Double Error Cellular Automata-Based Error Correction with Skip-mode Compact Syndrome Coding for Resilient PUF Design

Anthony Mattar El Raachini, Hussein Alawieh, Adam Issa, Zainab Swaidan, Rouwaida Kanj, Ali Chehab and Mazen A. R. Saghir
American University of Beirut, Beirut, Lebanon, 1107 2020

ABSTRACT

Physical Unclonable Functions (PUFs) present an attractive security primitive due to their volatile key generation capability. Subject to environmental conditions, the PUF response, however, is prone to errors which may undermine the reliability of the system when left unaddressed. An error-correction scheme is typically used alongside the PUF circuit when used for cryptographic applications. In this paper, we propose the use of Cellular-Automata Error-Correcting Codes (CAECC) due to their simplicity and regularity. An efficient implementation of (15, 7, 5) CA-ECC encoder/decoder targeting a Xilinx Zynq-7000 device is demonstrated, and the design is validated on design compiler targeting 40nm TSMC technology. We also propose a skip-mode compact syndrome coding scheme for relaxed per-block BER. CAECC is tested in conjunction with the skip-mode scheme, and the approach is verified on ring oscillator PUF data. The skip-mode scheme is found to reduce the ring oscillator overhead up to 20% and enhance the entropy up to 23% compared to no-skip schemes.

1. INTRODUCTION

From Tesla's smart vehicles to Dell and Intel's telemedicine solutions [1, 2], the IoT industrial revolution has already started. With a wide span of applications, Gartner and IDC estimate around six to nine billion connected devices are in use worldwide [3]; they expect this number to explode to 20 billion in 2020. A security breach on any of these smart devices can infiltrate the network and may affect millions of users around the world. In 2016, a distributed denial of service (DDoS) attack targeted the KrebsOnSecurity website. The malware powering the IoT botnet responsible for the attack was made publically available, raising the specter of additional attacks [4]. It is, therefore, very critical for IoT devices to carry out operations and communications in a highly secure but efficient fashion. Nearly all security models rely on a stored secret key which is vulnerable to invasive attacks. A physical unclonable function (PUF) is a circuit primitive that extracts secrets from physical properties of integrated circuits. PUFs generate volatile secret keys available in digital form when the chip is running making it very difficult to discover compared to nonvolatile keys [5].

IC PUFs employ a variety of underlying circuit structures such as SRAMs, Arbiters and Ring Oscillators (RO) [5, 6, 7]. RO PUFs are the most popular due to their ease of implementation. In its most basic implementation, a response bit is generated by comparing the delays of a pair of ROs. However, RO PUFs are vulnerable to environmental conditions which can affect the frequency of the ROs, and hence the reproducibility of the response. The authors in [5] propose a 1-out-of-8 masking scheme that picks the slowest and fastest ring oscillators in each group of 8 oscillators to determine the bit response. This minimizes sensitivities to small frequency variations due to enviornment effects and hence enables a reliable output bit response. An entropy-maximization technique is proposed in [8], where ROs with a frequency difference above a certain stability threshold are grouped by the longest increasing subsequence algorithm (LISA). The permutation of ROs in the same group is then mapped into bits using compact syndrome coding (CSC). This allows the realization of a near maximal entropy corresponding to a fixed number of ROs. An alternative non-compact mapping into bits based on Kendall Syndrome Coding (KSC) was devised in [9] to correlate the error weight distribution of the resulting bit string to the underlying error probability distribution of the rank flips within a certain permutation. This would relax the error correction requirements at the expense of extra computational effort in terms of KSC implementation and the need for an entropy packing stage that remaps permutations compactly into bits after error correction. The choice of the RO frequency comparison technique and its stability threshold together with the choice of the error correction code guarantees that the PUF produces the same response consistently. The role of ECCs is stressed out in guaranteeing the PUF response reproducibility. In [10], Chowdhury et al. proposed a cellular automata (CA) based error correction code. The circuit structure of the CAECC is amenable to VLSI implementations due to its simplicity, regularity, and cascadability. This is compared to conventional ECCs such as BCH codes whose irregularity and decoding structure complexity increase with the number of information bits [10]. The authors demonstrate a single-error-correction double-error detection Null Boundary Cellular Automata-based (NBCA) ECC which reduces decoding complexity compared to Hsiao code. Cho et al. [11] propose a double-error correcting code with reduced number of check-bits based on Periodic Boundary CA (PBCA).

In this paper, we evaluate for the first time the CA-based ECC code in the context of a newly proposed skip-mode CSC scheme on RO PUF data obtained from [12]. The skip-mode scheme is proposed to relax the per-Block bit error rate in the event of most probable adjacent RO rank flips. We also evaluate the design complexity of CAECC by simulating a (15, 7, 5) CAECC encoder/decoder blocks using the Xilinx Vivado tool-suite. The paper is organized as follows. Section 2 provides a background review on ECC for PUF, and cellular automata basics. Section 3 discusses the CA encoding and decoding algorithms. The proposed skip-mode CSC scheme for group-based ROPUFs is presented in Section 4. An efficient design of a double-error CAECC encoder and decoder targeting xc7z020clg484-1 FPGA used in the Zynq-7000 ZedBoard is presented in Section 5. This is followed by validation of the proposed scheme on RO PUF data. Conclusions follow in Section 6.

2. BACKGROUND REVIEW

2.1 Error Correction for Physical Unclonable Functions

Temperature, supply voltage and thermal noise may force a PUF cell to produce an erroneous output. Conventional methods for error correction in PUFs use XOR masking, also known as the code-offset construction, which performs a bitwise XOR of the parity information with the PUF output (Fig. 1) [13]. The code-offset construction steps for an n-bit PUF response vector **r** are summarized as follows.

(1) The n-bit response **r** is divided into a k-bit information vector **m** and an (n-k)-bit vector **c**.

(2) Upon initialization, the information vector **m** is fed into an encoder, which generates the (n-k) check-bit vector **cb**.

(3) Vectors **c** and **cb** are XORed to produce the helper bits, which are stored in nonvolatile memory as vector **h**. Even with helper data present, the adversary still needs to guess k information bits to find the correct PUF response [13].

(4) Upon deployment, the PUF response is r' with unknown error vector **E**. Vector **c'** is XORed with **h**.

(5) The result, along with vector **m'** (information vector from the second reading), is fed into the decoder, which produces the correct k-bit **m** used to build the cryptographic key; large keys can be derived from multiple smaller k-bit blocks.

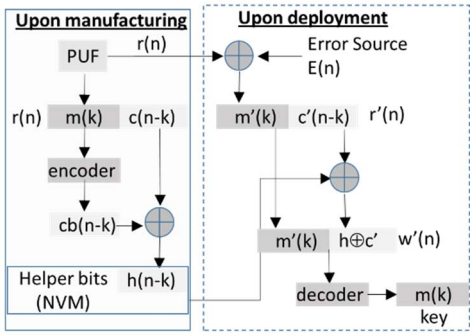


Figure 1. Typical Error correction flow for PUF response.

2.2 Cellular Automata

Cellular Automata are mathematical idealizations of physical systems in which space and time are discrete [11, 14].

Cellular Automata cell: Each cell consists of a D flip-flop and a next-state function, which depends on the cell current state and that of its two neighbors according to specific rules.

$$q_i^{t+1} = f(q_{i-1}^t, q_i^t, q_{i+1}^t)$$

$\boldsymbol{q_i^t}$: The state of the ith cell at instant t

$\boldsymbol{f}$: Logic function generating next state

k-cell CA: the different next-state functions can be all represented by the $k \times k$ characteristic matrix T_k. The next-state vector of the entire CA can be obtained by multiplying the current-state vector by the characteristic matrix. To obtain the state of the CA after 'p' cycles, it suffices to multiply the current state vector by the characteristic matrix 'p' times.

$$f_{t+1} = T \times f_t$$
$$f_{t+p} = T^p \times f_t$$

$\boldsymbol{f_t}$: The current state vector of the CA at instant t

$\boldsymbol{T}$: k × k characteristic matrix of the CA

CA ECC: The CA can be used to generate a set of codewords with any specified hamming distance. It is proven that a k-cell CA with a characteristic matrix T generates a *t*-distance code if for some integer p, T^p satisfies $\forall i$, $0 < i < t$, the bitwise sum of any *i* columns of T^p contains at least $(t - i)$ 1's [10].

3. CELLULAR AUTOMATA ECC

We follow the method proposed by [11] which improves on the original code of Chowdhury et al., [10] by reducing the number of check bits needed for the same error-correction capability for (n, k, 5) codes. For purposes of our implementation, we focus on (15, 7, 5) 2-bit error correction codes, and the PUF response is divided into several 15-bit blocks each with 7 information bits.

3.1 Encoder Design

Fig. 2 presents the high-level scheme for check-bit generation. The k information bits are fed into the k-cell CA, which is run for 3 cycles. The resulting vector state is used as the input of a logic circuit g(Q) which generates the (n-k) check bits ***cb*** [10]. The entire encoding process is represented by an (n-k)×k matrix T. Multiplying the matrix by a k-bit information vector yields the corresponding (n-k) check bits [11]. The algorithm used to generate T for an (n, k, 5) CAECC is described in Fig. 3. Fig. 4 presents, the k-cell CA and the corresponding FSM for the (15, 7, 5) encoder.

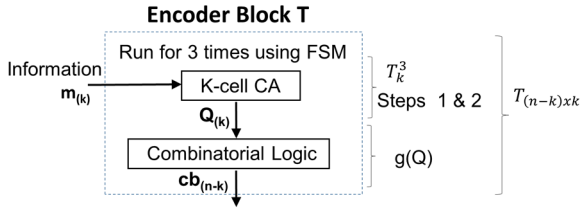


Figure 2. Encoder Block.

Algorithm 1: CAECC Encoder Block T

1. Construct the kxk characteristic matrix T_k to model the k-cell CA according to the rules <102,102,…,102,150>
2. Calculate T_k^3
3. Change the (1, k)-element of T_k^3 with 1
4. Add the matrix $\begin{pmatrix} 1 & 0 & 1 & 0 & \cdots & 1 & 0 \\ 0 & 1 & 0 & 1 & \cdots & 1 & 1 \end{pmatrix}$ to the new matrix. The new matrix is labeled T'.
5. Add the rows which don't have 1's at the same positions to obtain the matrix with the minimum number of rows which satisfies the following two conditions:
   a. Every column contains at least four 1's.
   b. The sum of any number of columns is not a zero vector.

Figure 3. Algorithm for (n, k, 5) encoder block matrix [11].

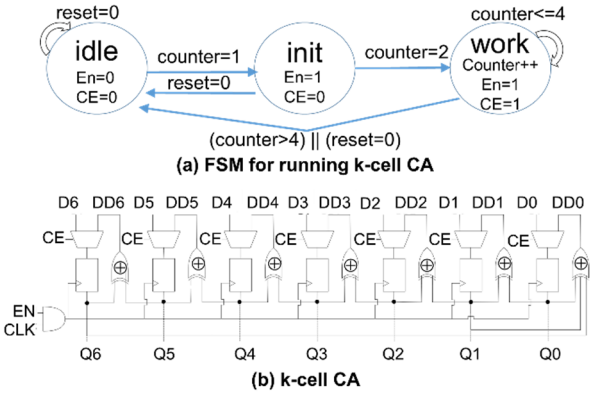


Figure 4. (a) FSM to run the k-cell CA 3 times. **(b)** k-cell CA**.**

In the 'init' state, the CE signal latches the information vector. In the 'work' state, the CE signal latches the next state as computed per the CA rules. Reset signal sets the outputs of the k-cell to zero. Enable signal is used in conjunction with the clock to activate the latches during 'init' and 'work' states. Note that block g(Q) is derived from T_k^3 and T, and together with the k-cell CA, they are of comparable complexity and perform the same function as T.

3.2 Decoder

The decoding scheme is based on an inverse transformation that leverages the properties of the CA. Given a new response **r'** with unknown error **E**: to find E, we first calculate a syndrome vector **S** using the parity matrix H formed by concatenating T and an (n-k)x(n-k) identity matrix I_{n-k} as illustrated in eq. (1). The error vector can be derived from S by relying on an augmented matrix version of H, called T_{aug}, the details of which are well explained in [11]. Thus, E can be derived uniquely using T_{aug}^{-1} according to eq. (2), where S_{aug} is a kx1 vector whose value depends on S. To determine the mapping that generates S_{aug} from S, we construct a table based on all permissible error vectors [10], E_p, and generate (S, S_{aug}) pairs according to eq. (4). This table is mapped to a combinatorial logic block whose input is S and output is S_{aug} and whose size increases as the number of permissible error vectors increase. We refer to this block as *S_{aug}Map*. For purposes of the (15, 7, 5) code, the block maps the syndromes resulting from all permissible (0, 1 or 2) errors for the double-error correcting code to S_{aug}. The full decoding scheme is summarized in Fig. 5 below.

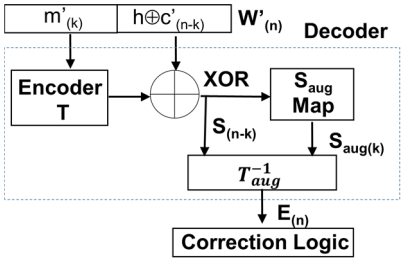


Figure 5. The decoding scheme. 'Map S_{aug}' block is derived from all permissible error vectors.

$$r'_n = [\, m'_{0,k-1} \mid c'_{0,n-k}] \quad (1)$$
$$m'_{0,..,k-1} = [\, m_{0,..,k-1} \oplus E_{0,..,k-1} \,]$$
$$c'_{0,..,n-k} = [\, c_{0,...,n-k} \oplus E_{k,...,n} \,]$$
$$w'_{n\times 1} = [m'_{0,..,k-1} \,|\, h \oplus c'_{0,...,n-k}]$$
$$H_{(n-k)\times n} = [T_{(n-k)\times k} | I_{n-k}]$$
$$S_{n-k} = H_{(n-k)\times n} * w'_{nx1}$$
$$\mathrm{E} = \mathrm{T}_{\mathrm{aug}}^{-1}\left(\mathrm{S}\middle|\mathrm{S}_{\mathrm{aug}}\right)^{\mathrm{T}} \quad (2)$$
$$\mathrm{T}_{\mathrm{aug}}^{-1} = \begin{bmatrix} 0_{k\times n-k} & I_k \\ I_{n-k} & T_{n-k\times k} \end{bmatrix} \quad (3)$$
$$T_{aug}\mathrm{E_p} = \left(\mathrm{S}\middle|\mathrm{S}_{\mathrm{aug}}\right)^{\mathrm{T}} \quad (4)$$

4. CSC WITH SKIP-MODE

4.1 Group-based Response Vector Generation

To generate the PUF response, the group-based technique compares multiple ROs at once in an attempt to enhance the entropy compared to the pairwise-comparison approach. For *n* ROs in a group, there are *n!* ways to order these ROs by speed. For example, suppose a group has 4 ROs {A, B, C, D} then there are 4! = 24 possible permutations of the ROs in this group. These permutations can be represented in $\lceil \log(n!) \rceil$ bits. Mapping the permutation to bits is performed using CSC syndrome coding. This technique, has two challenges [15].

1) The RO groups are subject to systematic errors on the chip: this is often resolved via regression techniques that identify systematic components and extract randomness.

2) High number of bit errors corresponding to most probable adjacent–RO flips is recorded when utilizing CSC. For example: rank permutation 'ADCB' is encoded by '00101'. It is possible that upon regeneration a flip occurs between RO_A and RO_D ranks, ('DACB') which is encoded by '10011'. KSC technique was proposed in [15] to better correlate RO flips to bit errors, such that more probable adjacent RO flips would result in less bit errors. This requires, however, an additional KSC encoding step, ECC on a much longer sequence compared to CSC, and inverse KSC prior to converting the bits to CSC representation. In what follows, we propose to enhance the tolerance to the number of bit errors by spreading the group bits across different response blocks as opposed to changing the encoding scheme.

4.2 Preliminaries

We elaborate on a 256-bit key example to highlight the concept of response blocks. We also discuss numerical limitations of the CSC encoding scheme to explicate the underlying assumptions in the following sections.

256-bit key Example and (15, 7, 5) CAECC: We use the PUF response to generate a 256-bit key (s_k=256), and we rely on (n=15, k=7, t=5) CAECC to recover from possible RO rank flips. Accordingly, the length of the corresponding PUF response vector should be $s_r = \left\lceil \frac{s_k}{k} \right\rceil * n =$ 555 bits to accommodate for the information bits and the redundancy bits. For purposes of error correction, the response is divided into 15-bit blocks whose number $nBlocks = \left\lceil \frac{s_k}{k} \right\rceil = 37$.

CSC code limitations and a maximum number of 18-ROs per group: Figure 6 presents the pseudo-code for the CSC encoding scheme. It invokes large factorial computations ($\sum_{i=1}^{|g|-1} i!$), where |g| is the number of ROs in a group, (lines 2 and 7). Due to the limited number of significant digits (15) in the Mantissa part of the double precision integer representation, large group sizes would result in factorial numbers that require more than 15 significant digits and thus cannot be represented accurately. Hence, a maximum of 18-RO groups can be formed for accurate encoding.

Given: a group g containing ordered ROs [RO_1 …. $RO_{|g|}$]
CSC Encoding: outputs a CSC encoded integer c_g in $\log_2(\lceil |g|! \rceil)$ bits

```
1: c_g = 0
2: for i = |g| to 2 do
3:   { inv = 0 // number of inversions
4:     for j=1 to (i-1) do
5:       { if RO_i < RO_j then
6:           inv = inv + 1}
7:     c_g = (c_g + inv) *(i - 1) }
8: return c_g
```

Figure 6. CSC encoding scheme[15].

4.3 Skip-mode CSC

In CSC, a highly probable adjacent RO rank flip can result in multiple bit flips within the same group. Figure 7 illustrates a sketch of an example CSC response vector. Assume that the ECC can correct a maximum of 1 bit in a 4-bit block. An RO rank Flip in Group 1 leads to error in bits x2 and x3. This results in the inability of the ECC to correct block1 in the Base CSC response. We propose skip-mode CSC to enhance error correction capability by relaxing the number of per-Block bit fails for small codewords. This is achieved by distributing the group bits across the different ECC blocks and hence reduce the per-block BER. Figure 8 presents a possible way to distribute the bits of the same group across the different blocks. In a *more conservative skip-mode* CSC, one can restrict the maximum number of ROs per group such that the number of bits per group is equal or less than the number of ECC blocks of the target response. In this scenario, only one group bit appears in a block. This comes at the expense of entropy loss. Thus, for the 256-bit key example, the conservative skip-mode restricts the group size to |g|=14 ROs: ($\lceil \log_2(|14|!) \rceil = nBlocks = 37$).

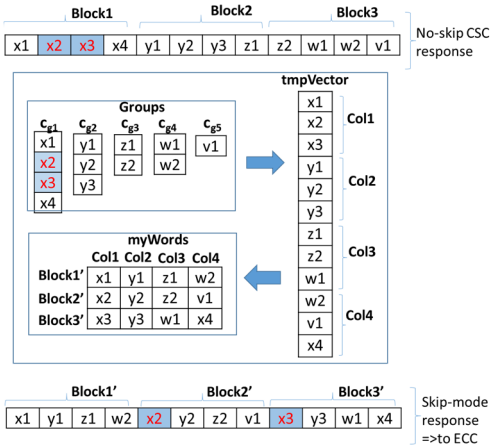


Figure 7. Skip mode relaxes number of error bits within ECC block for cases of most probable adjacent RO flips.

To understand the effect of skip-mode on reducing per-Block BER, we construct 1000 replications of 555-bit responses and record the maximum number of bit errors per 15-bit blocks based on the following three scenarios: ***(1)*** The regular no-skip scenario, ***(2)*** skip-mode scenario (18-RO groups), and ***(3)*** conservative skip-mode (14-RO groups). Fig. 9 presents the results for 1, 2, 3, and 4-adjacent RO rank flips. We note that the maximum number of error bits per 15-bit blocks for a given response reaches 12 for the no-skip CSC scenario. It is limited to 5 for the skip-mode, and 4 for 14-RO groups in the conservative skip-mode. Most importantly, for the skip-modes, all 1-adjacent RO flips and most of 2-adjacent RO rank flips are limited to a maximum of 2-errors per 15-bit block and hence can be resolved by 2-bit error correction codes. *It is worth noting that in the presence of environmental variations and as the grouping threshold increases, the group sizes naturally decrease, so the regular skip-mode would act conservatively naturally.*

Skip-mode CSC	
Given:	$\Omega=\{c_{g1}, \ldots, c_{g\|\Omega\|}\}$ // group bits as derived from CSC encoding
Initialize:	$ptr=\{1, 1, ..\}_{\|\Omega\|}$,cnt=1, $s_g=\{\|c_{g1}\|, \ldots, \|c_{g\|\Omega\|}\|\}$
Build tmpVector	**while** ($\|\Omega\|$)> 0) **for** each Group i { step=min(nBlocks, s_g(i)) **if** (step) {tmpVector(cnt:cnt+step-1)<-c_{gi}(ptr(i):ptr(i)+step-1) ptr(i)<- ptr(i)+step cnt=cnt+step s_g(i)=s_g(i)-step **if** (s_g(i)==0) { ptr(i)<- null $\Omega=\Omega$ - c_{gi} }}}
Map Blocks	cnt=1 **for each** Col j ∈ myWords { Col j <- tmpVector (cnt: cnt+nBlocks-1) cnt<-cnt +nBlocks }

Figure 8. Skip mode pseudo-code.

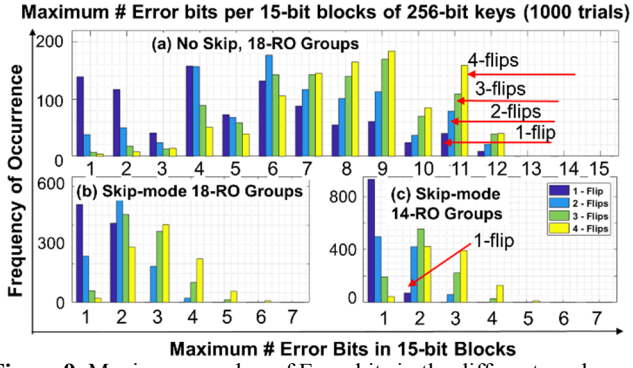


Figure 9. Maximum number of Error bits in the different modes.

5. RESULTS AND ANALYSIS

In this section, we evaluate the efficiency of the CAECC with skip-mode CSC on real RO PUF data. First, we present the CAECC encoder/decoder design.

5.1 (15, 7, 5) CAECC Encoder/Decoder Design

The (15, 7, 5) double error correcting CAECC encoder was designed using the Xilinx Vivado tool-suite and targeting the xc7z020clg484-1 FPGA device. Figure 10 presents the schematic of the encoder K-cell CA and the decoder S_{aug}Map blocks respectively. Figure 11 presents the behavioral simulation plot when vector w'≡[m, cb]⊕E is fed to the decoder. For this example, w'=[010100111010101] corresponding to [m, cb]=[010110111011101]. The output of the decoder correctly identifies the corresponding error vector E=[000010000001000]. The device utilization summary of the different blocks is presented in Table I. Based on the post implementation timing analysis and partitioning the path into 4 key blocks, we identify a critical path delay of 3.7ns. The design consumes 4 clock cycles for the k-cell CA 'init' and 'work' cycles, and an addition 3 cycles for the remaining blocks resulting in a total latency of 26 ns using a 270 MHz clock. It is worth noting that replacing the k-cell CA and combinatorial logic of the encoder with the T block consumes around 1.8ns and eliminates the need for 3 work cycles, thereby reducing the latency to 14.8ns.

Finally, we validated the design using Synopsys Design Compiler [16] targeting TSMC 40nm tcbn40lpbwpbc

technology [17]. Table II. presents the cell utilization and timing summary of the decoder block (including the encoder block). Fig. 12 presents the synthesized S_{aug}Map block. The critical path represents the delay from the k-cell CA output to Eout of the full decoder block.

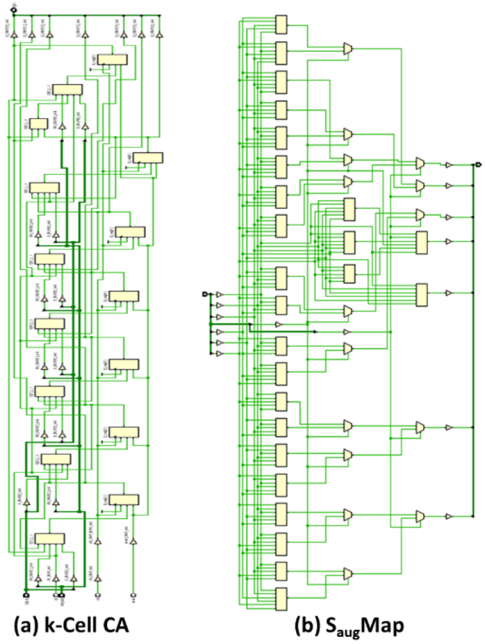


Figure 10. (a) Encoder k-cell CA and **(b)** Decoder S_{aug}Map blocks.

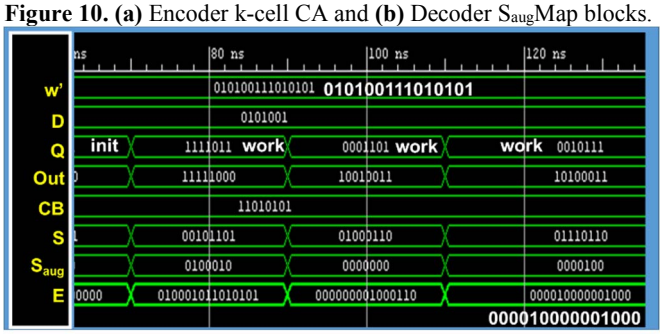


Figure 11. Behavioral Simulation for Decoder Output.

Table I. Summary of the Device Utilization and post implementation timing analysis of the different blocks (Xilinx).

FPGA	component	#LUTs	#FF	Delay (ns)	Power (mW)
Encoder	k-cell CA	7	7	1.597	0.05
	g(Q)	3	-	1.570	16
Decoder	$Taug^{-1}$	8	-	1.874	47
	XOR+S_{aug}Map	25	-	3.689	84

Table II. Overall Design Synthesis on 40nm TSMC technology.

ASIC Critical Path delay	0.71ns
Levels of Logic	12
Leaf cell count	202 (151 combinatorial)
Design Area	360 μm^2

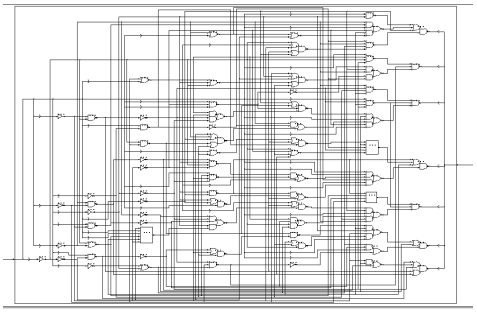
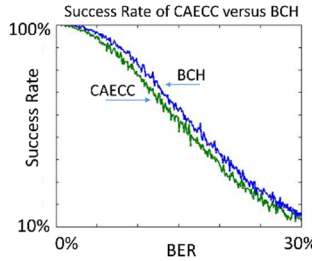


Figure 12. S_{aug}Map block synthesis using Design Compiler.

5.2 Cellular Automata Error Correction Capability

We compare the performance of CA(15, 7, 5) [13] to BCH(15, 7, 5). We subject 1000 information messages to the same increasing bit-error rates. We record the number of information messages that are accurately recovered for each of the two ECCs to measure their success rates. At low BERs (<2%), both CAECC and BCH have a success rates of nearly 100%; i.e., all 1000 information messages were recovered correctly. As the BER increases, the success rates of both ECCs start to drop, with BCH having a very slight edge over CAECC (see Fig. 13). We attribute this to few non-unique S|Saug pairs obtained when deriving the (S|Saug) pairs according to T_{aug} derived in [13].



	Success Rate (%) (15, 7, 5)	
BER	BCH	CAECC
1%	100	99.8
5%	97.1	94.9
10%	82.2	77.2
15%	61.5	58.5

Figure 13. The decoding scheme. 'Map S_{aug}' block is derived from all permissible error vectors.

5.3 Virginia-Tech Data RO-PUF with Skip-mode CSC

Grouping analysis was carried for frequency data for 512 ROs arranged in an array of (32, 16) ROs from the Virginia-Tech PUF dataset chip D059546 [12]. The (1.2V, 25ºC) operating condition was used to build a 2nd order linear regression model for randomness extraction (Fig. 14). The model was then applied to all other test data of variable temperature and voltage values to test the efficacy of the skip-mode approach versus no-skip both in the presence of CAECC. Figures 15 and 16 present the number of Error bits post ECC for the no-skip and skip modes. It is clear that for the various operating conditions of voltage and temperature, the skip-mode converges faster to lower bit errors in the response. Note that we do not report the conservative skip-mode since the groups' sizes were small enough with frequency thresholding.

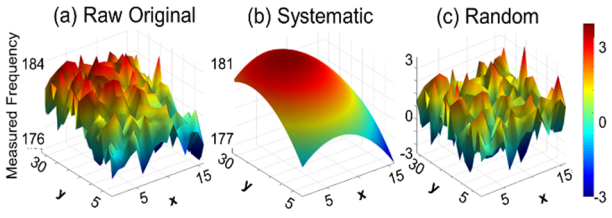


Figure 14 Extracting Randomness with 2nd order model.

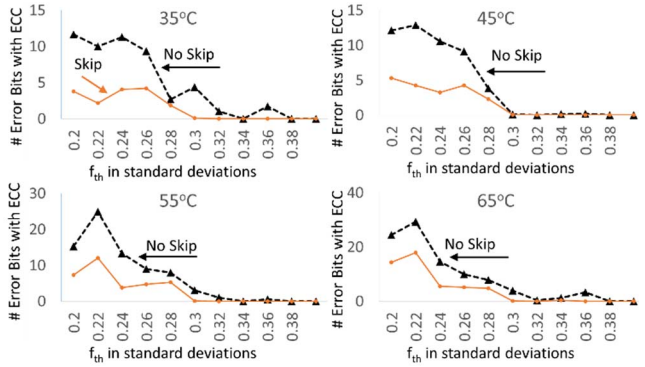


Figure 15. Number of information bit errors as function of the grouping frequency threshold f_{th} for no-skip and skip scenarios (both with CAECC). Variable temperature Case. 1.2V Vdd.

The grouping frequency threshold f_{th} is used to guarantee that the frequency difference of any two ROs in a group exceeds f_{th}. With no thresholding, we generate 555 bits from 193 ROs (10x18-RO groups and 1x13-RO group). As f_{th} increases, the chances for rank flips and hence bit error decreases; however, also, the number of ROs satisfying the f_{th} criteria decreases. Thus, we require more hardware resources in terms of ROs to generate the 555-bits. We define: (1) the normalized hardware overhead: the ratio of the number of ROs (nRO_{fth}>193) needed to generate the 555-bit response at a given grouping threshold f_{th} compared to that with no thresholding (nRO_0=193); (2) the normalized entropy as the ratio of the corresponding number of bits generated at f_{th} (555) to the number of bits that would have been generated from (nRO_{fth}) if there was no thresholding. We define the stability f_{th} to be the threshold beyond which a given scenario with ECC does not result in any bit error. It is clear that the stability f_{th} is higher in the no-skip mode, and hence the hardware overhead needed to reach stability f_{th} is higher in the no-skip scenario compared to the proposed skip-mode scenario. We note hardware overhead to be upto 20% higher for the no-skip versus the skip-mode (Fig. 17). We also notice up to 23% (average 9%) entropy improvement for the skip-mode.

6. CONCLUSIONS

An efficient implementation of (15, 7, 5) CA-ECC targeting a Xilinx xc7z020clg484-1 device is demonstrated. The design is also validated in a 40nm TSMC process. In conjunction with CAECC, skip-mode CSC is proposed to relax the number of per-block bit errors for the most probable adjacent rank flips. This helps enhance the correction capability and reduce stability f_{th}. Theoretical studies as well as grouping analysis based on RO-PUF data demonstrate enhanced entropy and reduced hardware overhead by up to 20% for the skip-mode CSC compared to no-skip mode.

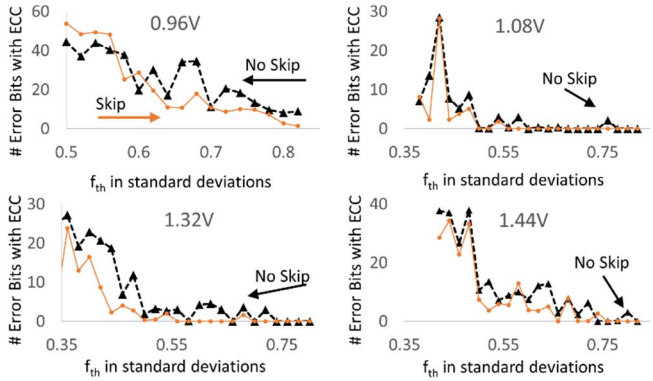


Figure 16. Number of information bit errors as function of f_{th} for no-skip and skip-mode (both w/ CAECC). Variable Voltage. 25C.

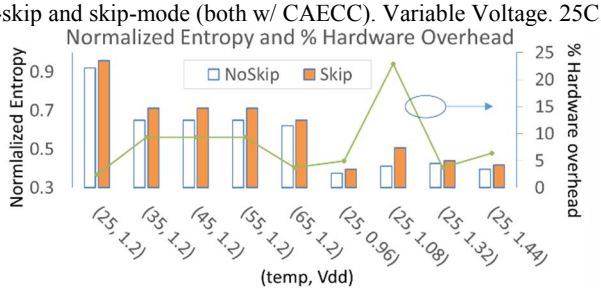


Figure 17 Normalized Entropy and relative % hardware overhead for no-skip versus skip-mode measured at their respective critical f_{th}.

Acknowledgement. The authors thank the American University of Beirut University Research Board for supporting this work. The authors would also like to thank Prof. Louay Bazzi for discussions.

7. REFERENCES

[1] http://www.dell.com/learn/

[2] http://www.tesla.com

[3] https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated

[4] L. Hardesty, "Someone Released the Code That Turns IoT Devices into Evil Bots", SDxCentral, 2016.

[5] Suh, G. Edward, and Srinivas Devadas. "Physical unclonable functions for device authentication and secret key generation." *44th DAC*. 2007.

[6] J. Guajardo et al., "FPGA intrinsic PUFs and their use for IP protection," 9th Int. Worksh. Crypt. Hardw. and Emb. Syst. (CHES'07), 2007, pp. 63-80.

[7] J. Lee et al, "A technique to build a secret key in integrated circuits for identification and authentication applications,"VLSI'04, Jun 04, pp. 176-179.

[8] C.-E. D. Yin and G. Qu, "Lisa: Maximizing ro puf's secret extraction," 3rd IEEE Intl Worksh on Hardw Oriented Security and Trust (HOST), 2010.

[9] Yin, Chi-En, Gang Qu, and Q Zhou. "Design and Implementation Of A Group-Based RO PUF". DATE 2013.

[10] Chowdhury, D. R., et al., "Design of caecc-cellular automata based error correcting code." IEEE Trans. on Comp: 43(6), 759-764, 1994

[11] Cho, S., Choi, U., &Heo, S., "Design of double error correcting codes based on cellular automata" Jnl of app. Math and Comp, 21(1/2), 545, 2006.

[12] A. Maiti and P. Schaumont, "A large scale characterization of ro-puf," 3rd IEEE Intl Worksh on Hard Oriented Security and Trust (HOST), 2010.

[13] Bohm, Christoph and Maximilian Hofer. Physical Unclonable Functions In Theory And Practice. 1st ed. New York: Springer, 2012. Print.

[14] S.Wolfram, Statistical mechanics of cellular automata, Rev. Mod. Phys., 55, July 1983, pp. 601-644.

[15] Yin, Chi-En. A group-based ring oscillator physical unclonable function. Thesis. University of Maryland, College Park, 2012.

[16] Synopsys Inc., Design Compiler User Guide, 2011.

[17] TSMC, TCBN40LPBWP version 120A Release Note, 2009.