

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) α_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 α_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 α_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 β_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 β_l . Otherwise, the estimated-bit vector β_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 β_l and β_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

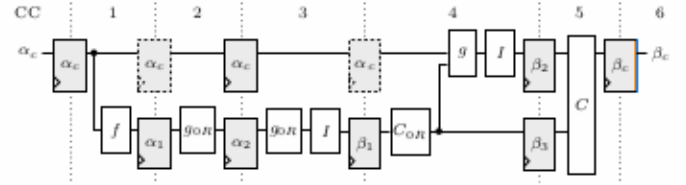


Fig. 2. Fully-unrolled deeply-pipelined decoder for a systematic (8, 5) polar code. Clock signals omitted for clarity. CC stands for clock cycle.

TABLE I
Block types used in the unrolled decoders.

Name	Description
f	Application of (??)
g	Application of (??)
g_{0R}	Application of (??), where β_l is an all-zero vector
I	Application of (??), note that $i \notin \mathcal{F}$
C	Application of (??)
C_{0R}	Application of (??), where β_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 α and β 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 α_c in the figure, during the 2nd, 3rd, and 4th 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

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 Δ 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 gf, gg_R functions [?] 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

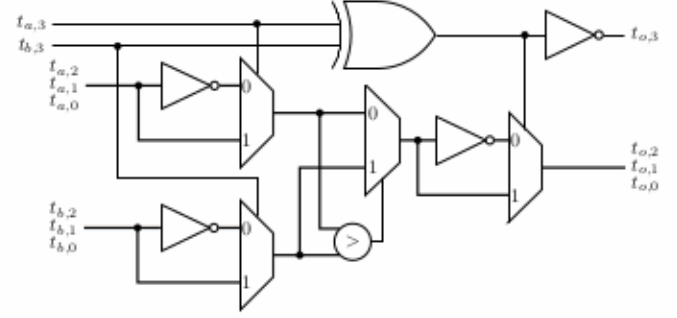


Fig. 4. min-sum for \mathcal{T} in the MS-IB decoder.

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 α 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.

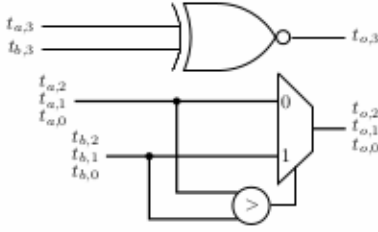


Fig. 5. min-sum for T_{re} in the re-MS-IB decoder.

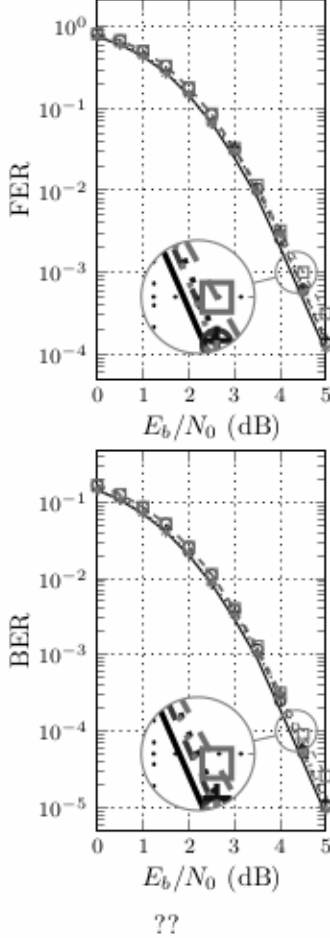


Fig. 6. Error-correction performance of the systematic (128, 64) polar code.

?? 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 ²)	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 ²)	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 2.23% smaller and 3.3% faster than the fixed-point decoder, leading to 35% better area efficiency.

V. Conclusion

In this work, we showed that replacing blocks of logic by LUTs in unrolled decoders for polar codes may not be a wise choice to gain in terms of LUT performance metrics. We presented three variants of LUT-based decoders, and compared them against a regular fixed-point decoder. We used full synthesis to find the best LUT-based decoder. Beyond carefully crafting the LUTs to be keying LUTs, it is also a good performance to determine LUT implementations using truth tables by deploying efficient implementations. This was achieved by deploying the approximate the LUT design together with appropriate relabeling of the

