CHALMERS
UNIVERSITY OF TECHNOLOGY

Reliable and Area-Efficient Polar Decoders for SRAM PUF-based Key Generation

Reliable and Area-Efficient Polar Decoders for SRAM PUF-based Key Generation

Xu Wang and Per Larsson-Edefors
Chalmers University of Technology, Gothenburg, Sweden
xuwang@chalmers.se

***Abstract*—Physical unclonable functions (PUFs) offer several advantages for chip authentication, but their implementations must ensure reliable key generation while maintaining hardware efficiency. During key generation, error correction is required to suppress noise caused by environmental variations. Polar codes have emerged as a promising candidate, offering strong error-correction capability at short block lengths. However, their high decoding complexity is at odds with the constrained resources required by a PUF implementation. Sequential decoder architectures can mitigate this issue, however, storing log-likelihood ratios (LLRs) values still requires significant memory, especially when targeting a stringent block-error rate (BLER) of 10^{-9}. To address this challenge, we propose two techniques: inline quantization eliminates the storage of input LLRs, while LLR operation fusion reduces the storage required for intermediate values. Synthesized in a 22-nm CMOS technology, our area-efficient polar decoders provide up to 29.4% memory usage reduction and 16.7% total area reduction over conventional decoders. These reductions enable area-efficient decoder implementations that meet stringent BLER requirements. For example, a $\mathcal{P}(1024, 64)$ decoder, occupying 4098 μm^2, tolerates a 0.18 input bit-error probability under a target BLER of 10^{-9}.**

***Index Terms*—Physical unclonable function, Key generation, Polar code, Memory optimization, ASIC.**

I. Introduction

Physical unclonable functions (PUFs) [1] have garnered increasing attention owing to their advantages over traditional security primitives in chip authentication and cryptography [2], [3]. Unlike conventional approaches, PUFs generate cryptographic keys directly within the chip and on demand, eliminating the need for external memory storage. They leverage intrinsic variations inherent in the manufacturing process to generate unique keys. SRAMs [3]–[6] and ring-oscillator circuits [7], [8] are two commonly used sources from which intrinsic variations can be extracted. However, the raw response bits generated by these sources cannot be directly utilized as reliable cryptographic keys since environmental factors—including temperature fluctuations, humidity, and device aging—introduce variability, causing instability and inconsistency across repeated evaluations.

To counter this problem, error control codes (ECCs) can be employed within a concept known as secure sketch [9], [10]. However, there are several challenges to incorporating ECCs into PUFs, including (i) ensuring sufficient error-correction performance for a given input bit-error probability p, (ii) accomplishing an area-efficient implementation, and (iii) minimizing the number of source bits needed to generate fixed-length secret key. Polar codes have emerged as a compelling candidate [5], [6], providing high error-correction performance at short block lengths.

The problem is that decoding of polar codes is inherently complex and incurs significant computational and memory overhead, which is difficult to reconcile with the hardware efficiency constraints of PUFs. Two approaches to hardware-efficient decoder implementations are log-likelihood ratio (LLR) quantization [11], [12] and sequential decoder architectures [6]. The former reduces memory usage by representing LLRs with lower precision, and the latter reduces hardware complexity by traversing the polar decoding tree sequentially. While these approaches can reduce logic complexity, quantization may degrade error-correction performance, and the memory structures required for intermediate values still occupy a substantial portion of the decoder area [6].

The stringent requirements on PUF reliability, with $p > 0.15$ and a block-error rate (BLER) target of 10^{-9}, further exacerbate the issue of area overheads. These reliability requirements necessitate longer polar codes to achieve sufficient polarization and higher precision to provide adequate error-correction performance. Defining N as block length and q as the number of bits in a quantized LLR, the memory-bit requirement of conventional polar decoders scales as $\mathcal{O}(Nq)$. Therefore, increasing either parameter proportionally increases memory usage. Consequently, achieving high reliability while maintaining area efficiency remains a critical challenge for adopting polar codes in PUF-based key generation.

To address this challenge, we first integrate an inline quantization (ILQ) technique that computes LLRs on the fly during decoding, eliminating the (Nq)-bit storage requirement for input LLRs. Second, we propose an LLR operation fusion (LLR-OF) technique which fuses LLR operations across adjacent-stage decoding nodes. Taking s to denote the stage number in a polar decoding tree, LLR-OF provides a reduction of $2^s q$ bits for the intermediate LLR storage. A maximum of 50% reduction in LLR bits can be achieved when employing LLR-OF at $s = n - 1$, where $n = \log_2(N)$. Employing both ILQ and LLR-OF reduces the required memory bits by $3q/(4q + 6)$.

These techniques enable area-efficient polar decoder implementations capable of meeting the stringent BLER target of 10^{-9} while supporting input bit-error probabilities $p > 0.15$. To the best of our knowledge, this is the first polar decoder for PUF key generation that achieves these reliability requirements. We describe additional decoder designs to illustrate the tradeoffs between reliability and hardware efficiency.

II. PUF-based Key Generation

PUF-based key generation schemes [7], [9] typically employ a secure sketch for reliable reconstruction from noisy PUF responses. Common secure sketch approaches include syndrome and code-offset constructions [13].

A. Code-Offset Construction

In this work, we adopt code-offset construction in which helper data are generated by combining the enrolled PUF response with a randomly selected ECC codeword. During reconstruction, the noisy PUF response and the stored helper data are used to recover the original codeword through ECC decoding. For SRAM PUFs, the noisy PUF response during reconstruction can be modeled as the output of a binary symmetric channel (BSC), where the bit-error probability is determined by the PUF noise characteristics.

B. Error Control Codes For PUFs

During key generation, ECCs are required to provide sufficient error-correction performance, while minimizing the number of required PUF source bits. Hard-decision (HD) codes, such as Reed-Muller and Bose–Chaudhuri–Hocquenghem (BCH), offer low decoding complexity but typically cannot achieve sufficiently low BLER. Additionally, they require a large number of source bits when used alone [3]. To overcome these limitations, concatenated HD codes have been implemented [3], such as PUFKY [7]. In contrast, soft-decision (SD) codes can offer higher error-correction performance and improved code rates at the expense of increased decoding complexity. Maes et al. [14] proposed an SD decoder for generalized concatenated repetition and Reed–Muller codes, while [15] adopted convolution codes with reliable bit selection. Polar codes have attracted interest in secure sketch designs due to their strong error-correcting capability at short block lengths [5], [6], with [6] further demonstrating a hardware implementation for a target BLER of 10^{-6}.

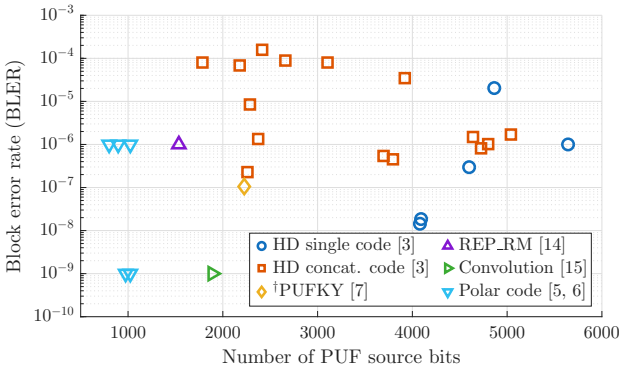

Fig. 1. Comparison of block error rate and required PUF source bit count (e.g., SRAM bits) for various code constructions with $p=0.15$. (†PUFKY is adapted to $p=0.15$.) Note that for polar code schemes, no design with a target BLER of 10^{-9} was implemented in hardware for PUF-based key generation.

Figure 1 summarizes the aforementioned codes for an input bit-error probability of $p=0.15$ comparing their number of required PUF source bits and the resulting BLER. While some standalone and concatenated HD codes can achieve a BLER below 10^{-6}, they require more source bits than SD codes do. For the stringent BLER of 10^{-9}, polar codes provide sufficient error-correction performance with fewer PUF source bits. Several PUF-based key generation schemes using polar codes have been proposed, however, hardware implementations remain scarce, especially for a BLER of 10^{-9} and $p>0.15$.

III. Polar Codes

Taking N to represent block length and K message length, a polar code $\mathcal{P}(N,K)$ with code rate $R=K/N$ is constructed based on channel polarization, which transforms a set of identical channels into a mixture of reliable and unreliable ones. The $N-K$ least reliable channels form the frozen-bit set $\mathcal{F}$, whose values are predetermined and known to both encoder and decoder, while the remaining K channels carry information bits. Various construction methods can be employed, e.g., Bhattacharyya parameter estimation [16], density evolution [17], and Monte-Carlo simulation. Taking the constructed error-correction performance into account, density evolution and Monte-Carlo simulation are used in this work.

A. Successive Cancellation (SC) Decoder

The decoder represents the most complex hardware unit in the secure sketch concept. Successive cancellation (SC), introduced in [16], can be adopted to decode the polar code, however, its performance degrades for short codes (low N) as a result of insufficient polarization. Successive cancellation list (SCL) decoding [18] enhances error-correction performance by maintaining multiple decision paths, but increases decoding complexity and hardware overhead. Therefore, we focus on SC decoding here, since minimizing hardware overhead is the primary priority in PUF scenarios.

The SC decoding process can be represented by a polar decoding tree [19], [20], where LLRs, denoted by α, are propagated from right to left, whereas the partial sum of the binary message, denoted by β, is propagated from left to right as shown in Fig. 2a. The information and frozen bits are represented by black and white, respectively.

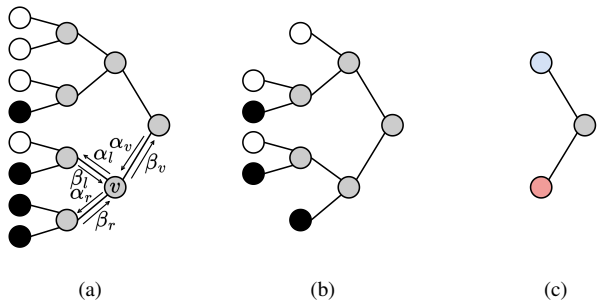

Fig. 2. Polar tree representation of (a) SC, (b) Simplified SC (SSC), and (c) FastSSC decoding algorithms for an (8, 4) polar code.

We assume BSC for an SRAM response. Here, the initial LLR, α_{root} for the i-th received bit $y[i]$ can be calculated by

$$\alpha_{\text{root}}[i] = \log\left(\frac{1-p}{p}\right)(1-2y[i]), \tag{1}$$

where p is the input bit-error probability. These initial LLRs are fed to the tree's root node and recursively propagated to

the leaf nodes. Fig. 2a illustrates the propagation process for node v at stage s in the decoding tree, assuming leaf nodes are at stage 0. Node v receives an LLR vector α_v of length $N_v = 2^s$ from its parent node, from which the left and right child LLR vectors, α_l and α_r, are computed using

$$\alpha_l[i] = f(\alpha_v[i], \alpha_v[i + N_v/2]), \tag{2}$$

$$\alpha_r[i] = g(\alpha_v[i], \alpha_v[i + N_v/2], \beta_l[i]), \tag{3}$$

where i goes from 0 to $N_v/2 - 1$. Functions f and g can be implemented in a hardware-efficient manner by eqs 4 and 5.

$$f(a, b) = \text{sign}(a)\ \text{sign}(b)\ \min\big(|a|,\ |b|\big) \tag{4}$$

$$g(a, b, c) = \big(1 - 2c\big)\, a + b \tag{5}$$

The binary message vector β_v is computed by

$$\beta_v[i] = \begin{cases} \beta_l[i] \oplus \beta_r[i], & \text{when } i < N_v/2, \\ \beta_r[i - N_v/2]\,, & \text{otherwise.} \end{cases} \tag{6}$$

For a leaf node corresponding to an information bit, $\beta_v = 0$ when $\alpha_v \geq 0$, and $\beta_v = 1$ otherwise; for a frozen bit, $\beta_v = 0$. After the binary message is propagated to the root node, the final decoded message, β_{root}, can be extracted.

Simplified successive cancellation (SSC) [19] and fast simplified successive cancellation (FastSSC) [20] were proposed to reduce decoding complexity. SSC prunes Rate-0 and Rate-1 nodes by directly decoding them without traversing their descendant subtrees, as shown in Fig. 2b. FastSSC further extends this idea by incorporating additional node types, such as repetition (REP) and single parity check (SPC) nodes, shown as blue and red nodes in Fig. 2c. While this can decrease decoding latency, it will likely increase hardware complexity. As area efficiency is the primary design objective in PUF-based key generation, we adopt SSC and incorporate only REP and SPC node simplifications from FastSSC.

B. Data Arrangement and Access Pattern

In conventional SC-based polar decoders [6], [12], [21], to store both the input and intermediate LLRs, a $(2Nq)$-bit memory (α-MEM) is required, where q is the number of quantization bits. The memory address is partitioned into two regions: $\mathcal{R}1$ for storing the intermediate LLRs (α_{inter}) and $\mathcal{R}2$ for storing input LLRs (α_{root}), as shown in Fig. 3a. During decoding, $\mathcal{R}1$ is accessed following the stage-wise schedule of the polar decoding tree in a read-compute-write pattern. Fig. 3b illustrates an example of this access pattern for a node v at stage $s = 4$ in a $\mathcal{P}(32, 8)$ decoding tree. First, a $\phi_v \in \{f_v, g_v\}$ is initialized using the α_{root} from the $\mathcal{R}2$ region. The resulting α_v, occupying $16q$ bits, is stored at the $s4$ location in $\mathcal{R}1$. Then, an f or g function is executed, reading this vector and producing either an $8q$-bit α_l or α_r, which is written to the $s3$ location in the $\mathcal{R}1$ region of α-MEM.

In addition, a $4N$-bit β-MEM is required and partitioned to three address regions to store the N-bit binary input message (y) in $\mathcal{R}1$, N-bit output message (β_{root}) in $\mathcal{R}2$, and $2N$-bit partial sums (β_{inter}) in $\mathcal{R}3$, as shown in Fig. 3a. Similar to α-MEM, the $\mathcal{R}3$ region for storing β_{inter} can also be divided into subregions for storage at different decoding stages.

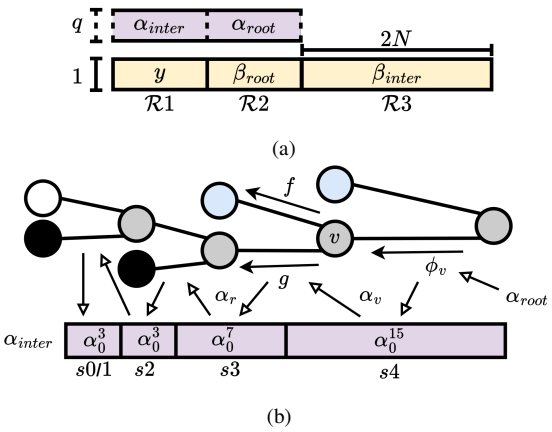

Fig. 3. (a) Data arrangement for LLR vector α and binary vector β. (b) Data arrangement and access pattern of α-MEM $\mathcal{R}1$ region for $\mathcal{P}(32, 8)$.

IV. Area-Efficient Polar Decoder

In contrast to high-throughput communication systems, polar decoders in PUF-based key generation must prioritize area efficiency and low hardware overhead over decoding speed. Thus, the proposed decoder employs a sequential, processor-like architecture. But since memory tends to dominate processor area, we propose two techniques—inline quantization and LLR operation fusion—to reduce memory storage.

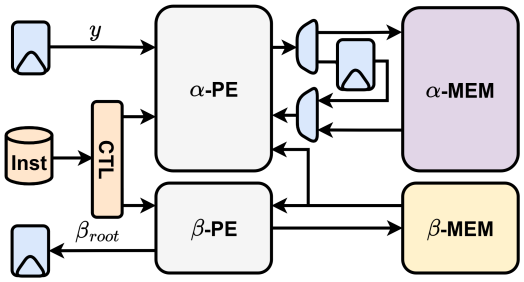

Fig. 4. Overview structure of the SC decoder, where Inst represent instruction storage, PE is processing element, and MEM represents memory.

Figure 4 shows an overview of the proposed decoder. 8-bit instructions are compiled using an in-house compiler and stored in the decoder. With the compiled instructions, the control logic (CTL) fetches the corresponding one according to the program counter and orchestrates PEs and data transfers. To limit the area overhead, the number of input and output registers is set to 8 and 16, respectively. The compute parallelism for α-PE is configured to four, allowing eight LLRs to be processed per clock cycle. The β-PEs operate with a parallelism degree of eight, resulting in up to 16 binary bits to be generated per cycle. To provide and consume these data, the bit-widths of α-MEM and β-MEM are set to $8q$ and 8 bits, respectively, where q denotes the number of quantization bits. In our design, to limit area, a single-port memory is used for both α- and β-MEM. The size of β-MEM is set to $3N$ by reusing its $R2$ region for input and decoded message storage.

A. Processing Elements

Equations 4-6 are implemented in the α- and β-PEs. Compared to α-PE, β-PE has a simpler structure as it primarily

consists of XOR gates and direct bit assignments of 0 and 1 depending on the node types. Processing element α-PE is built on four f and g function units, allowing eight input LLRs (α) to be computed and four output LLRs (α') to be generated.

To perform inline quantization (ILQ) of input LLRs, an ILQ unit is integrated, as shown in Fig. 5. To support REP operations, an accumulator with $n+2$ bits is implemented, accumulating α' from either f or g units. As the parallelism degree of α-PE is limited to four, for α' vectors longer than four, accumulation is performed incrementally. After accumulation, a sign unit is employed to determine the corresponding β'. For SPC operations, hard decisions are first generated from the α' values by a sign unit. Then the parity bit is computed incrementally using XOR accumulations. Meanwhile, the minimum absolute values of the α' elements are computed and compared to identify the smallest magnitude and its corresponding memory address, which is recorded to facilitate correction in the event of a parity mismatch.

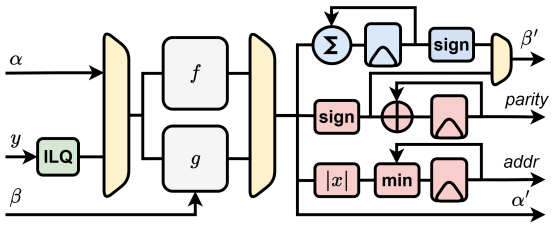

Fig. 5. Structure of α-PE, where ILQ denotes the inline quantization unit.

B. Inline Quantization (ILQ)

Unlike communication systems, where soft LLR values are typically provided by the channel, PUF-based key generation requires the decoder to compute LLRs internally from the PUF response. LLR quantization is commonly used in polar decoders to reduce hardware complexity. For BSC, the input LLRs are calculated according to eq. (1), which yields two discrete values: $\log\left((1-p)/(p)\right)$ for 0s and $\log\left((p)/(1-p)\right)$ for 1s. In addition, the intermediate LLR values are also discrete. A q-bit quantization scheme can be applied by mapping the LLRs to a range from -2^{q-1} to $2^{q-1}-1$, reducing both computational complexity and memory usage.

In our design, instead of storing the input LLRs in a dedicated memory region, ILQ performs on-the-fly quantization using the input message (y) stored in β-MEM. Therefore, for the computation of α_v at $s=n-1$ nodes, ϕ_v first reads the required input message y from β-MEM and then quantizes this inline to generate the corresponding LLR inputs for the computation. This approach removes the need for an (Nq)-bit $\mathcal{R}1$ region in α-MEM, resulting in a 50% memory bit reduction for α-MEM and a total $q/(2q+3)$ bit reduction when both α-MEM and β-MEM are considered.

C. LLR Operation Fusion

To further reduce memory storage in the decoding process, we propose an LLR operation fusion (LLR-OF) scheme that merges α (LLR) operations across adjacent decoding nodes. In the conventional decoding scheme described in Fig. 3b, LLR propagation at node v in stage s begins with the computation $\phi_v \in \{f_v, g_v\}$ of α_v from its parent node. These LLR values are stored in α-MEM and require $2^s q$ bits for storage. Then α_l is computed by the f function and propagated to the left child node to determine the corresponding β_l values. Based on β_l, the right-child LLR vector α_r is computed using the g function, requiring an additional $2^{s-1}q$ bits for storage.

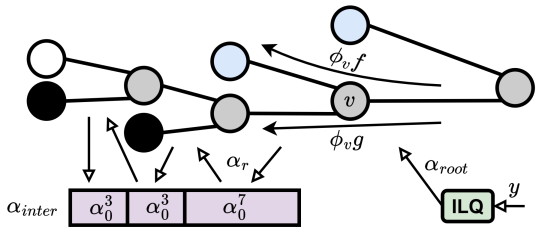

Fig. 6. Adjacent nodes LLR operation fusion of the root node ($s=4$) and its child nodes ($s=3$) for a $\mathcal{P}(32, 8)$ code. Here ϕ_v is a g_v function.

Applying LLR-OF at stage s eliminates the requirement of memory storage for α_v. It fuses the operation ϕ_v with the f and g functions to compute α_l and α_r, forming the $\phi_v f$ and $\phi_v g$ functions, respectively. However, to support this operation fusion, an $8q$-bit register is integrated (see Fig. 4) to buffer α_v values and forward them to the subsequent child operation. This results in a $(2^s/N)$-bit saving for α-MEM, and a total reduction of $2^s q/(2Nq+3N)$ bits (when including β-MEM) compared to non-optimized decoders. Note that, as the stage index decreases, the length of α_v decreases quadratically; therefore, applying LLR-OF at higher-stage nodes (closer to the root) yields greater memory storage reductions.

Figure 6 shows an LLR-OF applied at an $s=4$ stage node for a $\mathcal{P}(32, 8)$ code. Fusion operations ($\phi_v f$ and $\phi_v g$) consume the LLRs from the inline quantizer and produce α_l or α_r directly, eliminating the storage of the intermediate $16q$-bit α_v. As for $s=4$, the length of α_v is half the block length $N=32$ and, thus, a maximum of 25% memory reduction can be achieved for α-MEM, and a maximum of $q/(4q+6)$ reduction if β-MEM is included.

TABLE I
MEMORY CONFIGURATIONS OF DIFFERENT MEMORY OPTIMIZATIONS.

Technique	α-MEM	β-MEM	Total	Saving
Baseline	$2Nq$	$3N$	$N(2q+3)$	0
ILQ	Nq	$3N$	$N(q+3)$	$q/(2q+3)$
ILQ & LLR-OF	$Nq/2$	$3N$	$N(q/2+3)$	$3q/(4q+6)$

Table I summarizes the memory configurations with optimizations applied incrementally. The size of β-MEM is fixed to $3N$ for all configurations. The baseline corresponds to the memory configuration of conventional decoders [6], with a $(2Nq)$-bit α-MEM and a total memory size of $N(2q+3)$ bits when including β-MEM. ILQ reduces α-MEM from $2Nq$ to Nq, resulting in a total memory size of $N(q+3)$ bits. This corresponds to a $q/(2q+3)$-bit reduction compared with the baseline. After applying LLR-OF, the size of α-MEM is further reduced to $Nq/2$ bits, leading to a total memory size of $N(q/2+3)$ bits, thus achieving a reduction of $3q/(4q+6)$ compared with the baseline. It is worth to note that the relative

memory reduction in bits is independent of the code length N and that the reduction is greater for larger q.

V. Candidate Code Selection

We consider reliable decoders which can robustly handle input bit-error probabilities p from 0.15 to 0.20 and target BLERs of 10^{-6} and 10^{-9}. This scenario reflects the reliability requirements commonly considered in PUFs [5], [13]. In addition, a 128-bit secret key length is assumed, with the number of required source bits reduced to the greatest possible extent. With these requirements, candidate polar codes with $N \in \{512, 1024\}$ and $K \in \{32, 64\}$ are constructed, resulting in 2048 PUF source bits. Polar decoders with $q \in \{3, 4, 5, 6\}$ are implemented to explore the effect of quantization.

Evaluating BLER performance in the deep 10^{-9} region is difficult as it necessitates substantial analysis runtimes. Therefore, FPGA-based emulations are conducted across different candidate code configurations, with at least 10 block-error events collected at each operating point to ensure reliable BLER estimation. Our emulator implements the complete BSC chain together with the proposed decoders.

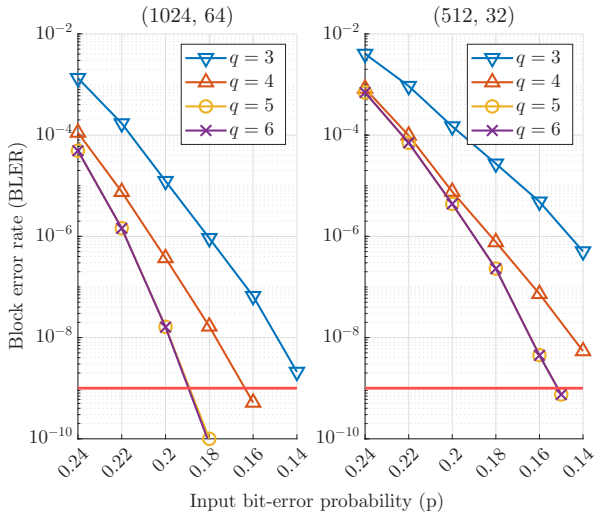

Fig. 7. BLER of $\mathcal{P}(1024, 64)$ and $\mathcal{P}(512, 32)$ as function of different input bit-error probabilities and different number of quantization bits.

Figure 7 shows the BLER performance of the candidate codes with varying quantization. Quantizations exceeding 5 bits do not provide any noticeable improvements. For the stringent BLER target of 10^{-9}, the 5-bit $\mathcal{P}(1024, 64)$ decoder demonstrates superior error-correction performance, supporting input bit-error probabilities up to 0.18. In contrast, the 5-bit $\mathcal{P}(512, 32)$ decoder satisfies the target only at $p = 0.15$. When the BLER target is relaxed to 10^{-6}, both codes can tolerate high input bit-error probabilities: $\mathcal{P}(1024, 64)$ satisfies the target up to $p = 0.20$ with $q = 4$ and 5, while $\mathcal{P}(512, 32)$ meets the requirement up to $p = 0.18$ with $q = 4$ and 5.

VI. Results

All decoder configurations were synthesized in Cadence Genus on a 22-nm FD-SOI CMOS technology, at $V_{DD} = 0.72$ V, 125 °C, and slow corner. To reduce area, α-MEM and β-MEM are using high-density single-port register files generated by Synopsys Memory Compiler. To evaluate the impact of different quantizations and memory optimization strategies on area, $\mathcal{P}(512, 32)$ and $\mathcal{P}(1024, 64)$ are implemented with a fixed $p = 0.18$, but with different quantizations.

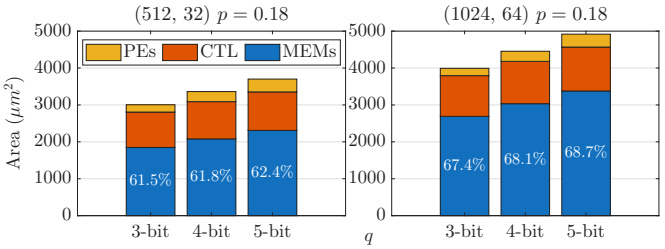

Fig. 8. Area of baseline decoders (no optimizations are applied) with varying quantization $q \in \{3, 4, 5\}$.

Figure 8 shows the non-optimized (baseline) decoder area when synthesized at 500 MHz. In general, the memory (α-MEM and β-MEM) dominates the area, with a maximum of 68.7% for a 5-bit $\mathcal{P}(1024, 64)$ decoder. The control logic (CTL) is the second largest component, however, its area does not vary significantly since the decoders are implemented in a processor-like architecture. Among all modules, the PEs use the least area. $\mathcal{P}(1024, 64)$ decoders occupy larger area compared with $\mathcal{P}(512, 32)$ and this is because longer block length codes require more memory.

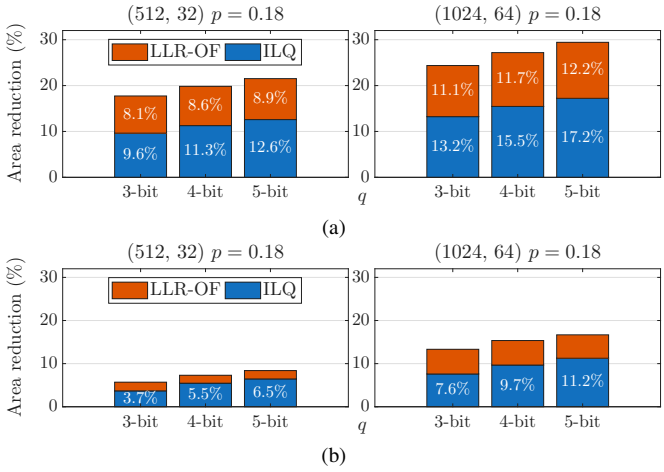

Fig. 9. (a) Memory area reduction and (b) total area reduction over the baseline, as a result of different memory optimization techniques.

Figure 9a illustrates the memory area reduction for different memory optimization techniques. The proposed decoder with inline quantization (ILQ) and LLR operation fusion (LLR-OF) can achieve a 29.4% memory area reduction over the non-optimization baseline. The memory area reduction after synthesis is smaller than the theoretical memory savings because the memory is implemented using high-density register-file macros, whose area does not scale linearly with the number of stored bits. The proposed techniques yield greater memory savings for $\mathcal{P}(1024, 64)$ decoders than for $\mathcal{P}(512, 32)$ decoders due to the former's higher memory storage requirements. When the number of bits of the quantized LLRs (q) increases, the memory area reduction becomes more pronounced.

Figure 9b shows that the proposed memory optimization techniques achieve up to 16.7% total area reductions relative to the baseline. Compared to the memory area reduction, the overall area reduction is limited since other components (PEs and CTL) constitute approximately 30–40% of total area. In addition to supporting ILQ and LLR-OF, CTL incorporates additional states and logic, increasing area overhead. ILQ achieves greater overall area reduction than LLR-OF, as it provides 2X higher memory reductions compared with LLR-OF, as shown in Table I.

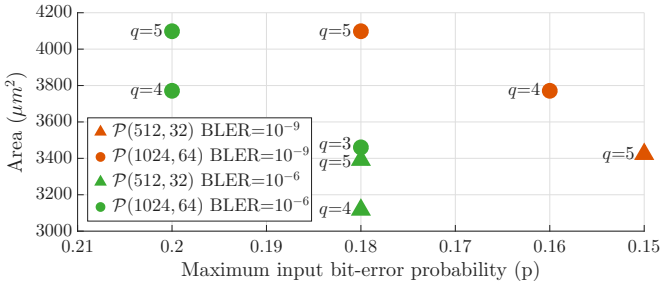

Fig. 10. Area of different decoder configurations versus maximum tolerated input bit-error probability p under the target BLER of 10^{-6} and 10^{-9}.

The aforementioned memory reduction techniques enable area-efficient decoder designs that tolerate higher input bit-error probability ($p > 0.15$). Fig. 10 shows the area and maximum tolerated p of the proposed decoders. All decoders implement the ILQ and LLR-OF techniques to reduce memory usage. Under the BLER requirement of 10^{-6}, the 4-bit $\mathcal{P}(512, 32)$ decoder is the most area-efficient candidate ($3117\,\mu m^2$) with a maximum tolerated $p = 0.18$. If p increases to 0.20, then only $\mathcal{P}(1024, 64)$ with $q = 4$ and 5 can meet the requirement, resulting in an area of 3771 and $4098\,\mu m^2$, respectively. For the more stringent BLER requirement of 10^{-9}, the 5-bit $\mathcal{P}(512, 32)$ decoder can handle a p up to 0.15 at an area of $3423\,\mu m^2$. In contrast, $\mathcal{P}(1024, 64)$ achieves higher error tolerance: with $q = 4$ and $q = 5$, the maximum p increases to 0.16 and 0.18, respectively, resulting in an area of 3771 and $4098\,\mu m^2$. To put these numbers in perspective, the placed-and-routed 22-nm decoder in [6] provides a BLER of 10^{-6} for $p = 0.15$, occupying an area of $4211\,\mu m^2$.

VII. Conclusion

Reliable PUF-based key generation requires the decoder to provide sufficient error-correction performance while maintaining high area efficiency. Achieving this using polar decoders leads to longer block lengths and higher precision LLRs, which increase the memory required to store intermediate values. To address this issue, we propose inline quantization and LLR operation fusion techniques, eliminating the storage requirement of input LLRs and reducing by 50% the storage requirement of intermediate LLRs. Synthesized in a 22-nm CMOS technology, under a BLER target of 10^{-9}, a 5-bit $\mathcal{P}(1024, 64)$ decoder supports up to $p = 0.18$ at an area of $4098\,\mu m^2$. For the same input bit-error probability, a 4-bit $\mathcal{P}(512, 32)$ decoder requires only $3117\,\mu m^2$ to achieve a BLER of 10^{-6}, which demonstrates that the decoder area scales well with different reliability requirements. This scalability also shows there is a potential for area-efficient implementations under more stringent conditions, such as $p = 0.25$ or BLER targets below 10^{-12}.

Acknowledgment

The authors would like to thank Area of Advance ICT for financial support and GlobalFoundries for design kit access.

References

[1] R. Pappu, B. Recht, J. Taylor, and N. A. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, pp. 2026–2030, 2002.
[2] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *ACM/IEEE Design Automation Conf. (DAC)*, pp. 9–14, 2007.
[3] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2008.
[4] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2007.
[5] B. Chen, T. Ignatenko, F. M. J. Willems, R. Maes, E. van der Sluis, and G. N. Selimis, "A robust SRAM-PUF key generation scheme based on polar codes," in *IEEE Glob. Commun. Conf. (GLOBECOM)*, 2017.
[6] C. Kestel, C. Frisch, M. Pehl, and N. Wehn, "Towards more secure PUF applications: A low-area polar decoder implementation," in *IEEE 35th Int. System-on-Chip Conf. (SOCC)*, 2022.
[7] R. Maes, A. V. Herrewege, and I. M. R. Verbauwhede, "PUFKY: A fully functional PUF-based cryptographic key generator," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2012.
[8] A. Maiti and P. Schaumont, "Improving the quality of a physical unclonable function using configurable ring oscillators," in *Int. Conf. on Field Programmable Logic and Applications*, pp. 703–707, 2009.
[9] Y. Dodis, R. M. Ostrovsky, L. Reyzin, and A. D. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *IACR Cryptol. ePrint Arch.*, vol. 2003, 2004.
[10] Y. Gao, S. F. Al-Sarawi, and D. Abbott, "Physical unclonable functions," *Nature Electronics*, vol. 3, pp. 81–91, 2020.
[11] S. H. Hassani and R. L. Urbanke, "Polar codes: Robustness of the successive cancellation decoder with respect to quantization," in *IEEE Int. Symp. on Information Theory (ISIT)*, pp. 1962–1966, 2012.
[12] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Trans. Signal Process.*, vol. 61, pp. 289–299, 2013.
[13] M. Hiller, L. Kurzinger, and G. Sigl, "Review of error correction for PUFs and evaluation on state-of-the-art FPGAs," *J. Cryptographic Engineering*, vol. 10, pp. 229–247, 2020.
[14] R. Maes, P. Tuyls, and I. M. R. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs," in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2009.
[15] M. Hiller, M.-D. Yu, and G. Sigl, "Cherry-picking reliable PUF bits with differential sequence coding," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, pp. 2065–2076, 2016.
[16] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, pp. 3051–3073, 2008.
[17] R. Mori and T. Tanaka, "Performance and construction of polar codes on symmetric binary-input memoryless channels," in *IEEE Int. Symp. on Information Theory (ISIT)*, pp. 1496–1500, 2009.
[18] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Int. Symp. on Information Theory (ISIT)*, pp. 1–5, 2011.
[19] A. A. Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 15, pp. 1378–1380, 2011.
[20] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 946–957, 2014.
[21] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1665–1668, 2010.