Cryptography in Ultra-Low Power Microcontroller MSP430

Radek Fujdiak[1], Jiří Mišurec[2], Petr Mlýnek[3], Ondřej Rášo[4]

Department of Telecommunications
Faculty of Electrical Engineering and Communication
Brno University of Technology, Czech Republic

Abstract **– This article describes an implementation for the method from Texas Instruments for a random number generator in the ultra-low power microcontroller MSP430x5xx Families and the analysis of this generator in the concrete microcontroller MSP430f5438A. The generator may be used to generate numbers for cryptography security methods (for example in the cryptosystem Diffie-Hellman). A short theoretical introduction to the cryptography generators, a description of implementation, some practice examples and an analysis (and its description) for con- Crete example of the generator are provided in this article.**

Keywords **- Cryptography, Implementation, Low-power Micro-controller, MSP430, Random Generator**

I. INTRODUCTION

The random number generators (RNG) are today used in everyday life in banking systems [1], data transfers [2], lottery systems [3], electronic communications [4], it is evident the random numbers today touching nearly all possible technical and also everyday life fields.

Two basic types of number generating are existing, hard-ware (called also as Non-Pseudo Random generator – NPRGN or True Random Generator – TRGN) and software (called also as Pseudo Random Generator – PRGN). The hardware generating using physical random events for example shot noise [5], radioactive decay [6], spontaneous parametric down- conversion [7], etc. The hardware generators are slower (than software) with pretended higher randomness, this is mostly mean truly random numbers. These generators require special modules (events) for creating random process and it is necessary periodically test the randomness, because the events are changing in time and that is also mean the reconstruction of the same generating process is not possible). The software generating is mostly faster with smaller randomness, pseudo- randomness. Pseudo-random numbers mean that these numbers are periodically repeated after some concrete time (value of number). The software generators using mathematical apparatus and the randomness depend on difficulty of this apparatus, which also define the periodicity. [8][9]

In general all types of generators are used, but it is necessary to count with advantages and disadvantages of these generators and choose the best (or better) one for concrete situation.

This article is focused on random number generators for low-power devices, concretely the microcontrollers MSP430x5xx Families from Texas Instruments Company. It is used in many devices for its good energy properties as battery saving in portable devices [10], real-time capability with ultra-low power consumption, active and low power mode [11] and many more.

II. POSSIBILITIES OF RNG IN MSP430

The random number generator for MSP430 can be created with the function `rand()`, which is included in the library `stdlib`. This generator (method) is PRGN and is called Linear Congruential Generator (LCG). This genera-tor is really fast and do not need big memory space. But it is used only for basic examples of numbers generating or some beta-testing in phases, where is not necessary generate truly (or enough strong) random numbers. For practical devices (real use) the randomness of this numbers is insufficient [12], even when the better seed is used is not poss ible this function use in situation, where we need strong security level of random number generating.

It can be also used many kind of implementations from various different authors, but this generators will every time generate only pseudo-random combinations of numbers [13] and from basic description of PRGN is evident it have bigger requirements for memory or energy compare to hardware generators, which use for generating physical events and not mathematical apparatus . This is big disadvantages for ultra-low-power devices and also the reason for choosing some hardware solution. When the generator is used in practical situations (devices) it is also necessary to count the final price. When it will be used some external module, it is evident the final price will grove and also reason for using internal modules (if it is possible).

A. BASIC WORK WITH MSP430

This is a basic introduction to the work with MSP430. It is simply described work with the parts of MSP430, which are used for random number generator.

1) Diode and Watchdog: One green diode is included in MSP430x5xxx. This diode is used as a control device, for example each state can blink by different frequency. In Code 2.1 the basic control of the diode is showed. For blinking is necessary to put the control orders in some loop function [14].

```
P1DIR |= 0x41; // Set the diode PIN
P1OUT = 1; // 1 for ON, 0 for OFF diode
P1OUT ^= 0x41; // negation of diode stat
```

Code 2.1: Diode control

The Watch Dog Timer (WDT) is a hardware timer used to trigger a system reset if software neglects to regularly service the watchdog (after a certain amount of time). The WDT have some events, what may control some situations, but for most situations it is better to create own events and the WDT is better to set off (Code 2.2). [15] In general the WDT was created for system control to not fall in to the infinitive loop or some other system collapse [16], but for development new applications may be really difficult to set the WDT right and that is also the major reason for put it off.

```
WDTCTL = WDTPW + WDTHOLD; // Stop WDT
```

Code 2.2: Stop watch-dog timer

But the WDT may be also used for creating interruption between diode blinking but this may be also handled by simple `for` cycle (Code 2.3).

```
for (int i;i<10;i++) {
 for (int j;j<6000;j++) {}
  P1OUT ^= 0x41;}
```

Code 2.3: Cycle for diode blinking

First `for` (with variables `i`) provide blinking, number of blinks is in example 5 (10 times change state). Second `for` (variables `j`) provide pause between state changing. Variants for this are many (for one more example "delay cycle", which may be used as pause between blinking (Code 2.4)).

```
for (int i;i<10;i++) {
  __delay__cycles(5000)
  P1OUT ^= 0x41;}
```

Code 2.4: Delay cycle for diode blinking

But the final influence for program function is minimal.

2) Clock modules: MSP430 has three basic clock modules Auxiliary clock (ACLK), Master clock (MCLK), and Subsystem Master Clock (SMCLK) (figure 2.1). Two of them can be used as a signal for Timers [17].

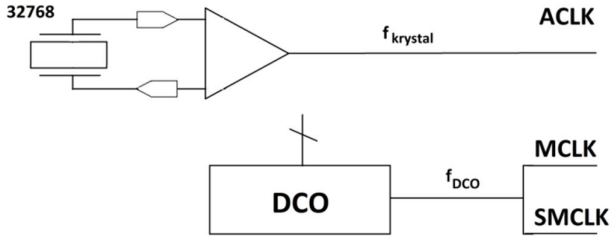

Figure 2.1: Clock system in MSP430

The SMCLK is using a digital controlled oscillator (DCO), which runs on a frequency around 1 MHz. The DCO can be also re-set by divider to 1, 2, 4, 8, 12 and 16 MHz (this is a basic divider possibilities, but the DCO may be also set to any other frequency by this divider). The frequency of the clock module may be changed by changing registers `BCSCTL` (basic clock system control registers) and `DCOCTL` (digital clock control register). Code 2.5 shows how to set the frequency to 1 MHz.

```
BCSCTL1 = CALBC1_1MHZ;// Set range
DCOCTL = CALDCO_1MHZ; // DCO step,modul.
```

Code 2.5: Setting 1 MHz in DCO

The ACLK is running on a much lower frequency, crystal with basic frequency 12 KHz. Similar to DCO the frequency of crystal can be re-set by divider (but here it may be only strict value of 1, 2, 4, 8 and 16). It is also used the register `BCSCTL1`, but with a different variable. All of the registers and possibilities may be found in the library `MSP430f5438`.

3) Timers and Interrupts: Two different timers `TIMER_A` and `TIMER_B` are exist. The differences between them are only question about queue priority of some register [18], what is irrelevant. The `TIMER_A` (figure 2.2) will be described, because the work with the both timers is similar and the `TIMER_A` is in the RNG used.

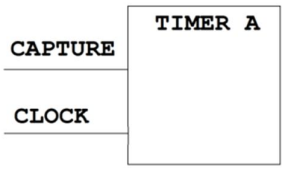

Figure 2.2: TIMER A

The `TIMER_A` is a 16-bit counter, having 4 modes of operation (stop, up, continuous and up/down), 3 capture/compare registers (`CCR1` – `CCR4`) and 2 interrupts vectors (`TACCR0` and `TAIV`) [19].

With using Timer Mode Control (MCx) may be set all of the four timer modes (Table 2.1).

MCx	00	01	10	11
Mode	Stop	Up	Continuous	Up/Down

Table 2.1: Timers Modes

The Stop Mode halts the timer from counting. Up Mode repeatedly counts from zero to the value of `TACCR0` register. Up/Down Mode is repeatedly count from zero to the value of `TACCR0` register and back to zero. Continuous Mode repeatedly counts from zero to 0xFFFFh value. The example of continuous mode function is showed on the figure2.3.

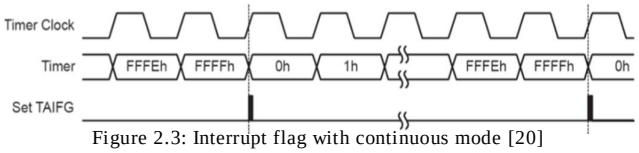

Figure 2.3: Interrupt flag with continuous mode [20]

The timer has three basic parts. The clock input is first, which has its own source (ticks at a specified rate) for example SMCLK or ACLK. The counter is second, which

has a specified mode and the interrupt procedure, which is called when the concrete value (counter limit) is reached.

For a better understanding of how to set a timer, it is possible to look in the library `MSP430f5438`, where the basic registers of `TIMER_A` can be found.

It is showed (Code 2.6) only simple example, when the count limit should be set to 12000 (`TA0CCR0`), with enabled interrupt flag (`TA0CCTL0`) and sources (`TA0CTL`) ACLK with 12 KHz (`TASSEL_1 + MC_2`). Values as `MC_2` or `TASSEL_1` are pre-set values, they may be also found in the library `MSP430f5438` (it is only a different and easiest way for set the bits in register).

```
TA0CCR0 = 12000; // count lim.,used CCR0
TA0CCTL0 = 0x10; // enable interrupts
TA0CTL = TASSEL_1 + MC_2;
```

Code 2.6: Simple settings for TIMER_A0

From the code is evident the `TIMER_A` may have sub-timers (threads) of its self (for example in Code 2.6 is used thread `TIMER_A0`).

The Timers of MSP430 are very complicated and it is no space to show all possibilities. This could be a topic for another article.

III. HARDWARE RNG USING TIMERS

How it is evident from previous chapters this article will describe the hardware random number generator, which will use internal modules. The basic idea, which will be in this article followed coming from Texas Instruments Company [21], the implementation of this method is showed in Fig. 3.1.

```
1. Set and start ACLK, SMCLK timers
2. Count ticks of SMCLK
3. Wait till ACLK will tick
<ACLK TICK>
5. Read SMCLK ticks
6. Save LSB from number of SMCLK ticks
(in left-shifted register)
7. Repeat everything again
(till will be reached required size of
random number)
```

Figure 3.1: Basic idea of hardware RNG for MSP430

Concretely it will be two clocks (SMCLK, ACLK) and one timer (TIMER_A and two of threads of it). The clocks SMCLK and ACLK are two independent clocks. This fact can be used for generating numbers if it is done right [21]. The ACLK running in basic on 12 KHz, the SMCLK clock running in basic on 1 MHz (in real the frequency circling around
0.9 to 1.1 MHz). One of the clocks is ticking slowly and one fast (Figure 3.2), then when the slower (ACLK) will have first tick the faster (SMCLK) will have n-ticks.

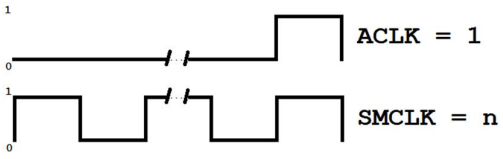

Figure 3.2: Comparison of ACLK and SMCLK

Because the SMCLK is circling around 1 MHz, the number `n` will be each time different (random). This event may be used as physical source for hardware random number generator. The LSB (least significant bit) from the number `n` may be used as one single random bit. This means, that if this operation is done in loop (circle), it can be easily generate a line of random numbers. The basic scheme of functionality for this type of generator is showed in the Figure 3.3.

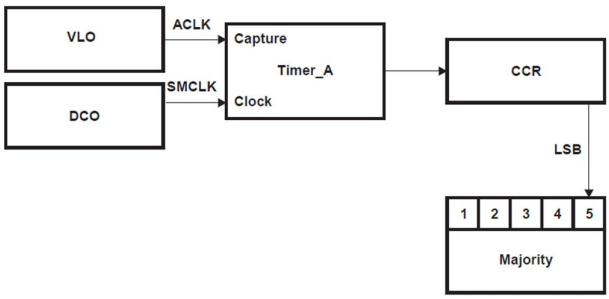

Figure 3.3: RNG in MSP430 using timers

It is used the `TIMER_A` (two threads of its), two clocks (SMCLK, ACLK), CCR register for load a LSB bit and left-shifter register for saving random numbers (single LSBs). First it is necessary to set an input of `TIMER_A` (concretely two threads of `TIMER_A`), first `Timer_A0` for VLO (Code 3.1) and second `TIMER_A 1` for DCO (Code 3.2).

```
TA0CCTL0 = CAP | CM_1 | CCIS_1;
TA0CTL = TASSEL_2 | MC_2;
```

Code 3.1: First sub-timer (`TIMER_A0`)

The tag `TA0` shows to what thread of `TIMER_A` belongs the register. The register `CCTL0` chooses the mode of the counter and `CTL` register chooses the clock source.

```
TA1CCR0 = 1;
TA1CTL = TASSEL_1 | MC_1;
TA1CCTL0 = CCIE | OUTMOD_3;
```

Code 3.2: Second sub-timer (`TIMER_A1`)

The tag `TA1`, register `CCTL` and `CTL` doing the same as in previous code. The register `CCR0` set the limit value of the counter. This is necessary because this thread will count the ticks of ACLK.

In the Figure 3.4 is showed little differences compare to Figure 3.3. These differences will be closer described in the following text.

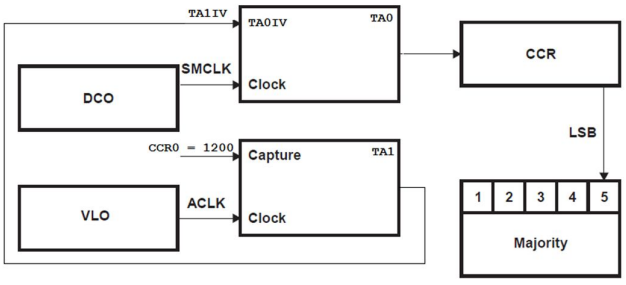

Figure 3.4: Real schema of the RNG

The Figure 3.4 shows procedure for generate one single random bit. Two threads of TIMER_A (TA0, TA1) are used. The DCO source (SMCLK) is used for the timer TA0 and the VLO source (ACLK) is used for the timer TA1. The timer TA1 is set as a counter, which is counting (in up mode) ticks of VLO (in this settings it counts till one tick is reached). Between the counting is good use the low power mode (one of the main feature of MSP430) for saving energy (Code 3.3) [22].

```
__bis_SR_register(LPM3_bits | GIE);
```

Code 3.3: Set the low-power mode

It is very important it will not occur to double interruption (Figure 3.5 part A) and also the main generating process will be not included in interrupt vector function, because this may also create double interruption [23]. The interruption must be handled like in Figure 3.5 B. This mechanism will be more described in following text.

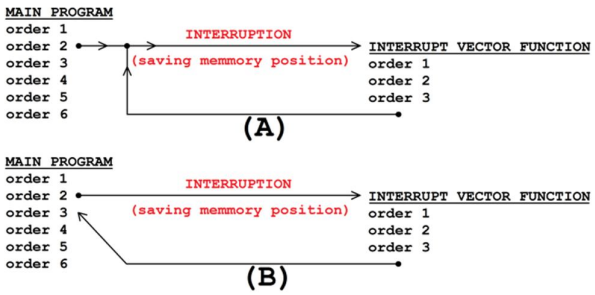

Figure 3.5: (A) Double-Interruption loop (B) Normal Interruption loop

The interruption is handled by interrupt vector TA1IV (Timer_A1 interrupt vector) and it occur to interruption, when is reached the value of register CCR0 of TA1. The TA0 and TA1 in that interruption time should be stopped (Code 3.4).

```
#pragma vector=TIMER1_A0_VECTOR
  __interrupt void Timer1_A0 (void) {
// TA1IV interrupt service routine
  TA0CTL = MC_0;
  TA1CTL = MC_0;
__bic_SR_register_on_exit(CPUOFF);
```

Code 3.4: Interrupt vector of TIMER_A0

The MC_0 in CTL register set the stop mode for timers. The last line is for going out from the low-power mode. For better understanding it may be followed Figure 3.6, where is the mechanism is described.

```
1. Set timers
2. Set interuption events
3. Set low-power mode ON
<SAVE THE POSITION>
<PROGRAM GO IN LOW POWER MODE>
<INTERRUPTION>
4. Orders in interrupt vector function
(program still in low-power mode)
5. Set low-power mode OFF
<PROGRAM GO OUT FROM LOW POWER MODE>
<LOAD THE POSITION>
6. Following orders in lines
```

Figure 3.6: Mechanism of Interruption and Low Power Mode

After the program step back to normal mode, the timers are in stop mode and it is possible to read the register TA0R, where is number of ticks of the Timer A0 (the number n, which was described in previous text).

This LSB from this number it should be saved to left-shifted register and after that the register TA0R must me re-set to zero (code 3.5) and whole mechanism can be repeated for generating more random numbers.

```
if ( (TA0R % 2) != 0 ) {
   Left_Shifted_Register |= (1 << i); }
```

Code 3.5: Left Shift Register saving random LSB

In Code 3.5 is used modulo operator (%), for decide if the last bite is logical "zero" or "one" (also odd or even) [24]. This operator includes many instruction cycles, which waste time and energy [25]. The program uses modulo order x-times, where x will be the number of required random bits (then for example 192-random bit number will use 192-times modulo operations). The easier and energy gentle solution is showed in Code 3.6, where is only controlled bit position of the number from TA0R register.

```
if (TA0R & (1 << 0))
```

Code 3.6: Gentle solution for LSB control

In following chapter will be showed some possibilities for grow a randomness and analysis of this generator.

A. GROWING THE RANDOMNESS

In this subchapter will be given some tips for increasing the randomness of the generator. On the Figure 3.7 some additional modules/register are showed, which can be used for increasing the randomness.

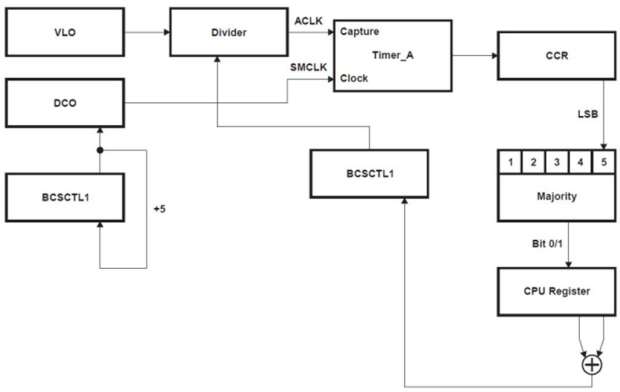

Figure 3.7: Additional modules for RNG [21]

Register BCSCTL1 (as it was showed in chapter II.A.2) is used for set the speed of DCO/VLO. Number five may be used for change the speed (increase the speed) of the DCO every time when is the LSB shifted (any other number may be also used, but five shows the biggest differences between DCO/VLO clock steps) and VLO change is depend on XOR function of two LSB bits.

The next thing what might be done is changing n (number of ticks). It could be static or random change in every round,

when is generated the LSB bit, but in this case is necessary to be careful, because this number even affects the speed of the number generating process.

Figure 3.8 showing the final ideas in order, for growing randomness this implementation should be followed.

```
1. Set and start ACLK, SMCLK timers
(with different TA1CCR0)
2. Count ticks of SMCLK
3. Wait till ACLK will tick
<ACLK TICK>
5. Read SMCLK ticks
6. Save LSB from number of SMCLK ticks
(in left-shifted register)
7. Increase speed of DCO by 5
8. Change speed of VLO by divider
(depend on xored LSB)
10. Repeat everything again
(till will be reached required size of
random number)
```

Figure 3.8: Idea for growing the randomness

The first idea (changing VLO/DCO) is already tested by series of statistical tests, described by the Federal Information Processing Standards (FIPS), concretely in implementation of FIPS 140-2 test from Texas Instruments Company [21]. It is necessary count the test was only implementation and not certified battery of tests. Every time is necessary to create model of the system and evaluate the entropy that is generated [26].

B. ANALYSIS OF THE GENERATOR

The following analysis will be focused on model described in the Chapter III. From the definition of statistical analysis is evident, that any analysis cannot prove the generating process is truly random. In each technique we trying to prove two assumptions, the generating process is not random (weak) or the generating process may be random. Simply we are trying to prove one of the input hypotheses. Many analytical techniques are existing for example from NIST, DIEHARD, STS
etc. [27].

The following tests from NIST organization will be used for this analysis:

- Binary Matrix Rank Test
- Discrete Fourier Transformation (Spectral)

The Random Number Analyzer (RNA) from NIST will be used for these tests [28]. The 240.000 bits was generated (used) for the following analyses.

1) The NIST RNA testing: The NIST testing strategy trying apply the statistical tests with appropriated parameters to the generated sequence of bits and after that trying to examine (and analyze) the Probability (P-value) for fixed critical value (α) a certain percentage are expected to failure. This techniques proving one from three hypotheses (scenarios):

- The analysis proving non-randomness character (inappropriate character). This is the Alternative Hypothesis (H_A).
- The analysis proving appropriate character for the analysis. This is the Null Hypothesis (H_0).
- The analysis is inconclusive.

First hypothesis is proved when critical value is higher than P-value, secondary is proved when critical value is lower than P-value and the last hypothesis coming from facts of test description [29]. This test is called Kolmogorov-Smirnov test and it is defined as

$$d = Max\left|F(x) - E(x)\right|,$$

where $F(x)$ and $E(x)$ are the theoretical and empirical distribution function evaluated at x, respectively [30].

2) Binary Matrix Rank Test: This test is included in DIEHARD, also in NIST battery of tests and in the end it is recommended also in online random generator (*www.random.org* – the services for generating random numbers via atmospheric noise) as one of the analytical methods. [31][32] That it is also reason for using this analysis for test this generator.

The test divides the generated sequence into blocks whose length is determined by the sub-matrices and it is counted the rank of these matrixes. The truly random number sequence should have only a few linear correlations (the number of linear correlation depends on ranks of matrixes). The X^2 distribution for (M, Q) matrixes

$$x_{obs}^2 = \frac{(F_M - E_1 N)^2}{E_1 N} + \frac{(F_{M-1} - E_2 N)^2}{E_2 N} + \frac{(F_r - E_3 N)^2}{E_3 N},$$

where the F are the calculation of the rank of every sub-matrix and the result for them is

F_M = *full rank (the number of matrixes with rank is M),*
F_{M-1} = *full rank -1 (the number of matrixes with rank is M-1),*
$F_r = N - F_M - F_{M-1}$ *(the number of matrixes remaining)*

and the M is number of rows and the Q is number of columns. For the matrix 32x32 is the (E_1, E_2, E_3) [28],

$(E_1, E_2, E_3) = (0.2888, 0.5776, 0.1336)$.

For final result of *P-value* is necessary to use the complementary incomplete gamma function (`igamc`)

$$P\text{-}value = \textbf{igamc}\left(1, \frac{x^2_{obs}}{2}\right) = e^{\frac{-x_{obs}^2}{2}},$$

If the *P-value* is between values 0.01 and 0.99, the sequence passes the test, otherwise fails [33].

The result of the NIST test is for *P-value* is:

P-value = 0.7170200592.

It is evident the value falling to the (0.01, 0.99) interval and the null hypothesis is accepted.

3) Discrete Fourier Transformation: The Fourier transformation is used in the randomness analytical methods really often and really long, the variants can be different as FFT, DFT, but the basic and important stuffs is that this analysis is again one of the most used for analysis [34][35][36]. This type of analysis is also used in NIST test suite and for the online random generator (*www.random.org*). These facts are used for choosing this type of analysis.

This test detects the periodic features (repetitive patterns that are near each other) in the generated sequence. These

features would indicate a deviation from the assumption of randomness. The detecting is based on searching the peak heights in the DFT. The basic description of this test is coming from [26], the main parts of this analysis is *sliding window*, which is using for converting the input sequence (ε) to the sequence (X) of -1 and +1. Next parts are *DFT function, modulus function and the error function (erfc)*. The simple description of this test may be as:

(1) The basic values are length of the bit string n, the periodic sequence PS (for example "10").

(2) Use sliding window for create the sequence $X = x_1, x_2, \ldots x_n$, where $x_i = 2\varepsilon_i - 1$. The ε is transformed to -1 or +1 values, depend if the PS match (+1) or not (-1)

(3) Apply DFT on X, $S = DFT(X)$ and calculate $M = mod(S') \equiv |S|$, where S' is the substring consisting of the first $\frac{n}{2}$ elements in S and *mod* function produces a sequence of peak heights.

(4) Compute $T = \sqrt{\left(\log\frac{1}{.05}\right)n}$ = the 95% peak height treshold value.

(5) Compute N_i (the actual observed number of peaks in M that are less than T)

(6) Compute $d = \frac{(N_1 - N_0)}{\sqrt{n(.95)(.05)/2}}$.

(7) Compute $P\text{-}value = \boldsymbol{erfc}\left(\frac{|d|}{\sqrt{2}}\right)$.

The result of the NIST test for *P-value* is:

$$P\text{-}value = 0.1897834012.$$

Compare to critical value the null hypothesis is also accepted.

IV. CONCLUSION

The generator, which is described in this article, was successfully tested. The results of this test showing, that the generator can be used in practical devices for generating random numbers. In the text is also showed the implementation and the way how the generator should be programmed, followed by practical advices for realization.

ACKNOWLEDGE

Thank to the Project TAČR 02020856 – Aplicated research of intelligent systems for monitoring energy networks and to the University of Technology Brno.

REFERENCES

[1] RONCIN, Marcel. Succession des protocols ETEBAC. November 2008. Available from: http://www.cfonb.org/Web/cfonb/cfonbmain.nsf/DocumentsByIDWeb/7KUEQA/$File/ETEBAC%2017112008%20%20V2.pdf

[2] RANGARAJAN A.V., SUGATA S.S., AJITH A., DHARMA P.A. Jigsaw-based Secure Data Transfer over Computer Networks. 2004. Available from: http://arxiv.org/ftp/cs/papers/0405/0405061.pdf

[3] SCHWARTZ, Jim. Advanced Lottery Theory. 2008. Available from: http://www.satorimediaworks.com/software/LS/Lottery_theory.pdf

[4] GALLAIS A., CARLE J., SIMPLOT-RYL D., STOJMENOVIČ I. Localized Sensor Area Coverage With Low Communication Overhead. 2006. Available from: http://hal.archives-ouvertes.fr/docs/00/07/08/79/PDF/gallais-percom-06.pdf

[5] JUN B., KOCHER P. The Intel® Random Number Generator. 1999. Available from: http://www.cryptography.com/public/pdf/IntelRNG.pdf

[6] KIMURA, Hiroshi. Physical Random Number Generator using Isotropic Radiation. 2001. Available from: http://isi.cbs.nl/iamamember/CD2/pdf/545.PDF

[7] XAVIER G.B., FERREIRA da S.T., VILELA de F., TEMPORAO G.P., WEID J.P. Practical Random Number Generation protocol for entanglement-based quantum key distribution. Available from: http://arxiv.org/ftp/arxiv/papers/0810/0810.0483.pdf

[8] GLOSEMAYER D., KNAPP R. Random Number Generation. 2010. Available from: http://www.wolfram.com/learningcenter/tutorialcollection/RandomNumberGeneration/RandomNumberGeneration.pdf

[9] TURNER, Noah. Software vs. Hardware RNG's. 2005. Available from: http://www.tstglobal.com/assets/downloads/1268986797a16.pdf

[10] Texas Instruments. MSP430 Ultra-Low-Power Microcontroller. 2008. Available from: http://www.ti.com/lit/sg/slab034w/slab034w.pdf

[11] Texas Instruments. Noah. MSP430 Microcontroller Family. 2000. Available from: http://gse.ufsc.br/~bezerra/disciplinas/Microprocessadores/MSP/Application%20Book/chp1.pdf

[13] PRESS, H. William. The Art of Scientific Computing (2nd ed.). 1992.

[14] LITOVSKY, Gustav. Beginning Microcontrollers with the MSP430. 2010. Available from: http://www.glitovsky.com/Tutorialv0_2.pdf

[15] Recursive Labs. Programming the watchdog timer. 2011. Available from: http://recursive-labs.com/static/courses/rl100/samples/watchdog.pdf

[16] DAVIES, H. John. MSP430 Microcontroller Basics. 2008.

[17] WISMAN, Ray. MSP430 Timers and PWM. 2012. Available from: http://homepages.ius.edu/RWISMAN/C335/HTML/msp430Timer.HTM

[18] QUIRING, Keith. MSP430 Timers In-Depth. 2006.

[19] Texas Instruments. MSP430x2xx Family User's Guide (Revised January 2012). December 2004.

[20] WANG, Yin. MSP430 Clock System and Timer. 2007. Available from: http://www.ccs.neu.edu/home/noubir/Courses/CSU610/S07/MSP430-Clock-Timers.pdf

[21] Texas Instruments. Random Number Generation Using the MSP430. 2006.

[22] OLSON, David. Tutorial 11-a: Going Low Power. October 2010. Available from: http://mspsci.blogspot.com.es/2010/10/tutorial-11-going-low-power.html

[23] KING, Chung-Ta. CS 4101 Introduction to Embedded Systems (LAB4). 2011.

[24] ALLAIN, Alex. Bitwise Operators in C and C++: A Tutorial. 2011. Available from: http://www.cprogramming.com/tutorial/bitwise_operators.html

[25] Texas Instruments. ULP Advisor for MSP430 (5.1). 2013. Available from: http://processors.wiki.ti.com/index.php/Compiler/diagnostic_messages/MSP430/1530

[26] NIST. FIPS PUB 140-2. May 2001. Available from: http://web.archive.org/web/20070817151620/http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[27] SOTO, Juan. Statistical Testing of Random Number Generators. 2012.

[28] NIST. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. August 2008.

[29] KRHOVJÁK, Jan. Statistical Testing of Randomness. 2005. Available from: http://www.fi.muni.cz/~xkrhovj/lectures/2005_PA168_Statistical_Testing_slides.pdf
[30] WANG, Hsiao-Mei. Comparison of the Goodness-of-Fit Tests: the Pearson Chi-square and Kolmogorov-Smirnov Tests. 2008.
[31] FOLEY, Louise. Analysis of an On-line Random Number Generator. 2001. Available from: http://www.random.org/analysis/Analysis2001.pdf
[32] CHARMAINE, Kenny. Random Number Generators: An Evaluation and Comparison of Random.org and Some Commonly Used Generators. 2005. Available from: http://www.random.org/analysis/Analysis2005.pdf
[33] LI, Liang. Testing several types of random number generators. 2012.Available from: http://www.cs.fsu.edu/research/theses/Liang_L_Thesis_2012.pdf
[34] DAVIS, Richard A. Introduction to Statistical Analysis of Time Series. 2003.Available from: http://www.stat.columbia.edu/~rdavis/lectures/Session6.pdf
[35] STEINFOLWF, Alexander. Random Vibration testing Beyond PSD Limitations. 2007.Available from: http://www.sandv.com/downloads/0609stei.pdf
[36] HINICH, Melvin J., NEBDES, Eduardo M. A. M. A New Statistical Approach to Evaluating Random Number Generators. 2009.Available from: http://www.la.utexas.edu/hinich/files/Statistics/Bisprandom.pdf
[37] HAMONO, Kenji. The distribution of the Spectrum for the Discrete Fourier Transform Test Included in SP800-22. 2005. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.99.3252&rep=rep1&type=pdf