International Journal of Civil Engineering and Technology (IJCIET)
Volume 9, Issue 7, July 2018, pp. 190–197, Article ID: IJCIET_09_07_019
Available online at http://www.iaeme.com/ijciet/issues.asp?JType=IJCIET&VType=9&IType=7
ISSN Print: 0976-6308 and ISSN Online: 0976-6316

COLLABORATIVE SOFTWARE INTEGRITY VERIFICATION USING BIT INVERSION ERROR CHECKING CODES IN CROWD-COUPLED IOT APPLICATIONS

Yongtae Kim, Jeonghun Cho and Daejin Park*
School of Electronics Engineering, Kyungppok National University - 41566, Korea
*Corresponding Author Email: boltanut@knu.ac.kr

ABSTRACT

Abstract Internet-of-Things (IoT)-based pedestrian crowd activity monitoring is now being applied in our lives. The application as a service, that is stored in on-chip flash memory in an IoT sensor processor, performs collaborative software execution between the connected IoT systems and the monitored human activity. The software code memory integrity of embedded flash memory in the IoT microcontroller (MCU) is becoming important for applications requiring safety-critical operations because the latest services directly affect human activity. Software-driven or hardware supports for the memory protection are required to guarantee the safe-conscious code execution of the downloaded firmware in MCUs. The protection method requires more power consumption in decoding the additional padding bits. In this paper, the protection hardware and software technique in the IoT systems are proposed to improve the safety and integrity of the software execution with a small amount of logic gates overhead. The error correction code (ECC) hardware data path is integrated with our newly designed binary bit-inversion decoder with zero overhead inversion flags to decrease the power consumption in decoding the binary code blocks. The 8-bit ECC per 64-bit code blocks (72:64 SEC-DED) is applied to a 64KB flash memory with the separated region of the flash memory for ECC padding bits. The proposed robust software execution unit is successfully integrated with human crowd monitoring system in the library.

Key words: Human-coupled, Internet-of-Thing, and Smart object software integrity.

Cite this Article: Yongtae Kim, Jeonghun Cho and Daejin Park, Collaborative Software Integrity Verification Using Bit Inversion Error Checking Codes in Crowd-Coupled IOT Applications, International Journal of Civil Engineering and Technology, 9(7), 2018, pp. 190–197.
http://www.iaeme.com/IJCIET/issues.asp?JType=IJCIET&VType=9&IType=7

1. INTRODUCTION AND MOTIVATION

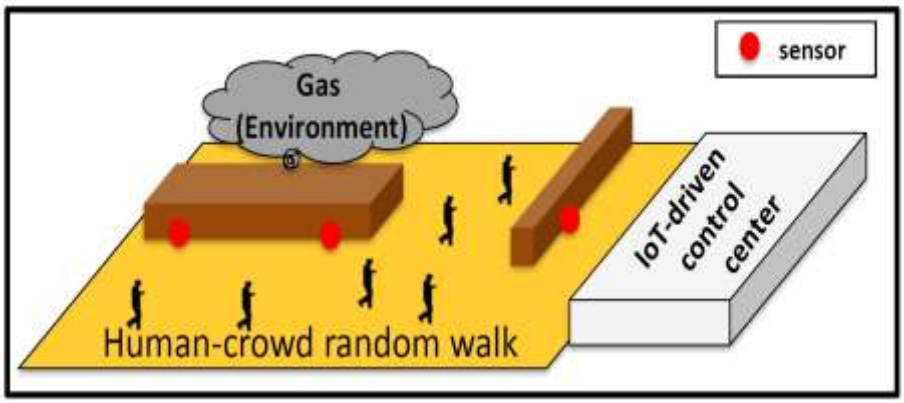

Figure 1 Human-crowed coupled IoT applications - system software execution is based on human movement

The smart electronic devices that are is powered by Internet-of-Thing (IoT)-driven connected services enable the interactive services between humans and things. The software services in IoT devices perform their tasks triggered by human interactions and the collaboration between connected IoT devices, as shown in Figure 1. Therefore, these human-coupled IoT devices and their soft-ware is based on the complex coupled relationship across the hardware, soft-ware, and human-ware (Seok et al, 2016) [13]. The traditional design approaches in human-coupled IoT applications tend to ignore the coupled effect by the crowd as a collection of human body (Kim et al, 2017) [3] so that they have limited capability to join the IoT systems, embedded software services, and human interactions in the crowd.

Although the atomic function embedded in each IoT device is defined by the on-chip embedded software, successful services of the entire IoT system are the result of cross-layered cooperation of all IoT subsystems. This feature causes complex-connected behavior while waiting for the results of the adjacent IoT node operations (Sivrikaya and Yener, 2004) [14]. Thus, IoT-driven applications require more design efforts in building large-scale IoT systems considering the complex-connected relationship (Lee et al, 2012) [12]. In addition to the successful integration of the IoT devices, the implementer has been always aware of the unexpected propagation of a small abnormal state in a few sets of IoT devices, which may affect the entire IoT system, as shown in Figure 2.

Although the on-chip software in IoT devices has no safety issue in standalone service for human, the collaborative operations across IoT devices have complex side effects (Sarma et al, 2014) [12]. If all cases that may result in harmful effects to human body (Maurer et al, 2006) [6] weren't considered, the entire service for human crowd interactions may be broken. In this paper, we focus on protecting the software execution failure to decrease the possibility of the total system freezing state, as shown in Figure 3.

The microcontroller (MCU), which is the central processing unit (CPU) in IoT devices, has been conventionally used in IoT-driven home appliances for human lifecare, which do not require the safety of the program execution. The electrical environment applied to the MCU is stable enough to directly read the instruction code from the embedded flash memory in the MCU. The CPU directly accesses the code memory via the memory bus interface without additional protection hardware (Park and Kim, 2014) [10].

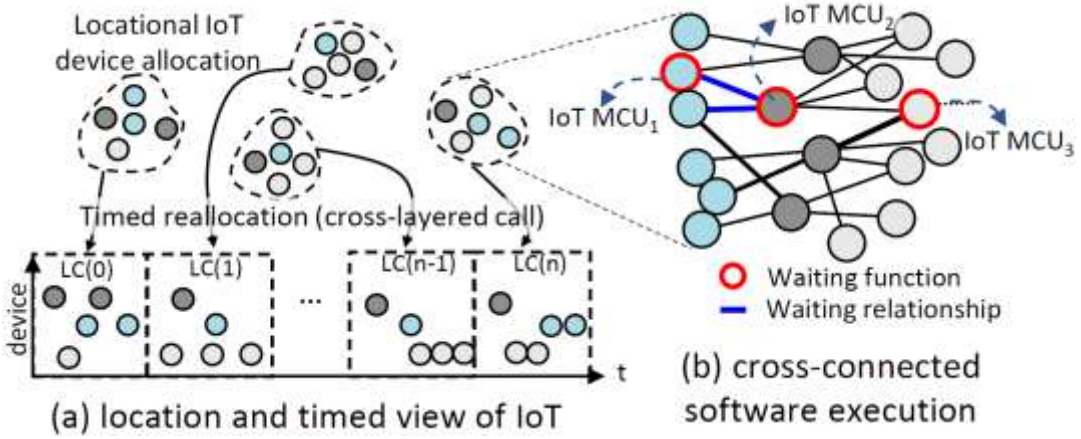

Figure 2 Freezing-effect of software code execution in IoT

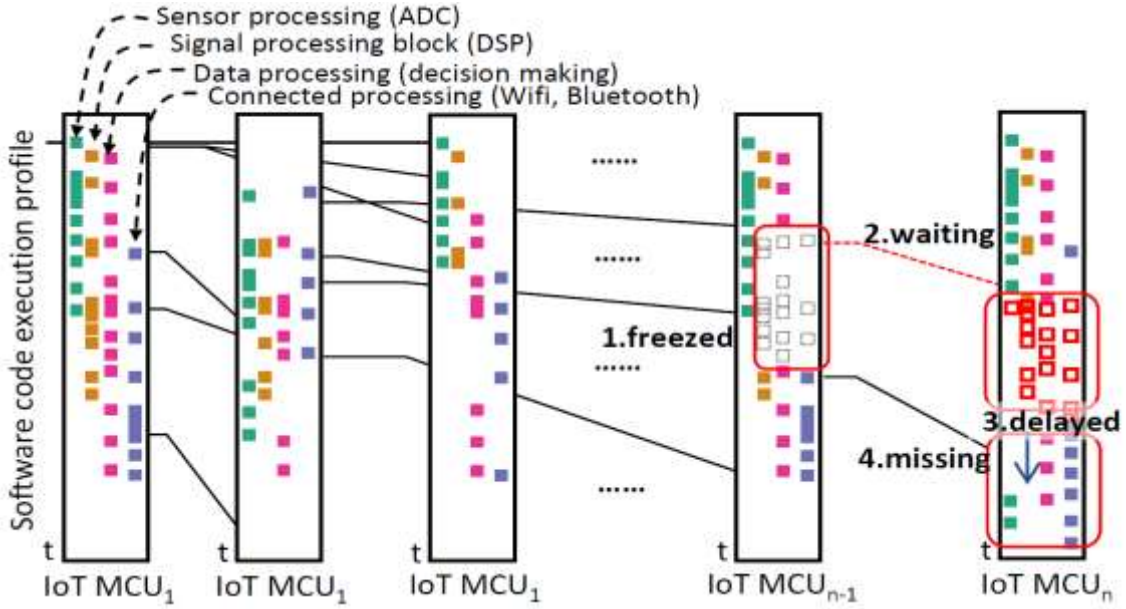

Figure 3 Freezing-effect of software code execution in IoT

The MCU is gradually being applied to mission-critical applications, such as human body wearable devices, and the direct code memory access is easily affected by external interferences, that may cause abnormal instruction decode results in the CPU. There are two reasons for memory read fail operations (Micheloni et al, 2003) [7]. One is a bus-signaling interruption during the code-fetch operation from the code memory. The other is caused by an electrical degradation like the threshold variation in the flash memory bit cell.

Our previous study (Park et al, 2012; Lin and Costello, 2004; Park and Kim, 2013) [11, 5, 9] presented the silent background mode memory verification method with the dedicated cyclic redundancy check (CRC) scanner hardware to pro-vide the memory integrity check with zero overhead, which does not require explicit interruptions of the user code execution and a small size of additional logic gates. This method focuses on detecting the memory integrity fail and branches the instruction flow to the emergency service routine in the memory fail condition. In this paper, we provide the hardware data path improvement for the safe memory read operation through on-the-fly read data protection hardware, which is operated in low-power consumption with zero overhead, in addition to the logic gates to

decode the conventional ECC padding bits (Lin and Costello, 2004) [5]. With the proposed approach, we evaluate the IoT-driven services for human activity monitoring (Park and Cho, 2014) [8] so that the more robust service execution is possible.

2. PROPOSED ARCHITECTURE

The linear block code for the error correction (Lin and Costello, 2004) [5] is based on the binary code conversion with padding bits. The previous study shows that the binary code inversion enables reduction of the memory access current (Daejin and Kim, 2011) [2]. The flash memory binary code inversion, which is proposed to lower the access current (Daejin and Kim, 2011) [2], requires an additional memory area for the inversion flags and identifies the inversion of the binary code with inversion flags in the instruction fetch cycle. The binary code translation, based on the bit-inversion method, requires the minimization method of the inversion flags to reduce the area overhead.

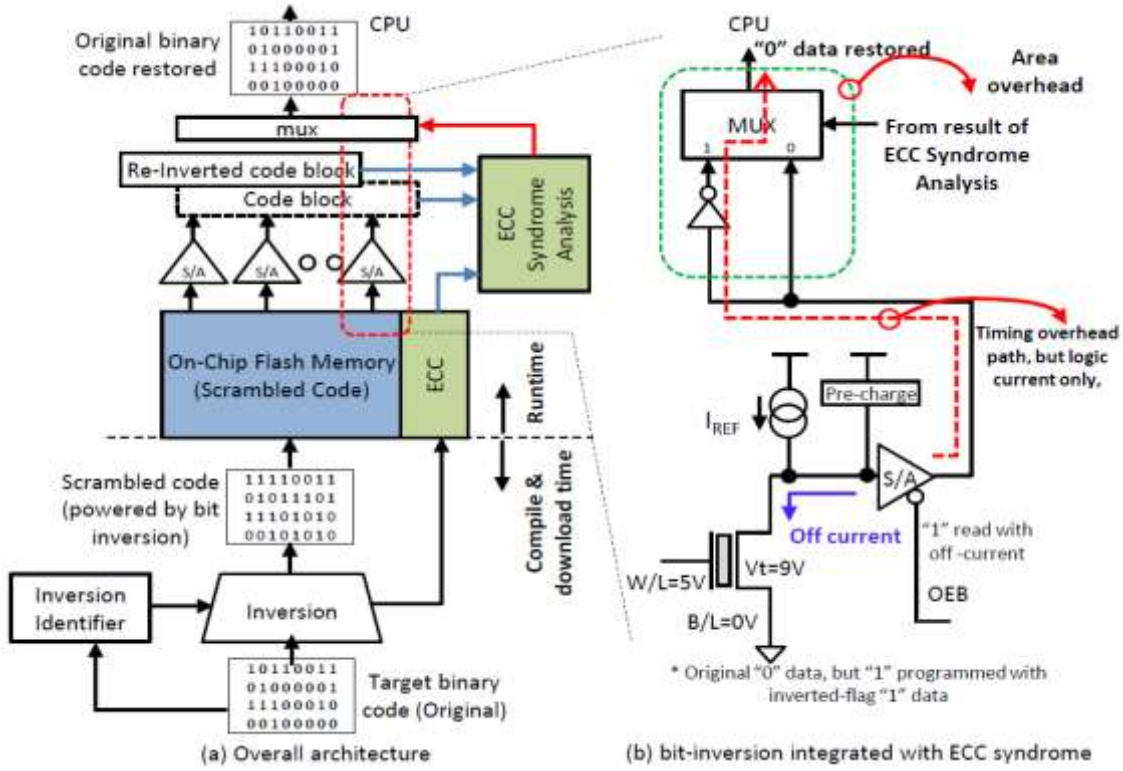

Figure 4 The proposed memory data path architecture based on ECC protection and bit inversion technique

The proposed architecture integrates the ECC memory protection with the binary inversion technique for the flash access current reduction by embedding the inversion flags into ECC padding bits. The inversion flags can be automatically extracted from the ECC padding bits. The flash read-path architecture, using the proposed motivation, could provide the memory protection capability in a relatively low power access current, compared to the conventional ECC syndrome only.

Figure 4(a) shows the overall architecture of the revised MCU architecture including the proposed ECC protection integrated with the bit-inversion hard-ware. The software code is compiled to assembly, converted to the target executable binary code, and finally stored in the on-chip flash memory, in the blue-colored box as shown in Figure 4. The malfunction by executing user software is caused by the broken bits in the software binary code. The error

checking code is embedded as a flag for the unique pattern for the stored bi-nary code. In code execution runtime, ECC syndrome calculation is performed and try to compare the expected ECC with the calculated pattern for the bi-nary. We proposed the random inversion to scramble the original binary code, and then the binary bits of executable codes were broken, which we could easily detect by using embedded ECC flags. If the ECC comparison is not matched, the proposed verification unit reattempts to compare the ECC for the reinverted binary code patterns.

Figure 4(b) describes the current reduction by the bit-inversion for a 1-bit data path controlled by the ECC syndrome result. The inversion-based access current reduction is based on our previous approach Park and Kim (2014) [10], but this work extends the inversion and reinversion technique to protect the binary code.

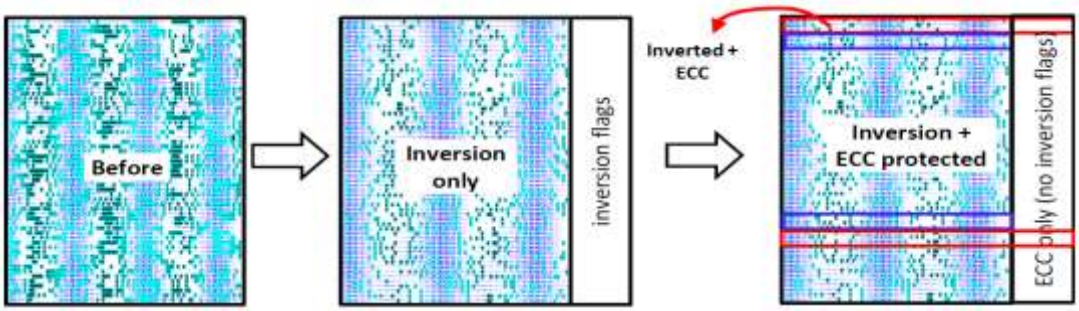

Figure 5 Proposed bit-inversed binary code pattern with ECC padding flags

Figure 5 illustrates that the original binary code is translated into new binary code, including ECC padding bits. The main code area is only inverted with the bit-inversion method, but the ECC padding bits, which include syndrome information for the original code, is not inverted. Through the proposed approach, we could simultaneously receive the benefit of using binary inversion techniques by scrambling and protecting the binary code for the integrity check validation.

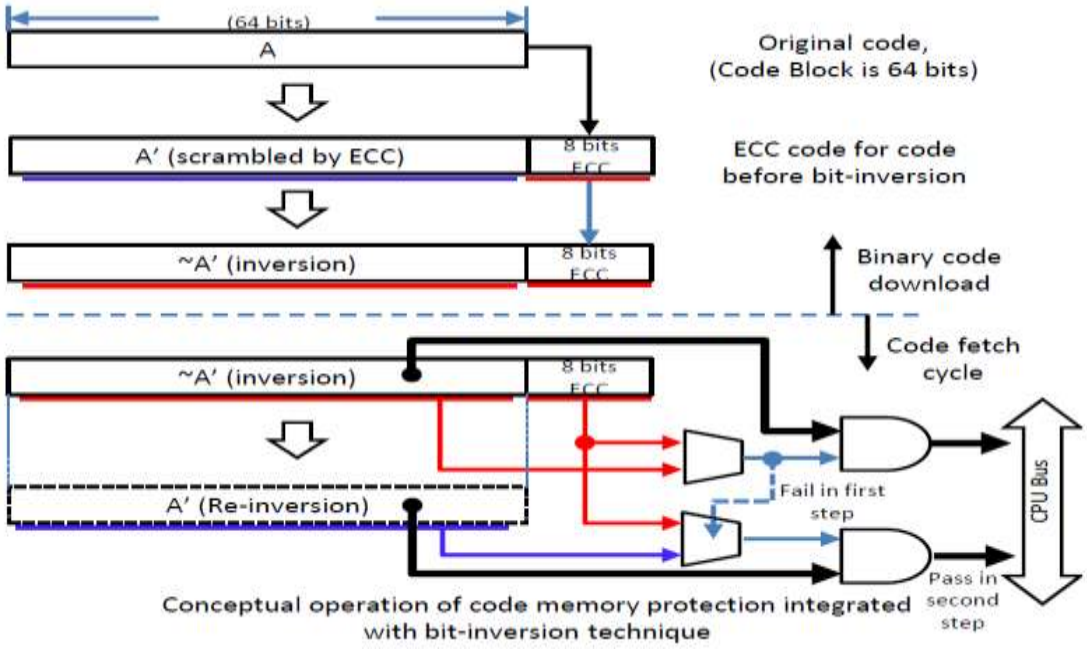

Figure 6 The ECC protection encoding/decoding procedures

If the data verification of the scrambled binary code block with the ECC block results in a fail, the second step of binary verification for the reinverted code block is simultaneously performed with the same ECC. If the second step passes, the main binary code is automatically identified as a bit-inversion sector of the code block. The conceptual operation of the proposed protection code method is described in Figure 6.

The additional hardware overhead, with the exception of the data path for the ECC syndrome calculation only requires a few inverters, registers, and muxes for 64-bit read bus width of one ECC- protected code block, which is nearly zero overhead, compared to the large flash memory size. The details of the proposed hardware data path to implement the ECC protection, which is integrated with a bit-inversion block, are described in Figure 7.

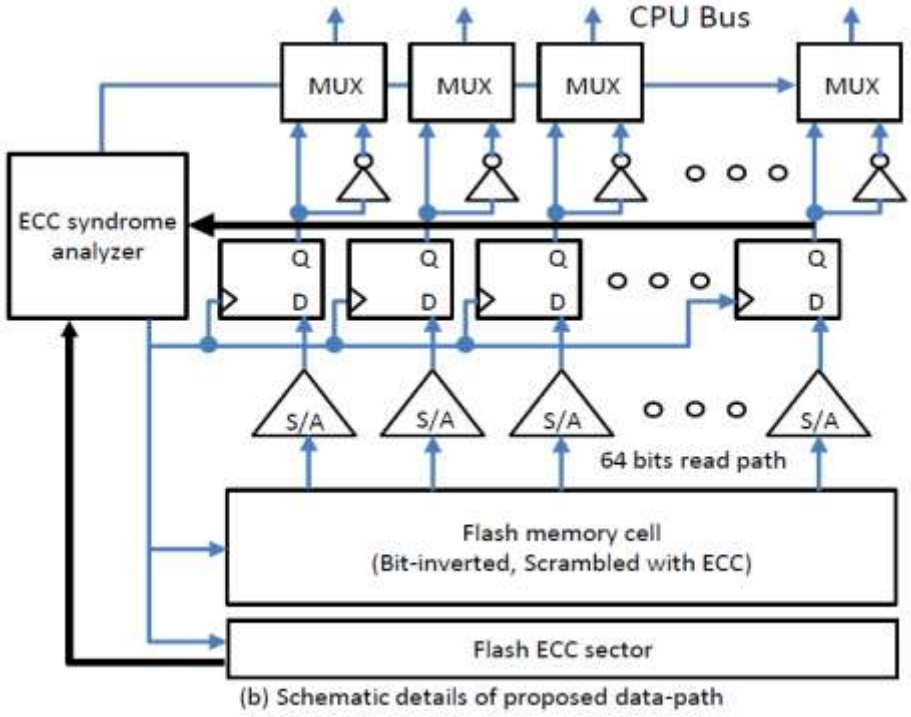

Figure 7 The ECC protection encoding/decoding hardware details of the proposed data path

3. RESULTS AND DISCUSSION

Figure 8 Random bit error injection caused by human-activity and recovered binary pattern by using the proposed approach

We evaluated our approach for human activity monitoring applications. When the IoT sensor executes the on-chip binary software code to monitor the human activity, we described the error injection situation to flip the binary bit pattern randomly, as shown in Figure 8. The proposed binary inversion/re-inversion with ECC code successfully recovers the binary software code, so that the collaborative software execution for crowd-sensing models has no freeze effect, as a result, shown in Figure 9.

Figure 9 The collaborative software execution without any freezing effect

The implementation of the proposed method provides the backward compatibility by modifying the interface only between the CPU and flash code memory bus. The improved hardware presented in this paper is implemented with an 8-bit ECC for 64 bits (72:64 SEC-DEC code (Chen and Hsiao, 1984) [1]).

There is no need for inversion flags to provide the bit inversion method, but it is still able to reduce the flash memory access through the code inversion. The ECC syndrome calculation is simultaneously performed for the original code and extracts the flags to identify the code inversion information during the code fetch cycle. The conventional thirteen 3-input XOR gates for the ECC syndrome calculation, 64 inverters, 64 muxes, and related control data paths are needed to implement the proposed technique with only logic gates. The current consumption reduction is also provided as the same result of the previous study, but this work provides the same current reduction without any hardware overhead through the inversion flags.

4. CONCLUSIONS

This paper introduces the collaborative software service code protection in human-coupled IoT applications. The additional error checking codes are embedded into the target binary code, but they are also used to reduce the access current to the software memory. The proposed method could success-fully scramble the binary code to prevent from malicious bit manipulation, and easily detect the broken bit through reinversion method of the target binary code. Without any analog circuit technique, the proposed architectural modification and small digital logic overhead enable ECC protection and the bit-inversion technique without any inversion flags to reduce the access current for the flash memory with ECC protection to improve the safety in the read operation from the code memory.

ACKNOWLEDGEMENTS

This study was supported by the BK21 Plus project funded by the Ministry of Education, Korea (21A20131600011). This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2014R1A6A3A04059410) and (NRF-2018R1A6A1A03025109).

REFERENCES

[1] Chen C. and Hsiao M. Error-correcting codes for semiconductor memory applications: A state-of-the-art review. IBM Journal of Research and Development, 28(2), 1984, pp. 124–134.

[2] Daejin P., Kim S. and Kim T. G. A sense amplifier using binary code inversion encoder-decoder for on-chip flash read current reduction. 2012 International Conference on Electronics, Information and Communication, 2012, pp. 235–236.

[3] Kim S., Choi G., Lee S., Cho J. and Park D. Event-detection microcontroller using interoperation-based acoustic surface sensing for individualized things human interaction. International Journal of Applied Engineering Research 12(12), 2017, pp. 3546–3552.

[4] Lee Y., Lee S., Kim B., Kim J., Rhee Y. and Song J. Scalable activity-travel pattern monitoring framework for large-scale city environment. IEEE Transactions on Mobile Computing, 11(4), 2012, pp. 644–662.

[5] Lin S. and Costello D. Error Control Coding: Fundamentals and Applications. Pearson-Prentice Hall, 2004.

[6] Maurer U., Smailagic A., Siewiorek D. and Deisher M. Activity recognition and monitoring using multiple sensors on different body positions. International Workshop on Wearable and Implantable Body Sensor Networks, 2006.

[7] Micheloni R., Crippa L., Sangalli M. and Campardo G. The flash memory read path: building blocks and critical aspects. Proceedings of the IEEE, 91(4), 2003, pp. 537–553.

[8] Park D. and Cho J. Accuracy-energy configurable sensor processor and IoT device for long-term activity monitoring in rare-event sensing applications. The Scientific World Journal, 2014, pp. 546–563.

[9] Park D. and Kim T. G. Safe microcontrollers with error protection encoder-decoder using bit-inversion techniques for on-chip flash integrity verification. IEEE 2nd Global Conference on Consumer Electronics (GCCE), 2013, pp. 299–300.

[10] Park D. and Kim T. G. Built-in binary code inversion technique for on-chip flash memory sense amplifier with reduced read current consumption. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 22(5), 2014, pp. 1187–1191.

[11] Park D., Kim T. G., Cho G., Lee K. and Kim C. A safe microcontroller with silent crc calculation hardware for code rom integrity verification in iec-60730 class-b. 2012 IEEE 1st Global Conference on Consumer Electronics (GCCE), 2012, pp. 197–200.

[12] Sarma S., Venkatasubramanian N. and Dutt N. Sense-making from distributed and mobile sensing data: A middleware perspective. Design Automation Conference (DAC), 2014, pp. 1–6.

[13] Seok M., Kim T., Choi C. and Park D. A scalable modeling and simulation environment for chemical gas emergencies. Computing in Science and Engineering, 18(4), 2016, pp. 25–33.

[14] Sivrikaya F. and Yener B. Time synchronization in sensor networks: a survey. IEEE Network, 18(4), 2004, pp. 45–50.