) Time Comparison between Accurate and SAC Computations

(b) Time Comparison between Accurate and TAC Computations

Fig. 7. Time comparison between the accurate computation and the approximate computation conducted by the SAC/TAC with different parameters.

C. Efficiency Evaluation

Furthermore, we evaluate the efficiency of the SAC and TAC. Fig. 7 shows the time consumption of the approximate computation stage, since it is the primary overhead in the proposed mechanism. In Fig. 7(a), we notice that the SAC consumes at least 71% less time compared to the accurate computation conducted by the benign IP in the FPGA. Also, as shown in Fig. 7(b), the TAC consumes at least 48% less time than the accurate computation conducted by the IP. The results indicate that the SAC and TAC are able to verify the security with low performance overhead.

VII. Conclusion

In this paper, we developed a novel approximate computing based hardware security verification mechanism by leveraging spatial and temporal approximation strategies to verify 3PIP security in the CPU-FPGA system. We deployed a video motion detection system in Zynq SoC as a case study and evaluated the efficiency and effectiveness of the proposed mechanism. In the future, we will leverage our mechanism to a diverse set of applications, such as security verification for neural networks to defend against adversarial attacks.

Acknowledgement

We appreciate the constructive reviews provided by the anonymous reviewers. This work was supported in part by the National Science Foundation under Award CNS-1912593.

References

[1] "Intel FPGA acceleration hub for Intel Xeon CPU with FPGAs," www.intel.com/content/www/us/en/programmable/solutions/acceleration-hub.

[2] "Xilinx Zynq SoCs with hardware and software programmability," www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[3] "Amazon EC2 F1 instances," aws.amazon.com/ec2/instance-types/f1/.

[4] "Microsoft Azure cloud computing platform & services," azure.microsoft.com.

[5] L. Gazzola *et al.*, "Automatic software repair: A survey," in *IEEE Transactions on Software Engineering*, 2019, pp. 34–67.

[6] M. Tehranipoor *et al.*, "A survey of hardware trojan taxonomy and detection," in *IEEE Design Test of Computers*, 2010, pp. 10–25.

[7] T. Huffmire *et al.*, "Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems," in *S&P*, 2007, pp. 281–295.

[8] W. Hu *et al.*, "Property specific information flow analysis for hardware security verification," in *ICCAD*, 2018.

[9] E. L. Horta *et al.*, "Dynamic hardware plugins in an FPGA with partial run-time reconfiguration," in *DAC*, 2002, pp. 343–348.

[10] T. El-Ghazawi *et al.*, "The promise of high-performance reconfigurable computing," in *Computer*, 2008, pp. 69–76.

[11] A. P. Johnson *et al.*, "Fault attack on AES via hardware Trojan insertion by dynamic partial reconfiguration of FPGA over ethernet," in *WESS*, 2014.

[12] X. Wang *et al.*, "Hardware Trojan detection and isolation using current integration and localized current analysis," in *DFT*, 2008, pp. 87–95.

[13] X. Li *et al.*, "Caisson: A hardware description language for secure information flow," in *PLDI*, 2011, pp. 109–120.

[14] P. M. Frank, "Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy," in *Automatica*, 1990, pp. 459–474.

[15] W. Hong *et al.*, "Applying observer based FDI techniques to detect faults in dynamic and bounded stochastic distributions," in *International Journal of Control*, 2000, pp. 1424–1436.

[16] M. Walfish *et al.*, "Verifying computations without reexecuting them," in *Communications of the ACM*, 2015, pp. 74–84.

[17] B. Parno *et al.*, "Pinocchio: Nearly practical verifiable computation," in *S & P*, 2013, pp. 238–252.

[18] R. S. Wahby *et al.*, "Full accounting for verifiable outsourcing," in *CCS*, 2017, pp. 2071–2086.

[19] J. Shin *et al.*, "A hardware-based technique for efficient implicit information flow tracking," in *ICCAD*, 2016, pp. 1–7.

[20] S. Mittal, "A survey of techniques for approximate computing," in *ACM Computing Surveys*, 2016, pp. 62:1–62:33.

[21] M. Gao *et al.*, "Approximate computing for low power and security in the internet of things," in *Computer*, 2017, pp. 27–34.

[22] W. Liu *et al.*, "Approximate computing and its application to hardware security," in *Cyber-Physical Systems Security*, 2018, pp. 43–67.

[23] S. Hashemi *et al.*, "Drum: A dynamic range unbiased multiplier for approximate applications," in *ICCAD*, 2015, pp. 418–425.

[24] V. K. Chippa *et al.*, "Scalable effort hardware design," in *IEEE Transactions on VLSI Systems*, 2014, pp. 2004–2016.

[25] M. Ye *et al.*, "HISA: Hardware isolation-based secure architecture for CPU-FPGA embedded systems," in *ICCAD*, 2018, pp. 1–8.

[26] A. Coughlin *et al.*, "Breaking the trust dependence on third party processes for reconfigurable secure hardware," in *FPGA*, 2019, pp. 282–291.

[27] F. Schellenberg *et al.*, "An inside job: Remote power analysis attacks on FPGAs," in *DATE*, 2018, pp. 1111–1116.

[28] M. Zhao *et al.*, "FPGA-based remote power side-channel attacks," in *S & P*, 2018, pp. 229–244.

[29] "Video sample," www.youtube.com/watch?v=Bnk6dMQFNa8&t=3s.

[30] J. Sykora, "Motion detection in VGA video (640x480) on ZC702 (Zynq FPGA)," www.youtube.com/watch?v=LT2mGCnVoYA.

[31] X. Feng *et al.*, "Towards the security of motion detection-based video surveillance on IoT devices," in *Thematic Workshops of ACM Multimedia*, 2017.

[32] C. J. V. Rijsbergen, "Information retrieval," in *Butterworth-Heinemann*, 1979.