Exploring ICMetrics to Detect Abnormal Program Behaviour on Embedded Devices

Xiaojun Zhai[1], Kofi Appiah[2], Shoaib Ehsan[3], Gareth Howells[4], Huosheng Hu[3], Dongbing Gu[3] and Klaus McDonald-Maier[3]

[1]University of Leicester, United Kingdom

[2]Nottingham Trent University, United Kingdom

[3]University of Essex, United Kingdom

[4]University of Kent, United Kingdom

xzhai@leicester.ac.uk, kofi.appiah@ntu.ac.uk, sehsan@essex.ac.uk, w.g.j.howells@kent.ac.uk, hhu@essex.ac.uk, dgu@essex.ac.uk, kdm@essex.ac.uk

***Abstract*—Execution of unknown or malicious software on an embedded system may trigger harmful system behaviour targeted at stealing sensitive data and/or causing damage to the system. It is thus considered a potential and significant threat to the security of embedded systems. Generally, the resource constrained nature of Commercial off-the-shelf (COTS) embedded devices, such as embedded medical equipment, does not allow computationally expensive protection solutions to be deployed on these devices, rendering them vulnerable. A Self-Organising Map (SOM) based and Fuzzy C-means based approaches are proposed in this paper for detecting abnormal program behaviour to boost embedded system security. The presented technique extracts features derived from processor's Program Counter (PC) and Cycles per Instruction (CPI), and then utilises the features to identify abnormal behaviour using the SOM. Results achieved in our experiment show that the proposed SOM based and Fuzzy C-means based methods can identify unknown program behaviours not included in the training set with 90.9% and 98.7% accuracy.**

***Index Terms*—Embedded system security, abnormal behaviour detection, intrusion detection, Self-Organising Map.**

I. Introduction

We are in the midst of a digital revolution where small, battery-operated and resource-constrained embedded devices can be seen deployed in a wide range of applications. These ubiquitous embedded devices have drastically transformed the manner in which the information is created, destroyed, shared, processed and managed. An example of this is the healthcare sector where embedded medical devices are utilized extensively on daily basis for processing sensitive medical data and for performing critical functions for multiple patients. Since these embedded systems usually handle sensitive information, it is desirable to have some security mechanism deployed on these devices, either in the form of software or the hardware. However, security of embedded devices is a challenging task and considered an open research issue due to the resource-constrained nature of these devices [1]. For example, these devices typically have strict limitations on the amount of memory, computational units and power consumption. Indubitably, security has been extensively explored in the context of general purpose computing and communications systems, such as cryptographic algorithms and security protocols [2]. However, such security solutions are often incompatible with many embedded architectures and cannot be utilized due to custom firmware (or operating systems), limited power budgets and highly constrained computational resources. Consequently, conventional protection software such as antivirus (AV) programs, which are widely used on general-purpose computers, are difficult to implement on these small, low-power embedded systems.

Indeed, researchers have explored both the hardware- and software-based solutions for securing embedded devices. Physical Unclonable Function (PUF) [3] or hardware intrinsic security [4], is a hardware-based security mechanism that was proposed to secure embedded devices physically. Integrated circuits are first identified by utilising manufacturing process variation and then the identities are further used for cryptography. There has also been research work focusing on detecting software failure, tampering and malicious codes in embedded architectures [1, 5]. A major drawback of these techniques is the storage of sensitive information in the system as "valid" samples or templates. For example, a basic-block control-flow graph (CFG) is usually stored and used to examine the running programs.

Although embedded devices are deployed in a wide range of application scenarios, they generally perform a small number of repetitive functions or operate in a simplified state space. The execution space may include activities such as actuating an electrical relay, controlling a pump, processing medical data, and collecting sensor

readings [6]. As opposed to computation intensive conventional antivirus, detection of compromised activities through deviation in normal program execution is a promising solution, which is light-weight and arises from the intrinsic behaviour of these embedded devices. There are currently alternative solutions that may secure vulnerable embedded architectures [7] [8], where machine learning and pattern recognition algorithms are employed on human-machine interaction. Another potential technology that can be utilized for embedded systems security is ICMetrics (Integrated Circuit metrics) [9], which relies on the unique trace generated on the embedded architecture by its regular user or environment. The concept of ICMetrics is analogous to biometrics in humans. Fig. 1 exhibits a typical embedded system and ICMetrics system. The ICMetric system is designed to be employed on previously unseen devices and to faithfully reproduce encryption keys for such devices on further application to them by examining a pre-defined set of measurable features of such devices. However, the system does require detailed knowledge of the likely distribution of such features within their domain of possible values for typical devices. Therefore, a significant calibration phase is required for each application domain for which the ICMetric system is to be employed prior to its employment in the generation of encryption keys. This calibration phase operates on samples taken from typical example devices which may or may not include devices for which encryption keys will subsequently be required. Although the system is, subsequent to the calibration phase, designed to operate on previously unseen devices, this is governed by the restriction that the measured features will behave approximately as predicted by analysis of the sample devices. The ICMetrics is therefore a two phase system: the calibration phase is applied only once per application domain employing a number of known circuits as a calibration set, while the operation phase is applied each time an encryption key is desired for a given circuit.

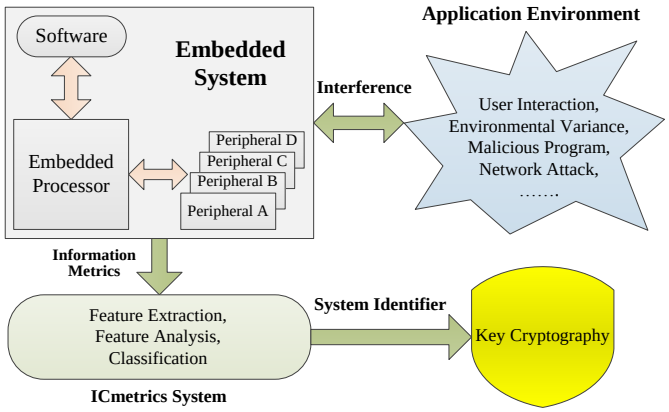

Fig. 1. A typical embedded system and ICMetrics system.

Importantly, the ICMetrics based system does not need to store any user data or template and does not require support from operating system, thus offering multiple benefits over traditional static AV approaches like scanning executable, instruction sequences and CFG of an application. Irrespective of the type of application scenario, our approach is suitable for any resource-constrained embedded device that performs repetitive tasks, and thus is a generic method of securing embedded systems. As an example, such devices are predominantly used in the medical and automation industry, which have limited cost and resource in the system.

In our pervious paper [10], we use Cycle per Instruction (CPI) to extract corresponding Program Counter (PC) values, and use it as ICMetric for correct program identification allowable to execute on the embedded architecture. In this paper, instead of using only unsupervised Self-Organising Map (SOM) [11] for the identification, both SOM and a Fuzzy C-means [12] based approaches are separately used to classify the ICMetric behaviour of the embedded system. Results achieved in our experiment show that the proposed SOM based and Fuzzy C-means based methods can identify unknown program behaviours not included in the training set with 90.9% and 98.7% accuracy respectively.

The remainder of the paper is structured as follows: An overview of the related work in this domain is given in Section II. Abnormal program behaviour detection algorithms are proposed in Section III, which are based on SOM and Fuzzy C-means algorithms. Section IV details the experimental design and results performed on an ARM Cortex-M3 embedded processor to demonstrate the utility of the proposed method. Finally, the conclusions are presented in Section V.

II. Related Work

This section provides a brief overview of the previous work related to embedded systems security. As

mentioned in Section I, information digitization to facilitate quick access has rendered digital privacy an important issue in protecting personal data [13]. While we believe our work to be the first demonstration of how on-chip debug information [14] can be used to identify anomalies in embedded system program execution, previous research has investigated the behaviour and prevalence of code modified with the intent of harming a system or its user. Arora et al [1] addressed secure program execution by focusing on the specific problem of ensuring that the program does not deviate from its intended behaviour. In their work, properties of an embedded program is extracted and used as the basis for enforcing permissible program behaviour.

Software piracy has enormous economic impact [15], making it important to protect software intellectual property rights. Software watermarks, unique identifier embedded in a protected software to discourage intellectual property theft is presented by Collberg and Thomborson [16]. In [17], Kolbitsch et al proposed a malware detection system to complement conventional AV software by matching automatically generated behaviour models against the runtime behaviour of unknown programs. Similar to [1], Rahmatian et al [5] used a CFG to detect intrusion for secured embedded systems by detecting behavioural differences between the correct system and malware. In their system, each executing process is associated with a finite state machine (FSM) that recognizes the sequences of system calls generated by the correct program. An attack is detected, if the system call sequence deviates from the known sequence. The system promises the ability to detect attacks in most application-specific embedded processors. Wang et al [15] proposed a system call dependence graph (SCDG) birthmark software theft detection system. Software birthmarks have been defined as unique characteristics that a program possesses and can be used to identify the program. Without the need for source code, a dynamic analysis tool is used in [18] to generate system call trace and SCDGs to detect software component theft.

Yang et al [19] presented an interesting approach for detecting digital audio forgeries mainly in MP3. Using a passive approach, they are able to detect doctored MP3 audio by checking frame offsets. Their work proves that frame offsets detected by the identification of quantization characteristics are good indication for locating forgeries. The experiment conducted on 128 MP3 speech and music clips shows a 94% rate of correctly detecting deletion and insertion using frame offset. Panagakis and Kotropoulos [20] proposed the random spectral features (RSFs) and the labelled spectral features (LSFs) as intrinsic fingerprints suitable for device identification. The unsupervised RSFs reduce the dimensionality of the mean spectrogram of recorded speech, while the supervised LSFs derives a mapping between the feature space, where the mean spectrograms lie on the label space. Experimental result shows that RSFs and LSFs are able to identify telephone handset with up to 97.58% accuracy.

Information hiding can be used in authentication, copyright management as well as digital forensics [21]. Swaminathan et al [21] proposes to enhance computer system performance with information hiding in the compiled program binaries. The system-wide performance is improved by providing additional information to the processor without changing the instruction set architecture. The proposed system employs look-up-tables for data embedding and extraction, which is subsequently stored in the program header and loaded into run-time memory at the beginning of program execution. In [22], Boufounos and Rana demonstrate with the use of signal processing and machine learning techniques, how to securely determine whether two signals are similar to each other. They also show how to utilize an embedding scheme for privacy-preserving nearest neighbour search by presenting protocols for clustering and authenticating applications.

As mentioned above, software birthmarks are a unique characteristic that a program possesses and can be used to identify the program [15]. Similarly, ICMetrics can be defined as a unique characteristic that a program possesses when running on a particular embedded device and can be used to identify the program and hardware. Let p, q be programs. Let $f(p)$ be a set of characteristics extracted from p when running on hardware f. We say $f(p)$ is the ICMetrics of p, only if the following two conditions are satisfied:

1) $f(p)$is obtained from p running on f.

2) Program q is a copy of $p \Rightarrow f(p) = f(q)$.

The limitations with the use of system calls for program identification [1], [5] have been pointed out in [15] and are more prevalent in embedded systems settings, which typically to not employ operating systems. The mentioned limitations are:

1) Programs with little or no system calls such as programs solely based on arithmetic operation, and

2) Programs which do not have unique system call behaviours may fail to exhibit a birthmark.

Using an unsupervised SOM to reduce the dimensionality of PC values, we introduce an offset rule similar to that presented in [19] to detect compromised programs. Thus using machine learning techniques [22] we are able to determine whether two PC values are similar to each other, with the use of the program binaries [21] and no prior knowledge of the source code. Our main contributions of this paper can be summarised as follows:

1) We introduce two anomaly detection systems which can be used to combine with ICMetrics system in the embedded devices, predominantly adopted in the medical and automation industry, one uses a SOM based

classifier and another one uses a Fuzzy C-means based classifier. A comparison of the two classifiers is also given to show the improvement.

2) Our approach introduces a way to extract and analyse the useful low level hardware information, and uses them as a feature to identify an embedded system's abnormal behaviour, which allows our system to be employed in a wider range of embedded systems, as it is independent of the high level software environments (e.g. Operating system, source programs).

III. Methods for Detecting Compromised Programs

This section provides details on the method for detecting compromised programs in an embedded device. Most processors found in embedded devices execute their programs with three structural levels [1]:

1) Function call relationships used to represent function calls within a program.
2) A basic-block CFG used to represent internal control flow and
3) Instruction stream within each CFG.

This is true for a single processor program, which comprises of a number of micro operations. Micro operations are very dependent on the instruction set architecture of the processor of the embedded system. The number of clock cycles for each instruction is dependent on the hardware architecture used, the type of instruction and the task to be executed. Typically, most modern pipelined processor architecture instructions such as Load, Store and Jump, only require a clock cycle to execute, as they need access to memory during processing. In particular, these multi-cycle instructions indicate where the function call or conditional branching occurs [4]. Multiprocessor SoCs are increasingly deployed in embedded systems with little or no security features built in. But the order of instructional execution on single processor architecture is not comparable to that of multi-processor architecture. For multi-processor embedded systems, one approach is to use a dedicated security (monitor) processor to oversee the application at runtime. Each processor communicates with the monitor processor through a FIFO queue, and is continuously checked [23].

For a single processor embedded systems, we can approximately detect the function call or condition branch based on the variance of the processor's performance. In addition, the value of PC register shows the instruction stream of a program, which is also a suitable source for monitoring changes at the instruction level. By monitoring the processor's performance, we detect changes in the function call and CFG, and then analyse the PC values within each CFG. Again, an overall evaluation could indicate whether the program is compromised or not. In this work, we measure the average CPI as the parameter of a processor's performance. A block diagram of our proposed program monitoring systems is presented in Fig. 2.

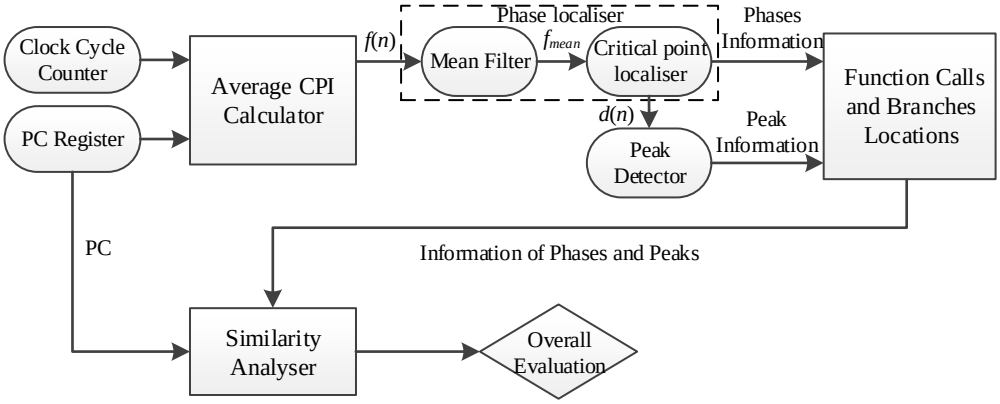

Fig. 2. Overall block diagram of the proposed monitoring system.

The average CPI calculator in Fig. 2 is first used to calculate the average Cycles per Instruction (CPI) value, and it continually reads clock cycle and PC data from the time counter and PC register. This information can be accessed through non-intrusive debug support interface of the processor. Also the phase localiser and peak point detector blocks are used to obtain the function call and conditional branch location information from average CPI profile respectively, and then the obtained information used to extract features for the SOM and Fuzzy c-means classifiers. The final evaluation is based on the results of the output of the various classifiers.

A. CPI Analysis

The complexity of instructions executed within a particular period of time represents the CPI. A good

description of the average CPI for a processor and how it can be calculated is given in [24]. Fig. 3 shows an average CPI profile while a program is running in an ARM cortex-M3 processor based embedded platform, where instruction counter and processor's running frequency are set to 2^{11} and 120 MHz respectively.

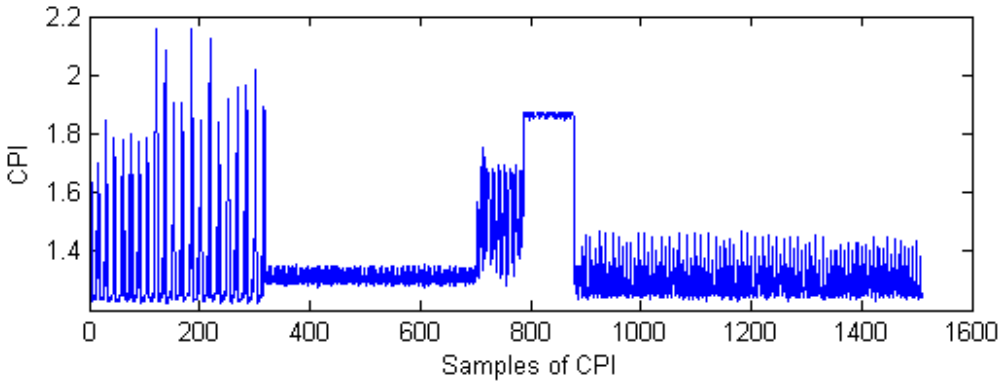

Fig. 3. Example of average CPI diagram.

In Fig. 3, the program mainly consists of five phases, and there are also many variances (i.e. peaks) within each phase. A method for obtaining the positional information of the phases and peaks will be presented in the following sections.

1) Phase localiser block

There are two main sub-blocks in the phase localiser block, these are the mean filter and critical point localiser. The mean filter is first used to smooth the original CPI diagram, the critical point localiser is then used to localise the positions of each phase.

2) Mean filter

A 1×w rectangular window is used as a mask in the mean filter, the local average value within the mask is then calculated. Let f(n) denote the CPI value at position n which is always the centre point of a window of size w. The window mean value f_{mean}(n) is calculated by (1):

$$f_{mean}(n) = \sum_{n \in B} f(n) / w \tag{1}$$

Fig. 4 shows the resulting diagram after applying the mean filter on the original CPI diagram (i.e. Fig. 3), where w is set to '5'. As can be seen from Fig. 4, the variances within each phase have been significantly suppressed, and the boundaries of each phase still stay intact.

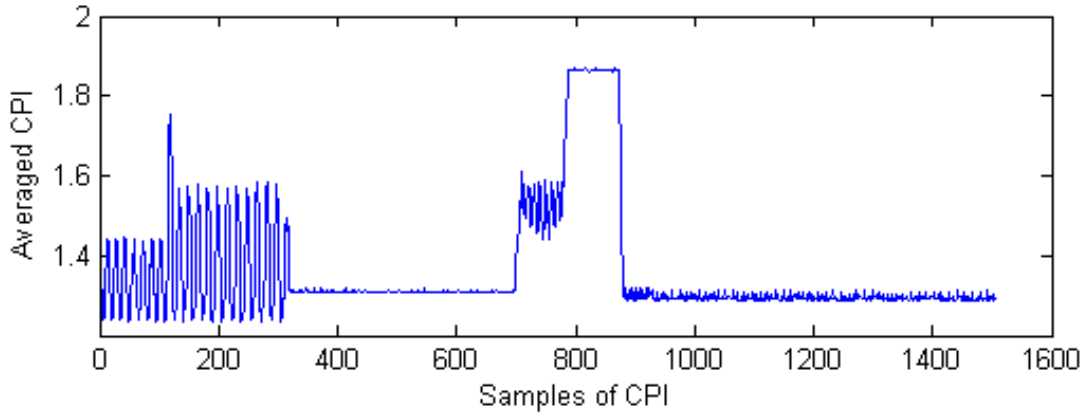

Fig. 4. Resulting CPI diagram after applying the mean filter.

3) Critical point localiser

As the values of two adjacent points at the boundary are normally significantly different, the proposed method is to localise the high variance points, and then select the best candidates based on pre-defined criterion.

Let f_{mean} denote averaged CPI, absolute differences between adjacent elements of f_{mean} can then be calculated by:

$$d(n) = \left| f_{mean}(n+1) - f_{mean}(n) \right| \tag{2}$$

where $1 \le n < N$, N is the total numbers of elements in array f_{mean}, $d(n)$ is n^{th} element in an array of absolute differences between adjacent elements of $f_{mean}(n)$.

A threshold t_1 is first used to select the high variance elements from array d , where the indices of the elements are greater than t_1 they are stored in array d_1. After that, absolute differences between adjacent elements of d_1 are calculated to form d_2. Finally, a threshold t_2 is used to select the boundary candidates, where elements greater than t_2 are selected as the candidates. Values of t_1 and t_2 are fixed based on experimental results. In this work, t_1 and t_2 are set to 0.03 and 9 respectively. Fig. 5 shows resulting diagram after applying the critical point localiser on Fig.4.

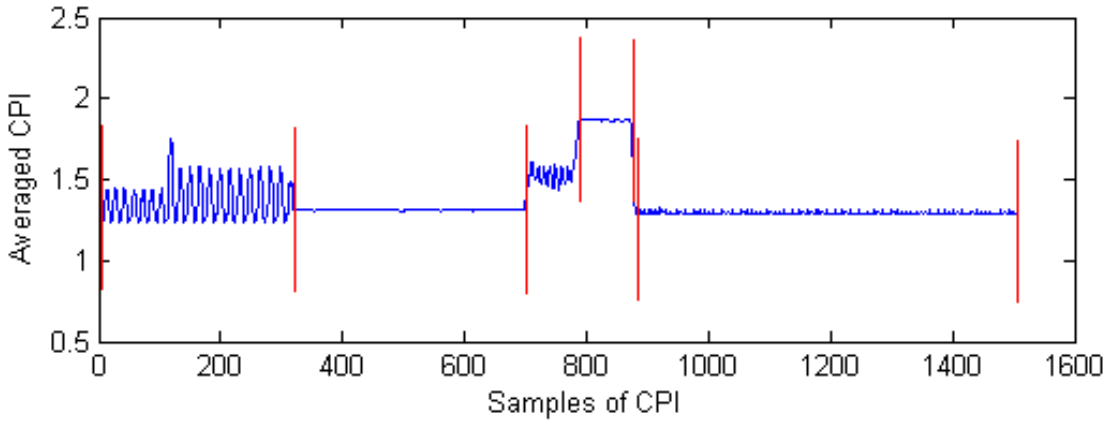

Fig. 5. Resulting diagram after applying the critical point localiser.

4) Peak detector block

In order to obtain positions of peaks and valleys, we apply the peak detector on array d rather than the original array f_{mean}. Pseudo-codes for detecting the peaks are summarised as follows:

```
Peak detection procedure:
Input: d_i is an array of absolute differences between adjacent elements of
f_mean in the i^th phase.
Output: P = {p_1, p_2, p_3,..., p_i }where p_1 is a set of locations for the i^th phase.
for all samples in d_i do
    if d_i(n − 1) < d_i(n) and d_i(n) > d_i(n + 1) then
      d_i'(j)=d_i(n); /* record the amplitude in array d_i'(j)*/
    end
end
t_i=mean(d_i'(j)); /*t is mean of all the elements in d_i'*/
for all samples in d_i'do
    if d_i'(j) > t_i then
      p_i = j; /*mark j^th element as a peak*/
    end
end
```

Fig. 6 shows resulting diagram after applying the peak detector on array d.

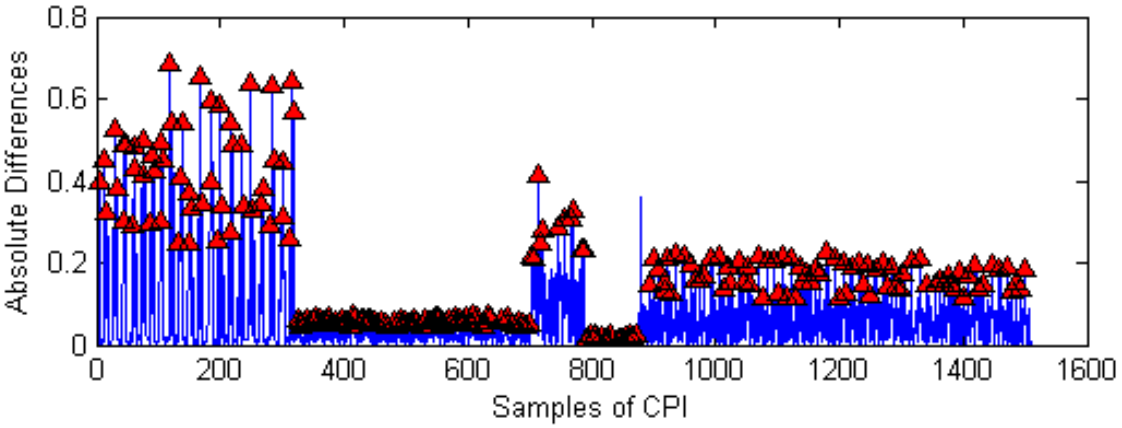

Fig. 6. Diagram following the application of the peak detector.

5) Similarity analyser

The similarity analyser has three different parts, each with a measure to ascertain the originality of the program in execution. The three parts are the phase, peak and classifier. The first part is used to verify if the number of known phases is the same as the number of phases in the executed program. Any mismatch shows that the number of function calls differ, signifying an insertion or deletion. The second part compares the number of identified peaks within each phase. It must be noted that any difference in the number of peaks does not necessarily mean the program is compromised, however a variation in CPI has taken place. The first two parts of the analyser become useful when the system has completed a cycle. The final part of the analyser uses the SOM/Fuzzy C-means to measure similarity between known programs and programs currently executed.

SOM based classifier

The basic principle of the SOM is to adjust the weight vectors until the neurons represent the input data, while using a topological neighbourhood update rule to ensure that similar prototypes occupy nearby positions on the topological map. PC values extracted from the program execution trace, corresponding to the peaks in the trace are used as inputs to the SOM during training and testing. For a given network with k neurons and N-dimensional input vector $\mathbf{K^i}$, the distance from the j^{th} neuron with weight vector $\boldsymbol{w_j}$ ($j<k$) is given by

$$D_j^2 = \sum_{l=1}^{N} \left(K_l^i - w_{jl} \right)^2 \tag{3}$$

where $\boldsymbol{w_{jl}}$ is the l^{th} component of weight vector $\boldsymbol{w_j}$. The vector components of the winning neuron $\boldsymbol{w_k}$ with minimum distance D_k are updated as follows, where $\eta \in (0,1)$ is the learning rate.

$$\Delta w_k = \eta \left(K^i - w_k \right) \tag{4}$$

Updates are only carried out during the training phase. Additionally, for every neuron in the network we maintain two extra parameters; the minimum and maximum distances of all input vectors associated with any particular neuron.

After training, the next step is to associate each of the network neurons with the corresponding program or sub-program. In this work, we use Vector Quantization (VQ) [11] to assign labels to the trained neurons in the network as follows:

- Assign labels to all the input training data. The label is an identifier for the program from which the training data has been extracted from.
- Find the neuron in the network with the minimum distance to the labelled input data.
- For each input data maintain the application label, the corresponding neuron and the distance measured. The distance is maintained as a tie breaker for applications that share similar address space.

For each network neuron, we estimate the number of programs that are associated with that neuron. If only one program is associated with a neuron and the number of data points exceeds 5% of the total number of program data points, the neuron is exclusively assigned to that very program. For all programs with more than 5% of data points associated with a neuron, we create a codebook with an entry for the neuron, and the corresponding

programs, each with its distance range (i.e. minimum distance and maximum distance).

Fuzzy C-means based classifer

Fuzzy c-means (FCM) is a method of clustering which allows one piece of data to belong to two or more clusters [12]. This method is based on minimisation of the following function:

$$J_m = \sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^{m} \left\| x_i - c_j \right\|^2 \tag{5}$$

where $1 \le m \le \infty$, u_{ij} is the degree of membership of x_i in the cluster j, x_i is the i^{th} of d-dimensional measured data, c_j is the d-dimension centre of the cluster, and ||*|| is any norm expressing the similarity between any measured data and the centre.

Fuzzy partitioning is carried out with aim of optimization of the objective function shown in (5), every iterative optimisation update membership u_{ij} and the cluster centres c_j by:

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\left\| x_i - c_j \right\|}{\left\| x_i - c_k \right\|} \right)^{\frac{2}{m-1}}} \tag{6}$$

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^{m} x_i}{\sum_{i=1}^{N} u_{ij}^{m}} \tag{7}$$

Similar to the SOM based classifier, the Fuzzy C-means based classifier will also record associate each of the cluster centre with the corresponding program or sub-program. In addition, assign labels to all the input training data with the minimum distance to the labelled input data. For each phase of program, the minimum and maximum distances to the corresponding centre of peak are also recorded. The information is maintained as a reference for testing unknown programs.

IV. Experimental Results

In this work, we use a Keil MCBSTM32F200 evaluation board equipped with an ARM 32-bit Cortex-M3 processor-based microcontroller for our experimental testing [25]. The ARM 32-bit Cortex-M3 processor-based microcontroller comes with 128KB of on-chip RAM and 2MB of external SRAM, which is widely used in many high-performance low-cost embedded applications (e.g. industrial control systems and wireless networking and sensors). We also use a combination of KEIL μVision IDE, and ULINKpro Debug and Trace Unit [26] to download the program and trace the instructions executed in the microcontroller, where high-speed data and instruction trace are streamed directly to the host computer allowing off-line analysis of the program behaviour [26]. It is worth noting that the experimental platform is a typical low cost ARM-based embedded development board, but it has very limited memory space, especially when the tracing port is enabled, there is only 1MB external SRAM can be used by users. Thus we can only analyse a limited number of programs at a time, and the complexity of the tested programs are also limited. These limitations fall within the scope of our initial embedded architecture, expected to have minimal memory, power and computational resources. The concept presented here is scalable; as the available resources increase, the complexity of applications can also be increased. In this work, MATLAB implementation is used for proof of concept prior to hardware implementation. Since the proposed method could use the debug interface of the processor together with a dedicated hardware module providing the classifier's functionality to extract data and identify the system behaviour, it is independent of the processor's architecture or Operating System's kernel.

A. Benchmark Test Suite

In the proposed work, five algorithms from the automotive package of the MiBench benchmark suite [27] are selected, those five algorithms are used to train and test the proposed analysers and also there is a unknown program is only used in the testing. Details of the used benchmarks are presented in Table I.

As can be seen from Table I, the six benchmarks are set with different parameters and performing various functions. For instance, the benchmark "*angle conversion*" is used to perform radians and degree conversion, where "NUM_TEST" indicates the number of sets of input test data stimuli. Overall, they do not only have different complexities and characteristics, but also contain similar sub-functions, which make them suitable test candidates for the proposed experiments.

In order to train with all five benchmarks, they are mixed together to form a *new program*, where each benchmark is treated as a separate function call. The new program is executed twice in order to generate enough training samples. For testing, a random function call generator is used to switch between benchmarks form the test samples. Various combinations of the five algorithms are used as compromised program, where AC, BC, CF, RN, and SR are executed twice in each combination. In addition to the above, we also use an "unknown" algorithm "Fibonacci Series (FS)" to replace AC, BC, CF, RN, and SR to represent another five compromised programs for testing. Since the FS algorithm consists of some similar sub-functions to the known algorithms, this experimental setup is more suitable for evaluating the proposed system. At the beginning of the test, we run the original program five times separately in the embedded platform, and all the program execution trace profiles are stored into five different files respectively. One of the files (i.e. the training file) is used for training the classifiers and the remainder are used for testing.

TABLE I
DETAILS OF THE USED BENCHMARKS

Benchmarks	Description	Parameters
angle conversion (AC)	Radians and Degree Conversion	Degree range: [0 360] Radians range: [0 2π]
bit count (BC)	Bit Counter	NUM_TESTS: 500
cubic function (CF)	Solve a cubic polynomial	NUM_TESTS: 8
random numbers (RN)	Random Number Generator	NUM_TESTS: 3000
square roots (SR)	Square Root Calculation	NUM_TESTS: 10000
Fibonacci Series (FS)	Fibonacci Series Generator	NUM_TESTS: 232

B. SOM classifier

The start and end locations of each peak can be used to select a series of PC addresses, and this forms an input vector with 1×2048 elements, the vector values are then normalised before fed into the SOM-based classifier. The maximum number of nodes and iterations for the SOM are set to 20 and 1000 respectively. The outputs of the training are network weights, a record of each phase, the corresponding neuron(s), and associated minimum and maximum distance for the phase. A statistical table for each phase and estimated outputs for each peak are generated after the training process. The same process is repeated during the testing.

C. Fuzzy C-means classifier

Similar to the SOM classifier, the same PC addresses of each peak are used as the input vector of the Fuzzy C-mean classifier, where each input vector contains 1×2048 elements. At beginning of the processing, the expected the number of clusters is set to 5, and the initial centres of each cluster are randomly initialised. The maximum number iterations and minimum amount of improvement are set to 100 and 10^{-5} respectively. The outputs of the training are matrix of final cluster centres, where each row provides the centre coordinates and final fuzzy membership function matrix, and values of the objective function during iterations. Records of each phase, the cluster, and associated minimum and maximum distance to its centre. A statistical table for each phase and estimated outputs for each peak are generated after the training process. The same process is repeated during the testing.

D. Evaluation metric

The measurements of the evaluation mainly includes correct recognition rate (true positive (T_p) and true negative (T_n)), rate of misclassified samples (false positive (F_p)), and rate of samples incorrectly classified as

unknown (false negative (F_n)). Based on the measurements, accuracy, precision and recall rates for the proposed system can be calculated.

Accuracy

It is the rate of correctly labelled samples, which can be calculated by (T_p+ T_n) / total number of samples.

Precision

It is the rate of positively labelled samples whose labels are correct, which measures the classifier's resistance to false positives and can be calculated by T_p / (T_p +F_p).

Recall

It is the rate of samples that should have been positively labelled that are correctly positively labelled, which measures the classifier's resistance to false negatives and can be calculated by T_p / (T_p +F_n).

A classifier's precision and recall results provide insight into what types of errors the classifier tends to make, rather than only reporting the number of misclassified samples.

E. Experimental results

In this experiment, the proposed system classifies the programs' peaks and phases into different categories, where the known peaks and phases will be assigned their corresponding names and unknown ones will be labelled as '-1'. For testing, each of the test files (27 files in total) is fed into the trained classifier to generate individual output files, and the 27 files are divided into three categories: 1) programs with original function call sequence; 2) programs with various function call sequences (including only known benchmarks); 3) Programs with various function call sequences (include known and unknown benchmarks).

1) Programs with original function call sequence

In this category, there are total 5 programs, which include 1144 peak samples. Overall, the proposed SOM and Fuzzy C-means based systems have 95.6% and 100% successful identification rates for the peaks respectively. Results of accuracy, precision and recall rates for each program are illustrated in Fig. 7.

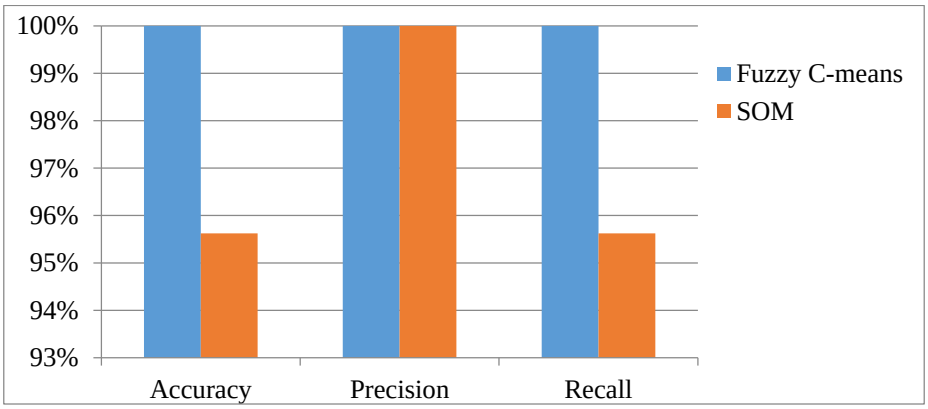

Fig. 7. Results of accuracy, precision and recall rates for category 1.

2) Programs with various function call sequences (including only known benchmarks)

In this category, there are total 10 programs, which include 2070 peak samples. Overall, the proposed SOM and Fuzzy C-means based systems have 90.9% and 99% successful identification rates for the peaks respectively. Results of accuracy, precision and recall rates for each program are illustrated in Fig. 8.

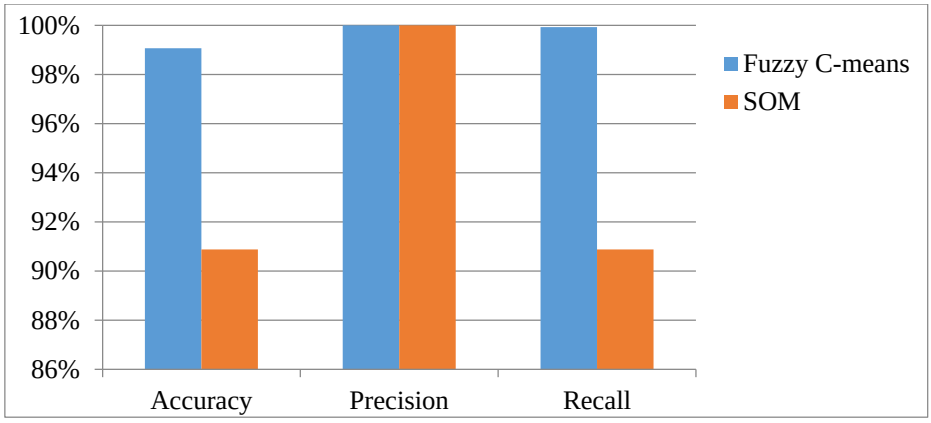

Fig. 8. Results of accuracy, precision and recall rates for category 2.

3) Programs with various function call sequences (include known and unknown benchmarks)

In this category, there are total 12 programs, which include 2661 peak samples. Overall, the proposed SOM and Fuzzy C-means based systems have 90.7% and 97% successful identification rates for the peaks respectively. Results of accuracy, precision and recall rates for each program are illustrated in Fig. 9.

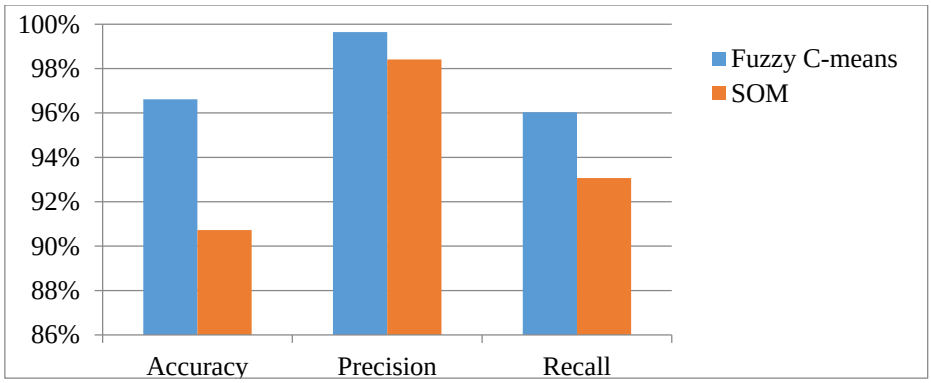

Fig. 9**.** Results of accuracy, precision and recall rates for category 3.

In general, as the complexity of the test categories are varied, both algorithms in the first category have the best accuracy, precision and recall rates. In contrast, the accuracy, precision and recall rates of both algorithms in the second and third categories are relatively lower than the first one. As the types and the lengths of each tested program in the last two categories are different, which causes the resulting rates of each program have relatively higher variance than the first one. The reason of most failure cases is due to the fact that some of the extracted PC patterns in the testing programmes are different to the patterns in the training programme, which further causes the misidentification of the similarity analysers. Especially, when the patterns of the unknown programme are very similar to the known programme (i.e. scenario 3 in the testing database), the recognition rate is further decreased. Overall, when comparing the performance of the Fuzzy C-means based algorithm with the SOM based algorithms, the former has better performance than the later. Overall, the Fuzzy C-means based algorithm has achieved 98.7% accuracy; however, the SOM based algorithm only can achieved 90.9% accuracy. This is due to SOM based classifier being much affected by the number of variables and clusters in the input data, whereas, the Fuzzy C-means based classifier is less affected by these factors.

It is worth noting that unlike other software-based [17] [18] and hardware-based [1, 5] approaches, our work is independent of the processor's architecture or operating system's kernel, thus making it compatible with most modern embedded systems. In [1], the hash-checking of all CFG significantly degrade the processor's performance. However, our scheme maintains the original length of critical path of the processor and monitors the executing programme in parallel through the debug interface, without performance penalty. Although the fastest intrusion detection system is proposed in [5], only system call sequence is used for the comparison of correct and compromised systems. In contrast, our system does not only monitor system call sequence, but also checks instruction sequence within the system call, which captures both coarse-grained and

fine-grained programme behaviour in a hierarchical manner. Hence, the proposed work is particularly suitable for providing possible security solutions to commercial off-the-shelf (COTS) products, where the products have many restrictions on modifying their internal programs or hardware architectures. The proposed system can be run on a non-intrusive debug facility, a non-intrusive infrastructure that is generally used during device software development at present in all production devices, that connects to the targeted embedded device through a debug interface [28, 29], which means that the proposed system would not affect the performance of the monitored embedded system in terms of additional memory and processor usage. In one of the authors' previous works [30], an implementation of the conventional SOM on a Xilinx Virtex-4 with 40 neurons required only 22.1% of the available 5,184 Kb Block RAM. Again, the Virtex-4 design implementation clocked at 25MHz could train with approximately 10,000 patterns per second. As a result of this, the hardware implementation of the SOM produces a significant speed improvement, which is 30 times faster than the original SOM implemented on a state-of-art PC [30]. Hence, the preferred implementation is to follow a hardware acceleration approach that facilitated rapid identification processing suitable for real-time execution.

V. Conclusion

In this paper, we have presented an approach for detecting compromised programs by analysing CPI and PC from an embedded system. Through monitoring the processor's CPI, we detect changes in the function call and CFG, and then analyse the PC values within each CFG using SOM and Fuzzy C-means based classifiers separately. The experimental results demonstrate that the proposed algorithm can be used to detect abnormal behaviour in embedded devices. Results achieved in our experiment show that the proposed SOM and Fuzzy C-means based systems can both be employed to identify unknown behaviours not in the training set, with 90.9% and 98.7% accuracy respectively. The proposed work provides protection at different levels for embedded architectures such as function call sequence, internal control flow and instruction stream within each function. Since the main aim of this research work is to implement a real-time security solution for complex embedded computer architectures, more evaluation on realistic attacks for the proposed algorithms will further be investigated. For evaluation parameters of real-time detection system, the proposed algorithm can also be implemented with a soft-core processor on FPGA as part of an on-line protection system. The online implementation will have the capability of extracting execution trace from customised tracing interfaces directly located on the processor, determine the behaviour in real-time, and subsequently halting the program to prevent any harmful effect on the embedded system architecture. The idea of this work can be extended to detect abnormal behaviour of multi/many-core processor based embedded system by applying the proposed system on the each core.

Acknowledgment

The authors gratefully acknowledge the support of the EU ERDF INTERREG IVA 2 Mers Seas Zeeën Cross-border Cooperation Programme – SYSIASS project: Autonomous and Intelligent Healthcare System (project's website http://www.sysiass.eu/). This work was also supported by the EU Project COALAS (INTERREG IVA France (Channel) – England European cross-border co-operation programme, which is co-financed by the ERDF).

References

[1] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha, "Secure embedded processing through hardware-assisted run-time monitoring," in *Proceedings Design, Automation and Test in Europe*, 2005, pp. 178-183

[2] P. Dongara and T. N. Vijaykumar, "Accelerating private-key cryptography via multithreading on symmetric multiprocessors," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2003, pp. 58-69.

[3] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *44th ACM/IEEE Design Automation Conference*, 2007, pp. 9-14.

[4] H. Handschuh, G.-J. Schrijen, and P. Tuyls, "Hardware Intrinsic Security from Physically Unclonable Functions," in *Towards Hardware-Intrinsic Security*, A.-R. Sadeghi and D. Naccache, Eds., ed: Springer Berlin Heidelberg, 2010, pp. 39-53.

[5] M.Rahmatian, H. Kooti, I. G. Harris, and E. Bozorgzadeh, "Hardware-Assisted Detection of Malicious Software in Embedded Systems," *IEEE Embedded Systems Letters*, vol. 4, pp. 94-97, 2012.

[6] Shane S. Clark, Benjamin Ransford, Amir Rahmati, Shane Guineau, Jacob Sorber, Kevin Fu, *et al.*, "WattsUpDoc: Power Side Channels to Nonintrusively Discover Untargeted Malware on Embedded Medical Devices," in *Proceedings of USENIX Workshop on Health Information Technologies*, 2013.

[7] D. Garcia-Romero and C. Y. Espy-Wilson, "Automatic acquisition device identification from speech recordings," in *IEEE International Conference onAcoustics Speech and Signal Processing* 2010, pp. 1806-1809.

[8] C. Hanilci, F. Ertas, T. Ertas, and O. Eskidere, "Recognition of Brand and Models of Cell-Phones From Recorded Speech Signals," *IEEE Transactions on Information Forensics and Security* vol. 7, pp. 625-634, 2012.

[9] Y. Kovalchuk, K. D. McDonald-Maier, and G. Howells, "Overview of ICmetrics technology-security infrastructure for autonomous and intelligent healthcare system," *International Journal of u- and e-Sevice, Science and Technology*, vol. 4, pp. 49-60, 2011.

[10] X. Zhai, K. Appiah, S. Ehsan, W. Cheung, G. Howells, H. Hu, *et al.*, "Detecting Compromised Programs for Embedded System Applications," in *Architecture of Computing Systems – ARCS 2014*. vol. 8350, E. Maehle, K. Römer, W. Karl, and E. Tovar, Eds., ed: Springer International Publishing, 2014, pp. 221-232.

[11] T. Kohonen, "Learning vector quantization," in *The handbook of brain theory and neural networks*, A. A. Michael, Ed., ed: MIT Press, 1998, pp. 537-540.

[12] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*: Plenum Press, New York, 1981.

[13] M. Deng, K. Wuyts, R. Scandariato, B. Preneel, and W. Joosen, "A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements," *Requirements Engineering*, vol. 16, pp. 3-32, 2011/03/01 2011.

[14] K. D. Maier, "On-chip debug support for embedded Systems-on-Chip," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, 2003, pp. V-565-V-568 vol.5.

[15] C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn, *et al.*, "Dynamic path-based software watermarking," in *Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation*, 2004, pp. 107-118.

[16] C. Collberg and C. Thomborson, "Software watermarking: models and dynamic embeddings," presented at the Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, San Antonio, Texas, USA, 1999.

[17] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," presented at the Proceedings of the 18th conference on USENIX security symposium, Montreal, Canada, 2009.

[18] X. Wang, Y.-C. Jhi, S. Zhu, and P. Liu, "Behavior based software theft detection," presented at the Proceedings of the 16th ACM conference on Computer and communications security, Chicago, Illinois, USA, 2009.

[19] R. Yang, Z. Qu, and J. Huang, "Detecting digital audio forgeries by checking frame offsets," presented at the Proceedings of the 10th ACM workshop on Multimedia and security, Oxford, United Kingdom, 2008.

[20] Y. Panagakis and C. Kotropoulos, "Telephone handset identification by feature selection and sparse representations," in *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2012, pp. 73-78.

[21] A. Swaminathan, Y. Mao, M. Wu, and K. Kailas, "Data Hiding in Compiled Program Binaries for Enhancing Computer System Performance," in *Information Hiding*. vol. 3727, M. Barni, J. Herrera-Joancomartí, S. Katzenbeisser, and F. Pérez-González, Eds., ed: Springer Berlin Heidelberg, 2005, pp. 357-371.

[22] P. Boufounos and S. Rane, "Secure binary embeddings for privacy preserving nearest neighbors," in *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2011, pp. 1-6.

[23] K. Patel, S. Parameswaran, and S. Seng Lin, "Ensuring secure program execution in multiprocessor embedded systems: A case study," in *2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2007, pp. 57-62.

[24] M. Annavaram, R. Rakvic, M. Polito, J. Bouguet, R. Hankins, and B. Davies, "The fuzzy correlation between code and performance predictability," in *the 37th International Symposium on Microarchitecture (MICRO)*, 2004, pp. 93-104.

[25] STMicroelectronics. *STM32F207IG Data Sheet*. Available: http://www.st.com/internet/mcu/product/245085.jsp

[26] KEIL. *Keil µVision IDE Data Sheet*. Available: http://www.keil.com/uvision/

[27] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE International Workshop onWorkload Characterization*, 2001, pp. 3-14.

[28] A. B. T. Hopkins and K. D. McDonald-Maier, "Debug support strategy for systems-on-chips with multiple processor cores," *IEEE Transactions on Computers*, vol. 55, pp. 174-184, 2006.

[29] A. B. T. Hopkins and K. D. McDonald-Maier, "Debug support for complex systems on-chip: a review," *IEE Proceedings -Computers and Digital Techniques*, vol. 153, pp. 197-207, 2006.

[30] K. Appiah, A. Hunter, P. Dickinson, and M. Hongying, "Implementation and Applications of Tri-State Self-Organizing Maps on FPGA," *IEEE Transactions on Circuits and Systems for Video Technology,*, vol. 22, pp. 1150-1160, 2012.