

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

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)

Pascal Giard*, Syed Aizaz Ali Shah†, Alexios Balatsoukas-Stimming§, Maximilian Stark†, and Gerhard Bauch†

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

†Institute of Communications, Hamburg University of Technology, Germany.

§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 $x = uF^{\otimes n}$, where $F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, $n \triangleq \log_2 N$, u is the vector of bits to be encoded, and $F^{\otimes n}$ is the n^{th} Kronecker product of F and $F^{\otimes 1} = F$. To obtain an (N, k) polar code of rate $R = k/N$, the k most-reliable bit locations in 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 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{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 [?]

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

LLRs and a very high throughput. The magnitude of the input message, magnitude associated to the LLR. The min-sum operation of the input integer messages can allow 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 [?]. In 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 directly be applied to integer-valued messages $t_i, t_o \in T_{re}$. An approximate mathematical functions. In this work, we use LUT-based MS-IB decoder that uses T_{re} is referred to as *re-MS-IB* decoder. Relabeling has no effect on error-correction in which arithmetic computations are replaced with look-up operations of integer-valued messages [?], [?]. The

A. Functional Blocks in LUT-based Decoders

The hardware implementations are generated using our software toolchain, first mentioned in [?] but significantly improved over the years to extend its functionality, including details regarding the design of these LUTs are given in the next section.

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

Our toolchain notably takes a polar decoder 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 length, rate, the types of nodes that can be used, and the type of decoder. The type of nodes that can be used, and the type of decoder. The type of nodes that can be used, and the type of decoder.

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 as the input and output, respectively.

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 $N_0 = 3.0$ dB using the method of quantized messages [?], 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 simulating the alphabet transitions. We denote quantization levels Q_L and Q_C as the total number of bits used to store a channel LLR and Q_C is the total number of processes of generating channel LLRs using 2's complement representation. All the LUT-based decoders are designed for transformed 3.0 dB and coded bits s_i 4-bit resolution as 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 σ_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 the bit channels using the IB method. For decoding u_0 , the output $y_0 = [y_0, y_1]$ of its bit channel are quantized into $t_0 \in \mathcal{T}$ such that $\max_{p(t_0|y_0)} I(U_0; T_0)$. The deterministic mapping $p(t_0|y_0)$ can serve as a LUT that replaces the f function. It maps y_0 to t_0 which, in turn, can be used to decode u_0 as

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

with $i = 0$. Similarly, the output $y_1 = [y_0, y_1, u_0]$ of the bit channel of u_1 is compressed to $t_1 \in \mathcal{T}$ aiming $\max_{p(t_1|y_1)} I(U_1; T_1)$. The LUT $p(t_1|y_1)$ can then replace the g function, where it is used to map y_1 to t_1 and decode u_1 using (5).

The circuits implementing the C and C_{OR} 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.

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 (5) and (7) in the SC decoding. This is illustrated for the root node in the decoder tree of Fig. ??, and LUTs $p(t_0|y_0)$ and $p(t_1|y_1)$ obtained in this section. In this case, the LLR f blocks are realized in the same way.

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

Fig. ?? illustrates the g 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 the SC decoding tree, i.e., 21 of N LUTs in total. At the root nodes, 3 multiplexers and 2 inverters. Furthermore, the amount of resources required is much less, as will be reflected in the results of Section ??.

B. Hardware Efficiency, Lookup and Impact of Quantization

The LUTs in this section are generated for the systematized (128,64) polar code of our unrolled decoders support, modulated by differential phase shift keying (DPSK) and transmitted over an AWGN channel. Fixed-point results for Q_L , Q_C and Q_C LUTs are presented as well as results for unrolled LUT-based updates. Full floating-point results are also included for 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 $\Delta = \frac{|T|-1}{2}$, $t_o \in \mathcal{T}$ 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 Fig. 4, implementation perspective, 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 pre-processed such that they belong to the relabeled finite alphabet $\mathcal{T}_{re} = \{3, 2, 1, 0, 4, 5, 6, 7\}$ instead of $\mathcal{T} = \{0, 1, 2, 3, 4, 5, 6, 7\}$. Under the relabeled alphabet, the min-sum expression of (??) can directly be applied to integer-valued messages $t'_a, t'_b \in \mathcal{T}_{re}$. All the MS-IB decoders that uses \mathcal{T}_{re} is referred to as re-MS-IB decoder. Relabeling has no effect on error-correction performance.

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^{-5} or at a BER of 10^{-5} compared to the floating-point representation. Meanwhile, the coding loss of the IB LUT-based decoders is under 0.15 dB at the same BER and FER values. For this work we further modified it in order to add support for substituting the f, g, g_{OR} 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. 8 synthesis result using a 28 nm FD-SOI CMOS technology. A strong STE Microelectronics IP blocks 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) performance to outperform the baseline fixed-point decoder on all metrics, notably offering 35% better area efficiency with similar or better error correction performance. 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, like the total number of bits used to store a channel LLR. All LUTs in the decoders were designed for 28 nm FD-SOI CMOS technology. The decoders were supported by ISSI SRC/T Discrecy Grant #651824on.

ACKNOWLEDGEMENT

The author would like to thank Manjinder Hishinipand Nazare (Bindhoven University of Technology) for providing the ASIC synthesis tool. The decoders were designed for 28 nm FD-SOI CMOS technology. The decoders were supported by ISSI SRC/T Discrecy Grant #651824on.

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_{OR} 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_{OR} 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.

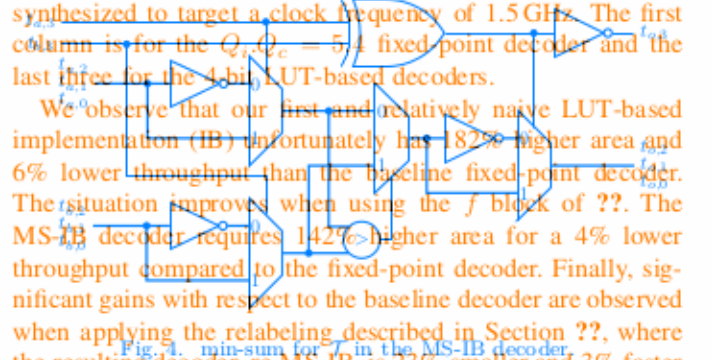


Fig. 4. min-sum for f in the MS-IB decoder.

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_{OR} 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. 8 synthesis result using a 28 nm FD-SOI CMOS technology. A strong STE Microelectronics IP blocks 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.