IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 27, 2021, accepted July 12, 2021, date of publication July 19, 2021, date of current version July 27, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3098004

Hardware Architecture Proposal for TEDA Algorithm to Data Streaming Anomaly Detection

LUCILEIDE M. D. DA SILVA[1,2], MARIA G. F. COUTINHO[1], CARLOS E. B. SANTOS, JR.[1], MAILSON R. SANTOS[3], M. DOLORES RUIZ[4], LUIZ AFFONSO GUEDES[3], AND MARCELO A. C. FERNANDES[1,3]

[1]Laboratory of Machine Learning and Intelligent Instrumentation, Federal University of Rio Grande do Norte, Natal 59078-970, Brazil
[2]Federal Institute of Education, Science and Technology of Rio Grande do Norte, Santa Cruz, State of Rio Grande do Norte 59200-000, Brazil
[3]Department of Computer Engineering and Automation, Federal University of Rio Grande do Norte, Natal, State of Rio Grande do Norte 59078-970, Brazil
[4]Department of Statistics and Operations Research, University of Granada, 18071 Granada, Spain

Corresponding author: Marcelo A. C. Fernandes (mfernandes@dca.ufrn.br)

This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) under Finance Code 001.

ABSTRACT The amount of data in real-time, such as time series and streaming data, available today continues to grow. Being able to analyze this data the moment it arrives can bring an immense added value. However, it also requires a lot of computational effort and new acceleration techniques. As a possible solution to this problem, this paper proposes a hardware architecture for Typicality and Eccentricity Data Analytic (TEDA) algorithm implemented on Field Programmable Gate Arrays (FPGA) for use in data streaming anomaly detection. TEDA is based on a new approach to outlier detection in the data stream context. The suggested design has a full parallel input of N elements and a 3-stage pipelined architecture to reduce the critical path and thus optimize the throughput. In order to validate the proposals, results of the occupation and throughput of the proposed hardware are presented. The design reached a speed of up to 693x, compared to other software platforms, with a throughput of up to 10.96 MSPs (Mega Sample Per second), using a small portion of the target FPGA resources. Besides, the bit accurate simulation results are also presented. This work is a pioneer in the hardware implementation of the TEDA technique in FPGA. The project aims to Xilinx Virtex-6 xc6vlx240t-1ff1156 as the target FPGA.

INDEX TERMS Data streaming, FPGA, anomaly detection, TEDA.

I. INTRODUCTION

Outlier detection or anomaly detection consists of detecting rare events in a data set. When data is captured and processed continuously in an online way they are considered as data streams [1]. Due to the increasing number of sensors in the most diverse areas and applications, streaming data is currently generated from many different sources and there is a huge rise in the availability of time-series data. It is a central problem in many application areas such as time-series forecasting, medical systems, industrial process monitoring, telecommunications, sensors networks, internet traffic and others [2], [3]. These systems provide users with real-time information and continuously seek to extract knowledge from structures of unified analysis from within massive data flows. Time-series anomaly detection helps to monitor the different metrics and parameters of industrial and corporate applications and services in real-time. It supervises the time-series continuously and sends alerts for probable risky events related to incidents instantly [4]. Consequently, outlier detection of data streams is a prominent research area in data mining [5], as well as an important task in various industrial applications.

Industry 4.0 projects are an area where anomaly detection has been increasingly applied to [3]. One of the challenges of Industry 4.0 is the detection of production failures and defects [6]. New technologies aim to add value and increase process productivity, but face difficulties in performing complex and massive-scale computing tasks due to the large amount of data generated [7]. Many solutions presented in the literature require a knowledge of full data set processes and the systems for modelling and making a series of initial assumptions. These, in most cases, are not applied [8], hence, traditional anomaly detection techniques may not be an applicable approach to detect anomalies in real-time streaming data series [9].

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

Typicality and Eccentricity Data Analytic (TEDA) algorithm has been presented as a possible solution for anomaly detection with streaming algorithms [10]–[12]. It is based on a new approach of outlier detection in a data stream context [13] and, for example, it can be applied with an algorithm to detect autonomous behavior in an industrial process operation. TEDA can be used as an alternative to statistical framework for analyzing most data. It is based on new metrics, all based on the similarity/proximity of data in the data space, not in density or entropy, as per traditional methods.

Real-time analysis stream data is critically dependent on computational infrastructure. The huge accumulation of real-time data to flow in a network, for example, can quickly overload traditional computing systems due to the large amount of data that originates from sensors and a requirement for intensive processing and high performance. Therefore, software-only approaches cannot keep up with the growing computational demands of real-time analysis, given the barriers to reducing latency in large volumes. In addition, as the number of nodes increases to handle the ever-growing amounts of data, performance is not scaled linearly [14]–[16].

The ability to process massive data flows from different sources at high speed is a critical problem. Some important aspects need to be considered when choosing an anomaly detection method, such as the computational effort required to handle large streaming data. Since they are dynamic, unknown and unlimited, the received information needs to be stored and analyzed without compromising memory and run-time. Handling this type of data requires three fundamentals, which are high-throughput, ultra-low-latency, and low-power [14]–[19].

Thinking about the challenges presented, this work proposes a specialized hardware architecture of TEDA in reconfigurable computing using FPGAs for Real-Time Anomaly Detection of Streaming Data. The development of the hardware technique allows systems to be made even faster than their software counterparts. This extends the possibilities of use especially during situations where time constraints are at their most severe. Reconfigurable computing is an emerging area, and it provides the possibility of developing hardware architectures customized to the algorithm. The main objective is to fit hardware to the algorithm, and it is not to fit the algorithm to hardware, unlike the traditional model where the algorithm fits the instructions of the processor. In the traditional model, complex algorithms with high degrees of parallelization are inhibited in terms of computational performance, as they often have to shape themselves sequentially to the target hardware [20], [21]. The development of a more specialized hardware architecture has been presented as an exciting alternative for overcoming bottlenecks, making it possible to create solutions for mass data processing and, at the same time, consider ultra-low-latency, low-power, high-throughput, security and ultra-reliable conditions [22], [23], all-important requirements for increasing productivity and quality in industry 4.0 processes.

One of the motivations for this work is the possibility of accelerating the TEDA algorithm for time-constrained applications, massive data, and energy efficiency by using a hardware-based implementation [24], [25]. All validation and synthesis results were made using an FPGA Virtex 6 xc6vlx240t-1ff1156. The FPGA choice was because it has high performance and flexible architecture.

This paper is organized as follows: This first section is presented as an introduction to the work, explaining the motivation behind it and major contributions. Section II discusses some related works and the state of the art. Section III presents a theoretical foundation regarding the TEDA technique. Section IV presents the implementation description details for the architecture proposed. Section V presents the validation and synthesis results of the proposed hardware, as well as comparisons with software implementations. Finally, Section VI presents the conclusions regarding the obtained results.

II. RELATED WORK

Real-time anomaly detection in data streams has the potential to be applied to many areas such as; preventive maintenance, fault detection, fraud detection and signals monitoring. These are concepts that can be used in different ranges of the industry, among others these can include technology, finance, medicine, security, energy, e-commerce, agriculture and social media.

In the literature, there are some uses of the TEDA technique for anomaly detection and even for classification. The article presented in [26] shows a proposal for a new TEDA-based anomaly detection algorithm. The proposed method compares the accumulated proximity information of all samples against previous specific point pairs suspected of being anomalies. The method/technique uses local spatial distribution information about the vicinity of the suspect point. In the journal, TEDA is compared to an approach using traditional statistical methods, emphasizing that the set of initial assumptions is different. TEDA is a generalization of traditional statistics, although TEDA does not need the initial assumptions. Besides, it has been shown that due to the recursion feature, TEDA is computationally more efficient and suitable for online and real-time applications. Other works in the literature also present similar results and conclusions [8], [10], [12], [27], [28]. The characteristics and advantages of the TEDA algorithm presented in these references have justified and motivated the choice of this algorithm, among several anomaly detection algorithms, for the implementation of the hardware architecture for streaming processes.

Another motivation for using the Algorithm TEDA in hardware implementation is the possibility of paralleling it. In the work shown in [11], a new algorithm is presented with a parallel structure within big data processing classification. The main feature of the proposed algorithm, called TEDAClassBDp, is the processing of block data, where each block uses the TEDAClass so that all blocks operate in parallel. As with TEDAClass, the proposed algorithm does

not require information from previous data, and its operation occurs recursively, online and in real-time. The parallel structure presented is very suitable for FPGA, its implementation in hardware would improve the performance and efficiency of the system in terms of latency, throughput and energy consumption. This enables the use of the proposal in response to problems where these restrictions may be limiting the processing of large volumes of data.

As the amount of data that needs to be processed grows exponentially, and autonomous systems become increasingly important and necessary, the implementation of machine learning, fault and anomaly detection and streaming algorithms on hardware have been studied in the literature. In work [29], an implementation of target and anomaly detection algorithms for real-time hyper-spectral imaging was proposed on FPGA. The algorithms were implemented in streaming fashion, similar to this work. The results obtained from a Kintex-7 FPGA using fixed-point structure were satisfactory for their application. However, they present an occupied area much larger than our proposal, using twice as many LUTs, almost 70 times more registers and 10 times the number of multipliers. The outcome also shows a very low throughput, with a processing time in the range of seconds.

The work [30] presented a study on the impact of Neural Network architectures compared to statistical methods which are used in the implementation of an Electrocardiogram (ECG) anomaly detection algorithm on FPGA. The fixed-point implementation contributes to reducing the number of required resources, however the design was created with High-Level Synthesis (HLS), which, perchance, was not able to optimize the FPGA resource consumption, when regarding higher resource consumption compared with manual optimization. In this work, only occupation and accuracy results are analyzed, no throughput or time analyzes are presented.

The implementation presented in [31] describes an accelerated FPGA architecture for anomaly detection based upon a Neural Network in pipeline. The design has a 200MS/s throughput and a 105ns latency likewise used in Radio Frequency Signals. The work presents a high throughput and ultra-low latency for inference, considerably faster than other similar implementations. However because the proposed hardware uses an NN-based algorithm, it needs a training stage, which is hosted in a PC (heterogeneous architecture) and this considerably limits the systems adaptability. As presented, the throughput and latency of the architecture are 1290ns (or 775KS/s) and 1285ns respectively, with both their NN-weights and biases to be updated. This makes the true throughput of the system, taking into consideration its adaptability, almost 10 times lower than ours. Even though all processing is done using 16-bit-fixed point with truncated rounding. Fixed-point with low resolution allows the possibility of keeping everything on-chip although with considerable impact on the weight system accuracy.

Concerning the TEDA algorithm, to date, no literature or studies have been aimed at exploring the implementation of its hardware on FPGA. This paper proposes to accomplish this feat in a pioneering manner.

III. BACKGROUND

TEDA was introduced by [32] as a statistical framework, influenced by recursive density estimation algorithms. Unlike algorithms that use data density as a measure of similarity, TEDA uses concepts of typicity and eccentricity to infer whether a given sample is normal or abnormal to the dataset.

The typicality of TEDA is the similarity of a given data sample to the rest of the dataset samples to which it belongs. Eccentricity, on the other hand, is the opposite of typicality, which indicates how much a sample is dissociated from other samples within its set. Therefore an outlier can be defined as a sample with high eccentricity and low typicality, when taking into consideration the established threshold for comparison. It is important to note that for eccentricity and typicality calculations no parameter or threshold is required.

To calculate the eccentricity of each sample, TEDA utilizes the sum of the geometric distances between the analyzed sample $\boldsymbol{x}_k$ and the other samples in the set. Thus, the greater the value, the greater the eccentricity of the sample, and consequently, the lower its typicality. [26] proposed recursively calculating the eccentricity. The eccentricity, ξ can be expressed as

$$\xi_k(x) = \frac{1}{k} + \frac{(\boldsymbol{\mu}_k^x - \boldsymbol{x}_k)^T(\boldsymbol{\mu}_k^x - \boldsymbol{x}_k)}{k[\sigma^2]_k^x}, \quad [\sigma^2]_k^x > 0 \tag{1}$$

where k is the discretization instant; $\boldsymbol{x}_k$ is an input set of N elements in the k-th iteration, $\boldsymbol{x}_k = [x_k^1\ x_k^2\ \ldots\ x_k^N]$; $\boldsymbol{\mu}_x^k$ is also an N elements vector, equal to the average of $\boldsymbol{x}_k$ at the k-th iteration and $[\sigma^2]_k^x$ is the variance of $\boldsymbol{x}_k$ at the k-th iteration. The calculation of $\boldsymbol{\mu}_x^k$ and $[\sigma^2]_k^x$ are also done recursively, using the following equation

$$\boldsymbol{\mu}_k^x = \frac{(k-1)}{k}\boldsymbol{\mu}_{k-1}^x + \frac{1}{k}\boldsymbol{x}_k, \quad k \geq 1,\ \boldsymbol{\mu}_0^x = 0 \tag{2}$$

and

$$[\sigma^2]_k^x = \frac{(k-1)}{k}[\sigma^2]_{k-1}^x + \frac{1}{k}\left\|\boldsymbol{x}_k - \boldsymbol{\mu}_k\right\|^2, \quad k \geq 1,\ [\sigma^2]_0^x = 0. \tag{3}$$

The typicality of a given sample $\boldsymbol{x}_k$, at the k-th iteration, can be expressed as a complement to eccentricity, as follows [26]

$$\tau_k(x) = 1 - \xi_k(x). \tag{4}$$

Besides, [26] also defined that normalized eccentricity can be calculated as

$$\zeta_k(x) = \frac{\xi_k(x)}{2}, \quad \sum_{i=1}^{k} \xi_k(x) = 1,\ k \geq 2. \tag{5}$$

To separate normal state data from abnormal state data, it is necessary to define a comparison threshold. For anomaly detection, the use of the $m\sigma$ [33] threshold is widespread, however this principle must first assume the distributional

behaviour of the analyzed data, such as the Gaussian distribution [26]. Chebyshev inequality can be used for any data distribution, assuming that the probability that the data samples are more than $m\sigma$ from the average is less than or equal to $1/m^2$, where σ is the standard deviation of the data [34].

The condition that produces the selfsame results as Chebyshev's inequality, discarding any assumptions about data and its independence, can be expressed as

$$\zeta_k > \frac{m^2+1}{2k}, \quad m > 0 \tag{6}$$

where m corresponds to the comparison threshold [26].

For the hardware implementation, we have used the TEDA algorithm as presented in Algorithm 1 in a 3-stage pipelined architecture, based upon the equations presented above.

Algorithm 1 TEDA

Input: $\mathbf{x}_k$: k-th sample; m: threshold
Output: outlier: sample classification as abnormal or normal

```
1:  begin
2:      while receive x do
3:          if k=1 then
4:              μ_k^x ← x_k;
5:              [σ^2]_k^x ← 0;
6:          else
7:              update μ_k^x using equation 2;
8:              update [σ^2]_k^x using equation 3;
9:              update ξ_k(x) using equation 1;
10:             update ζ_k(x) using equation 5;
11:             if ζ_k(x) > (m^2+1)/2k then
12:                 outlier ← true;
13:             else
14:                 outlier ← false;
15:         k ← k + 1;
```

The inputs of the Algorithm 1 are the data samples, $\boldsymbol{x}_k$, and a comparison threshold, m. The output for each entry, $\boldsymbol{x}_k$, is the indication of the sample's classification as abnormal (outlier = true) or normal (outlier = false).

IV. PROPOSED ARCHITECTURE

In this work, a TEDA FPGA proposal was implemented using Register Transfer Level (RTL) such as the works presented in [25], [35]–[38]. A design overview can be seen in Figure 1. It has four different block structures: The MEAN module, the VARIANCE module, the ECCENTRICITY module and the OUTLIER module. The characteristics of the proposal and modules description will be presented and discussed in the following section.

A. ARCHITECTURE PROPOSAL OVERVIEW

The architecture was developed to pipeline the operations presented in Algorithm 1. The MEAN module implements the average described in Equation 2 and it is one clock cycle ahead of the VARIANCE module, which is responsible for calculating the variance as presented in the equation 3; The ECCENTRICITY module calculates the eccentricity as presented in the equation 1 and it is one clock cycle delayed to the VARIANCE module and two to the MEAN module; The OUTLIER module is used to normalize the eccentricity as in Equation 5 and to compare with the threshold, as shown in Equation 6 and it operates at the same clock cycle as ECCENTRICITY. Each module is detailed later in the following section.

3-stage pipelined architecture reduces the critical path and thus optimizes the throughput. The critical path is reduced through the placement of pipeline registers between the modules. The placements of the registers are, 1) at the output of MEAN module, N registers illustrated as MREGn, and also at the input of the VARIANCE module, N registers illustrated as VREG1n, where N is the size of the input vector $\mathbf{x}_k$, 2) at the output of VARIANCE module, one register represented as VREG1, and at the input of ECCENTRICITY module, two registers illustrated as EREG3, EREG4. A total of three clock cycles are required to classify the input signal in outlier or non-outlier. The first cycle is required to calculate the mean, μ_k^n, (Equation 2), the second one to calculate the variance, $[\sigma^2]_k^x$,(Equation 3) whereas the third cycle is required to calculate the eccentricity, (Equation 1) and ultimately classify the sample in outlier or not-outlier. it causes a 3 clock cycle delay, however, it decreases the critical path and consequently improves throughput.

The implementation has the Algorithm 1 as reference. The system receives the FPGA clock and the k-th sample vector $\mathbf{x}_k$ as inputs. The k-th iteration number is updated from the increment of a counter and the m threshold is used as a constant, stored in the OUTLIER module. As in the algorithm line 7, the MEAN module calculates each single element average of $\mathbf{x}_k$ vector. It is possible to observe that there are N MEAN blocks, where N is the vector size. This block will be detailed in section IV-B. After this step, moving to the next line (8), the calculation of variance is done in the VARIANCE Module, this block is detailed in the section IV-C. The ECCENTRICITY block has, as inputs, the signals that left the block VARIANCE and k, referenced at line 9 and detailed in subsection IV-D. The OUTLIER block is detailed in subsection IV-E. It receives the ECCENTRICITY, $\xi_k(x)$, and calculates the normalized eccentricity to compare against the threshold as presented in lines 10, 11 and 12.

B. MODULE I - MEAN

Each n-th MEAN module computes the average of each individual n-th element vector $\boldsymbol{x}_k$ acquired at run time. The implementation is based on Equation 2 and it is detailed in Figure 2. In addition to receiving the n-th element of vector $\boldsymbol{x}_k$ as an input, the MEAN block uses a counter to define the number of sample interactions, k. The implementation uses a comparator block identified as in the Figure 2 as MCOMPn

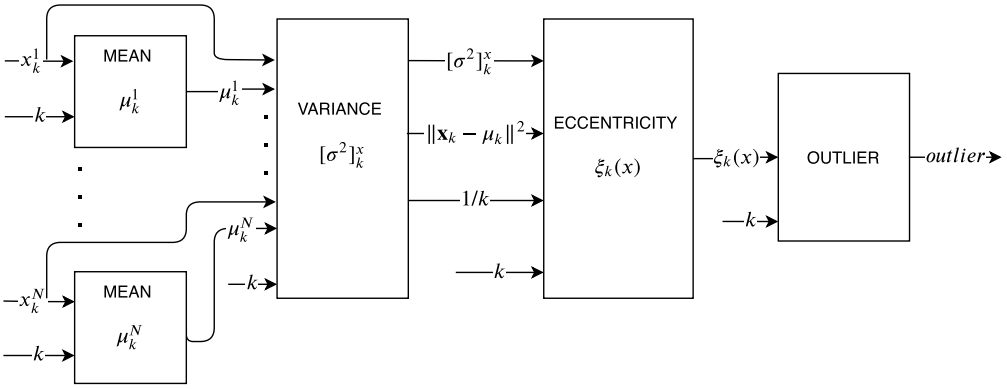


FIGURE 1. General architecture overview.

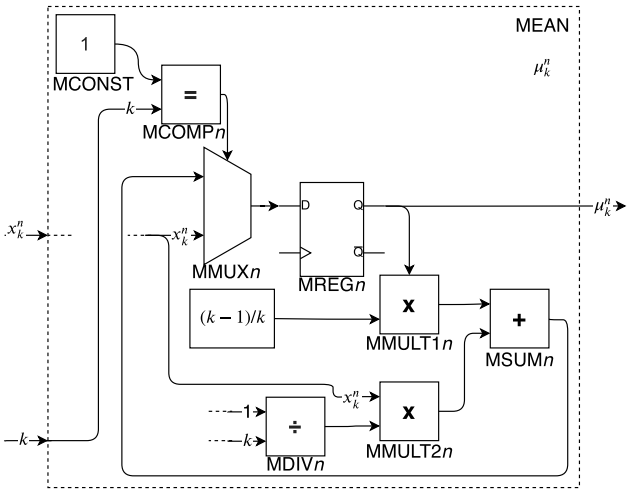


FIGURE 2. MEAN module.

which is used to verify if the system is in the first iteration as Line 3 of Algorithm 1. The MMUXn is a multiplexer that acts as a conditional evaluation, using as a selecting value, the output of MCOMPn comparator. The register MREGn stores the vector n-th $\boldsymbol{\mu}_k^x$ elements (μ_k^n). The μ_k^n value stored in MREGn is multiplied with $\frac{k-1}{k}$ in MMULT1n and added in MSUMn with the output of MMULT2n that has as input x_k^n and the inverse value of k. Each n-th element of vector $\boldsymbol{x}_k$, x_k^n, requires a MEAN block. The MEAN module computes every single element average of $\mathbf{x}_k$ in parallel, regardless of the size of the vector, thus enabling the processing of several signals in parallel at the same clock.

C. MODULE II - VARIANCE

The VARIANCE module is illustrated in Figure 3. It computes the variance of $\boldsymbol{x}_k$ vector samples by receiving the $\boldsymbol{x}_k$ vector itself and its average, $\boldsymbol{\mu}_k^x$, which were calculated in the previous MEAN blocks.

The VARIANCE module, akin to the MEAN module, uses a comparator identified at the Figure 3 as VCOMP1 to verify if the system is in the first iteration (Line 3 of Algorithm 1). The VMUX1 is a multiplexer that also implements a conditional evaluation to release the value of 0 in the register output VREG1 in the first iteration. The register VREG1 stores the variance value, $[\sigma^2]_k^x$, from the second iteration. The other registers in the block, VREG2 register and the N VREGn registers, are used to delay the iteration number k by one clock cycle and the elements of $\boldsymbol{x}_k$ respectively.

As demonstrated in Equation 3, the variance calculation is done recursively. It is necessary to calculate $\left\|\boldsymbol{x}_k - \boldsymbol{\mu}_k\right\|^2$ and to do that, N subtractors (VSUBn) and N multipliers (VMULT1_n) are used, as well an adder (VSUM1) with N inputs. Each element of vector $\boldsymbol{\mu}_k^x$ is subtracted from its respective element in vector $\boldsymbol{x}_k$ and the result of this operation is multiplied by itself (squared) and then added to the other results. The $\left\|\boldsymbol{x}_k - \boldsymbol{\mu}_k\right\|^2$ value is then multiplied (at VMULT2) by $1/k$. It is then added at VSUM2 adder with the variance calculated in the previous iteration, $[\sigma^2]_k^x$, multiplied (VMULT3) by $(k-1)/k$. From the second iteration on, this value passes through the VMUX1 multiplexer to the VREG1 register, delivering the calculation of the variance value to the VARIANCE block output. The values of $\left\|\boldsymbol{x}_k - \boldsymbol{\mu}_k\right\|^2$ and $1/k$ are also delivered to the output of the VARIANCE block to avoid redundant operations as they will be implemented in the next block, the ECCENTRICITY block.

D. MODULE III - ECCENTRICITY

The ECCENTRICITY module is a simpler block than those previously presented. This is because it uses operations that have already been performed in the VARIANCE block to calculate eccentricity. The geometric distance $\left\|\boldsymbol{x}_k - \boldsymbol{\mu}_k\right\|^2$ (equivalent to $(\boldsymbol{\mu}_k^x - \boldsymbol{x}_k)^T(\boldsymbol{\mu}_k^x - \boldsymbol{x}_k)$) is stored in register EREG3 and $1/k$ is stored in EREG4 register. As the ECCENTRICITY module is the architectural design of Equation 1 (Algorithm 1 line 9), the variance value $[\sigma^2]_k^x$ is multiplied by k (EMULT1) and used to divise (EDIV1) the geometric distance $(\boldsymbol{\mu}_k^x - \boldsymbol{x}_k)^T(\boldsymbol{\mu}_k^x - \boldsymbol{x}_k)$. This operation output is added to $1/k$ in the ESUM1 adder, subsequently calculating the eccentricity of the samples ($\xi_k(x)$) and delivering them to the ECCENTRICITY block output.

E. MODULE IV - OUTLIER

Finally, in the OUTLIER block, the samples are classified as abnormal (outlier = true) or normal (outlier = false).

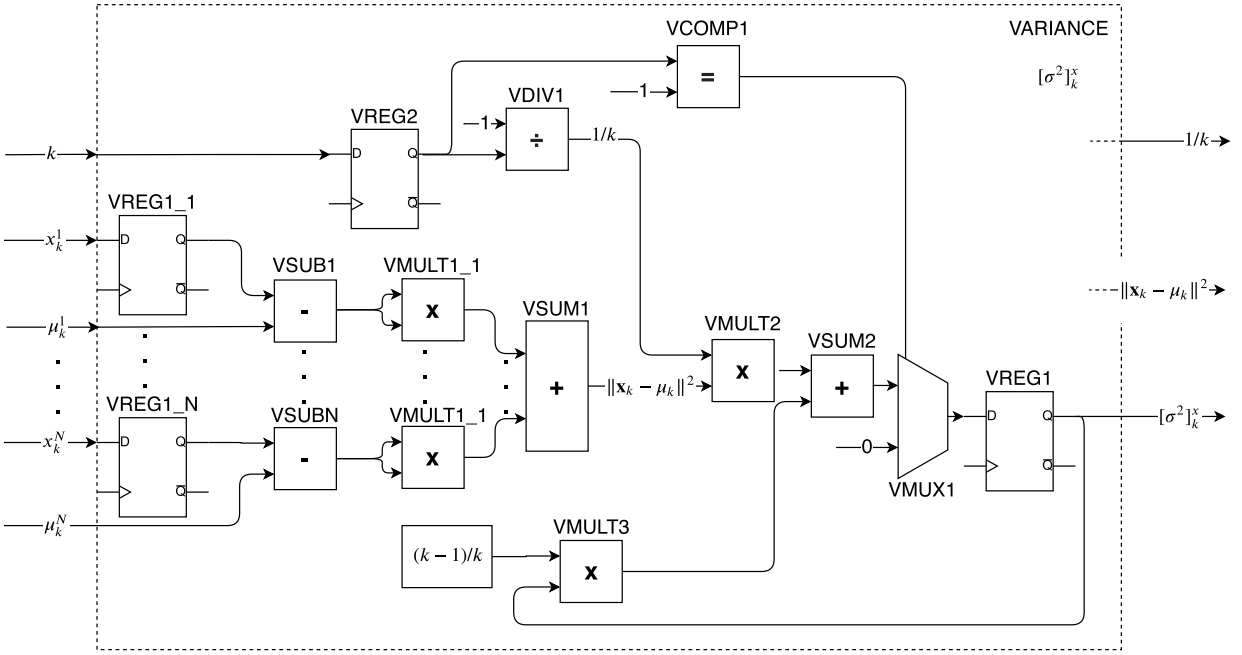


FIGURE 3. VARIANCE module.

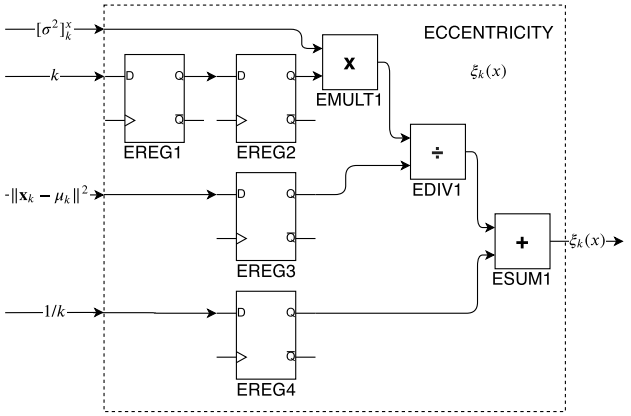


FIGURE 4. ECCENTRICITY module.

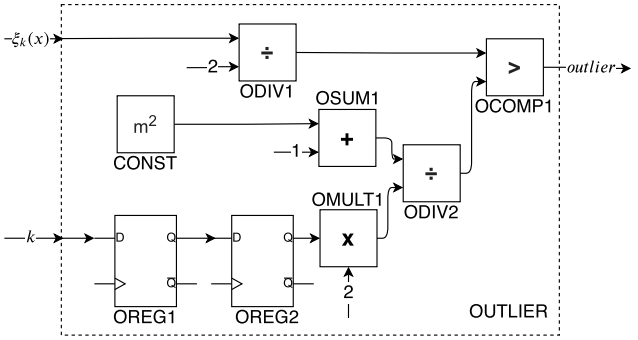


FIGURE 5. OUTLIER module.

The design module can be seen in Figure 5. To classify the samples, the OUTLIER block normalizes eccentricity by dividing (ODIV1) with a constant, as shown in Equation 5,. The comparative block (OCOMP1, compares this normalized eccentricity with a threshold as shown in the Lines 11, 12, 13 and 14 of the Algorithm 1. The registers OREG1 and OREG2 are used to synchronize the iteration number k, since as the modules act in pipeline, the operations carried out in the OUTLIER block (as well as ECCENTRICITY module) are delayed by two clock cycles concerning the system input.

F. PROCESSING TIME

To parallelize the input of data from the sensors, we use the technique of data parallelization, as described in [39]. All N main modules are identical operators processing, in parallel, different portions of the input data. Each of the N input signals ($[x_k^1\ x_k^2\ \ldots\ x_k^N]$) is processed independently. When the number of input signals increases, this parallel structure can be updated, keeping the OUTLIER and ECCENTRICITY modules unchanged and changing the VSUM1 adder of the VARIANCE module. VSUM1 will perform a tree sum. Exploring this type of parallelism justifies speed up compared to other solutions. In sequential architectures, in cases where the arrival rate or amount of date items exceeds the processing rates of the sequential structure [39], input signals from each sensor are processed one after the other increasing the operator's input queue.

The use of the pipeline technique allows the system to be divided into consecutive operators. Each operator receives the output data from the previous one. Because it's a 3-step pipeline design, the proposed architecture has an initial delay, d, that can be expressed as

$$d = 3 \times t_c \tag{7}$$

where t_c is the system critical path time.

TABLE 1. Fault types [40].

Fault	Description
f16	Positioner supply pressure drop
f17	Unexpected pressure change across the valve
f18	Fully or partly opened bypass valves
f19	Flow rate sensor fault

The circuit implemented for the TEDA algorithm has an execution time which is determined by the system critical path time, t_c. After the initial delay, the execution time of the proposed TEDA, t_{TEDA}, can be expressed as

$$t_{TEDA} = t_c \tag{8}$$

Therefore, in every t_{TEDA} it is possible to obtain the output of an inserted sample, that is, the sample classification of abnormal or normal.

The throughput of the implementation, th_{TEDA}, in samples per second (SPS) can be expressed as

$$th_{TEDA} = \frac{1}{t_{TEDA}}. \tag{9}$$

V. RESULTS

The hardware validation and synthesis results for the architecture proposed in this work are presented in this section. The behavioural simulation results of Algorithm 1 are first validated with their corresponding Python-based functional models. Then, the RTL models are synthesized on a Virtex 6 (xc6vlx240t-1ff1156) FPGA device. Validation results were used to verify the hardware functionality, while synthesis results allowed the system to be analyzed for important parameters for the design of hardware architectures. Analysing parameters such as hardware occupancy and processing time whilst also considering factors such as throughput and speedup. All cases were validated and synthesized on floating-point.

A. VALIDATION RESULTS

To validate the hardware architecture of the TEDA algorithm, we used the DAMADICS (Development and Application of Methods of the Actuator Diagnosis in Industrial Control Systems) benchmark dataset [40]. The benchmark provides a real data set of the water evaporation process in a Polish sugar factory. It is a plant with three actuators; a control valve, which controls the flow of water in the pipes; a pneumatic motor, which controls variable valve openings and a positioner. This dataset has faults at different times of the day on specific days. There are four different fault types, as shown in Table 1.

Artificial failures were introduced on specific days to plant operation data. The dataset has a set of 19 faults in these 3 actuators. As a way to validate the architecture, actuator 1 failures were simulated. Table 2 shows a detailed description of some introduced faults for actuator 1.

The behavioral results, after synthesis, are presented in Figures 6, 7, 8 and 9. The hardware architecture was designed with a floating point number format.

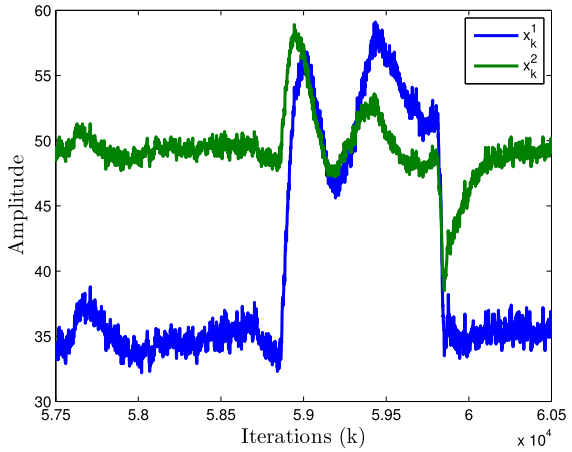


FIGURE 6. Behavior of fault item 1 - input vector x_k.

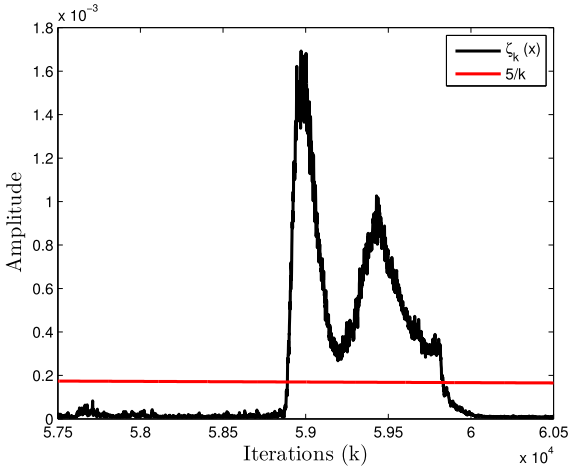


FIGURE 7. Behavior of fault item 1 - normalized eccentricity $\zeta_k(x)$ with $5/k$ ($m = 3$) threshold.

Figures 6 and 7 show the results obtained for the fault tabulated as item 1 in Table 2. Figure 6 illustrates the behavior of two simulated input variables ($\boldsymbol{x}_k = [x_k^1\ x_k^2]$). It is possible to observe that a failure happens between the moments k=58900 and k=59800. In Figure 7 it is possible to observe that there is a sudden change in the behavior of the eccentricity (black curve), surpassing the value of the comparison threshold with $m = 3$ (red curve).

In Figures 8 and 9 it is possible to observe the results obtained for the fault tabulated as item 7 in Table 2. Figures 6 and 7, Figures 8 illustrate the behavior of two elements of input $\boldsymbol{x}_k = [x_k^1\ x_k^2]$ and Figure 9 shows the eccentricity and the threshold. The failure happens between moments $k = 37700$ and $k = 38400$. It is possible to observe that there is a change of eccentricity (black curve), surpassing the value of the comparison threshold (red curve) also to $m = 3$.

B. SYNTHESIS RESULTS

After validation of the implemented circuit, the hardware synthesis was wielded to obtain the FPGA resource occupation report, as well as the critical time information and clock cycles used to calculate the proposed implementation processing time.

TABLE 2. List of artificial failures introduced to actuator 1 [40].

Item	Fault	Sample	Date	Description
1	f18	58800-59800	Oct 30, 2001	Partly opened bypass valve
2	f16	57275-57550	Nov 9, 2001	Positioner supply pressure drop
3	f18	58830-58930	Nov 9, 2001	Partly opened bypass valve
4	f18	58520-58625	Nov 9, 2001	Partly opened bypass valve
5	f18	54600-54700	Nov 17, 2001	Partly opened bypass valve
6	f16	56670-56770	Nov 17, 2001	Positioner supply pressure drop
7	f17	37780-38400	Nov 20, 2001	Unexpected pressure drop across the valve

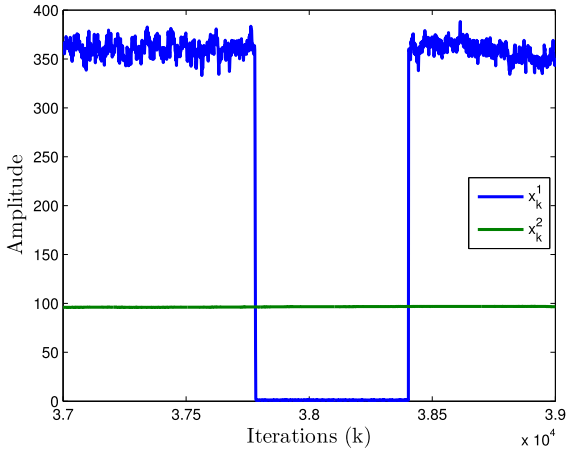


FIGURE 8. Behavior of fault item 7 - input vector x_k.

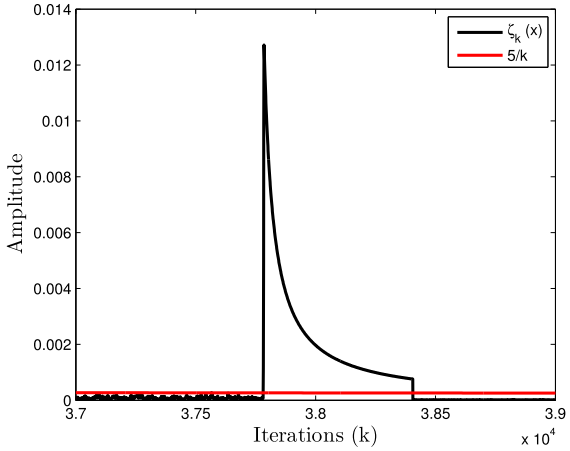


FIGURE 9. Behavior of fault item 7 - normalized eccentricity $\zeta_k(x)$ with $5/k$ ($m = 3$) threshold.

1) HARDWARE OCCUPATION

Table 3 presents data related to the hardware occupation of the circuit in the target FPGA for 5 scenarios, with a different number of input sensors for each of them. The first column shows the number of input sensors, the second column displays the number of multipliers used, the third column displays the number of registers, and the fourth column shows the number of logical cells used as LUT (n_{LUT}) throughout the circuit.

Analyzing the data presented in Table 3, it was identified that even whilst using a floating-point resolution, only a small portion of the resources were occupied from the target FPGA. For the first scenario, these resources had a total of only 3% from multipliers, less than 1% from registers, and about 7% from logical cells, used as LUT. This demonstrates that the proposed circuit, when used with a small number of input sensors, could also be applied to low-cost FPGAs, where the amount of available hardware resources are even smaller. In addition, multiple TEDA modules could be applied in parallel, for anomaly detection in the same dataset to further reduce processing time.

TABLE 3. Hardware occupation.

Inputs	Multipliers	Registers	n_{LUT}
2	27 (3%)	792 (< 1%)	11.586 (7%)
4	45 (5%)	911 (< 1%)	17.720 (11%)
8	81 (10%)	1189 (< 1%)	29.963 (19%)
16	153 (19%)	1690 (< 1%)	54.439 (36%)
32	297 (36%)	2802 (< 1%)	103.324 (68%)

TABLE 4. Processing time.

Inputs	Critical time	Delay	TEDA time	Throughput
2	91.2 ns	273.6 ns	91.2 ns	10.96 MSPs
4	95.0 ns	285.0 ns	95.0 ns	10.53 MSPs
8	93.8 ns	281.4 ns	93.8 ns	10.65 MSPs
16	102.4 ns	307.2 ns	102.4 ns	9.76 MSPs
32	122.1 ns	366.3 ns	122.1 ns	8.19 MSPs

2) SAMPLING RESULTS

The results for the operational processing time of the architecture are presented in Table 4. The first column indicates the number of input sensors for each scenario, the second column presents the circuit critical time, t_c, the third column shows the initial delay, expressed by Equation 7, the fourth column, the TEDA computation-time, expressed by Equation 8, and the last column, the implementation throughput in samples per second (SPS), expressed by Equation 9. The throughput consists of the number of samples processed and classified (as normal or outlier) by TEDA every second.

The data presented in Table 4 is quite expressive. The circuit critical time, which also corresponds to the TEDA runtime, was only $t_c = 122.1$ ns in the worst scenario. Hence, after the 366.3 ns delay, it is possible to get output for a processed sample sorted every 122.1 ns, which guarantees a throughput of 8.19 million sorted samples per second, for the 32 input sensors case. These results indicate the viability of using the proposal presented in this work to manipulate large data flows in real-time.

Another outcome of using the architecture was low consumption systems. As presented in [41] there is a strong dependency between operational clock frequency and the dynamic power. Using the clock rate lower than the maximum theoretical operating frequency causes a decrease in the dynamic power, making it possible to reduce energy

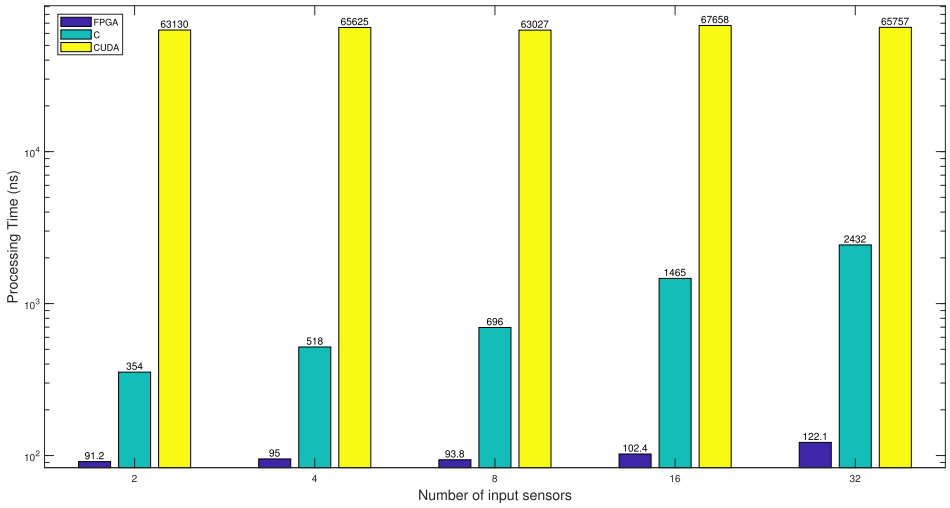


FIGURE 10. Comparison of processing time for different platforms.

consumption in applications where the processing speed is not limited and/or there is a need for low energy consumption.

C. PLATFORMS COMPARISON

To date, no previous literature has been found to explore TEDA hardware implementations. Thus, this paper presents, for the first time, a proposal to implement the TEDA technique on FPGA. To verify the advantages of the hardware application proposed here over implementations on other software platforms, some comparisons of the FPGA processing time with the processing time of other software implementations were made. Table 5 presents the results of the comparisons made. The first column indicates the number of input sensors, the second the hardware used, the third presents the processing time required to obtain the classification of each sample, and the fourth column, the speedup achieved by the proposal presented in this paper.

The hardware implementation on FPGA proposed here has been able to achieve speedups of up to 3.9 times faster when compared to a C implementation on Intel Core i7-9750H CPU in the first scenario and 19.9 times in the scenario with the highest number of input sensors. For a CUDA implementation using the NVIDIA GeForce GTX 1660 GPU, a speedup of 693 and 538 times was obtained for 2 and 32 sensors input respectively.

Figure 10 presents a bar chart with processing time for this FPGA proposal and the two other platforms compared in this work, CPU (with C) and GPU (with CUDA) respectively. The vertical axis is presented in a logarithm scale due to the asymmetry between the CUDA processing time and the other two implementations. The number of input sensors for the 5 simulated and synthesized different scenarios are presented in the horizontal axis. Analysing the Figure is possible to notice that the processing time with CUDA is much more substantial than the others but it remains almost constant, the C data presents an expressive growth with the increase in the number of input sensors while the FPGA processing time performs a slow augmentation.

TABLE 5. Software implementations comparison.

Inputs	Platform	Time	Speedup
2	This work proposal on FPGA	91.2 ns	–
	CPU (C)	354 ns	3.9×
	GPU (CUDA)	63130 ns	693×
4	This work proposal on FPGA	95.0 ns	–
	CPU (C)	518 ns	5.5×
	GPU (CUDA)	65625 ns	690×
8	This work proposal on FPGA	93.8 ns	–
	CPU (C)	696 ns	7.5×
	GPU (CUDA)	63027 ns	672×
16	This work proposal on FPGA	102.4 ns	–
	CPU (C)	1465 ns	15.6×
	GPU (CUDA)	67658 ns	660×
32	This work proposal on FPGA	122.1 ns	–
	CPU (C)	2432 ns	19.9×
	GPU (CUDA)	65757 ns	538×

Regarding the GPU implementation with CUDA, this happens because the process of copy between host and device has an impact in small groups threads due to the system bus bandwidth, latency and also because Single Instruction Multiple data (SIMD) turns into a important constraint for any innate different task [42]. An increase in the number of sensors improves the relative performance of a CUDA implementation as it is possible to handle more threads in parallel, but there is a limit of 1024 threads per block on GPU devices. Threads from different blocks share data through global memory using different kernel invocations, adding overhead [43]. All the CUDA scenarios used here for comparison were done in a single thread block.

The C implementation presents an expressive growth in the processing time due to the sequential architecture of the CPUs where augmenting the number of sensors has a significant impact. The FPGA sensors inputs (n MEAN modules - μ_k^n) are parallel, this makes it possible to increase the number of sensors without losing a lot of performance and throughput. Despite this, the tree sum in the VSUM1 adder (VARIANCE module) causes a slight increase in the FPGA processing time.

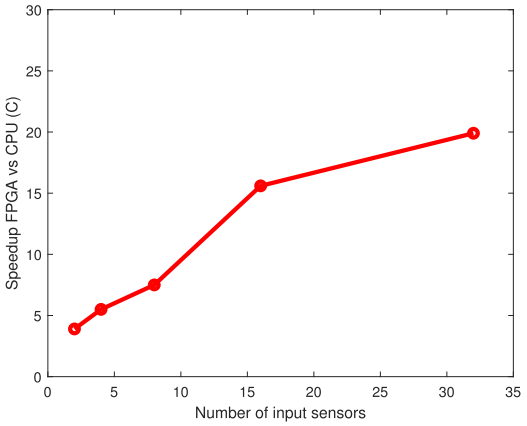


FIGURE 11. FPGA speedup in relation to the CPU (Intel Core i7-9750H) programmed with C.

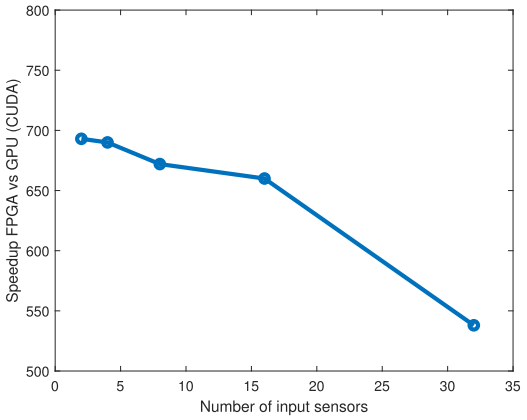


FIGURE 12. FPGA speedup in relation to the GPU (NVIDIA GeForce GTX1660) programmed with CUDA.

Figure 11 illustrates the FPGA speedup relative to CPU (with C) for different numbers of input sensors. It is possible to observe that there is a tendency to increase the speedup of the FPGA in relation to the CPU (C). Figure 12 illustrates the FPGA speedup, this time in relation to the GPU (programmed in CUDA). As previously mentioned, the relative performance of the GPU improves with the increase of the sensors, since its processing time remains practically constant and the processing time in the FPGA has a slight increase. It is important to note that even with a small decrease in the FPGA speedup in relation to the GPU, the processing time in the FPGA is still much smaller in all studied cases, as can be seen in Figure 10.

In an Industrial 4.0, or IoT environment, one of the most challenging tasks is how to handle streaming data from a large number of sensors. For example, there might be tens of thousands of various sensors in a factory. Supposing sensors that work with 50 samples/s (a sample each 20 ms), one specialized hardware on FPGA (with 125 ns) can work with about $20\,\text{ms}/125\,\text{ns} \cdot 32$ sensors $\approx$ 160,000 $\cdot$ 32 sensors $\approx$ 5,120,000 sensors. Other industrial sensors and devices like speed meters and motors can demand a high sample rate, which also justifies the specialized hardware solution. Another important aspect is the latency because the data anomaly detection can be associated with other algorithms (like machine learning). Spending a short time on TEDA execution is essential to reduce the holy anomaly detection system's latency.

VI. CONCLUSION

This work presents a proposal for the implementation of the hardware, TEDA data streaming anomaly detection technique. The hardware was implemented in RTL using a floating-point format. Synthesis results were obtained for a Xilinx Virtex 6 xc6vlx240t-1ff1156 FPGA. The proposed implementation used a small portion of the target FPGA resources, allowing the results to be obtained in a shortened processing time. The high speedups obtained, which were compared with other software platforms, coupled with the possibility of reducing power consumption by diminishing the clock frequency, reaffirms the importance of this work. The tendered architecture has the feasibility to operate, in practical fault detection applications in real industrial processes, with severe time constraints, and handling large volumes of data, as well as systems with power restraint, such as data streaming.

REFERENCES

[1] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, Nov. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231217309864

[2] M. Fahim and A. Sillitti, "Anomaly detection, analysis and prediction techniques in IoT environment: A systematic literature review," *IEEE Access*, vol. 7, pp. 81664–81681, 2019.

[3] X. Zhou, Y. Hu, W. Liang, J. Ma, and Q. Jin, "Variational LSTM enhanced anomaly detection for industrial big data," *IEEE Trans. Ind. Informat.*, vol. 17, no. 5, pp. 3469–3477, May 2021.

[4] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at Microsoft," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*. New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 3009–3017, doi: 10.1145/3292500.3330680.

[5] N. Giatrakos, E. Alevizos, A. Artikis, A. Deligiannakis, and M. Garofalakis, "Complex event recognition in the big data era: A survey," *VLDB J.*, vol. 29, no. 1, pp. 313–352, Jul. 2019, doi: 10.1007/s00778-019-00557-w.

[6] P. Napoletano, F. Piccoli, and R. Schettini, "Anomaly detection in nanofibrous materials by CNN-based self-similarity," *Sensors*, vol. 18, no. 2, p. 209, Jan. 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/1/209

[7] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of 'big data' on cloud computing: Review and open research issues," *Inf. Syst.*, vol. 47, pp. 98–115, Jan. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437914001288

[8] C. G. Bezerra, B. S. J. Costa, L. A. Guedes, and P. P. Angelov, "An evolving approach to unsupervised and real-time fault detection in industrial processes," *Expert Syst. Appl.*, vol. 63, pp. 134–144, Nov. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417416303153

[9] M.-C. Lee, J.-C. Lin, and E. G. Gan, "ReRe: A lightweight real-time ready-to-go anomaly detection approach for time series," in *Proc. IEEE 44th Annu. Comput., Softw., Appl. Conf. (COMPSAC)*, Jul. 2020, pp. 322–327.

[10] B. S. J. Costa, C. G. Bezerra, L. A. Guedes, and P. P. Angelov, "Online fault detection based on typicality and eccentricity data analytics," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–6.

[11] D. Kangin, P. Angelov, J. A. Iglesias, and A. Sanchis, "Evolving classifier TEDAClass for big data," *Procedia Comput. Sci.*, vol. 53, pp. 9–18, Jan. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915017779

[12] B. S. J. Costa, C. G. Bezerra, L. A. Guedes, and P. P. Angelov, "Unsupervised classification of data streams based on typicality and eccentricity data analytics," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jul. 2016, pp. 58–63.

[13] D. Osherson and E. E. Smith, "On typicality and vagueness," *Cognition*, vol. 64, no. 2, pp. 189–206, Aug. 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010027797000255

[14] P. Cappellari, M. Roantree, and S. A. Chun, "A scalable platform for low-latency real-time analytics of streaming data," in *Data Management Technologies and Applications*, C. Francalanci and M. Helfert, Eds. Cham, Switzerland: Springer, 2017, pp. 1–24.

[15] K. Nakamura, A. Hayashi, and H. Matsutani, "An FPGA-based low-latency network processing for spark streaming," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2016, pp. 2410–2415.

[16] J. Fang, Y. T. B. Mulder, J. Hidders, J. Lee, and H. P. Hofstee, "In-memory database acceleration on FPGAs: A survey," *VLDB J.*, vol. 29, no. 1, pp. 33–59, Jan. 2020.

[17] P. L. Noac'h, A. Costan, and L. Bougé, "A performance evaluation of Apache Kafka in support of big data streaming applications," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 4803–4806.

[18] T. Aung, H. Y. Min, and A. H. Maw, "Coordinate checkpoint mechanism on real-time messaging system in kafka pipeline architecture," in *Proc. Int. Conf. Adv. Inf. Technol. (ICAIT)*, Nov. 2019, pp. 37–42.

[19] O.-C. Marcu, A. Costan, G. Antoniu, M. Pérez-Hernández, B. Nicolae, R. Tudoran, and S. Bortoli, "KerA: Scalable data ingestion for stream processing," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 1480–1485.

[20] J. Liu and D. Liang, "A survey of FPGA-based hardware implementation of ANNs," in *Proc. Int. Conf. Neural Netw. Brain*, vol. 2, 2005, pp. 915–918.

[21] N. Fuengfusin and H. Tamukoh, "Mixed-precision weights network for field-programmable gate array," *PLoS ONE*, vol. 16, no. 5, pp. 1–26, May 2021, doi: 10.1371/journal.pone.0251329.

[22] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "Processing-in-memory: A workload-driven perspective," *IBM J. Res. Develop.*, vol. 63, no. 6, pp. 3:1–3:19, Nov. 2019.

[23] V. Sze, "Designing hardware for machine learning: The important role played by circuit designers," *IEEE Solid State Circuits Mag.*, vol. 9, no. 4, pp. 46–54, Nov. 2017.

[24] M. Rashid, M. Imran, and A. R. Jafri, "Exploration of hardware architectures for string matching algorithms in network intrusion detection systems," in *Proc. 11th Int. Conf. Adv. Inf. Technol. (IAIT)*. New York, NY, USA: Association for Computing Machinery, Jul. 2020, pp. 1–7, doi: 10.1145/3406601.3406608.

[25] L. M. D. D. Silva, M. F. Torquato, and M. A. C. Fernandes, "Parallel implementation of reinforcement learning *Q*-learning technique for FPGA," *IEEE Access*, vol. 7, pp. 2782–2798, 2019.

[26] P. Angelov, "Anomaly detection based on eccentricity analysis," in *Proc. IEEE Symp. Evolving Auton. Learn. Syst. (EALS)*, Dec. 2014, pp. 1–8.

[27] P. Angelov, "Typicality distribution function—A new density-based data analytics tool," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–8.

[28] R. S. Martins, P. Angelov, and B. S. J. Costa, "Automatic detection of computer network traffic anomalies based on eccentricity analysis," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jul. 2018, pp. 1–8.

[29] B. Yang, M. Yang, A. Plaza, L. Gao, and B. Zhang, "Dual-mode FPGA implementation of target and anomaly detection algorithms for real-time hyperspectral imaging," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 6, pp. 2950–2961, Jun. 2015.

[30] M. Wess, P. D. S. Manoj, and A. Jantsch, "Neural network based ECG anomaly detection on FPGA and trade-off analysis," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[31] D. J. M. Moss, D. Boland, P. Pourbeik, and P. H. W. Leong, "Real-time FPGA-based anomaly detection for radio frequency signals," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.

[32] P. Angelov, "Outside the box: An alternative data analytics framework," *J. Autom., Mobile Robot. Intell. Syst.*, vol. 8, no. 2, pp. 29–35, Apr. 2014.

[33] A. Bernieri, G. Betta, and C. Liguori, "On-line fault detection and diagnosis obtained by implementing neural algorithms on a digital signal processor," *IEEE Trans. Instrum. Meas.*, vol. 45, no. 5, pp. 894–899, Oct. 1996.

[34] J. G. Saw, M. C. K. Yang, and T. C. Mo, "Chebyshev inequality with estimated mean and variance," *Amer. Statistician*, vol. 38, no. 2, pp. 130–132, May 1984.

[35] M. G. F. Coutinho, M. Torquato, and M. A. C. Fernandes, "Deep neural network hardware implementation based on stacked sparse autoencoder," *IEEE Access*, vol. 7, pp. 40674–40694, 2019.

[36] M. F. Torquato and M. A. Fernandes, "High-performance parallel implementation of genetic algorithm on FPGA," *Circuits, Syst., Signal Process.*, vol. 38, no. 9, pp. 4014–4039, 2019.

[37] F. F. Lopes, J. C. Ferreira, and M. A. C. Fernandes, "Parallel implementation on FPGA of support vector machines using stochastic gradient descent," *Electronics*, vol. 8, no. 6, p. 631, Jun. 2019.

[38] A. L. X. Da Costa, C. A. D. Silva, M. F. Torquato, and M. A. C. Fernandes, "Parallel implementation of particle swarm optimization on FPGA," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 11, pp. 1875–1879, Nov. 2019.

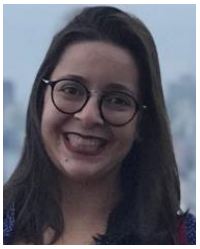
[39] G. Gracioli, M. Dunne, and S. Fischmeister, "A comparison of data streaming frameworks for anomaly detection in embedded systems," in *Proc. 1st Int. Workshop Secur. Privacy Internet-of-Things (IoTSec)*, Apr. 2018, pp. 1–4.

[40] EFRT Network. (2002). *DAMADICS RTN Information Web Site*. [Online]. Available: http://diag.mchtr.pw.edu.pl/damadics/

[41] S. N. Silva, F. F. Lopes, C. Valderrama, and M. A. C. Fernandes, "Proposal of Takagi–Sugeno fuzzy-PI controller hardware," *Sensors*, vol. 20, no. 7, p. 1996, Apr. 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/7/1996

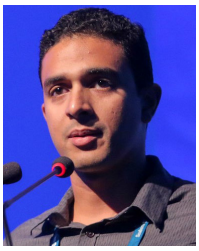
[42] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," in *Proc. 36th Annu. Int. Symp. Comput. Archit. (ISCA)*. New York, NY, USA: Association for Computing Machinery, 2009, pp. 152–163, doi: 10.1145/1555754.1555775.

[43] NVIDIA Corporation. (2021). *CUDA C++ Programming Guide, Release: 11.3.1*. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/

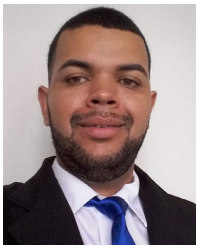


LUCILEIDE M. D. DA SILVA was born in Natal, Brazil. She received the B.S. degree in electronics engineering from the École Nationale Supérieure d'électronique, d'Electrotechnique, d'-Informatique et d'Hydraulique et des Télécommunications (ENSEEIHT), Toulouse, France, in 2011, and the B.S. and M.Sc. degrees in electrical engineering from the Federal University of Rio Grande do Norte (UFRN), Natal, in 2012 and 2016, respectively, where she is currently pursuing the Ph.D. degree in electrical and computer engineering. She is an Assistant Professor at the Informatics Department, Federal Institute of Rio Grande do Norte (IFRN), Santa Cruz, Brazil. She is currently a Visiting Researcher at the Centre Telecommunication Research, King's College London (KLC), London, U.K. She is part of the IFRN's Robotics Group, training teenagers in different competitions and developing assistive technology to children with special needs. She is also part of the Research Group on Embedded Systems and Reconfigurable Hardware, Department of Computer Engineering and Automation, UFRN, where the main research topics are the acceleration of artificial intelligence algorithms through reconfigurable computing on FPGA. Her research interests include artificial intelligence, embedded systems, reconfigurable hardware, tactile internet, and educational robotics for teenagers and children with special needs.

MARIA G. F. COUTINHO was born in Natal, Brazil. She received the B.S. degree in computer science from the State University of Rio Grande do Norte, Natal, in 2017, and the M.Sc. degree in electrical and computer engineering from the Federal University of Rio Grande do Norte, Natal, in 2019, where she is currently pursuing the Ph.D. degree in electrical and computer engineering and a Team Member of the Research Group on Embedded Systems and Reconfigurable Computing. Her main research topic is the acceleration of deep learning algorithms through reconfigurable computing in FPGA. Her research interests include artificial intelligence, embedded systems, and reconfigurable hardware.



CARLOS E. B. SANTOS, JR. was born in Recife, State of Pernambuco, Brazil. He received the A.S. degree in computer networking from the Federal Institute of Education, Science and Technology of Rio Grande do Norte, Natal, Brazil, in 2013, the latu-sensu certificate in network security from the Estacio de Sa University, Natal, in 2016, and the M.Sc. degree in computer engineering from the Federal University of Rio Grande do Norte, Natal, in 2018, where he is currently pursuing the Ph.D. degree in computer engineering. He is an Information Technologist Analyst and a Team Member of the Research Group on Embedded Systems and Reconfigurable Computing. His main research topics are computer and security network (attack and defence), FPGA, and acceleration of hash algorithms. His research interests include security, artificial intelligence, embedded systems, and reconfigurable hardware.



MAILSON R. SANTOS was born in Currais Novos, Rio Grande do Norte, Brazil. He received the degree in computer engineering from the Federal University of Rio Grande do Norte, Natal, in 2018, where he is currently pursuing the M.Sc. degree in electrical and computer engineering. He is a Team Member of the Scientific Data Analysis Laboratory. His main research topic is the fault detection and isolation in industrial processes, evolving systems. His research interests include artificial intelligence, data analysis, evolving systems, and system to industrial application.

M. DOLORES RUIZ received the degree in mathematics and the European Ph.D. degree in computer science from the Universidad de Granada, in 2005 and 2010, respectively. She held a non-permanent teaching position with the Universities of Jaén, Granada, and Cádiz. She has participated in more than ten projects, including the EU FP7 Projects ePOOLICE and Energy IN TIME. She is currently a Research Associate with the Computer Science and Artificial Intelligence Department, Universidad de Granada. She has organized several special sessions about data mining in international conferences and was part of the organization committee of the FQAS'2013 and SUM'2017 conferences. She belongs to the Approximate Reasoning and Artificial Intelligence Research Group and the Cybersecurity Laboratory, Universidad de Granada. She has been the Principal Investigator of the project exception and anomaly detection by means of fuzzy rules using the RL-theory, application to fraud detection. Her research interests include data mining, information retrieval, energy efficiency, big data, correlation statistical measures, sentence quantification, and fuzzy sets theory.

LUIZ AFFONSO GUEDES received the degree in electrical engineering from UFPA, in 1988, the master's degree in electronic engineering and computing from ITA, in 1991, and the Ph.D. degree in automation and computer engineering from Unicamp, in 1999. He is currently a Full Professor with the Department of Computer Engineering and Automation (DCA), Federal University of Rio Grande do Norte (UFRN), and a CNPq Research Productivity scholarship holder. He has expertise in fault diagnostic and operational reliability of industrial processes using evolving systems and statistic modeling. He supervised 14 doctoral theses and 39 master's dissertations, and has more than 15 years of experience in coordination of research and development projects, which have resulted in various software solutions in use in industries and with INPI certification.

MARCELO A. C. FERNANDES was born in Natal, Brazil. He received the B.S. and M.S. degrees in electrical engineering from the Federal University of Rio Grande do Norte, Natal, in 1997 and 1999, respectively, and the Ph.D. degree in electrical engineering from the University of Campinas, Campinas, State of São Paulo, Brazil, in 2010. From 2015 to 2016, he worked as a Visiting Researcher with the Centre Telecommunication Research (CTR), King's College London, London, U.K. From 2019 to 2021, he worked as a Visiting Scholar with the John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, USA. He is currently an Associate Professor with the Department of Computer Engineering and Automation, Federal University of Rio Grande do Norte. He is the Leader of the Research Group on Embedded Systems and Reconfigurable Computing (RESRC), and a Coordinator of the Laboratory of Machine Learning and Intelligent System (LMLIS). He is the author and coauthor of many scientific papers and practical studies with reconfigurable computing on FPGA to accelerate artificial intelligence algorithms. His research interests include artificial intelligence, digital signal processing, embedded systems, reconfigurable hardware, and tactile internet.