PRODUCING UNIQUE IDENTIFIERS AND RANDOM NUMBERS ON BASIS OF UNCLONABLE PARAMETERS OF MICROCONTROLLERS AND UNDESIRED EFFECTS

Josef Strnadel, Giuseppe Conte
Brno University of Technology, IT4Innovations Centre of Excellence
Bozetechova 2, 612 66 Brno, Czech Republic
strnadel@fit.vutbr.cz, xconte00@stud.fit.vutbr.cz

ABSTRACT:

Main goal of this paper is to utilize component-of-the-shelf microcontrollers (MCUs) to show how their unique and distinctive physical properties can be utilized to enhance the security of embedded systems with no necessity to modify the hardware for the purpose. It is shown in the paper that even if each MCU chip is produced by the same procedure and technology, there are some slight changes that make it being both unique and random in some sense. Almost 100k experiments were performed to verify the concept proposed in the paper on more than 100 MCU chips to show both simplicity and general applicability of our concept. During the first set of about 50k experiments, same piece of code with no middleware being utilized was executed on each MCU with the goal to produce its unique identifier as well as a unique sequence of random numbers based on the physical properties of that MCU. During the second set of experiments, the production was repeated, but on basis of a real-time operating system layer to observe impacts of the additional layer to results of the production process. Freescale's MC9S08JM60 MCU and Micrium's uC/OS-II kernel was utilized during the experiments.

Keywords: jitter, random number, unique identifier, generator

1 INTRODUCTION

The need for security belongs to typical requirements imposed on recent devices such as MCUs. A very important aspect in the security context is the utilization of unique identifiers (IDs) and random numbers (RNDs) for confidentiality and authentication purposes [5]. Typically, either a special hardware (HW) or software (SW) is utilized to produce the IDs and RNDs. However, this adds extra costs since an extra HW or SW must be added to enhance security of a device.

In this paper, we focus to the device identification (DEVID) problem, which is solved especially in the secured communication area where it is necessary uniquely identify both a sender and a receiver of a message. Typical solutions of the DEVID problem are based on writing a unique (ID) number into a non-volatile memory such as flash, post-production modification of the device, introducing special circuits designed to compute IDs

algorithmically or using non-conventional solutions such as polymorphic chip called REPOMO32 [6].

However, if the uniqueness requirement imposed on IDs is extended by non-reproducibility (unclonability) of IDs, many of the above-mentioned solutions fail to meet both the requirements – e.g., if the ID being stored in a flash of a device A becomes declassified (for example, on basis of monitoring the communication line), it is not a problem to make a copy of it and store that copy into the flash of another device B. As a result, the device B cannot be distinguished from the device A, so it can, e.g., falsify messages originating from the device A. Special post-fabrication modifications or special circuits such as REPOMO32 can meet the requirements, but at extra costs added for that purpose.

To minimize costs related to production of unique, non-reproducible IDs, we have decided to utilize inherent properties of common COTS (component-of-the-self) devices such as MCUs rather than to utilize a complex circuitry, technology or algorithms for that purpose. Since MCUs are typically digital, synchronous sequential circuits we have focused to inherent properties implying from uniqueness of their inner clock signals. Because of the production process variability, several undesired effects such as jitters can be measured and hopefully, utilized to identify those devices.

Jitter can be defined as the timing variation of a (real) signal edge from its ideal (simulated, theoretical) occurrence time. However, if a real hardware is utilized then signals such as clock are typically disturbed by factors such a thermal noise, power supply variations, loading conditions, device noise, and interference coupled from nearby circuits. Many types of jitter can be identified in the literature, e.g. Period Jitter, Cycle to Cycle Jitter, Long Term Jitter, Phase Jitter or Time Interval Error Jitter. Further jitter effects such as task-release jitter, response-time jitter etc. can be observed if a SW (e.g., operating system, OS) layer is utilized over the HW [1].

2 EXPERIMENTAL PLATFORM

Before utilizing the jitter effects and for utilization of their impacts for the security purposes, we have decided to measure them on a (simple, 8-bit) MC9S08JM60 MCU (briefly, JM60) produced by the Freescale company [2]. From our point of view, it is important that there are multiple clock signals distributed in the JM60's architecture – most of them (including bus clock, BUSCLK, being utilized to clock the CPU) are derived from the multi-purpose clock generator (MCG) module able to derive clock signals either from the external, e.g., 12 MHz, oscillator (XOSC) or inner 32kHz oscillator plus frequency-locked loop (FLL), phase-locked loop (PLL) circuits or from the inner, 1kHz low-power oscillator (LPO). Since the signals produced by LPO and MCG are not dependent, and LPO is loaded with an error that is negligible in comparison with MCG's error, we have decided to utilize the LPO signal as a reference for measuring jitter effects implying from MCG signals.

For the purpose of the measurement, we have utilized two modules realized on the JM60's chip: the Real-Time Counter (RTC) configured to be clocked by the LPO and the Time/Pulse-Width Modulator (TPM) configured to be clocked by the BUSCLK derived from MCG. Operational principles of RTC and TPM modules follow.

On basis of the 1kHz clock, RTC is able (among the others, e.g., calendar or alarm functions) generate an interrupt after a predefined number of real-time units (e.g., 1, 10, 100 or 1000 ms) being derived from the basic 1ms RTC period and the prescaler configuration programmed into the RTCPS select bits. For example, in order to adjust interrupts to be generated by the RTC each 1 ms, RTCPS bits must be set to 8. Alike, for 10 ms, 100 ms or 1000 ms periods, they must be set to 11, 13 or 15. Since the corresponding interrupt service

routine (ISR) is started with the period given by the RTCPS bits, it can be utilized to sample further quantities, such as jitters implying from MCG.

TPM is based on a free-running 16-bit counter, which can be clocked by BUSCLK divided by the programmed prescaler (PS bits in the TPM1SC register; for PS=0, the divisor is 1). The maximum value the counter can reach can be programmed into the TPM's modulo register (reaching the value is denoted as TPM's overflow) and an interrupt can be generated to signal this and to start the corresponding ISR.

3 EXPERIMENTAL SETUP AND RESULTS

Since the rate of BUSCLK (typically, 8-9 MHz) is much higher than the rate of RTC (typically, 1-0.01 kHz), we can observe the jitter effect (supposing it is there) by counting how many times TPM overflows (TPMovr value) in a single RTC period and to read the final state (TPMstate value) of the TPM counter at the end of the RTC period – for the illustration, see Fig. 1.

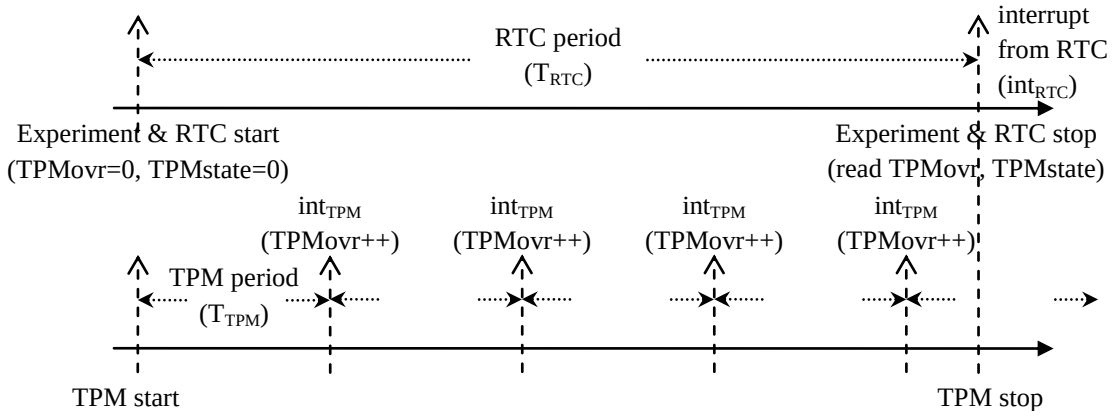

Fig. 1: Principle of the jitter effect measurement

To measure the jitter effect and to evaluate how it depends on physical parameters of particular devices, we programmed exactly the same firmware with the functionality shown in Fig. 1 into 21 different JM60 chips. For each chip, we have performed 1000 experiments (RTC periods) for each of the following RTCPS values: 8, 11, 13 and 15 (which corresponds to RTC periods (T_{RTC}) of the following lengths: 1ms, 10 ms, 100 ms and 1 s) – more than 80k experiments in total!

The result of the experiments can be summarized as follows. Number of TPM overflows (i.e., the TPMovr value) was same (0) for RTCPS=8 (corresponding to T_{RTC}=1 ms), almost the same (2 or 3) for RTCPS=11 (T_{RTC}=10 ms), more different (TPMovr ranged from 22 to 29) for RTCPS=13 (T_{RTC}=100 ms) and the most different (TPMovr ranged from 227 to 295) for RTCPS=15 (T_{RTC}=1 s).

Detail results for particular devices can be seen in Fig. 2 on basis of which it can be concluded the devices cannot be identified on basis of the TPMovr values achieved under RTCPS=8 (just 1 ID (0) is produced for 21 devices) or 11 (only 2 IDs (2, 3) are produced). However, for RTCPS=13, there are 8 different IDs (22 to 29) produced and for RTCPS=15, there are 19 different IDs produced, out of which just two are same (234) for devices No. 4, 13 and two (263) for devices No. 10, 11! On basis of the results, it can be said that (and it is expectable) that greater T_{RTC} contributes to classification of the devices just because an error implying from the jitter effect accumulates and grows with T_{RTC}.

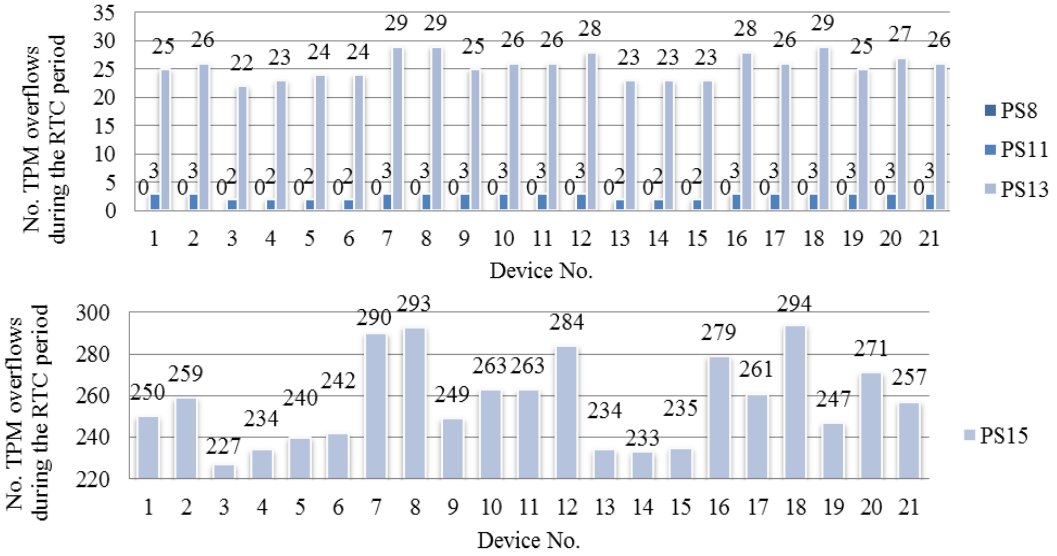

Fig. 2: Amount of TPM overflows within single RTC period as a function of the RTC prescaler

But, up to now we have not analyzed the TPMstate values – being measured along with TPMovr for each device – yet. If the values are studied further – e.g., for TPMovr=23-26 achieved for the RTCPS=13 – (see Fig. 3, a-d) then it can be observed that although some devices produce the same TPMovr value (for example, TPMovr=23 for devices No. 4, 13, 14, 15), they differ in their TPMstate value. The final conclusion is that the unique, non-reproducible ID of a JM60 device can be computed as a function of TPMovr and TPMstate values.

The Same conclusion can be made for RTCPS=15 (see Fig. 3, e, f).

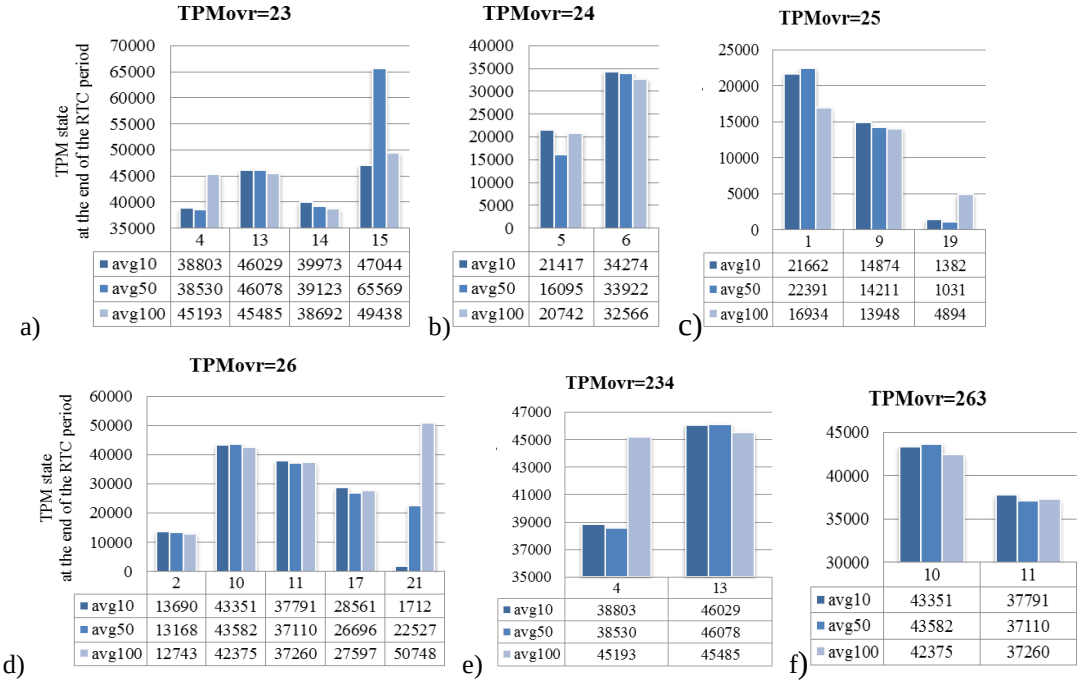

Fig. 3: JM60 chips with the same TPMovr value can be distinguished on basis of the TPMstate value – selected results for the RTCPS=13 (a-d) and for RTCPS=15 (e, f). avg10, avg50 and avg100 means the value was evaluated as a mean value (arithmetical average) from 10 %, 50 % and 100 % of TPMstate data

Since the ID generation is possible on basis of the jitter effects, we are going to focus to the second important aspect in the security context, i.e., to the RND generation problem. In relation to the generation, especially the following properties of the generator are important:

a) randomness (i.e., unpredictability) of generated RNDs,
b) uniqueness of generated RND-number sequences,
c) RND-number length (i.e., number of digits in a RND) the generator is able to achieve.

In general, the generator can be either a pseudo-RND generator (based typically on some kind of a seed able to start an algorithm able to produce a sequence of RNDs) or a true-RND generator (being dealt in this paper) based on an uncertain physical quantity rather than on an algorithm.

We have tested the properties mentioned in a) – c) using a set of experiments designed especially for that purpose. To increase the randomness, we have decided to generate RNDs at the task-level, which introduced an extra (task-release) jitter into the results and allowed the generator to be more scalable comparing to that based just on MCU's resources. For that purposes, we have utilized the Micrium's uC/OS-II kernel [3], but the RND-generation principle is general enough to be realized using a different operating system.

For the purpose, we have defined several tasks (below, the tasks are listed in the decreasing-priority order, i.e., the highest-priority task is mentioned first):

a) the Start-up task (StartupTsk), goal of which is to create remaining tasks (see below), to initialize JM60 and its peripherals such as TPM and then to suspend (blocks) itself to let the other tasks be executed,
b) the Message box task (msgBoxTsk) designed to wait – in a blocking manner – for a message with the TPMstate value written into by one of the User tasks (i.e., msgBoxTsk is blocked by the kernel – i.e., other tasks can be executed – until it receives a message); let it be noted there that at most 1 message can be in the its box,
c) the Random-number generator task (rndTsk) utilized to produce a "seed" from previous two (random-enough) TPMstate values being read by the task,

```
static void rndTsk (void *pdata)
{
   (void)pdata;
   for(;;) {
        rnd_c = get_rnd(0x04);
        OSTimeDly(1);
   }
}
```

```
unsigned char get_rnd(unsigned char mask)
{
   static unsigned char key[2], keyc=0;

   key[keyc]=TPM1CNT + rnd_c + 17;
   keyc^=1;
   return(mask & (key[1] + rnd_c));
}
```

d) the User task No. 1 (usrTsk1) designed to periodically (pseudo-periodically since the period is not accurate because of the jitter effect implying from higher-priority activities such as executions of tasks or ISRs) send a message to the msgBoxTsk with the actual TPMstate value (which changes in time, depending on the time the task was released). If a message arrives in to the full box then the message is discarded, which introduces further randomness into the generation process.

```
static void usrTask1 (void *pdata)
{
   INT8U err;                 // NOTE to utilized variables:
                              // INT16U par_arr[MAX_USR_TASK];
   for(;;)                    // par_arr[0] = 3;  par_arr[1] = 5;  par_arr[2] = 7;
   {
        OSTimeDly(rnd_c);                          // randomize the task`s period
        msg[0] = TPM1CNT;                          // write TPMstate into the message
        err = OSMboxPost(mBox, (void *)&msg[0]);   // put the message into the box
        if(err != OS_NO_ERR) OSTimeDly(par_arr[0]); // check if the box is full
   }
}
```

e) the User task No. 2 (usrTsk2) – see the comment by the usrTsk1,
f) the User task No. 3 (usrTsk3) – see the comment by the usrTsk1.

The (pseudo-)periods of User tasks were adjusted to avoid a soon repetition of the same task execution order. In general, the execution order repeats each number of time units equal to the least common multiple of the tasks's periods. In our experiment, the periods of tasks usrTsk1, usrTsk2, usrTsk3 were adjusted to 3, 5, 7, so the repetition is done after 3x5x7=105 operating system time (OSTime) units. To make the repetition more random, . This concept was enough to construct an RND generator of relatively good randomness, sequence uniqueness and number's length (see Tab.1 for the results).

Tab. 1: TPM overflows

RND length		**Max RND value**	**0/1 occurence ratio in an RND**	
decimal digits	**bits**	**(Min was 0)**	**Zeros**	**Ones**
5	17	95017	53,83%	46,17%
6	20	950003	52,97%	47,03%
10	34	9077133871	57,59%	42,41%
15	50	928876388000220	52,47%	47,53%
18	60	947704142133787000	57,86%	42,14%

5 CONCLUSION AND ACKNOWLEDGMENTS

On basis of the results presented in Fig. 2, 3 and Tab. 1, being achieved for the JM60 device, it can be concluded that it is possible to generate both the unique IDs and RNDs on basis of physical, non-reproducible parameters of MCUs with no necessity of utilizing either a complex and/or very specialized software or hardware for the purpose. Instead, inherent jitter-effects of a device can be utilized along with a simple jitter-measurement technique based e.g. on the ISR-level or task-level executions.

This work has been partially supported by the internal university project No. FIT-S-14-2297 (Architecture of parallel and embedded computer systems) and the project No. ED1.1.00/02.0070 (IT4Innovations Centre of Excellence).

6 REFERENCES

[1] Conte, G.: *Enhancing Security on Basis of Unclonable Parameters of Embedded Systems*. MSc. thesis, Brno University of Technology, 2014, 74 p.
[2] Freescale.com web pages: *MC9S08JM60 Datasheet*. Available from http://cache.freescale.com/files/microcontrollers/doc/data_sheet/MC9S08JM60.pdf.
[3] Labrosse, J. J.: *MicroC/OS-II, The Real-Time Kernel*. CMP Tech. Books, 2002, 552 p.
[4] Lofstrom, K., Daasch, W. R., Tailor, D. *IC identification circuit using device mismatch*. In: Proc. of the IEEE Int. Solid State Circ. Conf., IEEE, 2000, pp. 372-373.
[5] Maurer, U. M.: *The strong secret key rate of discrete random triples*. In: Proceedings of Symposium on Communications, Coding and Cryptography, 1994, Kluwer.
[6] Sekanina, L., Ruzicka, R., Vasicek, Z., Simek, V., Hanacek, P.: *Implementing a unique chip ID on a reconfigurable polymorphic circuit*, Information technology and control, Vol. 42, No. 1, 2013, pp. 7-14.