Iterating Von Neumann's Post-Processing under Hardware Constraints

Vladimir Rožić[1], Bohan Yang[1], Wim Dehaene[2] and Ingrid Verbauwhede[1]
[1]ESAT/COSIC and iMinds, KU Leuven
[2]ESAT/MICAS, KU Leuven and IMEC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
E-mail: {Vladimir.Rozic, Bohan.Yang, Wim.Dehaene, Ingrid.Verbauwhede}@esat.kuleuven.be

***Abstract*—In this paper we present a design methodology and hardware implementations of lightweight post-processing modules for debiasing random bit sequences. This work is based on the iterated Von Neumann procedure (IVN). We present a method to maximize the efficiency of IVN for applications with area and throughput constraints. The resulting hardware modules can be applied for post-processing raw numbers in random number generators.**

***Index Terms*—Random number generators (RNGs), Entropy, Post-processing**

I. Introduction

True random number generators (TRNGs) are essential components in secure systems. TRNGs are used for generating session keys, challenges for authentication protocols and masks for countermeasures against side-channel attacks (SCA). The security of authentication protocols and SCA-resistant hardware implementations often rely on the TRNG's ability to produce statistically perfect and independent random bits.

Hardware entropy sources rarely provide perfect random numbers. Produced bits usually suffer from statistical defects such as bias and bit dependencies, so different post-processing algorithms are required to compress the raw bit stream into a statistically perfect, full-entropy bit stream. Special publication by the National institute of Standards and Technology (NIST) SP 800-90B [1] provides guidelines for TRNG design and evaluation. This document recommends to use cryptographic post-processing (hash function or a block cipher) unless a provably-secure alternative is provided. German standard AIS31 [2] requires arithmetic post-processing of the raw bits to reach the entropy level of at least 0.997 Shannons per bit before cryptographic post processing is used. Given the high area and energy demands of the cryptographic primitives as well as their throughput limitations, there is a need for lightweight, high-throughput post-processing techniques.

Von Neumann (VN) post processing [3] is a famous method for de-biasing bit sequences. This method guarantees the full entropy output provided that there are no dependencies between the raw bits, i.e. if the incoming bits are independent and the bias is the only statistical defect of the input sequence. The exact bias value doesn't need to be known at design time. One of the problems of this method is the fact that a lot of entropy is wasted in the process, causing a substantial reduction of throughput. An extension of the VN procedure that achieves improved efficiency of entropy extraction was proposed by Peter Elias in [4]. The theoretical paper of Yuval Peres [5] proposes a more practical solution based on the iterated Von Neumann (IVN) post-processing in order to improve the efficiency. The central idea of the IVN is to re-apply the VN post-processing on the information that is discarded in the first run. By iterating the procedure to infinity, it is possible to extract the theoretical maximum of entropy.

In this work, we explore the IVN under limited computational resources. The contributions of this paper are the following:

- We show that improvements of the original IVN are possible by changing the structure of the post-processing circuits. We are the first to explore the incomplete binary tree topologies of processing elements for throughput or area optimization.
- We provide an algorithm for finding the optimal circuit structure given the bias value and the hardware area budget. The produced post-processing circuit is optimal in terms of efficiency of the entropy extraction, which can be used for throughput or area optimization.
- We explore the design space for various bias values.

This paper is organized as follows. In Section II we explain the operation of the iterated Von Neumann post-processing. In Section III, we explain our contribution of finding the optimal configurations for performing the IVN with limited hardware resources. In Section IV experiment results are presented using the data from the FPGA implementation. Conclusion is given in Section V.

II. IVN post-processing

Here we provide an overview of the iterated Von Neumann (IVN) procedure [5].This method takes advantage of the fact that the information that is wasted during the classical Von Neumann post-processing can be recycled by reapplying the VN procedure on the discarded data.

We will use the following example to explain the principle of operation. A $100Mb/s$ entropy source produces bits with bias equal to $b = 10\%$. In this paper, we use the following definition of the bias:

$$b = \frac{|p_1 - p_0|}{2}, \tag{1}$$

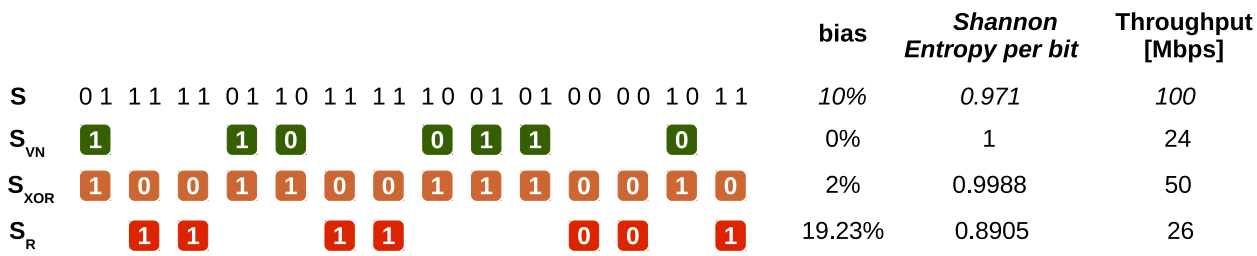


Fig. 1: Principle of operation of the iterated von Neumann post-processing.

where p_1 and p_0 are binary bit probabilities.

This source produces $97.1Mb/s$ of Shannon entropy. At the top of the Figure 1, we see an example of a sequence S generated by this source. We can decompose this sequence into three sequences which are denoted as S_{VN}, S_{XOR} and S_R. The S_{VN} sequence (shown in green) is constructed by applying the classical Von Neumann procedure on the input bits, i.e. by generating a bit 0 every time a bit pair 10 is detected in the original sequence and generating a bit 1 when 01 is detected. S_{VN} sequence is always unbiased and, in the presented example, its throughput is $24Mb/s$. In a general case, the relative throughput of this sequence is given by:

$$r_{VN} = \frac{Throughput(S_{VN})}{Throughput(S)} = (0.5 - b)(0.5 + b), \qquad (2)$$

which reaches the maximal value of 0.25 for $b = 0$.

The S_{XOR} sequence (shown in orange) is generated by producing a bit 1 every time blocks 01 or 10 are detected and producing a bit 0 when 00 or 11 is detected. This sequence has bias equal to $2{\cdot}b^2$ (2% in this example) and half the throughput of the original bit sequence ($r_{XOR} = 0.5$), in this example $50Mb/s$. The S_R sequence (shown in red) is the residual sequence that contains all information that is not extracted by the other 2 sequences. It is generated by producing a bit 1 whenever a block 11 is detected and producing 0 whenever 00 is detected in the original sequence. The bias of this sequence is $2{\cdot}b/(1{+}4{\cdot}b^2)$ (19.23% in this example) and the throughput is $26Mb/s$. In a general case, the relative throughput of the residual sequence is given by:

$$r_R = \frac{Throughput(S_R)}{Throughput(S)} = 0.25 + b^2. \qquad (3)$$

It is clear that no information (entropy) is lost during the decomposition because the original sequence S can be reconstructed from the sequences S_{VN}, S_{XOR} and S_R.

During the classical Von Neumann post-processing, only the S_{VN} (the green sequence) is sent to the output while the other two sequences are discarded. These two sequences contain all the discarded information. In [5] it is shown that these three sequences are independent and that Von Neumann procedure can be applied again on the discarded information (S_{XOR} and S_R). In the presented example, the S_{XOR} has a throughput of $50Mb/s$ and more than 0.99 bits of Shannon entropy per output bit. Note that this sequence has lower bias than the original one. The sequence S_R, on the other hand has higher bias. However, a substantial amount of information can be extracted from this sequence as well. With a throughput of $26Mb/s$ and around 0.89 bits of Shannon entropy per output bit there is a Shannon entropy throughput of more than $23Mb/s$.

TABLE I: Elementary IVN operation.

S		S_{VN}	S_{XOR}	S_R
0	0	-	0	0
0	1	1	1	-
1	0	0	1	-
1	1	-	0	1

The procedure can be iterated. S_{XOR} and S_R can be further decomposed into three components and the same procedure can be repeated. By iterating the procedure to infinity, the theoretical maximum of $97.1Mb/s$ can be achieved, i.e. all available entropy can be extracted. However, in order to extract all available entropy, an infinite amount of computational resources is required.

III. IVN Optimization

In this section, we explore the efficiency of IVN, using finite computational resources.

A. Elementary operation

VN is based on processing pairs of consecutive bits. The elementary operation of VN consists of reading a pair of bits and producing one or none output bits as the result. The elementary operation of IVN is only slightly more complex. Processing one pair of input bits results in exactly 2 bits at the output (one at the S_{XOR} and one at either S_{VN} or S_R) as shown in Table I. Full IVN implementation is obtained by cascading these elementary operations.

This work shows how to maximize the amount of extracted entropy using a fixed amount of elementary operations. This has applications in both hardware and software implementations. Elementary operations have a fixed cost in terms of the number of cycles for software operations or the number of processing modules operating in parallel for hardware implementations. In hardware designs, the proposed method

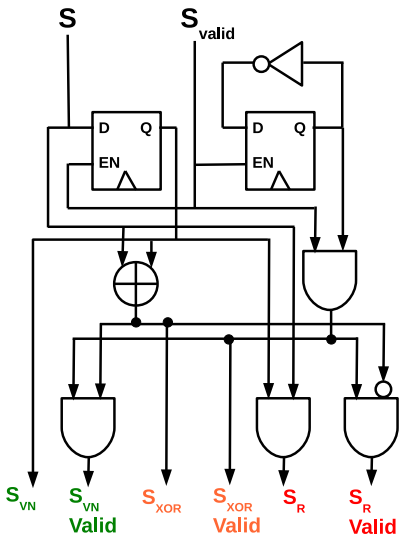


Fig. 2: Processing element architecture.

can be used to maximize throughput within the given area constraints. We will use hardware implementations to illustrate the benefits of the proposed method.

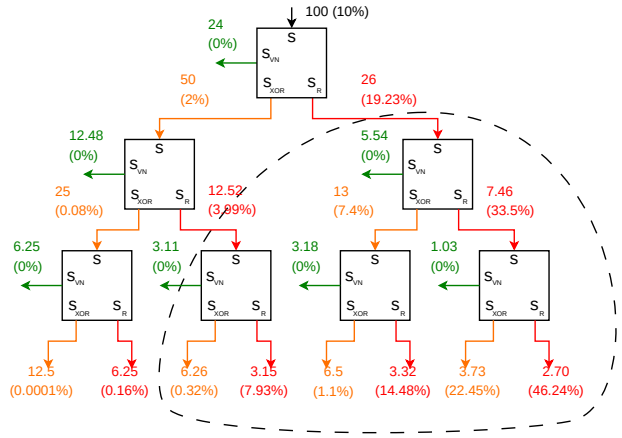
B. HW Processing Module

Figure 2 shows the architecture of the hardware module performing the IVN elementary operation. The design has 2 data inputs: signal S which represents the bit value and a signal S_{Valid}, which indicates that the value of S is valid. The output consists of 3 data signals (S_{VN}, S_{XOR} and S_R) and 3 signals to indicate when the output is valid ($S_{VN}Valid$, $S_{XOR}Valid$ and S_RValid). Hardware implementation of this module requires only 2 flip-flops and several logic gates. On Xilinx Spartan-6 the implementation consumes only 2 slices. ASIC implementation using standard cell methodology in open cell library NanGate 45nm consumes only 25 gate equivalents.

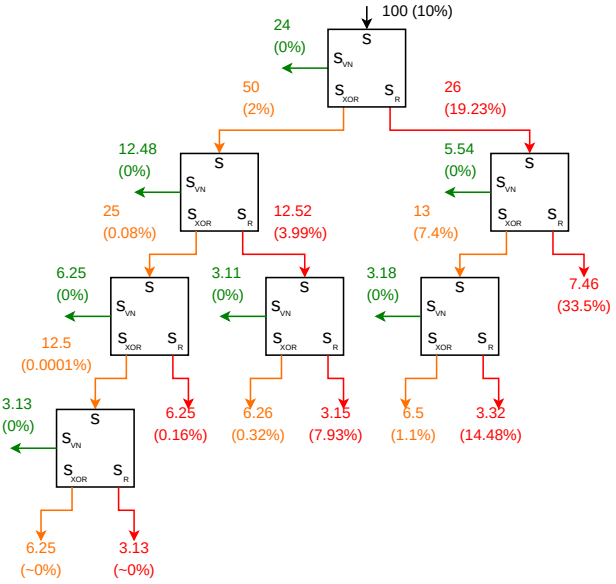
C. Optimization Strategy: An Example

To illustrate the benefits of the IVN optimization, we use the example of a biased generator discussed in Section II.

A $100Mb/s$ data stream is post-processed using the IVN procedure with 3 iterations. Figure 3a shows the post-processing block implemented using 7 processing elements. The throughput and bias of each branch is indicated in the figure. The S_{VN} output (shown in green) of each processing element is sent to the output, while the other 2 outputs are sent for recycling. It is immediately obvious that the elements don't contribute equally to the total output: the root node (classical 1-stage VN), contributes with $24Mb/s$, while the lower right element contributes with only $1.03Mb/s$. Moreover, the elements enclosed within the dashed line contribute with only $9.75Mb/s$ (out of $55.59Mb/s$) while consuming more than 50% of the area. The area optimization can be performed



(a) Classical IVN with 3 iterations. The part enclosed within the dashed line contributes very little to the total throughput.



(b) IVN structure with 7 processing elements, optimized for high throughput.

Fig. 3: Different IVN post-processing structures. The numbers indicate the throughput [Mb/s] and bias of each branch.

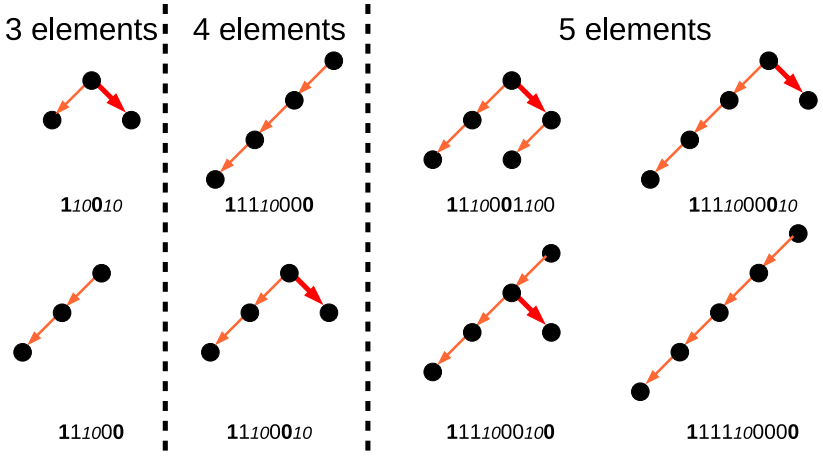
by removing these low-utilization elements, resulting in significantly more compact design with around 20% reduction in throughput. Throughput optimization can be performed by finding the optimal post-processing structure with the given number of processing elements. This optimal structure may depend on the bias. The optimal structure for 10% bias and 7 processing elements is shown in Figure 3b.

D. Binary tree representation

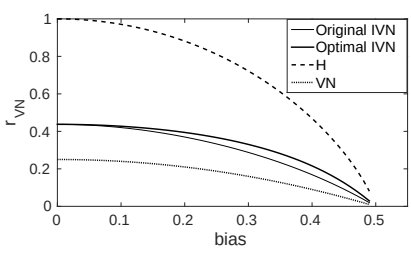
Note that the post-processing block using classical IVN has the structure of a complete binary tree of n levels where n is the number of iterations. This block can be implemented using exactly $2^n - 1$ processing elements. The optimized solution shown in Figure 3b also has a structure of a binary tree, however this tree is incomplete. The main contribution of this paper is exploring the incomplete binary tree structures

TABLE II: Throughput results after post-processing 100 Mb/s biased entropy source using the Worst case (W), Classical (C) and Optimal (O) IVN post-processing.

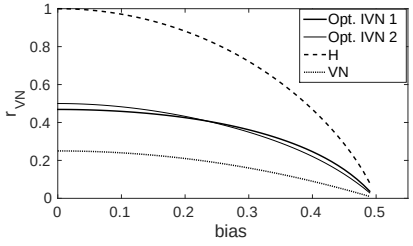
Bias (%)	3 elements			7 elements			15 elements		
	W	C	O	W	C	O	W	C	O
1	32.79	43.73	43.74	33.28	57.79	59.36	33.28	68.33	69.51
5	32.22	43.31	43.50	32.5	57.24	58.96	32.5	67.70	69.01
10	30.57	42.02	42.73	30.67	55.59	57.68	30.67	65.83	67.40
20	25.14	36.98	39.43	25.15	49.26	52.05	25.15	58.61	61.75
30	17.94	28.76	33.03	17.94	38.74	41.80	17.94	46.55	51.02
40	9.5	16.87	21.58	9.5	23.33	27.89	9.5	28.49	32.80



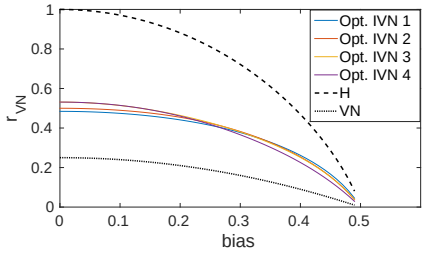
(a) Binary Trees



(b) Throughput efficiency of 3-element structures.



(c) Throughput efficiency of 4-element structures.



(d) Throughput efficiency of 5-element structures.

Fig. 4: Optimal binary trees and the corresponding throughput efficiency.

for IVN and providing the algorithm for finding the optimal structure for each bias value. Our results show that the classical IVN (complete binary tree) is rarely the optimal solution for any bias value. Note that using the incomplete binary trees enables us to use any number of processing elements, not only numbers of the form $2^n - 1$.

In the remainder of this paper we will use binary trees to represent different post-processing structures. Each node of the binary tree represents one processing element. The left branch of the node corresponds to the output S_{XOR} and the right branch corresponds to the output S_R. We will use the following notation to represent binary trees using strings of 1s and 0s. An empty tree is represented using the empty string '_'. Any other tree is represented using the following recursive definition:

$$< Tree >=' 1' < LTree >' 0' < RTree >, \qquad (4)$$

where $< LTree >$ and $< RTree >$ stand for the sub-trees connected to the left (S_{XOR}) and right (S_R) branch of the root node respectively. For example, a tree consisting of only one node is represented using a string '10'. More examples of binary trees and their string representations are shown in Figure 4a. For better readability, digits corresponding to the root node are shown using bold font and the digits corresponding to the leaf nodes are shown using italic font and a smaller font size.

Table II shows the achieved throughput for different bias values using different post-processing structures, assuming the $100Mb/s$ raw bits rate. Classical IVN can be performed using 3, 7, and 15 elements, these throughput results are shown in column (C). For the same number of elements, the optimal and the worst case binary trees were found, the corresponding throughput results are shown in columns (O) and (W). It can be seen that for low bias value and the high number of elements

there is little benefit over the classical IVN. This benefit is significantly higher for high bias and the low number of elements. Worst case throughput is always significantly lower than the optimal throughput so one needs to be careful when choosing the binary tree for post-processing.

For a small number of elements it is easy to find the optimal binary tree by searching through all combinations. The optimal post-processing methods using 3, 4 and 5 processing elements were explored and the results are summarized in Figure 4. Throughput efficiency of the 3-element binary trees (shown in Figure 4a) are computed for different bias values and the results are summed up in Figure 4b. Dashed line shows the Shannon entropy available in the sequence (maximal throughput efficiency) and the dotted line shows the efficiency of the Von Neumann procedure. Both binary trees show significant improvement of the post-processing efficiency over the VN procedure. The complete 3-element binary tree represents the classical IVN with one iteration of recycling, it is therefore denoted as *Original IVN*. The other binary tree is denoted as *Optimal IVN*. It results in the highest efficiency for all bias values. The other three 3-element trees were investigated, but they all show lower efficiency than the optimal IVN.

When using more than 3 elements, the optimal tree depends on the bias value. Two optimal trees can be constructed using 4 elements as shown in Figure 4a. One of these trees is optimal for bias lower than 25% and the other one for higher bias values, as shown in Figure 4c. Using 5 processing elements, we can construct 4 binary trees that are optimal for different bias intervals. Other 5-element trees are not optimal for any bias value. Therefore, the designer can choose the optimal post-processing configuration based on the bias of the original sequence. If the bias is not known at design time or it cannot be precisely estimated, wrong structure can be chosen. However, this is not a big problem because the structure that is optimal for one bias value has close-to-optimal performance for all bias values. This can be clearly seen in Figures 4c and 4d where all graphs corresponding to the optimal structures are clustered together. Therefore, the wrong estimation of bias doesn't cause high throughput reduction as long as the chosen structure is optimal for some bias value.

E. Finding the optimal trees

For higher number of elements, it becomes increasingly difficult to find the optimal binary tree by exploring all combinations. We propose an algorithm for finding the optimal tree given the bias and the number of processing elements. The intention is that the designer computes the available number of elements from the area budget and to use the algorithm to find the post-processing structure that extracts the most entropy.

Presented recursive function takes bias b and the number of elements n as input and produces the maximal throughput rate r and the corresponding binary tree structure T at the output. The optimal binary tree is constructed by assigning the root node and allocating the remaining $n-1$ nodes to the left and the right sub-trees. All n combinations are explored in a loop and the optimal sub-trees are found using the $FindTree$ function. The recursion stops when the number of nodes is 0. The combination resulting in maximal throughput is returned. The tree structure is reported as a string of ones and zeros according to the definition 4.

Algorithm 1 Algorithm for finding optimal trees.

function FINDTREE(b,n)
  if $n = 0$ **then**
    $r = 0$
    $T =' \_'$
  else
    $r_{VN} = 0.25 - b^2$
    $r_{XOR} = 0.5$
    $r_R = 0.25 + b^2$
    $b_{XOR} = 2b^2$
    $b_R = (2b)/(1 + 4b^2)$
    $r = 0$
    for $i = 0$ to $n - 1$ **do**
      $(nr_{XOR}, nT_{XOR}) = FindTree(b_{XOR}, i)$
      $(nr_R, nT_R) = FindTree(b_R, n - 1 - i)$
      $nr = r_{VN} + r_{XOR} \cdot nr_{XOR} + r_R \cdot nr_R$
      if $nr > r$ **then**
        $r = nr$
        $T =' 1' \, nt_{XOR} \, '0' \, nt_R$
      end if
    end for
  end if
  return (r, T)
end function

F. Results

Figure 5 shows the throughput efficiency for different bias values and the number of elements ranging from 1 to 20. The theoretical maximum is shown using dashed lines. This maximum corresponds to the Shannon entropy. The same trend is observed for all bias values: 1-stage post processing (VN) extracts around one quarter of the available entropy. Optimal configurations using 5 processing elements always extract more than half of the available entropy. According to the recommendations of NIST 800-90B [1], cryptographic post-processing should be used in such way that the amount of entropy at the input is twice the size of the output. Under these restrictions, the maximal efficiency of cryptographic post processing is equal to one half of the theoretical limit (entropy). Given its low area requirements, the 5-element IVN hardware module is always more beneficial for debiasing than the cryptographic post-processing. Adding more elements improves the throughput at smaller and smaller steps. After 15 elements are used, the additional contribution of each added element is less than 1%.

IV. EXPERIMENT RESULTS

The presented methodology was tested on Xilinx Spartan-6 FPGA. For testing purposes, we have produced biased random numbers using a carry-chain based TRNG [6] and

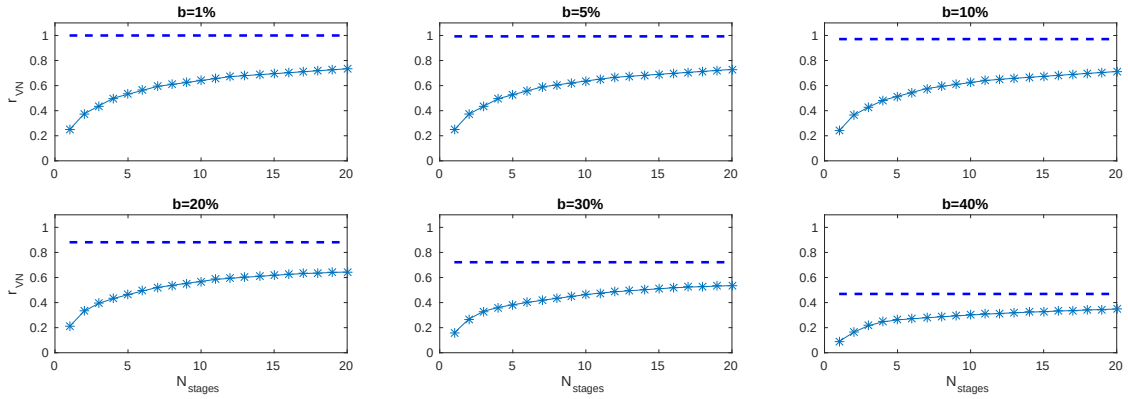


Fig. 5: Throughput efficiency given the number of elements for different bias values.

TABLE III: Relative Throughput.

Bias(%)	$r_{VN}(\%)$					
	3 elements		4 elements		5 elements	
	comp.	meas.	comp.	meas.	comp.	meas.
12.5	42.14	42.07	47.31	47.2	50.47	50.3
25.0	36.69	36.69	39.82	39.7	42.63	42.4
37.5	25.11	25.11	28.20	28.0	29.76	29.8

a biasing circuit. The biasing circuit takes three random bits at the input and produces one biased output bit. Bias can be controlled in steps of $1/8$. Biased sequences were post-processed using the optimal tree structures of 3, 4, and 5 elements. Table III summarizes the expected and the measured throughput values. Post-processed sequences pass all tests from the NIST suite [7].

V. Conclusions and Future Work

In this paper, we have proposed a lightweight hardware module for de-biasing binary data based on the iterated Von Neumann procedure. The main novelty of this work is the proposal to use the incomplete binary tree structures for VN iterations, unlike the classical approach which only uses complete binary tree structures. This approach gives us more freedom in exploring the design space allowing us to achieve area reductions with small penalties in throughput reduction. In addition, we present an algorithm for finding the optimal post-processing structure for any bias value and area budget.

It was found that the optimal post-processing structure depends on the bias of the input sequence, therefore the designer needs to estimate the bias of the raw numbers at design phase. However, bias estimation doesn't have to be very precise, especially when using a small number of elements. For example, for finding the optimal 4-element structure the designer only needs to determine if the expected bias is below or above 25%. Another observation is that the structures that are optimal for some bias value are close-to-optimal for any bias value. Therefore, a wrong bias estimation at the design phase doesn't result in high penalty in throughput reduction.

Throughput efficiency of optimal structures was evaluated for different bias values and it was found that 5-element optimal structures always extract more than half of the available entropy making them a preferable choice over cryptographic post-processing (used under restrictions of NIST 800-90B). We note that restrictions of VN and IVN procedures also apply on the proposed methods, i.e. the presented post-processing structures should only be used on sequences without dependencies between the generated bits. One possible direction for future work is using the optimized IVN procedures for debiasing PUF responses.

VI. Acknowledgments

This work is supported in part by the funding of the Research Council KU Leuven (C16/15/058), the Flemish Government through FWO G.0550.12N, G.0130.13N and FWO G.0876.14N, the Hercules Foundation AKUL/11/19, and by the European Commission through the Horizon 2020 research and innovation program under grant agreement No 644052 HECTOR. In addition, this work was supported in part by the Scholarship from China Scholarship Council (No.201206210295)

References

[1] E. Barker and J. Kelsey, "Recommendation for the Entropy Sources Used for Random BitGeneration," ser. NIST DRAFT Special Publication 800-90B, 2012.

[2] W. Killmann and W. Schindler, "A Proposal for: Functionality classes for random number generators ," ser. BDI, Bonn, 2011.

[3] J. Von Neumann, "Various Techniques Used in Connection with Random Digits," *National Bureau of Standards Applied Mathematics*, pp. 36–38, 1951.

[4] P. Elias, "The efficient construction of an unbiased random sequence," *Ann. Math. Statist.*, vol. 43, no. 3, pp. 865–870, 06 1972.

[5] Y. Peres, "Iterating Von Neumann's Procedure for Extracting Random Bits," *Ann. Statist.*, vol. 20, no. 1, pp. 590–597, 3 1992.

[6] V. Rožić, B. Yang, W. Dehaene, and I. Verbauwhede, "Highly efficient entropy extraction for true random number generators on FPGAs," in *Design Automation Conference (DAC)*, 2015, pp. 116:1–116:6.

[7] A. Rukhin et al., "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications ." Special-Pub:800-22 NIST, August 2008.