

=4

/Author (Pascal Giard, Syed Aizaz Ali Shah, Alexios  
Balatsoukas-Stimming, Maximilian Stark, and Gerhard  
Bauch) /Title (Unrolled and Pipelined Decoders based  
on Look-Up Tables for Polar Codes)

# Unrolled and Pipelined Decoders based on Look-Up Tables for Polar Codes

Pascal Giard\*, Syed Aizaz Ali Shah<sup>†</sup>, Alexios Balatsoukas-Stimming<sup>§</sup>, Maximilian Stark<sup>†</sup>, and Gerhard Bauch<sup>†</sup>

\*LaCIME, École de technologie supérieure, Montréal, Québec, Canada.

<sup>†</sup>Institute of Communications, Hamburg University of Technology, Germany.

<sup>§</sup>Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands.

Email: pascal.giard@etsmtl.ca, aizaz.shah@tuhh.de, a.k.balatsoukas.stimming@tue.nl, {maximilian.stark, bauch}@tuhh.de

**Abstract**—Unrolling a decoding algorithm allows to achieve extremely high throughput at the cost of increased area. Look-up tables (LUTs) can be used to replace functions otherwise implemented as circuits. In this work, we show the impact of replacing blocks of logic by carefully crafted LUTs in unrolled decoders for polar codes. We show that using LUTs to improve key performance metrics (e.g., area, throughput, latency) may turn out more challenging than expected. We present three variants of LUT-based decoders and describe their inner workings as well as circuits in detail. The LUT-based decoders are compared against a regular unrolled decoder, employing fixed-point representations for numbers, with a comparable error-correction performance. A short systematic polar code is used as an illustration. All resulting unrolled decoders are shown to be capable of an information throughput of little under 10 Gbps in a 28 nm FD-SOI technology clocked in the vicinity of 1.4 GHz to 1.5 GHz. The best variant of our LUT-based decoders is shown to reduce the area requirements by 23% compared to the regular unrolled decoder while retaining a comparable error-correction performance.

## I. Introduction

Unrolled decoders are known for their extremely high throughput [?], [?], [?], [?]. In particular, they offer at least one order of magnitude improvement in throughput with respect to standard decoders at the cost of larger area requirements. While this unrolling technique has been applied to successive-cancellation (SC)-based polar decoders before, e.g., [?], [?], it has not yet been combined with look-up table (LUT)-based decoding that has the potential to reduce the required quantization bit-width and, hence, the area and power consumption of the decoder.

**Contributions:** In this paper, we describe the design and implementation of unrolled and pipelined LUT-based hardware decoders for polar codes. We present three different variants and provide results for all three, along with results for a regular fixed-point decoder, illustrating the challenges of realizing the LUT-based decoders in hardware. In the end, we show that, even for a short (128, 64) polar code, a LUT-based decoder can reduce the area requirements by 23% while matching the error-correction performance and exceeding the throughput of a decoder using a standard fixed-point representation.

**Outline:** The remainder of this paper starts with ?? that provides the necessary background, consisting of a brief review of polar codes and an introduction to the SC and simplified successive-cancellation (SSC) decoding algorithms. Moreover, the concept of unrolled and pipelined hardware architectures is presented as well as that of using LUTs to implement functions. ?? describes our adaptation of the fully-unrolled and pipelined hardware architecture to LUT-based decoding. In particular, the generation of the LUTs, the architectures, and the decoders are discussed. ?? discusses implementation details and provides post-synthesis ASIC area and timing results using the 28 nm FD-SOI CMOS technology from ST Microelectronics. Finally, ?? concludes this paper.

## II. Background

### A. Encoding of Polar Codes

In matrix form, a polar code of length  $N$  can be obtained as  $\mathbf{x} = \mathbf{u}\mathbf{F}^{\otimes n}$ , where  $\mathbf{F} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ ,  $n \triangleq \log_2 N$ ,  $\mathbf{u}$  is the vector of bits to be encoded, and  $\mathbf{F}^{\otimes n}$  is the  $n^{\text{th}}$  Kronecker product of  $\mathbf{F}$  and  $\mathbf{F}^{\otimes 1} = \mathbf{F}$ . To obtain an  $(N, k)$  polar code of rate  $R = k/N$ , the  $k$  most-reliable bit locations in  $\mathbf{u}$  are used to hold the information bits while the other  $N-k$  bits, called frozen bits, are set to a predetermined value (usually 0). The bit-location reliabilities depend on the channel type and condition.

The encoding process can also be represented as a graph like that of Fig. ??, where  $\oplus$  represents modulo-2 addition (XOR). In that representation, a codeword is generated by setting the frozen-bit locations (the  $u_0$  to  $u_2$  in light gray) to 0 and the information-bit locations (the  $u_3$  to  $u_7$  in black) to the message to be encoded, and by propagating the data through the graph, from left to right. As described in [?], systematic encoding can be carried out by feeding the output values ( $x_0$  to  $x_7$ ) into the left-hand side, resetting the frozen-bit locations to 0, and propagating the data through the graph again.

### B. Successive-Cancellation Decoding and Simplified Successive-Cancellation Decoding

The SC decoding algorithm was proposed in the seminal work that introduced polar codes [?]. Illustrating its

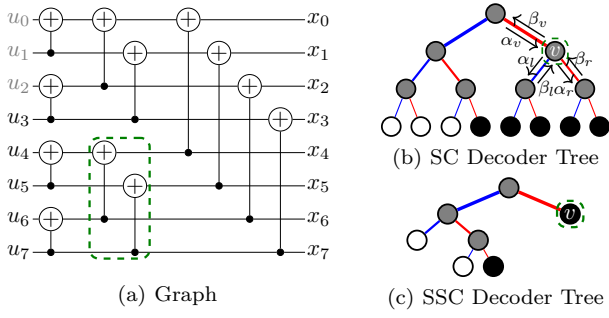


Fig. 1. Graph and decoder-tree representations of an (8, 5) polar code.

execution using a decoder-tree representation, it proceeds by visiting the tree—e.g., Fig. ??—sequentially, from top to bottom, from left to right, successively estimating  $\hat{\mathbf{u}}$  at the leaf nodes, from the noisy channel values. Visiting a left edge (blue) on this representation, the SC algorithm can calculate the soft-input log-likelihood ratios (LLRs)  $\alpha_l$  to the child node with the min-sum approximation [?]

$$\alpha_l[i] = \text{sgn}(\alpha_v[i]\alpha_v[i + N_v/2]) \min(|\alpha_v[i]|, |\alpha_v[i + N_v/2]|), \quad (1)$$

where  $\alpha_v$  and  $N_v$  are respectively the LLRs and node length from the parent node, and  $\text{sgn}(x)$  returns  $-1$  when  $x < 0$ ,  $+1$  otherwise. At the root node, the channel LLRs are used. Once a leaf node is reached, a bit  $\hat{u}_i$  (for a non-systematic polar code) is estimated as

$$\hat{u}_i = \begin{cases} 0, & \text{when } \alpha_v \geq 0 \text{ or } i \in \mathcal{F}; \\ 1, & \text{otherwise,} \end{cases} \quad (2)$$

where  $\mathcal{F}$  is the set of frozen-bit indices. For a systematic polar code under SC decoding, the estimated-bit vector can be obtained at the end of the decoding process by calculating  $\hat{\mathbf{u}}\mathbf{F}^{\otimes n}$  or its equivalent.

Visiting a right edge (red), the LLRs  $\alpha_r$  to the child node can be calculated [?] as

$$\alpha_r[i] = \begin{cases} \alpha_v[i + N_v/2] + \alpha_v[i], & \text{when } \beta_l[i] = 0; \\ \alpha_v[i + N_v/2] - \alpha_v[i], & \text{otherwise,} \end{cases} \quad (3)$$

where  $\beta_l$  is the bit-estimate vector generated by the left sibling in the decoder-tree. If the left sibling is a leaf node, its estimated-bit value  $\hat{u}_i$  is used as the  $\beta_l$ . Otherwise, the estimated-bit vector  $\beta_v$  at a node  $v$  is calculated as

$$\beta_v[i] = \begin{cases} \beta_l[i] \oplus \beta_r[i], & \text{when } i < N_v/2 \\ \beta_r[i + N_v/2], & \text{otherwise,} \end{cases} \quad (4)$$

where  $\beta_l$  and  $\beta_r$  are the bit-estimate vectors from the left- and right-child nodes, respectively.

The SSC algorithm is a variant that exploits the fact that subtrees solely composed of either frozen (rate-0 codes) or information nodes (rate-1 codes) do not need to be fully traversed [?]. Fig. ?? shows a decoder tree for the (8, 5) polar code of Fig. ??, where the SSC algorithm is applied, i.e., the tree of Fig. ?? is pruned by recognizing

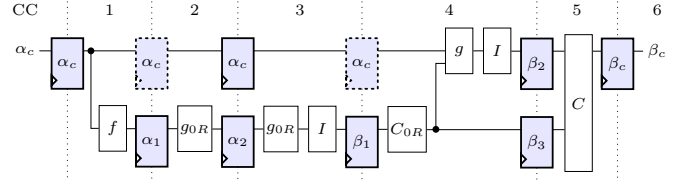


Fig. 2. Fully-unrolled deeply-pipelined decoder for a systematic (8, 5) polar code. Clock signals omitted for clarity. CC stands for clock cycle.

TABLE I  
Block types used in the unrolled decoders.

| Name     | Description  |
|----------|--|
| $f$      | Application of (??)  |
| $g$      | Application of (??)  |
| $g_{0R}$ | Application of (??), where $\beta_l$ is an all-zero vector |
| $I$      | Application of (??), note that $i \notin \mathcal{F}$      |
| $C$      | Application of (??)  |
| $C_{0R}$ | Application of (??), where $\beta_l$ is an all-zero vector |

that part of its left-hand-side subtree is a rate-0 code and that the right-hand-side subtree is a rate-1 code. In Fig. ??, the former subtree is replaced by a white node and the latter with a black node. Both cases are direct applications of (??), i.e., the estimated bit vector for a rate-0 code is always the all-zero vector and that of a rate-1 code is composed of hard decisions on the LLRs as it does not contain any redundancy.

### C. Unrolled and Pipelined Hardware Architectures

Unrolled decoder architectures provide extremely high decoding speeds. In an unrolled decoder architecture, each and every operation required is instantiated in hardware so that data can flow through the decoder with minimal control.

?? shows a fully-unrolled and deeply-pipelined decoder for the (8, 5) polar code illustrated in ??. Data flows from left to right. The  $\alpha$  and  $\beta$  blocks illustrated in light blue are registers storing LLRs or bit estimates, respectively. White blocks are the functions described in ?? and dotted registers are regular registers; they will be referred to when discussing partial pipelining in the following. As the  $C_{0R}$  and  $I$  blocks do not contain logic, i.e., they are equivalent to wires, they are inserted as preprocessing and postprocessing blocks, respectively. Among the registers, three are needed to retain the channel LLRs, denoted by  $\alpha_c$  in the figure, during the 2<sup>nd</sup>, 3<sup>rd</sup>, and 4<sup>th</sup> clock cycles (CCs). Such unrolled architectures for polar decoders were described at length in [?].

Deeply Pipelined Vs Partially Pipelined: In a deeply-pipelined architecture such as that illustrated in ??, a new frame is loaded into the decoder at every clock cycle. Therefore, a new estimated codeword is also output at each clock cycle. At any point in time, there are as many frames being decoded as there are pipeline stages. This

leads to a very high throughput at the cost of high memory requirements.

Partial pipelining, on the other hand, allows to reduce the required area, at the cost of reducing the throughput, by removing redundant registers in parts of the pipeline where data remains unchanged over multiple clock cycles [?]. Removing the dotted registers in ?? results in an initiation interval of 2, meaning that at every second clock cycle, a new frame can be fed into the decoder and a new codeword is estimated. We note that the interval only affects the memory, not the computational elements, in the decoder.

#### D. Functions as Look-up Tables

A LUT is a type of memory that maps a set of input values to output values. It can provide quick access to precomputed values that would otherwise require complex calculations. In the same vein, LUTs can also be used to approximate mathematical functions. In this work, we use the expression LUT-based decoders to refer to decoders in which arithmetic computations are replaced with look-up operations of integer-valued messages [?], [?]. The reliability information is embedded in the integer valued messages stored in the LUTs. We carefully craft LUTs to replace the logic that would normally go into the  $f$  and  $g$  functions of ??, where the  $g$  LUTs are also used for  $g_{0R}$ . Details regarding the design of these LUTs are given in the next section.

### III. Unrolled and Pipelined LUT-based Simplified Successive-Cancellation Decoding

#### A. Look-up Table Generation

The LUTs are designed using the information bottleneck (IB) method [?]. The IB framework clusters an observed quantity  $y$  into its compressed form  $t$  such that  $\max_{p(t|y)} I(X;T)$ . The random variable  $X = x$  is the designated quantity of primary relevance, e.g., a bit value, while the random variable  $T = t \in \mathcal{T} = \{0, 1, \dots, |\mathcal{T}| - 1\}$  is the compressed or quantized observation. The deterministic mapping  $p(t|y)$  can be treated as a LUT with  $y$  and  $t$  as input and output, respectively.

Each message  $t$  is associated with a LUT  $L_x(t)$  which can be computed from the distribution  $p(x, t)$  that is provided by the IB solution. Further, the finite alphabet  $\mathcal{T}$  for the quantized messages is chosen such that

$$L_x(t = |\mathcal{T}|/2 - 1 - j) = -L_x(t = |\mathcal{T}|/2 + j),$$

where  $j = 0, \dots, \frac{|\mathcal{T}|}{2} - 1$ . Moreover,  $L_x(t) > L_x(t')$  if  $t > t'$ ,  $\forall t, t' \in \mathcal{T}$ . Such a cluster assignment enables hard decisions of the bit  $x$  as the first half of the alphabet translates to negative LLRs while the second half translates to positive LLRs.

?? is used to illustrate the process of generating decoding LUTs for a polar code of length  $N = 2$ . In the figure, the uncoded bits  $\mathbf{u}$  are transformed into coded bits  $\mathbf{x}$  and received as  $\mathbf{y}$  after transmission through a quantized

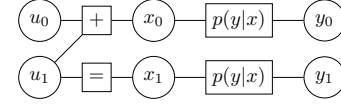


Fig. 3. Setup for generating decoding LUTs on a single building block

additive white Gaussian noise (AWGN) channel with transition probabilities  $p(y|x)$ . The AWGN channel quantizer is designed for a certain noise variance  $\sigma_N^2$  using the IB method with  $y \in \mathcal{T}$ .

With the quantized channel outputs  $[y_0, y_1]$  at hand, the LUTs for decoding  $u_0$  and  $u_1$  are generated by quantizing their bit channels using the IB method. For decoding  $u_0$ , the output  $\mathbf{y}_0 = [y_0, y_1]$  of its bit channel are quantized into  $t_0 \in \mathcal{T}$  such that  $\max_{p(t_0|\mathbf{y}_0)} I(U_0; T_0)$ . The deterministic mapping  $p(t_0|\mathbf{y}_0)$  can serve as a LUT that replaces the  $f$  function. It maps  $\mathbf{y}_0$  to  $t_0$  which, in turn, can be used to decode  $u_0$  as

$$\hat{u}_i = \begin{cases} 0, & \text{when } t_i > |\mathcal{T}|/2 \\ 1, & \text{otherwise.} \end{cases} \quad (5)$$

with  $i = 0$ . Similarly, the output  $\mathbf{y}_1 = [y_0, y_1, u_0]$  of the bit channel of  $u_1$  is compressed to  $t_1 \in \mathcal{T}$  aiming  $\max_{p(t_1|\mathbf{y}_1)} I(U_1; T_1)$ . The LUT  $p(t_1|\mathbf{y}_1)$  can then replace the  $g$  function, where it is used to map  $\mathbf{y}_1$  to  $t_1$  and decode  $u_1$  using (??).

For a polar code of length  $N > 2$ , the decoding LUTs are obtained by integrating the aforementioned mutual information preserving quantization into its density evolution [?]. These decoding LUTs then replace (??) and (??) in the SC decoding. This is illustrated for the root node in the decoder tree of Fig. ??, and LUTs  $p(t_0|\mathbf{y}_0)$  and  $p(t_1|\mathbf{y}_1)$  obtained in this section. In this case, the LLR carrying vectors  $\boldsymbol{\alpha}$  get replaced with vectors of integer valued messages  $\mathbf{t}$ . With  $N_v=8$ ,  $\mathbf{t}_v$  will carry the quantized channel outputs  $y_0, \dots, y_7$ . The LUT  $p(t_i|y_i, y_{i+N_v/2})$ , i.e., the mapping  $p(t_0|\mathbf{y}_0)$  with labels adjusted for  $N=8$ , will be used to generate the updates  $\mathbf{t}_l$  for the left child of the root node. Similarly, the updates  $\mathbf{t}_r$  for the right child will be generated using the LUT  $p(t_i|y_i, y_{i+N_v/2}, \beta_l[i])$ , i.e., the mapping  $p(t_1|\mathbf{y}_1)$  with adjusted labels. Note that a separate decoding LUT is used for each edge in the SC decoding tree, i.e,  $2N - N$  LUTs in total. At the leaf nodes, (??) can be used to estimate  $\hat{\mathbf{u}}$  from  $\mathbf{t}_l$  and  $\mathbf{t}_r$ . The decoder that uses the LUTs designed using the IB framework is henceforth referred to as the IB decoder.

#### B. Hardware-Efficient Look-up Tables

The LUTs in Section ?? are obtained from quantized density evolution of a polar code. Thus the  $N-1$  distinct  $f$ -operation LUTs are designed according to the check-node update rule, i.e., the so-called box-plus operation. The  $g$  LUTs are designed according to the variable-node update rule. From a hardware point of view, efficient

implementation of one, e.g.,  $f$ , LUT might not be valid for a different  $f$  LUT because the two will have different truth tables. However, it is beneficial to reduce the number of distinct LUTs so that the same hardware-efficient realization can be used for multiple LUTs.

In [?], LUT-based polar decoders were constructed where the min-sum approximation is utilized for designing the  $f$  LUTs while, like [?], the  $g$  LUTs are designed using the IB method. It was shown in [?] that the effect of using the approximate min-sum rule for LUT design on the error-correction performance of the decoder is negligible. Such a decoder is referred to as an MS-IB decoder.

In an MS-IB decoder, all the  $f$  LUTs have the same input/output relation. The number of distinct  $f$  LUTs, w.r.t. an IB decoder, thus reduces from  $N - 1$  to 1. Most importantly, the min-sum based  $f$  LUT can be realized as a min-sum of the integer valued message  $t_a, t_b \in \mathcal{T}$ . Recall (cf. Section ??) that the integer messages embed LLR information. The most-significant bit (MSB) of the LUT inputs can be treated as the sign of the associated LLR with MSB = 0 meaning negative LLR and vice versa. The remaining bits can be seen as the input-message magnitude associated to the LLR. The min-sum operation of the input integer messages can be expressed as:

$$t_o = f(t_a - \Delta, t_b - \Delta) + \Delta, \quad (6)$$

where  $\Delta = \frac{|\mathcal{T}|-1}{2}$ ,  $t_o \in \mathcal{T}$  and  $f(\cdot, \cdot)$  is computed using (??).

In comparison to (??), the argument  $\Delta$  in (??) is used for pre- and post-processing of the min-sum operation. From an implementation viewpoint, this is equivalent to inverting the magnitude carrying bits of any input message as well as the output message, if it falls in the first half of the finite alphabet  $\mathcal{T}$ . This extra logic can be discarded if a partially flipped finite alphabet is used. More precisely, all the integer messages are relabeled such that they belong to the relabeled finite alphabet  $\mathcal{T}_{re} = \{3, 2, 1, 0, 4, 5, 6, 7\}$  instead of  $\mathcal{T} = \{0, 1, 2, 3, 4, 5, 6, 7\}$  for  $|\mathcal{T}| = 8$ . Under the relabeled alphabet, the min-sum expression of (??) can directly be applied to integer-valued messages  $t'_a, t'_b \in \mathcal{T}_{re}$ . A LUT-based MS-IB decoder that uses  $\mathcal{T}_{re}$  is referred to as re-MS-IB decoder. Relabeling has no effect on error-correction performance.

### C. Unrolled-Architecture Generation

The hardware implementations are generated using our software toolchain, first mentioned in [?] but significantly improved over the years to extend its functionality, including to generate hardware unrolled decoders [?]. For this work, we further modified it in order to add support for substituting the  $f$ ,  $g$ ,  $g_{0R}$  functions of ?? by LUTs.

Our toolchain notably takes a polar code construction as well as a configuration file as input, and from that, it optimizes the decoder tree, e.g., going from the decoder tree of Fig. ?? to that of Fig. ??, and then generates the decoder. Among the configurable options are the code

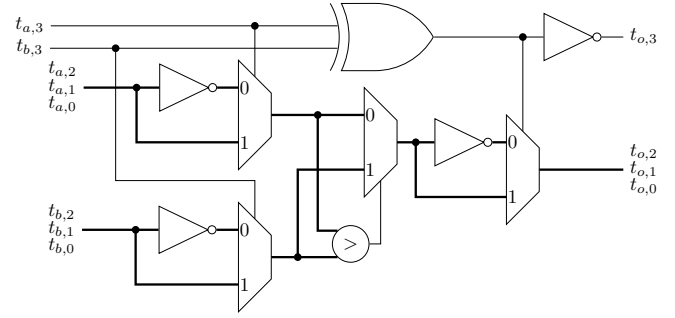


Fig. 4. min-sum for  $\mathcal{T}$  in the MS-IB decoder.

length, rate, the types of nodes that can be used, and the type of decoder. The type of nodes dictates the decoding algorithm. For this work, we generate hardware unrolled decoders in VHDL that implement the SSC algorithm.

## IV. Implementation and Results

In this section, we present implementation results for various LUT-based unrolled decoders. A fixed-point unrolled decoder is used for reference. Our decoders target a systematic (128,64) polar code optimized for  $E_b/N_0 = 3.0$  dB using the method of Tal and Vardy [?], and have an initiation interval of 10 and a fixed latency of 86 clock cycles. Without loss of generality, systematic coding is used as it offers better bit-error rate (BER) performance than non-systematic coding at the cost of a negligible complexity increase. The quantization used for the fixed-point decoder was determined by way of simulation with bit-true models. We denote quantization as  $Q_i, Q_c$ , where  $Q_c$  is the total number of bits used to store a channel LLR and  $Q_i$  is the total the number of bits used to store an internal LLR. All LLRs use 2's complement representation. All LUT-based decoders were designed for  $E_b/N_0 = 3.0$  dB and  $|\mathcal{T}| = 16$ , i.e, 4-bit resolution.

### A. Functional Blocks in LUT-based Decoders

This section presents how the block types of ?? were adapted to the LUT-based decoders.

The circuits implementing the  $C$  and  $C_{0R}$  blocks are identical to the fixed-point decoder. The  $I$  blocks are similar to those of the fixed-point decoder, with an added inverter per bit. It corresponds to a series of inverters that take the MSB of the soft messages  $\alpha$  or the integer messages  $t$  as input.

The  $g$  and  $g_{0R}$  blocks are realized by the synthesis tools as logic circuits according to the truth table of corresponding LUTs in the three LUT-based decoders. In the IB decoder, the  $f$  blocks are realized in the same way.

?? illustrates the  $f$  block circuit for the MS-IB decoder. The binary form of a 4-bit message  $t_x$  is denoted as  $[t_{x,3}, t_{x,2}, t_{x,1}, t_{x,0}]$ . Thick lines indicate vectors of bits. By slight abuse of symbolism, the inverters with vectors as input and output carry out bitwise inversions.

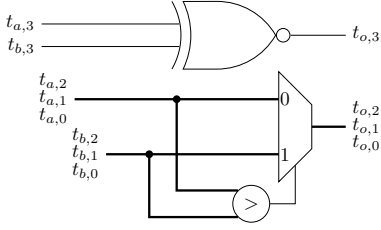


Fig. 5. min-sum for  $\mathcal{T}_{re}$  in the re-MS-IB decoder.

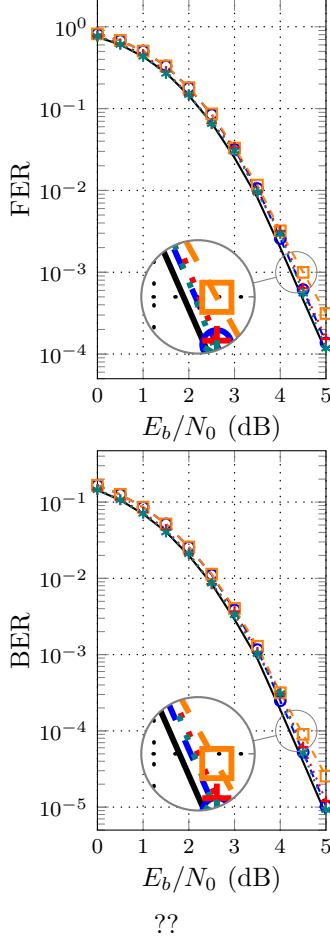


Fig. 6. Error-correction performance of the systematic (128, 64) polar code.

?? illustrates the  $f$  block circuit for the re-MS-IB variant. Comparing Figs. ?? and ??, it can be seen that the critical path for the re-MS-IB variant of  $f$  is much shorter than that of MS-IB. It consists of a single multiplexer as opposed to 3 multiplexers and 2 inverters. Furthermore, the amount of resources required is much less, as will be reflected in the results of Section ??.

#### B. Error-correction Performance and Impact of Quantization

?? shows the error-correction performance of the systematic (128, 64) polar code our unrolled decoders support, modulated with binary phase-shift keying (BPSK) and

TABLE II  
Comparison of unrolled decoders for a systematic (128, 64) polar code. All decoders have an initiation interval of 10.

| Variant                           | Fixed point | LUT based |       |          |
|-----------------------------------|-------------|-----------|-------|----------|
|                                   |             | IB        | MS-IB | re-MS-IB |
| Area (mm <sup>2</sup> )           | 0.090       | 0.254     | 0.218 | 0.069    |
| Frequency (GHz)                   | 1.47        | 1.38      | 1.40  | 1.51     |
| Latency (ns)                      | 58.6        | 62.2      | 61.3  | 56.8     |
| Info. T/P (Gbps)                  | 9.40        | 8.85      | 8.98  | 9.68     |
| Area Eff. (Gbps/mm <sup>2</sup> ) | 104.3       | 34.9      | 41.2  | 140.0    |

transmitted over an AWGN channel. Fixed-point results for  $Q_i.Q_c = 5.4$  and  $Q_i.Q_c = 4.4$  are presented as well as results for all LUT-based variants. Floating-point results are also included for reference. The  $Q_i.Q_c = 5.4$  fixed-point decoder and the MS-IB and re-MS-IB LUT-based variants are shown to have a coding loss of under 0.1 dB at a frame-error rate (FER) of  $10^{-3}$  or at a BER of  $10^{-4}$  compared to the floating-point representation. Meanwhile, the coding loss of the IB LUT-based decoder is little under 0.13 dB at the same BER and FER values. The LUT-based decoders initially have a smaller coding loss than the  $Q_i.Q_c = 5.4$  fixed-point decoder, but eventually come to match as the channel noise decreases.

#### C. Comparison of the Unrolled Decoders

?? shows synthesis results using a 28 nm FD-SOI CMOS technology from ST Microelectronics. All decoders were synthesized to target a clock frequency of 1.5 GHz. The first column is for the  $Q_i.Q_c = 5.4$  fixed-point decoder and the last three for the 4-bit LUT-based decoders.

We observe that our first and relatively naive LUT-based implementation (IB) unfortunately has 182% higher area and 6% lower throughput than the baseline fixed-point decoder. The situation improves when using the  $f$  block of ?. The MS-IB decoder requires 142% higher area for a 4% lower throughput compared to the fixed-point decoder. Finally, significant gains with respect to the baseline decoder are observed when applying the relabeling described in Section ??, where the resulting decoder, re-MS-IB, is 23% smaller and 3% faster than the fixed-point decoder, leading to a 35% better area efficiency.

#### V. Conclusion

In this work, we showed that replacing blocks of logic by LUTs in an unrolled decoder for polar codes may not necessarily lead to gains in terms of key performance metrics. We presented three variants of LUT-based decoders and compared them against a regular fixed-point decoder. We used a short systematic polar code to illustrate. Beyond carefully crafting the LUTs, the key ingredient to obtain good performance has been to determine LUT realizations having truth tables that lead to efficient implementation. This was achieved by deploying the min-sum approximate LUT design together with appropriate relabeling of the

inputs and output of LUTs. As a result, the third variant of the LUT-based decoders (re-MS-IB) was shown to outperform the baseline fixed-point decoder on all metrics, notably offering 35% better area efficiency with similar or better error-correction performance.

#### Acknowledgement

The authors would like to thank Marzieh Hashemipour Nazari (Eindhoven University of Technology) for providing the ASIC synthesis results. The work of Pascal Giard is supported by NSERC's Discovery Grant #651824.