A Low-Power Wearable Electronic Dice Using a Composite PRNG Seeded by Noisy On-Chip Temperature Acquisition

Vasile Surducan and Robert Gutt

***Abstract*—We describe a simple yet powerful wearable device containing an embedded system composed of proprietary designed hardware and firmware which is able to generate random seeds. Each seed generated after the hardware is powered on is further used by a classical pseudo-random number generator routine. The resulted pseudo-random number generator (PRNG) is stored on a laptop for testing purpose using a serial to universal serial bus (USB) interface and also truncated and displayed as dice symbols on seven ultra-low power LEDs. The random generator quality has been computed using various testing methods by acquiring a relevant number of samples of random buckets generated by hardware immediately after powering on. Statistical National Institute of Standards and Technology (NIST) and Diehard suite tests reveal the feasibility of the random seed and pseudo-random numbers generation. The electrical performance of 25 manufactured devices was also measured in ON and OFF state. The hardware has a maximum leakage current of 220 nA and draws an average of 0.8 mA at 3 V supply during displaying of ten successive dice symbols in 8 seconds, running with the same CR2032 battery for more than one year. The method used for variable seed generation does not require the use of any type of volatile or non-volatile memory. Our device and seed generation method can be used in any scientific application or toy industry requiring true random numbers generation at ultra-low power supply.**

***Index Terms*—wearable device, RNG, embedded system, ghosting, temperature measurements.**

Original Research Paper
DOI: 10.53314/ELS2630084S

I. Introduction

RANDOM number generators (RNGs) are crucial for many applications such as scientific computations [1] and cryptography [2]. The evolution of embedded systems such as sensor networks and the Internet of Things are increasingly reliant on cryptography. While early sensor networks depended on link-layer encryption and used pre-installed keys [3], modern practices require devices to use randomness to determine the pre-session keys [4]. RSA (Rivest-Shamir-Adleman) and DSA (Digital Signature Algorithm) security encryptions can fail catastrophically when used with malfunctioning random number generators [5]. Besides their indisputable necessity for information security and scientific computing, RNGs also provide the basis for many games, ranging for lottery simulations, card games to modern computer games. Therefore, the element of chance provided by RNGs is a key element for realism, continually-evolving scenarios, or re-playability for the game community. Hence, the need for "better" RNGs which are not predictable and cannot be easily decrypted is continuously growing.

Throughout literature there are a lot of Pseudo Random Number Generators (PRNG) based on computer algorithms such as Linear Congruential Generator (LCG), Lagged Fibonacci Generator (LFG), and a Linear-Feedback Shift Register (LFSR) [6]. All of these algorithm software oriented are limited by the periodic intervals of the output sequences. To provide the appearance of randomness, such algorithms use a long period compared to the random numbers used in any application (modern computer efficiency renders a RNG with period of $2^{31} - 1$ quite predictable). The usefulness of such PRNGs is given by the fact that they produce the same sequence given the same initial seed, which is important for testing different applications on different systems.

In order to generate a pseudo-random number, an initial seed stored in a number of words is required. The seed value has to be modified before each newly generated number. However, most of the proposed techniques require a volatile memory in order to store the previous seed to be further used. Sub-milliwatt hardware requires algorithms with intermittent (start–run–stop) power sequences to conserve the supply energy. Thus, before each stop, the last pseudo-random value has to be stored in the non-volatile memory and read at the next start as new seed, or else, the same pseudo-random number array will be available at each start. This is an unwanted situation. One approach to generate the seed is to use a custom built hardware RNG. These on-chip random number generators are commonly included on modern processors and high-end microcontrollers but not in low cost devices. These hardware RNG rely on entropy gathering processes, which constitutes the fundamental property of true random number

Manuscript received on April 8th, 2026. Received in revised form on August 5th, 2026. Accepted for publication on August 30th, 2026.

V. Surducan is with the National Institute for Research and Development of Isotopic and Molecular Technologies (INCDTIM), 67-103 Donath Street, 400293 Cluj-Napoca, Romania (email: vasile.surducan@itim-cj.ro, ORCiD: 0000-0002-8275-373X).

R. Gutt is with the National Institute for Research and Development of Isotopic and Molecular Technologies (INCDTIM), 67-103 Donath Street, 400293 Cluj-Napoca, Romania (email: robert.gutt@itim-cj.ro, ORCiD: 0000-0003-3184-0902).

This work was financed by the "Research projects to stimulate young independent teams (TE)", number PN-IV-P2-2.1-TE-2023-1272, implemented by UEFISCDI, Romania. Also, the authors acknowledge partial support through the "Nucleu" Program within the National Research Development and Innovation Plan 2022–2027, Romania, carried out with the support of MEC, project no. 27N/03.01.2023, component project code PN 23 24 02 01.

generators (TRNGs) [7]. Therefore, TRNGs employ physical phenomena in order to generate a sources of randomness, such as jitter noise [4], [8], metastable circuits [3], ring oscillators [9], thermal or shot noises of analogue devices and even quantum processes, such as quantum entanglement (i.e. a quantum mechanical phenomenon where two or more particles become linked together in such a way that they share the same fate, no matter how far apart they are) or potential barrier tunneling [10]. For example in [11], the authors designed a chaos-based True Random-Bit Generator (TRBG) implemented on a 0.18-µm CMOS technology with unique composition formed by a chaotic jerk oscillator, a modulator, and pre/post-processing circuit. Also new and ingenious non-electronics designs are proposed for RNG such as microfluidics devices [12] based only on a constant water-head pressure generated by the push action of a human or strain devices [13], which exploits the metastability of the stressed straintronics (i.e. the study of how folds and mechanically induced stresses in a layer of two-dimensional materials can change their electrical properties).

The question at which we expect to have answers at the end of our research described in this paper are the following: (i) can be implemented a simple but powerful RNG in a low power 8-bit microcontroller running with repetitive start-stop supply sequences? (ii) how can we generate a true random seed at each power up without auxiliary hardware and without saving the last seed value in non-volatile memory? (iii) which is the most effective way of displaying the random number and what is the impact on the power supply? (iv) how can we test such RNG in a simple but accurate manner?

In order to answers to the previous questions a novel hybrid RNG combining a deterministic PRNG with true random seed derived from on-chip temperature acquisition implemented in a low power wearable dice device is manufactured in a batch of 25 pieces and distributed as gifts. Each user can informally observe whether successive button presses produce differing dice sequences. However, since this qualitative indication is not scientifically rigorous, the RNG quality was separately tested using a number of trusted methods.

The novelty of our low-cost 8-bit RNG device consists of a seed-generation based on start-up time entropy arising from microcontroller temperature variations, combined with an analogue to digital conversion deliberately configured to produce erroneous data reads. Secondly, the device uses an intermittent operation (start-stop) for power reduction, producing a random seed at each microcontroller start-up which generates a set of 10 pseudo-random numbers. The numbers are displayed on ultra low power LEDs as a dice character, in a sub-µA-leakage implementation using a HiZ ghosting-suppression technique. Thirdly, the implementation avoids storing the last generated seed in EEPROM (Electrically Erasable Programmable Read-Only Memory) or high-endurance flash memory for use after the next power-up and requires start–stop RNG architecture requiring no EEPROM/flash seed persistence and no auxiliary hardware for TRNG.

Proof of functionality must be given by the pseudo-random number generator (RNG) statistical tests results. Guidelines for designing random number generators are provided by the NIST [5]. One fundamental requirement is to use a noise source whose functionality results from probabilistic behavior, but also to protect the noise source from adversarial observation and moreover to provide the necessary documentation and testing result. Beyond these requirements, low power, wearable embedded applications impose limitations on energy consumption, cost, and size. Our proposed wearable device was manufactured in a small lot, operates in the sub milliwatt region and was designed for low cost and compatibility with low power microcontrollers, making it suitable for a broad range of scientific applications or even to be used in the toy industry. As we are aware, this type of RNG implementation in a wearable "mărțișor" device it is absolutely new.

II. Materials and methods

A. Design and prototyping methodology

Electronic schematic (Fig. 1) and printed circuit board (PCB) have been designed (Fig. 2) and manufactured (Fig. 3) using some variations of the standard well known procedures: SCH and PCB design, design errors correction, machines files export and CNC conversion, PCB by CNC manufacture and components stuffing.

Front side of the PCB was painted after component placement (except for the LED lighting paths). The LEDs body (except the lighting area) was masked with a 0.2 mm copper foil, representing different symbols (human head, snail, four sheets clover, etc) obtained by CNC cutting technology.

The software of the embedded system was written with a free Pascal compiler named Just Another Language (JAL). Several firmware iterations were written and tested to completely remove the LED light ghosting. LED ghosting is a parasitic light which appears during the switching process of microcontroller IOs (input-output). 25 prototypes devices (two pieces as reserves) were manufactured, tested, programmed, each verified and offered to our female colleagues (Fig. 4) on March 1st of 2025 (a Spring celebration common in the East of Europe, particularly to Romania) to enjoy and interact with. This wearable device name is "mărțișor", pronounced as [ˈmər.tsi.ʃor].

B. The embedded system hardware

The detailed schematic of the circuit is presented in Fig. 1 and the printed circuit board design on Fig. 2. No component is placed on the solder side except the coin soldered battery and a safety pin.

Once the start button BUT is pushed, the microcontroller turns on and provides a low voltage through pin 2 to the gate of the MOS transistor, feeding the supply power to the microcontroller for the on state of 8 second. After the generation and display of 10 random numbers, pin 2 turns high, cutting the supply voltage of the microcontroller which goes back to the off state. Seven ultra-low power LEDs are used sequentially to display the truncated sequence of random numbers as dice faces. For this design, only four output pins of the microcontroller are used.

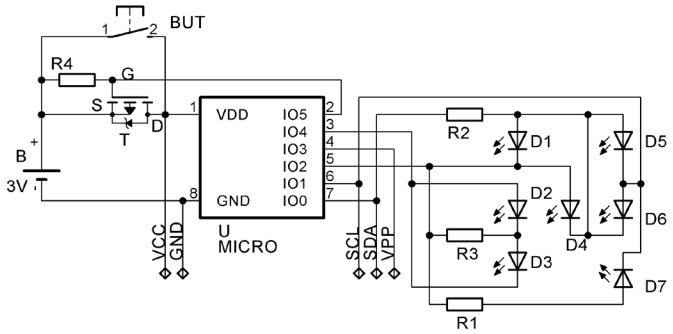


Fig. 1. The Hardware schematic.

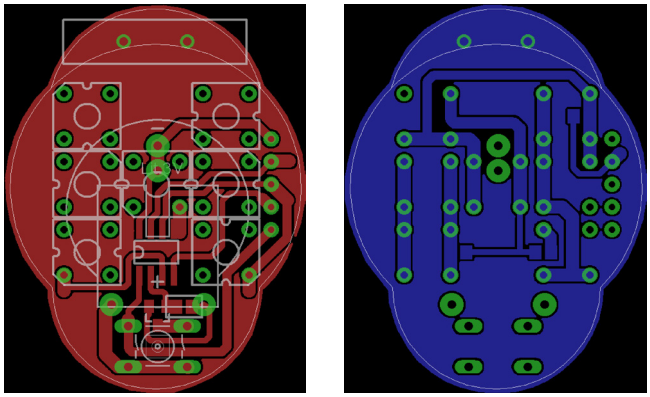


Fig. 2. Printed circuit board design (PCB), left: component side, right: solder side.

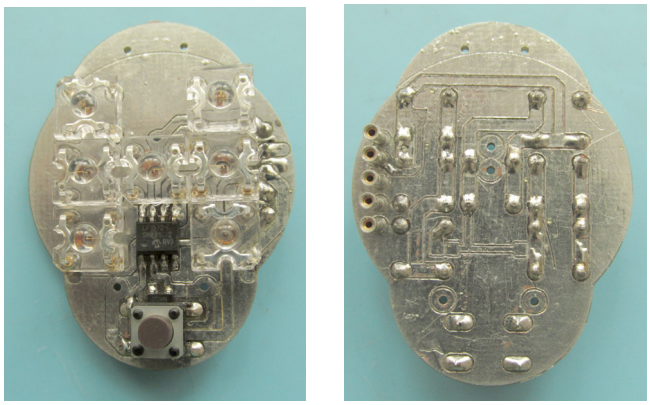


Fig. 3. The PCB equipped with electronic components.

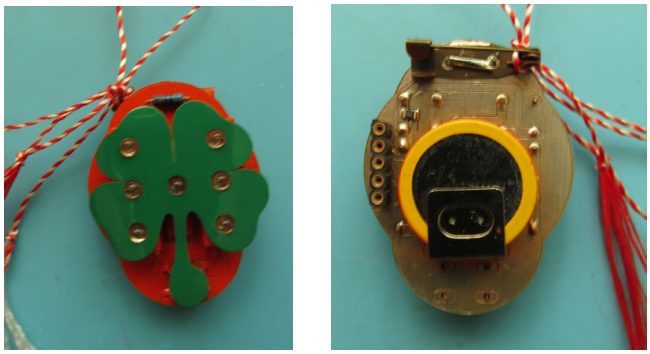


Fig. 4. The device aspect, left: front view, right: back view.

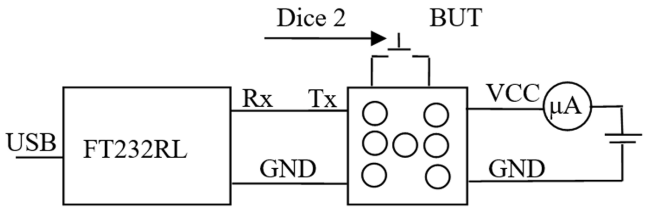


Fig. 5. Device hardware measurements and data communication.

C. Hardware measurements methods

The electrical performance of device was measured in on and off state by using a digital multi-meter (Owon B35) as micro-ammeter connected in series with the supply (Fig. 5). We measured the quiescent current during OFF state immediately after the LED dice display was turned off and the average of drawn current during the 8 seconds of LED light after each button push. Also a serial communication based on FT232RL converter IC was implemented for testing to send the pseudo-random arrays to a PC after each BUT command sent. To automatize the measuring process we actually used two similar devices where the first one was programmed for electrically "pushing" the button of the second device under test (Fig. 5), with a repetitive rate for 10 s, by using one IO pin as output command.

D. Software implementation techniques

We study a composite pseudorandom number generator (PRNG) that combines a *Linear Congruential Generator* (LCG) with a *Xorshift* generator, producing output as the sum of their respective states. This hybrid design, closely related to Marsaglia's `xorwow` generator [14], aims to combine the strengths of both components while mitigating their individual weaknesses.

The generator is defined over w-bit unsigned integers (typically $w = 32$), where all arithmetic is performed modulo 2^w.

Let x_n and y_n denote the states of the LCG and Xorshift components at step n, respectively.

1) Linear Congruential Generator

The recurrence of the LCG component is:

$$x_{n+1} = 69069 \cdot x_n + 123 \pmod{2^w}. \tag{1}$$

This is a standard LCG with multiplier $a = 69069$ and increment $c = 123$. By the Hull–Dobell theorem [15], this LCG has full period 2^w if and only if:

1) $\gcd(c, 2^w) = 1$ $(c = 123$ is odd),
2) $a - 1$ is divisible by all prime factors of 2^w, i.e., $2 \mid (a-1)$ $(69069 - 1 = 69068 = 4 \times 17267)$,
3) $4 \mid (a-1)$, if $4 \mid 2^w$ $(69068 = 4 \times 17267)$.

Hence, the LCG component has maximal period $T_{\text{LCG}} = 2^w$.

2) Xorshift Generator

The Xorshift state update consists of three successive xor-shift operations:

$$y_n^{(1)} = y_n \oplus (y_n \ll 13), \tag{2}$$

$$y_n^{(2)} = y_n^{(1)} \oplus (y_n^{(1)} \gg 17), \tag{3}$$

$$y_{n+1} = y_n^{(2)} \oplus (y_n^{(2)} \ll 5). \tag{4}$$

Each xor-shift operation can be expressed as multiplication by a matrix in $\mathbb{F}_2^{w\times w}$. Define the $w \times w$ identity matrix I and the *shift matrices*:

$$L_{ij}^k = \begin{cases} 1 & \text{if } j = i-k \text{ and } 1 \le j \le w, \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

$$R_{ij}^k = \begin{cases} 1 & \text{if } j = i+k \text{ and } 1 \le j \le w, \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where L^k represents a left shift by k bits and R^k a right shift by k bits. Interpreting y_n as a column vector $\boldsymbol{y}_n \in \mathbb{F}_2^w$, and noting that addition over $\mathbb{F}_2$ is XOR, the composite Xorshift update is:

$$\boxed{\begin{aligned} &\boldsymbol{y}_{n+1} = T\,\boldsymbol{y}_n, \\ &T = (I + L^5)(I + R^{17})(I + L^{13}) \in \mathbb{F}_2^{w\times w}. \end{aligned}} \tag{7}$$

3) Combined Generator

At each step the generator outputs the modular sum of the two states:

$$z_n = x_n + y_n \pmod{2^w}. \tag{8}$$

In summary, one iteration of the generator is:

$$\begin{aligned} x_{n+1} &= 69069 \cdot x_n + 123, \\ y_{n+1} &= y_n \oplus (y_n \ll 13), \\ y_{n+1} &= y_{n+1} \oplus (y_{n+1} \gg 17), \\ y_{n+1} &= y_{n+1} \oplus (y_{n+1} \ll 5), \\ z_{n+1} &= x_{n+1} + y_{n+1}. \end{aligned} \tag{9}$$

The combined generator has an order of $2^{32}(2^{32}-1)$ for the double-word state variables (x, y).

4) Seeding with Hardware Entropy

Secondly, a seeding algorithm based on the Xorshift RNG proposed by George Marsaglia [14] and modified for the JAL compiler by Stef Mientki was implemented and tested. To improve seed unpredictability, we combined the deterministic algorithm with a hardware entropy source: the chip temperature value is read by the analogue-to-digital converter (ADC) of the microcontroller with a lower acquisition time than specified in the datasheet and a variable reference for the ADC, yielding an unstable reading.

Starting from the default seed pair $(x, y) = (2282008, 362436069)$, the sampled ADC value is mixed into the state as follows:

$$\begin{aligned} x &\leftarrow x + ADC_value, \\ y &\leftarrow y - x. \end{aligned} \tag{10}$$

After seeding, the generator produces ten random numbers using the combined recurrence of (9). Since the sequence of interest is only ten numbers long, and given the difficulty of using variables larger than double words on the eight-bit microcontroller in use, this sequence length is satisfactory. We note, however, that long-period PRNGs are obtainable with Xorshift methods [14], [16].

5) Mapping to Dice Rolls

To convert a generator output z_n into a die face $d_n \in \{1, 2, \ldots, 6\}$, one computes $\varphi(z_n) = (z_n \bmod 6) + 1$. Because $6 \nmid 2^{32}$, the map φ introduces a small bias: the set $\mathbb{Z}/2^w\mathbb{Z} = \{0, 1, \ldots, 2^w - 1\}$ does not partition evenly into six residue classes.

For applications requiring exact uniformity the bias can be eliminated entirely via rejection sampling. Define the largest multiple of 6 not exceeding 2^w:

$$M = 6\left\lfloor \frac{2^w}{6} \right\rfloor = 6 \times 715{,}827{,}882 = 4{,}294{,}967{,}292. \tag{11}$$

The algorithm becomes:

$$d_n = \begin{cases} (z_n \bmod 6) + 1 & \text{if } z_n < M, \\ \text{resample} & \text{if } z_n \ge M. \end{cases} \tag{12}$$

The rejection probability is:

$$P(\text{reject}) = \frac{2^w - M}{2^w} = \frac{4}{2^{32}} \approx 9.31 \times 10^{-10}, \tag{13}$$

which is negligible.

E. Data acquisition procedure

The resulted pseudo-number sequence was stored on a laptop using a serial to USB interface based on FT232RL IC [17], an RS232 to USB converter widely used in embedded applications. On the laptop the free terminal software (Realterm 3.0) [18] is able to log data for long time intervals.

F. Statistical NIST and Diehard suite tests

In order to evaluate the "randomness" of the proposed device, we employ the National Institute of Standards and Technology (NIST) test suite for random and pseudo-random number generators [19], [20] and also the Dieharder test suite [21] for the generated sequences of numbers. Since the proposed device combines a pseudo-random number generator for multiple sequences of 10 numbers with a true random number generator based on the erroneous temperature read of the PIC12LF1572, considered as the seed of the RNG algorithm, we apply these statistical tests to the sequences of generated random numbers. In order to evaluate if the generated sequence is random, we applied four statistical tests for random number sequences: two statistical ones (Chi Square Test and the Serial Test [22]) and two empirical ones (Runs Test and The Binary Rank Test for 32x32 Matrices) of the Diehard test suit [23], [21]. Since no test can evaluate the randomness of a given sequence of numbers, we have applied those tests to show that the sequence is probably not non-random.

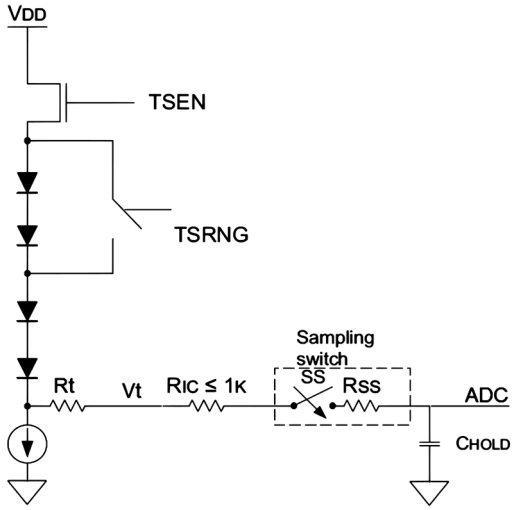


Fig. 6. Temperature and ADC measuring structure in PIC12LF1572.

III. Result and discussions

Implementing long and powerful TRNGs in any embedded systems with strong computational power (high frequency oscillator to provide very short time per instruction word) leads usually to serious amount of energy consumption.

One of the most important metrics of RNGs is entropy [5], which can be viewed as the number of bits of information not known to an adversary. Given sufficient entropy, a device can generate an unpredictable sequence using a deterministic algorithm called a pseudo-random number generator [2]. The authors in [16] have studied the entropy given by on board sensors by concatenating the least significant bit from the measurement of the sensor. The author found that best source of the entropy is the temperature sensor, at the cost of execution time, estimated at 1.7 second for one byte for their computing system. Since such a time interval was not acceptable for our device, we employ a deliberately erroneous hardware read of each temperature sample. The analogue to digital measuring procedure (Appendix, Fig. A-1) is altered by purpose in order to obtain a defective measured value of the temperature sensor by shorting the ADC sampling time below PIC12LF1572 datasheet value [24].

The ADC of PIC12LF1572 microcontroller has a sample & hold topology (Fig. 6), where a capacitor CHOLD of 12.5 pF is used to memorize the measured voltage. The time required to charge the capacitor depends on sampling switch resistor and interconnect resistors which are series connected between the capacitor CHOLD and the voltage source (Vt) to be measured. This includes the voltage source internal resistance (Rt). The sample time TACQ (14) is defined as the time between the moments when ADC is enabled (ADON bit set) and when the holding capacitor is disconnected from the measuring source by the sampling switch (SS).

$$TACQ = TAMP + TC + TCOFF, \tag{14}$$

where TAMP = 2 μs is the ADC amplifier settling time, TC = 1.5 μs is the CHOLD capacitor charging time with a voltage increase greater than 1/2LSB, TCOFF = (T - 25°C)(0/0.05 μs/°C) is the temperature coefficient.

At the maximum ambient temperature of +40°C and minimum supply voltage of 1.8 V, the internal resistance of the sampling switch (SS) exceeds 14 kΩ, thus minimum computed acquisition time it is at least TACQ = 3.5 µS (without consider the internal resistance Rt value of the temperature sensor). In real life there is a slight dispersion of TACQ between different microcontroller parts. For this reason TACQ is maintained in practice at it's highest value to acquire a stable input signal for any part taken from a lot of similar components. On the other hand, the internal temperature sensor output resistance is unknown, but its value is also contributing to the TACQ value. The voltage of the temperature sensor Vt is given in (15).

$$Vt = (Vdd - \mathrm{mode}) \cdot 0.659 - (\mathrm{T}[^\circ\mathrm{C}] - 40) \cdot 0.00132 \tag{15}$$

where Vdd is the power supply (1.8 ... 3 V), and mode = 2 (set from FVRCON_TSRNG bit) for low voltage supply.

Equation (15) indicates that a decrease in the sensor voltage of approximately -1.32 mV/°C reveals the silicon diode behavior of the temperature sensor. In terms of ADC result, (15) becomes

$$ADC_result = 1023 \cdot Vt/Vdd \tag{16}$$

Actually in (16), both the Vt and Vdd are variable over temperature and time, which is a bonus for RNG seed generation. Additionally, the ADC reference voltage which can be selected between an internal reference of 1024 mV and the Vdd supply it is tied to the external battery, since the battery voltage gradually decreases over time, it also affects the accuracy of the ADC measurements itself, producing variable ADC result.

Although the authors in [25] have analyzed the entropy given by the boot time of various embedded systems, such as DRAM decay behaviour and PLL locking latency on ARM-, MIPS-, and AVR32-based systems-on-a-chip, to the best of our knowledge no studies were employed on the boot-time entropy for internal temperature sensors of PICs microcontrollers. They also suggest the study include voltage scaling latency, GPIO pin voltage, flash memory corruption patterns, and power supply stabilization latency. While there are techniques for extracting randomness from low-entropy sources [26], [27], the cost of this extraction is prohibitive in resource-constrained embedded environments. From this perspective, our solution is less costly.

The entire software implementation of our device is shown in Appendix (Listing A-1) by using pseudo-logic language.

A. Hardware performance of the embedded system

Our "mărțișor" [28] device is to be offered to the ladies as present during the beginning of March and is accompanied by a red-white braided cord, and usually is worn on clothing to be seen for about one week. Obviously the size and weight of such device must be reduced to minimum (including the battery) as well the power consumption must be very low.

Two type of current measurement were performed on all 25 devices in agreement with Fig. 5: the leakage current measured in the off state immediately after microcontroller turns off and

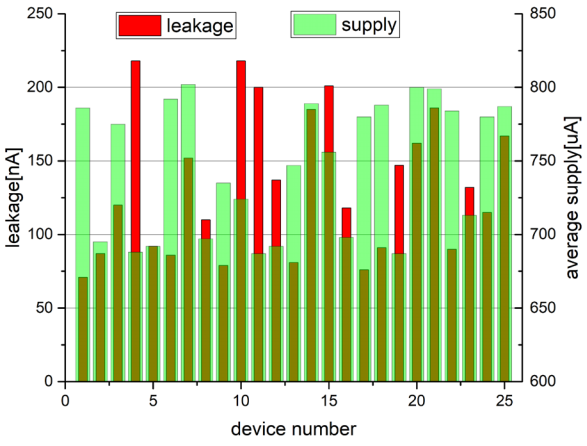


Fig. 7. Leakage current and supply current for 25 devices.

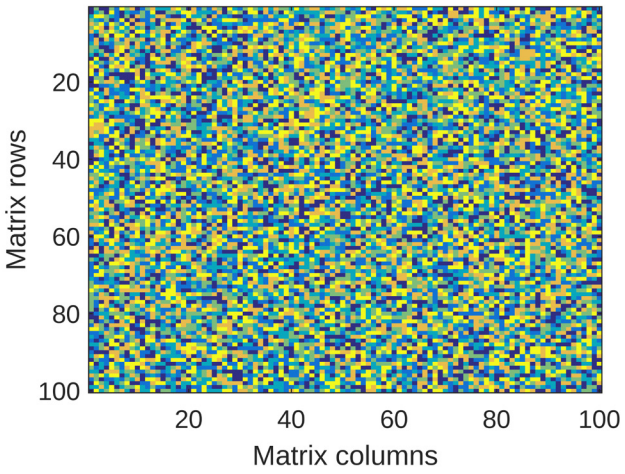


Fig. 8. The schematic representation of 10000 values.

the average supply current during display time ($\approx$ 8s) of 10 dice symbols. Fig. 7 shows the measured values.

The hardware we designed for RNG implementation in 25 devices has a leakage current of 70 nA - 220 nA (variable with device) and draws less than 800 μA at 3 V supply during displaying of ten successive dice symbols in 8 seconds.

As can be seen, there is no proportional relation between the leakage current and the supply current, leakage is given by the characteristics of the active switch (the PMOS transistor) and the parasitical resistance created by the paint used to protect the prototypes which interfere with the gate polarization of T transistor. Using factory solder mask for the PCB may significantly lower the leakage currents. The supply current heavily depends by the LED voltage drop characteristics, R1 to R3 resistors must be chosen as a good compromise between the LED lighting characteristic and the smallest current consumption for battery saves. Since a single LED is driven at a time, dice symbols are generated similar with the frames of a movie. The eyes inertia fixes the image still. The used LEDs were generating strong visible light below 500uA supply. The LEDs connected between two IO pins during HiZ (high impedance) outputs should remain dark; however they often generate a ghost light. This is caused by a parasitic voltage drop which appears between those two HiZ inputs and create a parasitic current flow through the LEDs. The ghosting intensity is also proportional with the period of time which passes until both IO pin directions are set. In between the time until both IOs are in the right state to set ON or OFF the LED, the ghosting effect is very annoying.

At least two methods can be used to decrease that parasitical voltage drop: (i) connecting a resistor in parallel with the LED [29] or (ii) quasi-simultaneously setting the IO pin function as input (HiZ) for both pins involved in LED ghosting. Even it is simplest, the method (i) cannot be used in low power devices because is increasing the overall consumption during LED lighting. Method (ii) was used in our device as described by $procedure_ONE$ in Appendix (Listing A-1). The order in which the LED pins IO direction was set is actually essential for removing LED ghosting. Without that particular order, instead of "one" the user may see in the same time a strong lighting "one" but also a pale lighting "three". Another good solution we found was to increase the contrast between the LED and the backside by using a color mask mounted above the LEDs. The switch BUT (Figs. 1 and 3) is also hidden below the mask by purpose, the user has to guess where it needs to be pushed.

B. Statistical NIST and Diehard performance results

Chi-square and Diehard suite test reveals the feasibility of pseudo-random generation. We have chosen a sequence of 10000 generated numbers which was tested statistically. Before the statistical evaluation of the pseudo-RNG, we have plotted (Fig. 8) the matrix associated to the random number sequence. It can be seen that no repeated visual pattern is present (meaning ill-constructed pseudo-random number generator). Table I shows the mean value of the sequence split into 1000 groups, which follows the normal distribution. The chi square goodness of statistic fit test confirms that the mean values have a normal distribution, at the 5% significance level.

TABLE I
STATISTICAL AND DIEHARD TEST RESULTS

Test	Result	Category	Observations
Chi square test	pass 100%	Statistical test	These tests were applied to groups of 1000 numbers.
Serial test	pass 100%	Statistical test	
Runs test	pass 90%	Diehard test suite	
Binary 32×32 matrix test	pass 100%	Diehard test suite	The binary representation of the numbers is used.

C. Long-term functionality assessment

Of the 25 manufactured units, 23 were distributed as gifts and remained in uncontrolled everyday use, leaving two de-

vices reserved as laboratory reference samples. This intentionally retained subset, although small, permits a controlled longevity assessment that the distributed units cannot provide: the reserved devices were stored and operated under known conditions with their original CR2032 batteries, and were retrieved for re-characterization in March 2026, one year after manufacture. Both devices remained fully functional. Table II summarizes the measured electrical parameters after one year: the battery voltages in ON state remain well above the 1.8 V minimum operating threshold of the PIC12LF1572, the leakage currents and the average supply currents during display are consistent with the initially measured leakage and average supply current.

TABLE II
ELECTRICAL PARAMETERS OF THE TWO RESERVED DEVICES AFTER ONE YEAR OF STORAGE AND OPERATION WITH THE ORIGINAL CR2032 BATTERY

Reserved device	U_{bat} [V] ON	U_{bat} [V] OFF	$I_{leakage}$ [nA]	$I_{avg\text{-}supply}$ [µA]	Obs
	3.06	3.23	100	996	upper left LED has lower light
	2.90	3.08	230	919	perfect LED appearance

D. Comparison with existing approaches

Table III is a comparison of the proposed method against the main alternatives for random number generation on low cost embedded platforms. Dedicated circuit-noise TRNGs [3], [4], [11] achieve high-quality entropy but require custom silicon and, therefore, not usable in our low-cost device, operating in a start-stop regime. Software-only sensor entropy harvesting [16] is compatible with such platforms but requires acquisition times (1.7 s/byte) incompatible with our fast design. The conventional stored-seed approach [30] supports start-stop operation only at the cost of non-volatile memory (NVM) writes at every power cycle, imposing energy consumption in a device designed for years of battery operation. Finally, the closest software-only, memoryless alternative for random number generation is the SRAM power-up state, which has been shown to provide insufficient entropy on PIC-class microcontrollers [31]. To the best of our knowledge, the deliberately sub-TACQ temperature acquisition proposed here is the only reported method combining true-random seeding, no auxiliary hardware, and no non-volatile state persistence on a low-cost 8-bit platform, which is validated on a manufactured series of 25 devices.

IV. CONCLUSIONS

We have designed and tested a low-power electronic device called “mărțișor”, named after the Romanian word for March,

TABLE III
COMPARISON WITH EXISTING RNG APPROACHES FOR EMBEDDED SYSTEMS

Ref.	Entropy source	Operating mode	Implementation	Limitations	Life time
[3], [4], [11]	circuit noise	-	experimental CMOS	custom IC	-
[16]	sensor LSBs	intermittent	1pcs MSP430 MCU	slow 1.7s/byte	-
[30]	stored-seed PRNG	intermittent	unknown, via NVM	number of NVM writes	-
[31]	SRAM power-up state	intermittent	1pcs PIC16F1825	insufficient entropy	-
This work	**sub-TACQ temp. ADC**	intermittent	25pcs PIC12LF1572	none	$>$ 1 year

“martie”. The device was operated and its behavior was observed over a one-year period. It incorporates a low-power random number generator whose output is symbolically visualized as a dice. Both statistical and empirical tests were conducted, demonstrating the randomness of the generated numbers and confirming that the proposed method is suitable for use in various social gaming applications. The low power consumption enables a long lasting life-time (as confirmed by one year of continuous use) without requiring battery replacement. The start–stop pseudo-RNG implementation does not rely on non-volatile memory, such as EEPROM or high-endurance flash. The pseudo-random generator met the general requirements proven by four different statistical tests. Moreover, the manufactured device is low cost, given by the few components in use and the software implementation method. Compared to other known RNG solutions, the proposed method offers one of the simplest possible implementations but also verification processes.

The results positively address all four questions posed at the begin of this research: (i) a low power 8-bit microcontroller is suitable for powerful RNG implementation, (ii) no auxiliary hardware is required for true seed generation, (iii) high efficiency LEDs can be used for RNG output visualization even powered by a coin-cell battery with multi-year lifetime (iv) Chi-square and Diehard test suite provide simple and accurate testing methods. Future work will focus on adapting the approach for other specific microcontroller architectures and application scenarios.

APPENDIX
WORKFLOW AND PSEUDOCODE REPRESENTATION OF THE IMPLEMENTED LOGIC

Listing A-1. Pseudocode representation of the implemented logic

```
include 12lf1572    ; the target microcontroller
pragma target clock 1_000_000    ; set 1MHz clock
pragma target OSC INTOSC_NOCLKOUT    ; internal
    oscillator
pragma target BROWNOUT DISABLED    ; no brownout reset
```

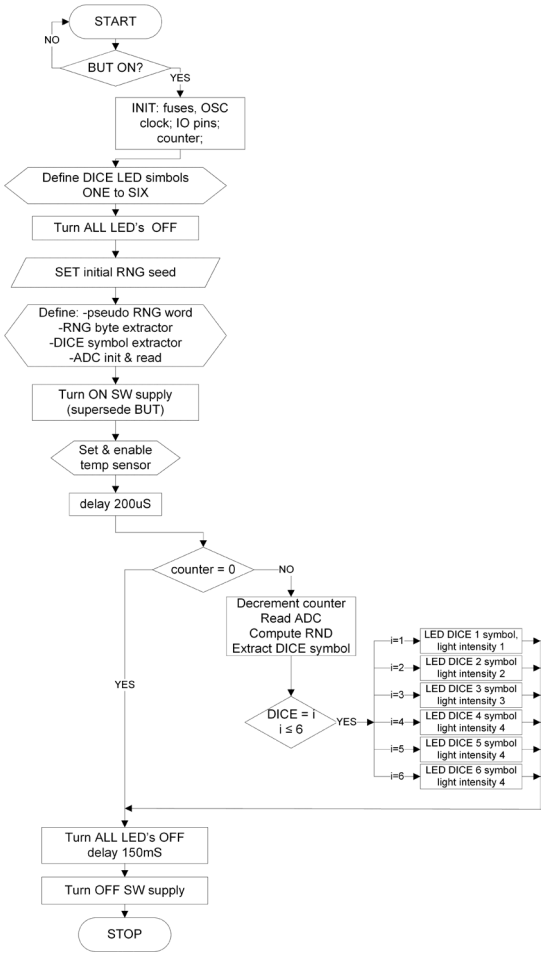


Fig. A-1. The simplified workflow of the implemented logic.

```
pragma target MCLR INTERNAL    ; use internal reset
pragma target CLKOUTEN DISABLED    ; no clock output
pragma target WDT DISABLED    ; no watchdog
OSCCON_SCS = 0    ; select primary oscillator
OSCCON_IRCF = 0b1011    ; 1MHz clock
OSCCON_SPLLEN = DISABLED    ; disable PLL
enable_digital_io()    ; make all pins digital I/O
OPTION_REG_WPUEN = high    ; weak pull-up disabled
ODCONA = 0    ; standard push-pull output
include delay    ; delay library

procedure ALL_OFF is    ; all LEDs OFF
  pin_A0_direction = input
  pin_A2_direction = input
  pin_A1_direction = input
  pin_A4_direction = input
end procedure

ALL_OFF    ; turn all LEDs OFF immediately

procedure ONE (byte in light) is    ; display one dice
    LED
  pin_A0_direction = input
  pin_A2_direction = input     ; avoid ghosting D1,
  pin_A1_direction = input     ; avoid ghosting D5, D6,
      D7
  pin_A4_direction = input     ; avoid ghosting D2, D3
  pin_A1_direction = input     ; keep dark D5, D6, D7
  pin_A4_direction = input     ; keep dark D2, D3
  pin_A0_direction = output    ; prepare IO pins for D4
  pin_A2_direction = output
  pin_A0 = low
  pin_A2 = high          ; D4 LED ON
  delay_10us (light)    ; multiply 10us with light value
  pin_A2 = low
end procedure

procedure TWO&FIVE (byte in light) is    ; similar to ONE
     or SIX
end procedure

procedure SIX (byte in light) is    ; display six dice
    LEDs
  pin_A0&A2_direction = output    ; D1 LED on
  pin_A1&A4_direction = input    ; no ghosting
  pin_A0 = high
  pin_A2 = low
  delay_10us (light)
  pin_A0 = low

  pin_A0&A1_direction = input    ; D2 LED on
  pin_A2&A4_direction = output
  pin_A4 = high
  pin_A2 = low
  delay_10us (light)
  pin_A4 = low

  pin_A0&A1_direction = input    ; D3 LED on
  pin_A2&A4_direction = output
  pin_A4 = low
  pin_A2 = high
  delay_10us (light)
  pin_A2 = low

  pin_A0&A4_direction = input    ; D7 LED on
  pin_A2&A1_direction = output
  pin_A1 = low
  pin_A2 = high
  delay_10us (light)
  pin_A2 = low

  pin_A0&A1_direction = output    ; D6 LED on
  pin_A2&A4_direction = input
  pin_A0 = low
  pin_A1 = high
  delay_10us (light)
  pin_A1 = low

  pin_A0&A1_direction = output    ; D5 LED on
  pin_A2&A4_direction = input
  pin_A0 = high
  pin_A1 = low
  delay_10us (light)
  pin_A0 = low
end procedure

var volatile dword _rndx = 2282008    ; initial dword
    seed
var volatile dword _rndy = 362436069

function random_word() return word is    ; pseudo random
    generator
  _rndx = 69069 * _rndx + 123
  _rndy = _rndy ^ (_rndy << 13)
  _rndy = _rndy ^ (_rndy >> 17)
  _rndy = _rndy ^ (_rndy << 5)
  return word(_rndx + _rndy)
end function

var word used_nr
function random_byte() return byte is    ; extract random
     byte
  used_nr = random_word
  var byte N[2] at used_nr
  var byte used_nr_hi = N[1]
  var byte used_nr_lo = N[0]
  return byte(used_nr_lo)
end function

function dice() return byte is    ; dice number 1-6
  var byte nr = 0
  while (nr < 1) | (nr > 6) loop
    nr = (random_byte & 0x07)
  end loop
  return nr
end function

procedure one_ch_ad_read (byte in channel, bit*2 in ref,
    bit in justify) is    ; temperature read
  pragma inline
  ADCON1_ADCS = 0b_000    ; fosc/2 at 1MHz
  ADCON1_ADPREF = ref    ; VDD reference
  ADCON1_ADFM = justify    ; result justification
```

```
  if channel == 4 then
    ANSELA = 0
    ADCON0_CHS = 0b_11101    ; internal temp sensor
  end if
  ADCON0_ADON = true    ; turn on ADC
  _usec_delay(2)    ; sample delay set below TACQ
  ADCON0_GO = true    ; start ADC
  while ADCON0_GO loop end loop    ; wait for finish
  _usec_delay(22)    ; delay minimum 11 TAD
  ADCON0_ADON = false    ; turn off ADC
end procedure

    ; ----- Main Program -----
var word wADRESH
var word adc_value
pin_A5_direction = output    ; T transistor ON
var bit switch_on is pin_A5
switch_on = low

VRCON_TSEN = high    ; enable temp
FVRCON_TSRNG = low    ; VDD >= 1.8V
delay_10us (20)    ; wait 200us

for 10 loop
  one_ch_ad_read(4, 0b00, 0)
  wADRESH = ADRESH
  adc_value = 256 * wADRESH + ADRESL
  _rndx = _rndx + adc_value
  _rndy = _rndy - _rndx
  read_value = dice

  if read_value == 1 then
    for 50 loop ONE(100) end loop
  elsif read_value == 2 then
    for 35 loop TWO(100) end loop
  elsif read_value == 3 then
    for 30 loop THREE(100) end loop
  elsif read_value == 4 then
    for 20 loop FOUR(100) end loop
  elsif read_value == 5 then
    for 20 loop FIVE(100) end loop
  elsif read_value == 6 then
    for 20 loop SIX(100) end loop
  end if

  ALL_OFF
  delay_1mS(150)
end loop

OSCCON_IRCF = 0b0000    ; set 31kHz, low power mode until
     complete turn off
FVRCON_TSEN = low    ; turn off temp sensor
enable_digital_io    ; disable analog input
switch_on = high    ; turn off T transistor
```

REFERENCES

[1] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong, "Monte Carlo simulations: Hidden errors from "good" random number generators," *Phys. Rev. Lett.*, vol. 69, no. 23, pp. 3382–3384, Dec 1992. DOI: 10.1103/PhysRevLett.69.3382

[2] B. Lampert, R. S. Wahby, S. Leonard, and P. Levis, "Robust, low-cost, auditable random number generation for embedded system security," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, ser. SenSys '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 16–27. DOI: 10.1145/2994551.2994568

[3] T. Figliolia, P. Julian, G. Tognetti, and A. G. Andreou, "A true random number generator using RTN noise and a sigma delta converter," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 17–20. DOI: 10.1109/ISCAS.2016.7527159

[4] E. Kim, M. Lee, and J.-J. Kim, "8Mb/s 28Mb/mj robust true-random-number generator in 65nm CMOS based on differential ring oscillator with feedback resistors," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 144–145. DOI: 10.1109/ISSCC.2017.7870302

[5] M. Sönmez Turan, E. Barker, J. Kelsey, K. McKay, M. Baish, and M. Boyle, "Recommendation for the entropy sources used for random bit generation," National Institute of Standards and Technology, Tech. Rep., 2018. DOI: 10.6028/NIST.SP.800-90B

[6] P. L'Ecuyer, "Uniform random number generation," *Ann Oper Res*, vol. 53, pp. 77–120, 1994. DOI: 10.1007/BF02136827

[7] W. Schindler and W. Killmann, "Evaluation criteria for true (physical) random number generators used in cryptographic applications," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, B. S. Kaliski, Ç. K. Koç, and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 431–449. DOI: 10.1007/3-540-36400-5_31

[8] B. Sunar, W. J. Martin, and D. R. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Transactions on Computers*, vol. 56, no. 1, pp. 109–119, 2007. DOI: 10.1109/TC.2007.250627

[9] S. Buchovecká, R. Lórencz, F. Kodýtek, and J. Bucek, "True random number generator based on ropuf circuit," in *2016 Euromicro Conference on Digital System Design (DSD)*, 2016, pp. 519–523. DOI: 10.1109/DSD.2016.36

[10] A. Marandi, N. C. Leindecker, K. L. Vodopyanov, and R. L. Byer, "All-optical quantum random bit generation from intrinsically binary phase of parametric oscillators," *Opt. Express*, vol. 20, no. 17, pp. 19 322–19 330, Aug 2012. DOI: 10.1364/OE.20.019322

[11] C. Wannaboon, M. Tachibana, and W. San-Um, "A 0.18-um CMOS high-data-rate true random bit generator through $\Delta\Sigma$ modulation of chaotic jerk circuit signals," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 28, no. 6, p. 063126, 06 2018. DOI: 10.1063/1.5022838

[12] K. Boonyaphon, S. Takayama, and S.-J. Kim, "Microfluidic random number generator driven by water-head pressure and human finger push," *Sensors and Actuators A: Physical*, vol. 302, p. 111802, 2020. DOI: 10.1016/j.sna.2019.111802

[13] M. Barangi, J. S. Chang, and P. Mazumder, "Straintronics-based true random number generator for high-speed and energy-limited applications," *IEEE Transactions on Magnetics*, vol. 52, no. 1, pp. 1–9, 2016. DOI: 10.1109/TMAG.2015.2478398

[14] G. Marsaglia, "Xorshift rngs," *Journal of Statistical Software*, vol. 8, no. 14, pp. 1–6, 2003. DOI: 10.18637/jss.v008.i14

[15] T. E. Hull and A. R. Dobell, "Random number generators," *SIAM Review*, vol. 4, no. 3, pp. 230–254, 1962. DOI: 10.1137/1004061

[16] M. P. Pawlowski, A. J. Jara, and M. J. Ogorzalek, "Harvesting entropy from on-board sensors of constrained devices for hardening security of IoT communication mechanisms," *Research Briefs on Information and Communication Technology Evolution*, vol. 1, pp. 21–34, Jan. 2015. DOI: 10.56801/rebicte.v1i.4

[17] *FT232RL datasheet*, FTDI, last visit: 29.05.2025. [Online]. Available: https://ftdichip.com/products/ft232rl/

[18] *Realterm software*, FTDI, last visit: 29.05.2025. [Online]. Available: https://sourceforge.net/projects/realterm/

[19] L. E. Bassham III, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, *et al.*, "Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications," 2010. DOI: 10.6028/NIST.SP.800-22r1a

[20] J. Zaman and R. Ghosh, "Review on fifteen statistical tests proposed by NIST," *Journal of Theoretical Physics and Cryptography*, vol. 1, pp. 18–31, 2012.

[21] R. G. Brown, D. Eddelbuettel, and D. Bauer, "Dieharder: A random number test suite," Duke University Physics Department, Durham, NC, 2018. [Online]. Available: https://webhome.phy.duke.edu/~rgb/General/dieharder.php

[22] D. Knuth, *The art of computer programming*. London: Second Edition, Vol 2., Addison-Wesley Pub, 1981.

[23] G. Marsaglia, "The marsaglia random number CDROM including the diehard battery of tests of randomness," 2008. [Online]. Available: http://www.stat.fsu.edu/pub/diehard/

[24] *PIC12LF1572 datasheet*, Microchip, dS40001723D.

[25] K. Mowery, M. Wei, D. Kohlbrenner, H. Shacham, and S. Swanson, "Welcome to the entropics: Boot-time entropy in embedded devices," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 589–603. DOI: 10.1109/SP.2013.46

[26] H. Krawczyk, "Cryptographic extraction and key derivation: The HKDF scheme," in *Advances in Cryptology – CRYPTO 2010*, T. Rabin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 631–648. DOI: 10.1007/978-3-642-14623-7_34

[27] L. Trevisan, "Extractors and pseudorandom generators," *J. ACM*, vol. 48, no. 4, pp. 860–879, July 2001. DOI: 10.1145/502090.502099

[28] I. Tataru, "Equivalence vs. adaptation: the dilemma of translating the culturebound word „Mărțișor" into English," *Studia Universitatis, Universitatea de Stat din Moldova*, no. 10, pp. 248–257, 2024. DOI: 10.59295/sum10(200)2024_32

[29] "Increasing efficiency in embedded digital IO lines," *Electronic Design News*, pp. 54–56, 2011, last visit 14.07.2025. [Online]. Available: https://www.edn.com/increase-efficiency-in-embedded-digital-i-o-lines/

[30] J. Kelsey, B. Schneier, and N. Ferguson, "Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator," in *Selected Areas in Cryptography*, H. Heys and C. Adams, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 13–33. DOI: 10.1007/3-540-46513-8_2

[31] A. Van Herrewege, V. van der Leest, A. Schaller, S. Katzenbeisser, and I. Verbauwhede, "Secure prng seeding on commercial off-the-shelf microcontrollers," in *Proceedings of the 3rd International Workshop on Trustworthy Embedded Devices*, ser. TrustED '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 55–64. DOI: 10.1145/2517300.2517306