LOW-COST ARM CORTEX-M0 BASED TRNG FOR IOT APPLICATIONS

Marek LABAN*, Milos DRUTAROVSKY**
*Department of Electronics and Multimedia Communications, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Park Komenského 13, 042 00 Košice, Slovak Republic
MICRONIC a.s., Sliacska 2/C, 83102, Bratislava, E-mail: laban@micronic.sk
**Department of Electronics and Multimedia Communications, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Park Komenského 13, 042 00 Košice, Slovak Republic, E-mail: milos.drutarovsky@tuke.sk

ABSTRACT

The Internet of Things (IoT) is one of perspective electronic sectors. In the near future a lot of common devices from a refrigerator to a door lock will be connected to the internet. Protection of the IoT devices should not be neglected. The device security is composed of many safety levels, where every countermeasure increases its robustness.

The paper describes an implementation of a True Random Number Generator (TRNG) used in many cryptographic algorithms and protocols. It is based on a modern low-cost and low-power STM32F050 ARM-M0 microcontroller, suitable especially for IoT applications. The main motivation for developing of such generator was its absence in lower members of microcontroller families. Integrated TRNG uses common features of the microcontroller, which may be portable across ARM-M0 architecture. A source of randomness is instability of internal RC oscillator, which is acquired using another faster clock and one timer.

The paper follows a previous research, but using the modern microcontroller with proposed on-line embedded tests which are designed in order to be simple and effective.

Keywords: *IoT, TRNG, ARM-M0, jitter, security, randomness*

1. INTRODUCTION

Random numbers are an important part of security and encryption systems. They are used e.g. as the keys in symmetric ciphers, stream in one time pad ciphers or primes in the RSA encryption [1]. The security in such systems is proportional to the randomness of the generated numbers.

Random number generators can be divided into two groups:

- pseudo-random number generator (PRNG),
- true random number generator (TRNG).

PRNG uses various mathematical algorithms to generate random numbers. A speed of the PRNG is equal to computing performance and generated numbers have very good statistical properties. However, the sequence of numbers is repeated with a very big period.

Their weakness consists in predictability of generated numbers. If an intruder recognize the generator equation, then he can predict its output. Typical example of abusing is AES cipher, where whole security depends on the strong and unknown key.

TRNG uses various physical factors like thermal noise, noise of semiconductor part or clock jitter to generate random numbers. Its output is hard to predict and unlike PRNG, its entropy raises in time. The TRNG throughput depends only on the quality and speed of the noise source.

In general, it is very usual approach to combine these two kinds of generators to obtain the best of both.

The following article is aimed on the integration of the TRNG and its simple on-line embedded test in the small and cheap microcontroller STM32F050K6 [2]. The source of noise is jitter of the low speed internal RC oscillator. Described generator can be applied in low-power, low-cost applications. The generator should be portable across various families of microcontrollers with a Cortex ARM-M0 core. The ARM-M0 core is not just a low-cost core, but it is also used in devices of miniature dimensions with few number of pins [3], that are very interesting for IoT application. Exploitation of Cortex ARM-M0 resources is described in detail, so the generator can be easily reimplemented. A typical example of use can be a car remote control, a hand-held transceiver or many other secured IoT applications.

2. GENERATION OF RANDOM NUMBERS

2.1. State of the Art

The principle was already proposed in 1955 by Bell Laboratories for LSI circuits [4]. The same principle could be used in modern microcontrollers as was shown in [5], [6]. The generator uses two independent oscillators – a high speed oscillator and a low speed oscillator. Several periods of slower oscillator are measured by fast oscillator (see Fig. 1).

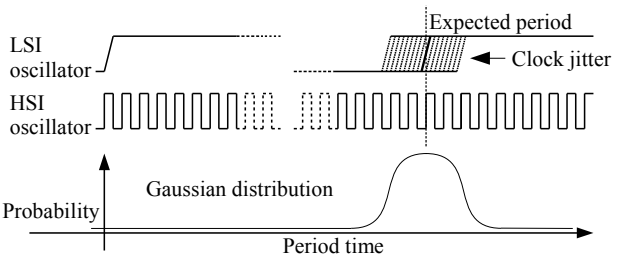

Fig. 1 Exploitation of the clock jitter in TRNG

In the ideal world, the value of the acquired counter should be a constant value. However, in the real world noise causes deviations in the acquired counter, which can be ex-

tracted as random numbers. The main source of noise is a clock jitter, which is defined as a clock instability in time.

Our proposed design of TRNG is very similar, but it differs in the following aspects:

- The previous generator was developed and focused on an 8-bit AVR microcontroller [7]. Our design is integrated on the modern low-cost 32-bit microcontroller, based on the ARM-M0 core. Since its architecture is similar to other ARM-M0 microcontrollers, the proposed generator may be portable to many other microcontrollers. We integrated whole TRNG using a one timer and two oscillators, which can be internal.
- The TRNG was tested mostly by statistical tests. They are sufficient for the early evaluation of the generator, but insufficient for the real operation due to their high computational complexity. In addition, the statistical tests don't ensure unpredictability of the generator. We designed embedded TRNG tests adapted to the generator, which monitors sufficient entropy and guarantees proper functionality of the generator.
- We optimized the throughput of the TRNG, by optimizing the counter reading period, in order to have sufficient entropy using only the last bit of the acquired counter. Additional bits of the counter are more sensitive. Therefore they are used in TRNG embedded tests for earlier detection of the failure. The previous work was oriented mainly on analysis of every counter bit randomness.

2.2. Description of the Implementation

We designed the generator on the STM32F050K6 (new marking STM32F031K6), 32-bit microcontroller based on ARM 32-bit Cortex®-M0 core [2]. The main advantage is its high performance in comparison to its small dimensions and low power consumption. Core can operate at a maximum frequency of 48 MHz and it contains 32 kB of flash and 4 kB of SRAM memory. These types of microcontrollers usually contain an integrated TRNG only in more complex devices (e.g. STM32F4xx).

The generator uses two oscillators, a slower 40 KHz (LSO) one and a faster 48 MHz (HSO) one. To measure the frequency we use a Timer 14 (TIM14) intended for internal or external clock measurement (see Fig. 2). The timer allows to measure several periods of slower LSO oscillator and to generate appropriate interrupt. TIM14 allows to capture clock signal from following sources:

- intended general purpose input pin (GPIO),
- real time clock,
- high speed external clock (divided by 32),
- microcontroller clock output (MCO) pin.

We are using the MCO pin, which can have various sources. In our case, internal 40 kHz LSO is used. The MCO clock frequency can be divided using the MCO clock divider (max. dividing by 128), which unfortunately does not feature all devices.

The input clock of TIM14 is sampled by scalable frequency, and the clock edge is detected. The edge detection is executed by several consecutive samples (number can be adapted) and a rising or falling edge can be selected. The next block, prescaler allows to generate the interrupt after 1/2/4 or 8 valid transitions.

Finally, capture register reads the 16-bit counter (clocked by HSO) and generates interrupt each time the prescaled edge is detected. If the capture register is read after several interrupts, the entropy is accumulated and the dispersion of the generator is bigger.

TIM 14 interrupt has to have a highest possible priority. Any other interrupts with higher priority as TIM14 interrupt may result in deterministic behaviour of the TRNG.

Every acquisition of the LSO period should be independent from the previous one. There are two solutions how to independently read data from the capture register during the interrupt routine. The first one is to read data from capture register and reset the counter. Its disadvantage is that a delay of the interrupt calling has a tiny impact on the acquisitions, but the realization is very simple. The second one is to remember the last capture register value and to calculate the difference between two consecutive samples. This calculation is computationally more complicated than previous one.

In order to achieve the simplest solution we chose the first one.

2.3. Optimization and Experimental Results

We have two main parameters which can adapt the output of the generator, the prescaler size (PRSC) and the in-

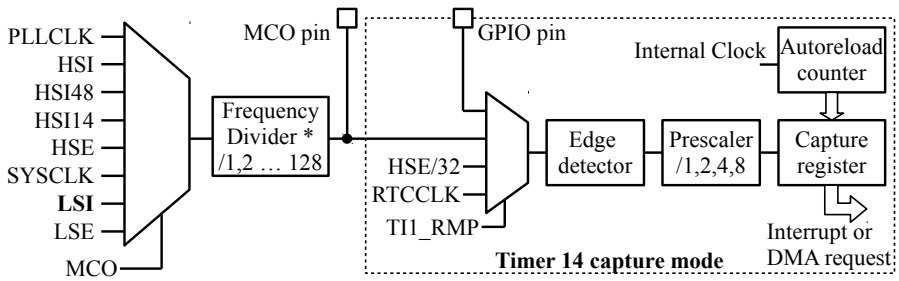

Fig. 2 Exploitation of the Timer 14 for TRNG

terrupt count (INT). PRSC defines how many periods of LSO are elapsed until the interrupt routine is called and INT defines how many interrupt routines are executed until one random bit is read. Minimization of executed interrupt routines per second minimizes a load of microcontroller. Therefore, we configured PRSC to the highest possible value - eight periods.

We did experiments to find the best INT value, in order to optimize its throughput. We acquired 524,288 of 16-bit capture counter words and tested individual bits by using ENT test suite [8], similar to [6]. The test suite consists of several smaller tests - Entropy, Chi-square Test, Arithmetic Mean, Monte Carlo Value for Pi, and Serial Correlation Coefficient. The binary sequence passes the test, if the test results are close or corresponding to the values shown in Table 1.

Table 1 The ENT test suite results of the ideal PRNG

Entropy	Compr.	Chi Square	Arithmetic Mean	Monte Carlo	Correl. Coeff.
> 7.976	0%	10 % < χ^2 < 90 %	~ 127.5	~ 3.14	~ 0

We did the tests for various values of INT = 1/2/3/4/5 that correspond with frequencies of counter reading of 5000/2500/1666/1250/1000 Hz (see Table 2).

Table 2 The ENT test suite results for individual TRNG bits

INT = 1; f = 5000 Hz

Bit	Entropy	Compr.	Chi Square	Arithmetic Mean	Monte Carlo	Correl. Coeff.
0	7.999708	0%	97.59%	127.4800	3.137204	-0.000592
1	7.980202	0%	0.01%	120.3084	3.280439	0.023683
2	7.873203	1%	0.01%	111.1374	3.367002	0.109169

INT = 2; f = 2500 Hz

Bit	Entropy	Compr.	Chi Square	Arithmetic Mean	Monte Carlo	Correl. Coeff.
0	7.999688	0%	90.06%	127.4712	3.145901	0.001630
1	7.997653	0%	0.01%	125.2827	3.197858	-0.000051
2	7.487666	6%	0.01%	91.1838	3.649581	0.117187

INT = 3; f = 1666 Hz

Bit	Entropy	Compr.	Chi Square	Arithmetic Mean	Monte Carlo	Correl. Coeff.
0	7.999650	0%	50.10%	127.5477	3.136059	-0.002757
1	7.999617	0%	14.84%	127.2944	3.136792	0.000997
2	7.831700	2%	0.01%	106.5012	3.484053	0.060202

INT = 4; f = 1250 Hz

Bit	Entropy	Compr.	Chi Square	Arithmetic Mean	Monte Carlo	Correl. Coeff.
0	7.999672	0%	76.59%	127.3845	3.139218	0.001754
1	7.999636	0%	32.90%	127.5111	3.140271	-0.000914
2	7.992287	0%	0.01%	125.4862	3.142239	0.045200

INT = 5; f = 1000 Hz

Bit	Entropy	Compr.	Chi Square	Arithmetic Mean	Monte Carlo	Correl. Coeff.
0	7.999630	0%	25.34%	127.5792	3.140362	0.000720
1	7.999675	0%	79.49%	127.4025	3.141049	0.001505
2	7.995464	0%	0.01%	124.627	3.193234	0.017945

These tests were done during standard operating conditions (room temperature, unaltered voltage) and we used the 8 MHz external clock source multiplied by PLL to 48 MHz as HSO and internal 40 KHz oscillator as LSO. These attributes may affect TRNG, but they are not discussed in detail in the paper. We will focus on them in the following work.

Table 2 shows the results of ENT test suite to the sequences of the zero bits, the first bits, and the second bits. According to Table 1, all sequences for INT = 1, and INT = 2 do not pass the tests – Chi Square test failed in every case. In other cases we can see, that the lower frequency of reading (lower throughput) increases entropy of individual bits. It also corresponds with mean value, where the Chi Square test passed in the cases, when the value was very close to 127.5. Results show that the most sensitive test is Chi Square test. These acquired data of different frequencies and quality are also used as the test vectors for the development of our embedded tests.

3. EMBEDDED TRNG TEST

The embedded TRNG test should be fast, simple, and reliable. Our proposal is to use the last bit of the counter as the generator output and to use additional higher two bits for embedded tests. The higher bits are more sensitive than the last bit (see Table 2), therefore they should indicate the TRNG failure faster and more accurately than the statistic tests of the generator output.

In the real application, the test can indicate TRNG failure, even if the failure didn't occur (so-called false alarm). These alarms are tolerable, if they are not very frequent. In this case, the corrupted data are not used and new ones are generated. However, the test should be very reliable and sensitive for the total failure.

3.1. Poker Test

Let us imagine an example that poker player has five cards, where the following combinations can be obtained:

- five of a kind – AAAAA,
- four of a kind – AAAAB,
- full house – AAABB,
- three of a kind – AAABC,
- two pairs – AABBC,
- one pair – AABCD,
- burst – ABCDE.

Every combination of cards on hand during "poker game" should be repeated with some statistical probability, which can be easily calculated. When we use these probabilities as reference for observation, we can estimate if the sequence of cards in a poker game is random or not. Then Chi-Square analysis is used for expression of the dependencies.

The same principle is used in Poker Test for testing of random data. The five consecutive integers are observed and one of the seven different patterns is recognized. Poker Test is also one of the NIST random number statistical tests defined in FIPS 140-1 (it was already superseded by FIPS 140-2, where the Poker Test is missing, but it is still used in many applications). There is also modification of Poker Test where only four or three consecutive numbers

are used [9]. Their advantage is mainly a lower computational effort.

We utilized such test in our embedded tests, where we used the last three bits of the counter. The test must be fast, so we took 1032 samples of counter (the number divisible by 3), where every sample consists of the last three counter bits. We used Poker Test with the hand of three consecutive numbers, due to its simplicity. The following Table 3 shows possible combinations and the probability of their occurrences in an ideal source of randomness:

Table 3 Probability of counter values occurrence using Poker Test with Hands of Three Numbers

Combination	Probability of the occurrence	Expected occurrence
Three of kind (AAA)	0.016	16
One pair (AAB)	0.328	339
Burst (ABC)	0.656	677
Sum:	1	1032

We analysed 4064 sequences (one sequence = 1032 bits) of our recorded data and calculated the Chi-Square (X^2) using the following formula:

$$X^2 = \sum \frac{(O-E)^2}{E} \tag{1}$$

The degree of freedom for our Table equals to two, so the critical value of $X^2_{.05}$ equals to 5.99. Every sequence which had a value higher than or equal to $X^2_{.05}$ was marked as failed. We carried out these tests for the same data as in the previous section, in order to see if the embedded test can detect worse TRNG quality.

As we can see earlier in Table 2, the first usable generator output is for the frequency of 1666 Hz. The results of Poker Test follow these results and there is also a considerable difference between 2500 Hz and 1666 Hz (Table 4). One of disadvantages of the test is its higher computational effort which can be crucial in low-power consumption applications.

Table 4 Results of Poker Test for various frequencies

TRNG Frequency	Number of successful	Number of failed
5000 Hz	0 (0 %)	4064 (100 %)
2500 Hz	250 (6 %)	3814 (94 %)
1666 Hz	2830 (70 %)	1234 (30%)
1250 Hz	3656 (90 %)	408 (10 %)
1000 Hz	3779 (93 %)	285 (7 %)

3.2. Counter Distribution Test

One of the simplest tests of randomness is to calculate the mean value. It does not monitor the patterns in a sequence similarly to Poker Test, however it is very easily implemented. Usually it counts the number of zeros (or ones) during some period.

We used the advantage of the additional counter bits and designed a similar test. The test counts and records an occurrence of the last three bits of 16-bit counter words, where only values from zero to seven can be obtained. In an ideal case, we should have a uniform distribution of these values. Fig. 3 shows the distribution of last three bits of the counter values for various frequencies (the same set of data as in Table 2). We can see that the distribution is more significant than just mean value of the last bit.

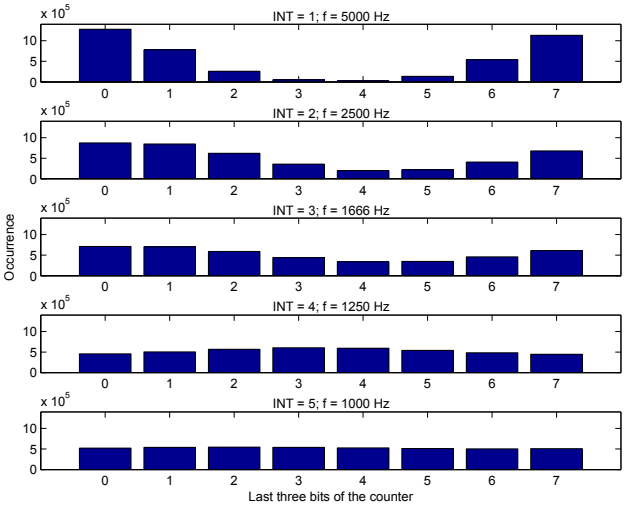

Fig. 3 Distribution of the last three bits of the generator

The next step was to find a proper threshold value for the TRNG test. As we can see in Fig. 3, the proper generator has a uniform dispersion, while wrong generator shows significant differences between values. Therefore we designed a double-sided threshold, in order to detect maximum and minimum values. The threshold is configured symmetrically on both sides from the estimated mean value. It is defined by percentage parameter s, which defines how much the sequence can be skewed from the mean value. If some value of the dispersion overruns one of the thresholds, the test is marked as failed. Top threshold t, and bottom threshold b for n samples of k bits are calculated as follows:

$$t = \frac{n(100+s)}{2^k 100} \quad b = \frac{n(100-s)}{2^k 100} \tag{2}$$

We did experimental tests for 4064 sequences of various frequencies with different threshold values (see Table 5). Our TRNG test uses sequence of size $n = 1032$ samples (one sample has $k = 3$ bits – last three bits of 16-bit counter). The test works very precisely, because for 2500 Hz and 5000 Hz it indicates the generator faults, similarly to the ENT statistical tests. We can also see some failures in frequency of 1666 Hz. They represent the false alarms, which don't mean the total failure of the generator. One of the total failure tests consist in a maximum allowed number of consecutive failures, the other consist in a maximum number of the threshold overruns (from one to eight).

Table 5 Number of failed sequences for different threshold values

Frequency	Top and bottom threshold value *s*			
	40%	50%	60%	70%
5000 Hz	4064 (100%)	4064 (100%)	4064 (100%)	4064 (100%)
2500 Hz	4064 (100%)	4064 (100%)	4064 (100%)	3815 (94%)
1666 Hz	3947 (97%)	2111 (52%)	367 (9%)	24 (1%)
1250 Hz	486 (12%)	19 (0%)	0 (0%)	0 (0%)
1000 Hz	21 (1 %)	0 (0%)	0 (0%)	0 (0%)

4. CONCLUSION

The paper presented TNRG implementation for low-cost ARM-M0 core based microcontroller, where the similar TRNG is typically missed and integrated only in higher families of the microcontroller. We described in detail its principle and implementation. We optimized its throughput and obtained the frequency of reading around 1666 bps. The achieved throughput is bigger than in the similar existing implementations.

We also designed two tests which can be used as embedded test - Poker Test and Counter Distribution Test. These tests use internal bits of the generator and they should detect the fault of the generator faster than statistical tests on the output sequence. The Counter Distribution Test is computationally simpler and can detect the generator fault similarly to (or even better than) Poker test. On the other hand, it just guarantees that the generator has proper distribution and it does not guarantee its randomness.

The following work will be focused on testing of the generator and its embedded on-line tests during various environmental conditions like temperature or power supply. We also plan to integrate the TRNG on similar ARM-M0 core based microcontrollers of various families.

Described TRNG design uses just noise of jittery clocks. Therefore one of the alternative to proposed TRNG can be in the future design with other clock sources, or design with different source of noise (e.g. thermal, or atmospheric noise).

ACKNOWLEDGEMENT

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-15-0692.

REFERENCES

[1] MENEZES, A. J. – Van OORSCHOT, P. C. – VANSTONE Scott: Handbook of Applied Cryptography, CRC Press 1996, ISBN 0-8493-8523-7, pp. 169.

[2] STM32-F0 datasheet of the http://www.st.com/resource/en/datasheet/stm32f031k6.pdf

[3] Freescale Semiconductor, KL03 Product Brief. Available at: http://www.nxp.com/docs/en/product-brief/KL03PB.pdf

[4] FAIFIELD, R. C. – MORTENSON, R. L. – COULTHART, K. B.: An LSI Random Number Generator (RNG), Lecture Notes in Computer Science, Vol. 196, Springer-Verlag, Berlin Heidelberg New York 1984, pp. 203-230.

[5] HLAVAC, J. - HADACEK, M. - LORENCZ, R.: True Random Number Generation on an Atmel AVR Microcontroller. Proceedings of 2nd International Conference on Computer Engineering and Technology ICCET 2010, 2010, Vol. 2, ISBN 978-1-4244-6347-3, pp. 493-495.

[6] Analysis of a True Random Number Generator, Master's Thesis, Bc. Simona Buchovecka, Czech Technical Universiry in Prague, Faculty of Information Technology, Department of Computer Systems, 2012.

[7] Atmel Corporation, 8-bit AVR Microcontroller with 16K Bytes In- System Programmable Flash ATmega169P, ATmega169PV, Document No. 8018N-AVR-08/09, 2009. Available: http://www.atmel.com/images/doc8018.pdf

[8] WALKER, J.: ENT A Pseudorandom Number Sequence Test Program, Switzerland, 2008. Available at: www.fourmilab.ch/random

[9] ABDEL-REHIM, W.M.F. – ISMAIL, I. A. – MORSY, E.: Testing Randomness: Poker Test with Hands of Three Numbers Journal of Computer Science, Vol. 8, 2012, ISSN 1549-3636, pp. 1353-1357. Available at: http://thescipub.com/PDF/jcssp.2012.1353.1357.pdf

Received September 12, 2017, accepted February 19, 2018

BIOGRAPHIES

Marek Laban was born in Vranov nad Topľou in the Slovak Republic, in 1992. He obtained his bachelor's and master's degrees in 2014 and 2016 in Electronics and Communication Technologies from Faculty of Electrical Engineering and Informatics, Technical University of Košice. He is currently working as a hardware engineer in MICRONIC a.s. – company focused on IT security products. Concurrently, he is an external Ph.D. student in Department of Electronics and Multimedia Communications. His current research focuses on secure implementations in FPGA circuits.

Milos Drutarovsky was born in Prešov in the Slovak Republic, in 1965. He obtained his Ing. (M.Sc.) degree and Ph.D. degree in Radioelectronics from the Faculty of Electrical Engineering, Technical University of Košice, in 1988 and 1995, respectively. He defended his habilitation work – Digital Signal Processors in Digital Signal Processing in 2000. He is currently working as an associate professor at the Department of Electronics and Multimedia Communications of the Faculty of Electrical Engineering and Informatics, Technical University of Košice. His current research focuses on embedded electronics, applied cryptography, algorithms and architectures for embedded cryptographic architectures, digital signal processing, digital signal processors, field programmable devices, and soft microcontrollers embedded into FPGA circuits.