ICmetrics for Low Resource Embedded Systems

Yevgeniya Kovalchuk, Huosheng Hu,
Dongbing Gu, Klaus McDonald-Maier
School of Computer Science & Electronic Engineering
University of Essex
Colchester, UK
yvkova@essex.ac.uk; hhu@essex.ac.uk;
dgu@essex.ac.uk; kdm@essex.ac.uk

Gareth Howells
School of Engineering and Digital Arts
University of Kent
Canterbury, UK
W.G.J.Howells@kent.ac.uk

***Abstract*—The ICmetrics technology is based on extracting features from digital devices' operation that may be integrated together to generate unique identifiers for each of the devices or create unique profiles that describe the devices' actual behaviour. Any changes in these identifiers (profiles) during consequent devices' operation would signal about a possible safety or security breach within the electronic system. This paper explores the program counter (PC) of a processor core as a potential source for ICmetrics features and discusses several methods of feature values acquisition with the aim to achieve a maximum level of information gain with a minimal impact on a system's performance. The main finding of this study is that while isolated PC values may not always allow to generate a stable identifier (profile) for a device that would distinguish the device from the rest in the considered set, the PC sequences and frequencies in the execution flow may serve as suitable ICmetrics features, which has yet to be tested in complex scenarios.**

ICmetrics; security; encryption; embedded systems; autonomous systems

I. INTRODUCTION

There is an ever growing number of domains and applications where ensuring security and safety of electronic devices' operation and communication is essential, however not easily linked to human input (e.g. passwords, biometrics) to facilitate security. Take for example environments where machine-machine communication occurs with no human intervention [1, 2] or medical and/or assistive devices (such as electronic wheelchairs), where human's input is difficult or not reliable (e.g. due to a patient's disability).

This paper presents some recent results of developing the ICmetrics technology – the technology of generating unique identifiers that may serve as encryption keys directly from characteristics of electronic systems' behaviour. As opposed to physical characteristics of integrated circuits (ICs) used in the Physical Unclonable Functions technology (PUF) [3], ICmetrics is based on features derived from the operation of ICs (software executing on programmable structures, circuits, sensors, communication peripherals, etc.) and their interaction with the environment. In our earlier work, we have presented the theory of the technology and formalized the tasks to implement it in real systems [4]. In this study, we are looking to investigate if the ICmetrics technology that has previously been designed for non-intrusive debug support architectures [5] can be deployed using intrusive methods on low resource embedded systems that do not feature a dedicated non-intrusive trace interface [6, 7].

In particular, we analyse two intrusive tracing methods for observing execution characteristics, namely (1) single stepping and (2) sample based tracing. At this early stage of our research, we have tested the methods only on obtaining the program counter (PC) values as a potential ICmetrics feature source. Our choice in favour of the PC is based on the observation that registered PC values and their sequences change slightly (if at all) for a certain devices during its operation in the given environment, but vary significantly across devices of different configuration or (if the configuration is identical) while operating in different environmental conditions.

Based on a detailed analysis of the PC logs presented in this paper, we provide some recommendations on designing tracing methods and suitable sources of ICmetrics features. We also propose alternative data analysis techniques in order to build a strong and effective security infrastructure.

In the remaining sections of the paper, we detail our experimental platform and methods of obtaining data for ICmetrics features; give interpretation of the results we have achieved when analysing the PC logs, and highlight future directions for ICmetrics research.

II. EXPERIMENTAL SETUP

For this study, we have employed a low resource embedded system based around an ARM7 processor core, in particular an Atmel AT91SAM7S256 microcontroller [8] and 64Kbytes SRAM memory. We have used the combination of Eclipse [9], Open On-Chip Debugger (OOCD) [10], and JTAG programming port for programming the microcontroller, as well as tracing the programs' execution.

We have used two intrusive tracing methods to log the PC: the single stepping method has provided us with the benchmark data against which we evaluated the second method, which is sampling. While these methods affect execution times, they do not change the execution flow (for the code executed here), meaning that the proposed methods provide the same PC values as would have been obtained with non-intrusive methods. To register the PC values, both methods halt the CPU by issuing OOCD commands [10] via a telnet port. The difference between the methods lies in the frequency and completeness of obtaining data. The single stepping tracing method logs every single CPU instruction, while the sampling method does this only at regular intervals, at the predetermined rate of 50Hz in this case. We have chosen this sampling rate since it is the highest throughput that our JTAG programming port supports and we aimed at testing a fast logging method to be practical in real time applications. Such settings mean that the single stepping method provides complete profiling of the program execution, whilst the sampling method – only its approximation. While the single stepping method is preferred to gain full profiles, it is very slow. The sampling method on the other hand allows speeding up logging considerably (thus, affecting the systems normal executing much less); however it does mean that not all data are logged and significant parts of the executing profile may not be identified. Below, we compare the two methods in more detail, explore their suitability for obtaining ICmetrics features, and provide recommendations on designing a better tracing method to be useful in ICmetrics research.

Since at this early stage we have only been interested to see if the PC could be used as a potential source for ICmetrics features, and also to compare the two methods of obtaining data, we have employed basic low complexity software routines to serve as a source of data so as to achieve visually representative and easily interpretable analysis results. More specifically, we have chosen several algorithms from the automotive package of the MiBench suite of benchmark algorithms [11] to design our programs, namely: angle conversion (AC); bit count (BC); cubic function (CF); and square roots (SR). In addition, we have included a program generating random numbers (RN). To increase the probability of logging all distinct PC values when using the sampling tracing method, we have run each program several times and recorded more PC values as compared to the single stepping method. The number of times to run each program has been determined to achieve the execution times comparable to those of the single stepping logging.

III. Data Analysis

Table I details a summary of statistics performed over the raw data obtained from the log files. In particular, "total steps" and "total samples" provide the total number of PC values recorded during the entire sessions of tracing by the single stepping and sampling methods respectively. Since a program may use the same memory address several times during its execution flow, we have calculated how many distinct PC values are present in the program profiles recorded by each of the two methods. This can be seen in "distinct @ step" and "distinct @ sam", where "sam" refers to the sampling tracing method.

TABLE I. Statistics of the Programs' Profiles

Param.\Program	AC	BC	CF	RN	SR
total steps	263565	104971	205482	138011	102131
total samples	758952	205315	1424747	202968	268034
distinct @ step	429	44	1695	52	94
distinct @ sam	429	44	1695	48	93
unique @ step	132	0	1376	7	16
unique @ sam	132	0	1376	7	16
total routine	13338	8964	14509	9002	4929
distinct routine	62	11	274	6	16
distinct interval	64	8	258	6	16
distinct routine incl. branches	37	4	200	4	8

For ICmetrics research, we are also interested in how the program profiles differ from each other. Therefore, we have further refined the number of distinct PC values by finding the number of addresses that occurred in the profile of a certain program, but not in the profiles of the rest of the programs. This is reflected in "unique @ step" and "unique @ sam". Note that despite the sampling tracing method, although being very close, has not always managed to trace all distinct addresses ("distinct @ step" as compared to "distinct @ sam"), it still catches exactly the same number of unique addresses for each of the programs ("unique @ step" and "unique @ sam").

The remaining figures in Table I are explained in the following sections, where we analyse the programs' profiles in more detail in order to compare the two tracing methods and to determine which potential ICmetrics features can be derived from the PC logs.

A. *Address Maps*

Figure 1 compares program profiles obtained with the two logging methods. It plots each occurrence (axis X) of the PC value (axis Y) during the program runs.

Note the non-repetitive PC values present at the beginning and end of the single step profiles, which result in their more frequent presence in the sampling profiles. These are addresses used for initializing variables. Since the sampling tracing method is designed to run the programs many times (to give more chances for each address to be recorded), the addresses corresponding to variable initialization appear in the sampling profiles often. Baring this in mind, it can be suggested from visual inspection of the address maps plotted in Figure 1 that both tracing methods provide similar program profiles for each of the five programs (i.e. each graph on the left is similar to its counterpart on the right).

At the same time, the profiles of the five programs differ from one another (the five pictures in the left column are different, so they are in the right column).

These two observations on similarities and differences between the program profiles suggest that:

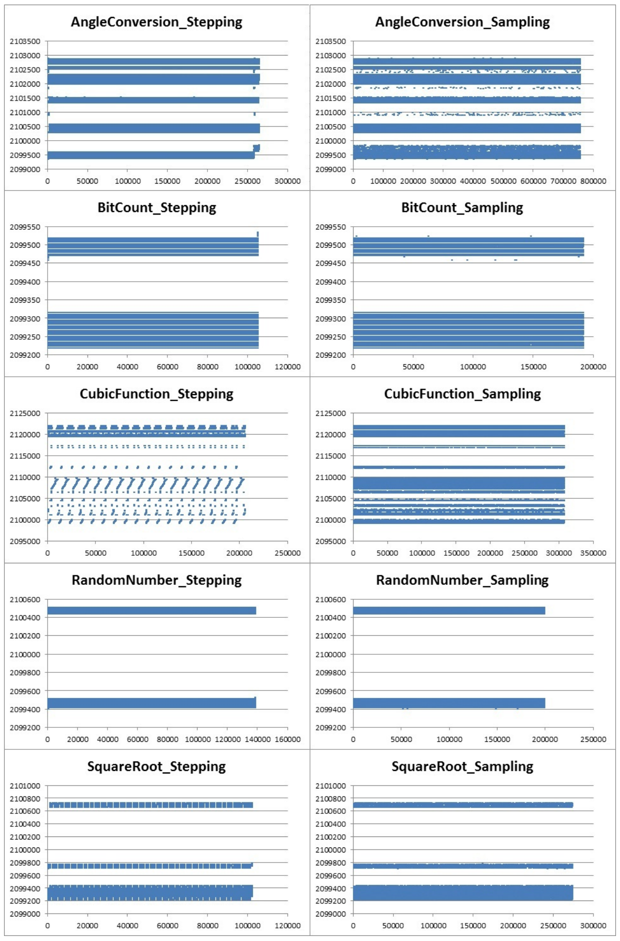

Figure 1. Program profiles by the two tracing methods.

1) the sampling tracing method can be used to approximate a program profile very closely to the complete one and therefore serve as a means for logging the PC as a potential source of ICmetrics features;
2) the PC might be a useful source for ICmetrics features, since it is possible to visually distinguish program profiles from each other based on the PC values only.

However, a more detailed analysis is required to verify these suggestions. The following sections provide better understanding of the data.

B. Information Gain

Since we have established that the sampling method could potentially provide us with the complete set of distinct addresses present in a program's profile (see "distinct @ step" and "distinct @ sam" in Table I), it would be interesting to determine the speed of gaining information during the sampling tracing. This exercise would allow us to justify and quantify gains in time versus losses in PC values when applying the sampling tracing method instead of the single stepping.

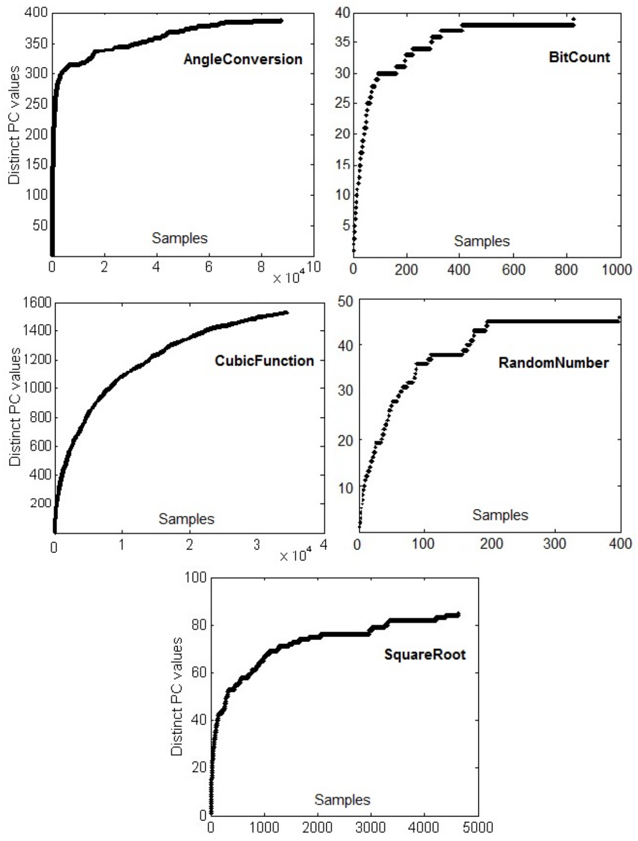

Figure 2. Information gain with the sampling method.

By plotting the number of samples against the number of unique addresses as picked up by the sampling tracing method (Figure 2), we have found that the information gain graphs for all programs considered in this study take a form of the logarithmic function. In other words, the longer the sampling logging is running, the lesser number of additional addresses is obtained; and at the same time, the more samples are taken, the closer is the profile to the complete one.

To demonstrate the advantage the sampling tracing method has over the single stepping method in terms of logging times, we have calculated the minimal times required to run sampling tracing until 90% (as an example) of all distinct PC values recorded in the single stepping logs are captured. Table II provides the results of this experiment. In particular, "distinct @" is the number of distinct PC values that is required to record with the sampling tracing method. "Samples needed" is the number of samples we have established it took the sampling method to capture the required number of distinct values. "Step time" is the time (hr:min:sec) of executing the programs when using the single stepping logging. "Sample time" is the time it took the sampling logger to record the required number of distinct addresses. Finally, "time saving" demonstrates what percentage of time as compared to the total "step time" it would allow to save if using the sampling logging instead of the single stepping.

TABLE II. STATISTICS ON TIMING AND INFORMATION GAIN

Param.\Program	AC	BC	CF	RN	SR
distinct @	386	39	1525	46	85
samples needed	87578	823	34447	398	4632
step time	1:28:00	0:35:00	1:08:00	0:46:00	0:34:00
sample time	0:32:28	0:00:16	0:11:32	0:00:08	0:01:33
time saving	63.1%	99.2%	83.0%	99.7%	95.4%

It can be noticed from Table II that time savings although vary across the programs, are very significant in all cases (at least 63% and up to 99.7% in our case). Matching the results in Table II to the program profiles depicted in Figure 1, it can be suggested that time savings depend on the complexity of the algorithm and sequence of its flow (i.e. how many and when distinct PC values appear in a program's execution profile). This means that the results may be not that high if more complex programs are involved (e.g., if there is a higher ratio of distinct PC values as compared to the total number of PC values involved in a program's execution flow).

These findings provide justification for our earlier suggestion that a sampling based tracing methods could be used for program profiling in embedded systems in order to reduce overheads related to obtaining data. Note however, that the main requirement for ICmetrics features is that they should allow for separation of considered embedded systems in the feature space. It is evident from Table I that the PC values themselves not always allow for distinguishing a program from the rest in the operational set (note zero values in "unique @ step" and "unique @ sam" for BC, meaning that there are no PC values that are unique for this program).

And yet, the PC logs could still serve as a potential source of ICmetrics features. For example, the uniqueness of each program profile could potentially be derived from their execution flows (i.e. not from separate PC values, but their sequences), and also frequencies of PC occurrences. It must be noted however that sampling at a certain frequency (i.e., our sampling tracing method) may not allow to reconstruct PC frequencies and sequences as they actually appear during the program execution flow.

To address this issue, in the next section we analyse PC sequences as recorded in the single stepping logs in order to determine if a better sampling method can be designed in such a way so it is useful for ICmetrics research. Section *D* presents the frequency analysis of the single stepping PC logs in order to estimate if ICmetrics features can be obtained from the PC frequency domain.

C. Branch Analysis

The PC in traditional processors is incremented sequentially after fetching a program instruction. In the case of an ARM7, i.e. the experimental setup presented here, the PC increments by 4 bit, and less often by 2 bit. However, there are certain instructions (e.g. branches, jumps, subroutine calls and returns) that interrupt the sequence by placing a new value in the PC. This results in larger changes in the log files. Here, we refer to the intervals of sequential increments in the PC values (i.e. by for 2 or 4 bits) as to "routines", while the points where a higher increment occurs (more than 4 bits) we call "branch" points.

We have extracted distinct cases of all routines present in the PC logs, noting their start and end addresses, the number of steps (increments) in between, and frequency of their occurrence. Depending on where branching points happen, we have noticed 4 types of routines in the program profiles: (1) without branching points (represented by routine R_1 in Figure 3); (2) with an early finish (routine R_2); (3) with a later start (routine R_3); (4) with both later start and early finish (routine R_4).

As can be seen from Figure 3, it is important to note all branching points within a larger routine (e.g. points A, B, C, D, and E for the routine R_1) as they split the later into smaller intervals that score different frequency values (intervals [A, B] and [D, E] occur with the frequency of 2, while the frequency of interval [C, D] is 3 and for interval [B, C] it is 4). The frequency of such intervals could potentially serve as an ICmetrics feature as it represents particularity of a program operation. For example, tracing two identical devices running the same software code but interacting with different environments could result in two different profiles since different subroutines would be executed during the execution flow within the program code depending on the characteristics of each environment, potentially resulting in different interval frequencies.

This observation also suggests an alternative, more intelligent, way of tracing the PC. In particular, samples can be taken at branch points instead of at regular intervals or at every single step, which is what modern debug support architectures present in advanced embedded processors facilitate [5]. Such approach combines the advantages of the two methods we propose here, overcoming their major pitfalls at the same time. More specifically, it would allow for obtaining full program profiles (as with the single stepping method), but much faster and with less intrusion (i.e. by sampling at branch points only).

Note, "distinct routine" in Table I refers to the distinct number of routines extracted from the single steps profiles as they occurred during the programs' operation despite their start or end in relation to bigger routines that involve them (i.e. we score the occurrences of routines R_1, R_2, R_3, and R_4, as in Figure 3, separately). In "distinct interval" we count the numbers of the smallest intervals found within larger routines (i.e. we score the occurrences of intervals [A, B], [B, C], [C, D], and [D, E]). Finally for "distinct routine incl. branches", we include smaller routines as part of counting the largest routines that involve them (i.e. we add occurrences of R_2, R_3, and R_4 to the number of R_1 occurrences). As can be seen from the results in Table I, all five programs behave differently; there is no pattern in relationship between the numbers in the last 3 rows. This confirms once again that the frequency of the smallest parts of routines is a good candidate feature for ICmetrics.

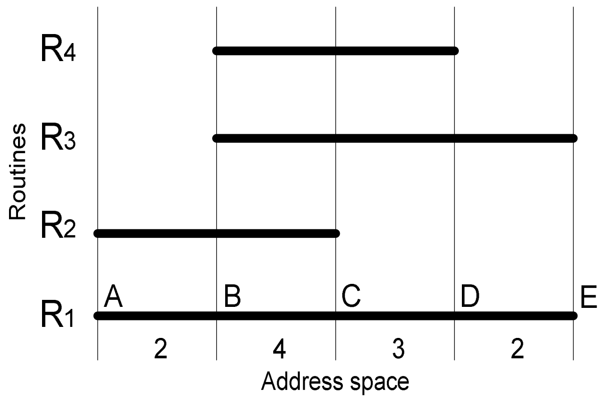

Figure 3. The four types of routines

D. *Frequency Matrices*

To get further insight on how the program profiles relate to each other and to establish if potential ICmetrics features can be derived from the PC frequency domain, we have brought all five programs into one space and built frequency matrices noting how many times a certain address occurred in the profile of each program. We have done this for both the single step and sampling logs, omitting entries with all zeros (i.e. when an address is not present in any of the program profiles).

By normalising the values in the matrices in different ways, we have obtained several colour maps that vividly show relationships between the programs. We used the grey colour scale to represent intensity of a certain feature as present in the profiles of each of the programs at each address point, from white colour corresponding to zero (a feature is not present) to black colour representing the highest value of the feature (Figures 4). The programs used 1864 distinct addresses all together; they are represented on axis Y from the lowest address at the top.

Figure 4a plots frequencies of address usage by each of the programs (i.e. how many times a certain address has occurred during the program execution as logged in the single step logs) to demonstrate once again (as in Figure 1) how different the program profiles are. Note, the more intense grey colour at a certain address is, the higher is the frequency of this address being used by the given program as compared to the rest of the programs. As there are spikes of high frequencies in the matrices (up to 6400 occurrences), we normalized the data with the logarithm transformation to bring all the values into the interval from zero to one.

Figure 4b demonstrates the frequency distribution of address usage across all addresses used by each program separately, independently from the rest of the programs. It can be noticed that the colour map is similar to the one depicted in Figure 4a. The difference lies in the peculiarity of normalization: in Figure 4a it is done over all values in the single step frequency matrix (to show relation of frequencies across all programs), while in Figure 4b it is done over the values of each column (i.e. program) separately (hence the more intense colours). In particular, we have first normalized the values in each column using formula (1) and then smoothed the data with the logarithm transform.

$$fr_{pi\_norm} = fr_{pi} / \sum_{i=1}^{N} fr_{pi} \qquad (1)$$

where fr_{pi_norm} is the normalized frequency value, fr_{pi} is the correspondent absolute frequency value, p is the program number, i is the address number and N is the total number of addresses (1864 in our case).

Finally, we have calculated probability distributions of each employed address to be encountered in each program profile. For this, we have applied formula (1) to every row (instead of column) of the frequency matrices. Figure 4c demonstrates these probability distributions for both the single step and sampling methods; the more intense the grey colour is, the higher is the probability of the address to represent a certain program, with black colour meaning the address is used by a particular program only. It can be noticed that the sampling tracing method provides the probability distribution colour map very close to the one obtained from the single step logs, but not exactly the same (note for example different intensities at the top of AC and CF lanes). This observation confirms our earlier suggestion (see section *C*) that a better sampling tracing method should be designed to get the accuracy of the single stepping method.

IV. Summary

This paper has investigated the program counter (PC) as a potential source for extracting ICmetrics features and compared two tracing methods, the single stepping and sampling, to obtain feature values. From the analysis results presented in the paper, we conclude that while the fact that PC values that may have been reached during execution does not serve as good ICmetrics features, a strong ICmetrics system can potentially be built based on the sequence analysis and frequency analysis of PC logs. However, further investigation and testing is required to confirm this suggestion. In particular, in this study we have used a small set of straightforward programs. In our future work, we plan to enrol more complex software and design scenarios simulating external environment and user interaction in order to verify if our results are valid for systems that can be find in real world.

We have also suggested a better tracing method to obtain data for ICmetrics, which is sampling at branch points. Our next step in developing methods for data acquisition will be to look at more course grain software instrumentation suited to observe larger and more complex software with limited disturbances.

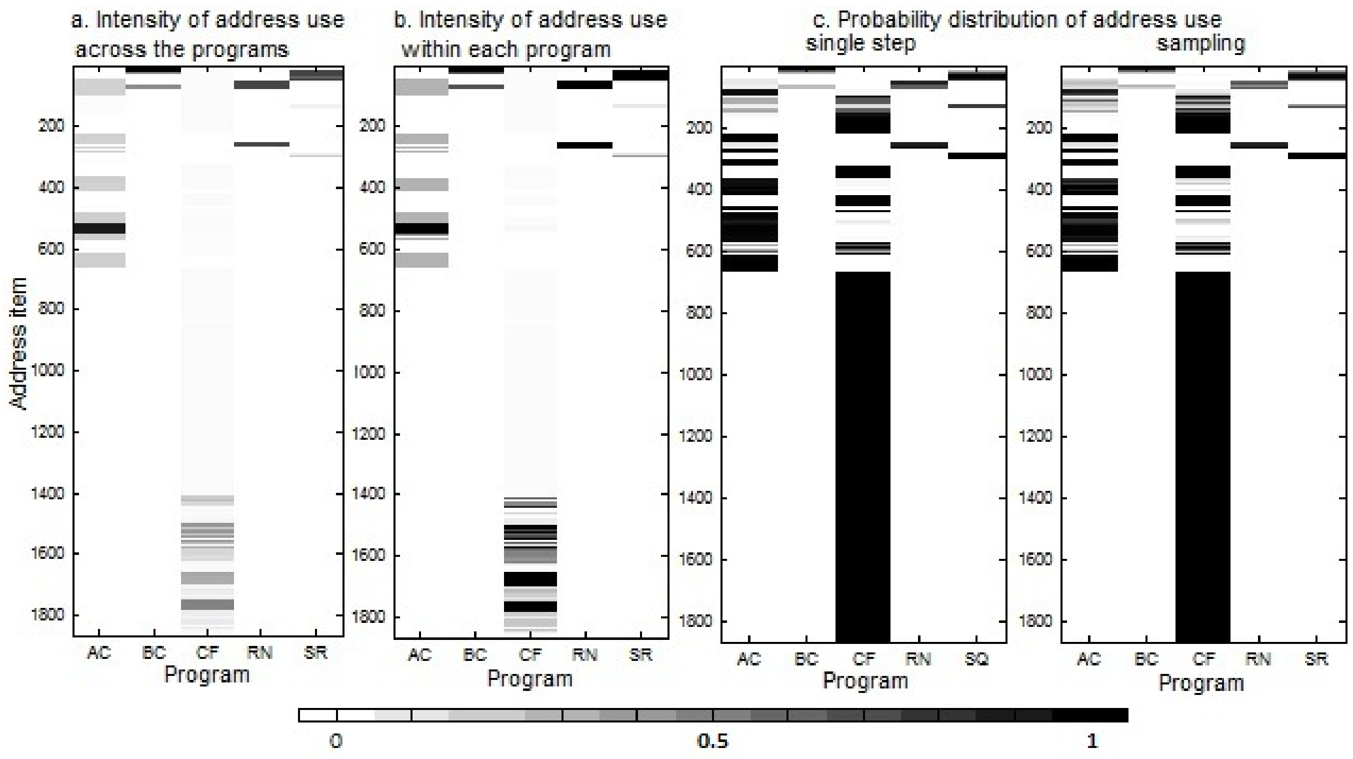

Figure 4. Frequency matrices colour maps.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of the UK Engineering and Physical Sciences Research Council under grant EP/K004638/1 and the EU Interreg IV A 2 Mers Seas Zeeën Cross-border Cooperation Programme – SYSIASS project: Autonomous and Intelligent Healthcare System (project's website http://www.sysiass.eu/).

REFERENCES

[1] D. Gesbert, M. Shafi, Da-shan Shiu, P. J. Smith, A. Naguib, "From theory to practice: an overview of MIMO space-time coded wireless systems", *IEEE Journal on Selected Areas in Communications, 21 (3)*, April 2003, pp. 281 -302.

[2] E. Papoutsis, W.G.J. Howells, A.B.T. Hopkins, K.D. McDonald-Maier, "Key Generation for Secure Inter-satellite Communication", *IEEE, NASA/ESA Conference on Adaptive Hardware and Systems* (AHS-2007), Edinburgh, UK, 2007, pp. 671–681.

[3] G. Edward Suh and Srinivas Devadas, "Physical unclonable functions for device authentication and secret key generation", In Proceedings of the 44th Design Automation Conference, IEEE, 2007, pp. 9-14.

[4] Y. Kovalchuk, G. Howells, and K.D. McDonald-Maier, "Overview of ICmetrics Technology – Security Infrastructure for Autonomous and Intelligent Healthcare System", *International Journal of u- and e- Service, Science and Technology*, Vol. 4, No. 3, September 2011, pp. 49-60.

[5] A.B.T. Hopkins, K.D. McDonald-Maier, E. Papoutsis, W.G.J. Howells, "Ensuring data integrity via ICmetrics based security infrastructure", *Proceedings of the IEEE, NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2007)*, Edinburgh, UK, 2007, pp. 75-81.

[6] A.B.T. Hopkins, K.D. McDonald-Maier, "Debug support for complex Systems on-Chip: A review", *IEE* Proceedings on Computers and Digital Techniques, 153 (4), 2006, pp.197-207.

[7] A.B.T. Hopkins, K.D. McDonald-Maier, Debug Support Strategy for Systems-on-Chips with Multiple Processor Cores, IEEE Transactions on Computers, Volume 55, Issue 2, 2006, pp. 174-184.

[8] Atmel's SAM7S datasheet: http://www.atmel.com/Images/doc6175.pdf

[9] Eclipse official web-site: http://www.eclipse.org/

[10] Online OpenOCD User's Guide: http://openocd.sourceforge.net/doc/html/index.html#Top

[11] 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", *Proceedings of the International Workshop on Workload Characterization*, 2001, pp. 3-14.