A Multi-Kernel Multi-Code Polar Decoder Architecture

A Multi-Kernel Multi-Code Polar Decoder Architecture

Gabriele Coppolino, Carlo Condo, *Member, IEEE*, Guido Masera, *Senior Member, IEEE*, Warren J. Gross, *Senior Member, IEEE*

***Abstract*—Polar codes have received increasing attention in the past decade, and have been selected for the next generation of wireless communication standard. Most research on polar codes has focused on codes constructed from a 2×2 polarization matrix, called binary kernel: codes constructed from binary kernels have code lengths that are bound to powers of 2. A few recent works have proposed construction methods based on multiple kernels of different dimensions, not only binary ones, allowing code lengths different from powers of 2. In this work, we design and implement the first multi-kernel successive cancellation polar code decoder in literature. It can decode any code constructed with binary and ternary kernels: the architecture, sized for a maximum code length N_{max}, is fully flexible in terms of code length, code rate and kernel sequence. The decoder can achieve frequency of more than 1 GHz in 65 nm CMOS technology, and a throughput of 615 Mb/s. The area occupation ranges between 0.11 mm^2 for $N_{max} = 256$ and 2.01 mm^2 for $N_{max} = 4096$. Implementation results show an unprecedented degree of flexibility: with $N_{max} = 4096$, up to 55 code lengths can be decoded with the same hardware, along with any kernel sequence and code rate.**

***Index Terms*—polar codes, multi-kernel, successive-cancellation decoding, hardware implementation.**

I. Introduction

Polar codes are capacity-achieving error correcting codes, characterized by a low-complexity encoding and decoding process [1]. They have been chosen to be adopted in the fifth generation of wireless communication standards (5G) [2], that foresees a variety of scenarios. Thus, coding schemes targeting low latency, low power, and high performance must be devised. Error correction performance and decoding speed are heavily influenced by the polar code block length, and the different scenarios demand a wide range of code lengths.

The majority of current research is focused on polar codes recursively constructed from a 2×2 polarization matrix, also called a binary kernel [1]. The code lengths of polar codes constructed from binary kernels are bound to powers of 2. This is a strong limitation, that is currently overcome with rate-matching schemes [3], [4], whose performance and optimality is hard to evaluate a priori. A few recent works have proposed construction methods based on multiple kernels of different dimensions [5]–[7]. Multi-kernel polar codes can have block lengths different from powers of 2, at the cost of more complex decoding algorithm update rules. In [6], it has been shown that multi-kernel codes can outperform codes of the same length obtained through the application of state-of-the-art puncturing and shortening schemes. At a frame error rate (FER) of almost 10^{-3}, multi-kernel codes yield gains ranging from 0.1 dB to 1.1 dB.

Polar code decoder architectures in literature focus mainly on design-time flexibility [8]–[10], with parametrized designs that can be implemented to decode a particular code. Some decoders guarantee code-rate online flexibility [11]–[14]: while the decoder can decode a single code length, any code rate is supported with the same hardware. The decoder architectures presented in [15], [16] target binary kernels only, and are online flexible in terms of both code rate and code length. However, a different decoding program must be stored for every considered combination of code length and rate, leading to huge area occupation. The unrolled architecture presented in [17] can decode a small set of binary nested code lengths and rates.

In this work, we consider multi-kernel polar codes constructed from binary and ternary (3×3) kernels, and we propose a flexible decoder architecture. The presented design can decode any code constructed from any combination of binary and ternary kernels, up to a maximum code length defined at design time, and any code rate. It is the first multi-kernel decoder in literature, yielding an unmatched degree of flexibility, with up to 55 supported code lengths in the considered case study. Implementation results in 65 nm CMOS technology show an achievable frequency of more that 1 GHz and 615 Mbps coded throughput.

The remainder of the paper is organized as follows. In Section II, we introduce polar codes construction and decoding, while in Section III we show the error-correction performance of some multi-kernel codes. Section IV details the proposed decoder architecture, while implementation results are given in V, together with a comparison with the state of the art. Conclusions are drawn in Section VI.

II. Preliminaries

A. Polar Codes

A polar code $\mathcal{P}(N, K)$ is a linear block code of length N and rate K/N, that relies on a phenomenon called channel polarization [1]. When N tends to infinite, the symmetric capacity of each bit-channel tends towards either 0 or 1, thus identifying very reliable and very unreliable channels.

Let us assume $N = 2^n$, where $n \geq 1$, and let $\mathbf{u} = (u_0, u_1, \ldots, u_{N-1})$ be the N-bit vector input to the encoder.

G. Coppolino and G. Masera are with the Department of Electrical and Telecommunications Engineering, Politecnico di Torino, Torino, Italy. e-mail: gabriele.coppolino@studenti.polito.it, guido.masera@polito.it. C. Condo, and W. J. Gross are with the Department of Electrical and Computer Engineering, McGill University, Montréal, Québec, Canada. e-mail: carlo.condo@mcgill.ca, warren.gross@mcgill.ca.

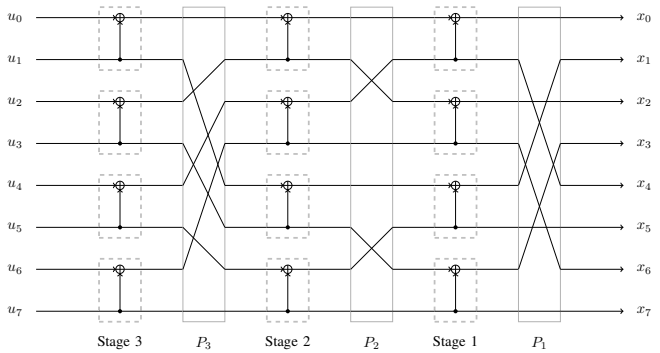
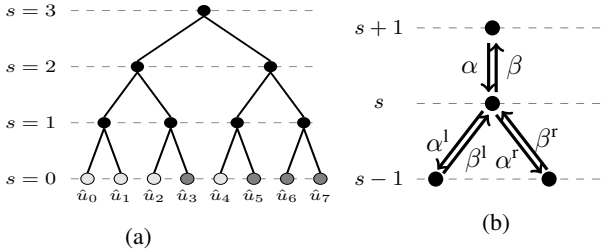


Fig. 1: Tanner graph for a $N = 8$ polar code.



(a) (b)

Fig. 2: (a) Decoding tree for a $\mathcal{P}(8, 4)$ polar code and (b) binary node message passing.

The K information bits are assigned to the K most reliable channels of $\mathbf{u}$, while the remaining $N - K$ are fixed to a known value (usually 0), and are known as frozen bits. The ensemble of their indices is the frozen set $\mathcal{F}$.

The encoding process can be represented through the linear transformation $\mathbf{x} = \mathbf{uG}$, where $\mathbf{G} = \mathbf{T_2}^{\otimes n}$ is the generator matrix, expressed through the *n*-th Kronecker product of the matrix $\mathbf{T_2}$. The matrix $\mathbf{T_2}$ is a binary polarization matrix, or kernel, defined as follows:

$$\mathbf{T_2} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

From the definition of $\mathbf{G}$, the recursive nature of the encoding process can be noticed: a polar code of length N can in fact be obtained as the concatenation of two $N/2$ polar codes. Polar code encoding can also be portrayed through a Tanner graph, as shown in Fig. 1 for an $N = 8$ code. Each stage depicts a Kronecker product, and the dashed boxes represent each $\mathbf{T_2}$ operation. Between neighbouring stages permutations are inserted, in which the bit-indices of the inputs are cyclically rotated to the right by one place [1].

B. Successive Cancellation Decoding

In [1] a first successive cancellation (SC) decoding algorithm has been proposed. It can be represented as a binary search tree where all the nodes must be explored, with priority being given to left branches. An example of a $\mathcal{P}(8, 4)$ polar code SC decoding tree is shown in Fig. 2a: the leaf nodes at stage $s = 0$ can be either information bits (dark gray) or frozen bits (light gray).

Let us call $\mathbf{y} = (y_0, y_1, \ldots, y_{N-1})$ the vector of logarithmic likelihood ratios (LLRs) obtained at the channel output, and $\hat{\mathbf{u}}$ the estimated vector output by the decoder. The decoding starts from the root node, and at each node information is passed from parent to child according to the scheme shown in Fig. 2b. The LLR value α is received and used to compute α^l, then β^l is obtained and used to compute α^r. Once β^r is available, β can be computed. Once a leaf node is reached, the value of $\hat{u}_i$ is estimated. If index $i \in \mathcal{F}$, its value is set to 0, otherwise a hard decision on the sign of α is performed. Calling N_s the length of the polar code at stage s, we can define $\forall i \in \left(0, 1, ..., \frac{N_s}{2} - 1\right)$:

$$\alpha_i^l = 2 \operatorname{arctanh}\left(\tanh\frac{\alpha_i}{2} \cdot \tanh\frac{\alpha_{i+\frac{N_s}{2}}}{2}\right) \simeq \varphi(\alpha_i)\,\varphi\left(\alpha_{i+\frac{N_s}{2}}\right) \min\left(|\alpha_i|, \left|\alpha_{i+\frac{N_s}{2}}\right|\right), \quad (1)$$

$$\alpha_i^r = \left(1 - 2\beta_i^l\right)\alpha_i + \alpha_{i+\frac{N_s}{2}}, \quad (2)$$

$$\left[\beta_i, \beta_{i+\frac{N_s}{2}}\right] = \left[\beta_i^l \oplus \beta_i^r, \beta_i^r\right], \quad (3)$$

where $\oplus$ represents the XOR operation and $\varphi()$ is a function returning the sign of the argument. In (1), both the exact and the approximate (hardware-friendly) computation, proposed in [8], are shown. At leaf nodes, β is initialized as $\hat{u}_i$ (4), where i is the index identifying the current leaf node.

$$\hat{u}_i = \begin{cases} 0 & \text{if } \alpha \geq 0 \text{ or } i \in \mathcal{F} \\ 1 & \text{otherwise} \end{cases} \quad (4)$$

C. Multi-kernel construction

In [6] a generalized construction method for polar codes has been presented: together with $\mathbf{T_2}$, larger kernels have been investigated. Thus, the matrix $\mathbf{G}$ is composed of a series of Kronecker products between kernels of different sizes. Ternary kernels, i.e. kernels of dimensions 3×3, have been considered in [6], where the proposed polarization matrix is

$$\mathbf{T_3} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Fig. 3 portrays the Tanner graph for an $N = 12$ code constructed with a kernel sequence $\mathbf{T_2} \otimes \mathbf{T_3} \otimes \mathbf{T_2}$. As in the binary case, inter-stage permutations are required to reshuffle indices. For each stage $i > 1$, the permutation matrix $\mathbf{P_i}$ can be found as

$$\mathbf{P_i} = (\mathbf{Q_i}|\mathbf{Q_i} + N_{i+1}|\mathbf{Q_i} + 2N_{i+1}|\ldots|\mathbf{Q_i} + (N/N_{i+1})N_{i+1}),$$

where $\mathbf{Q_i}$ is the so-called canonical permutation introduced in [6], $N_i = \prod_{j=1}^{i-1} n_j$, and $n_j \times n_j$ are the dimensions of the j-th kernel of the Kronecker product. Finally, $\mathbf{P_1}$ is computed in order to re-align output indices with those relative to the encoder input, considering all the previous permutations.

Fig.4a shows the SC decoding tree for the same code, and the message passing criterion in case of ternary nodes is shown in Fig. 4b. Defining (1) as f^b, (2) as g^b, and (3) as $comb^b$, for a ternary node at stage s the decoding rules $\forall i \in \left(0, 1, ..., \frac{N_s}{3} - 1\right)$ are:

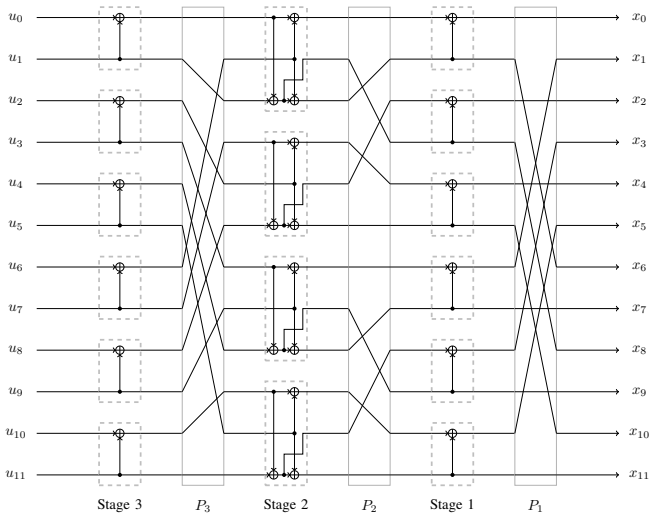
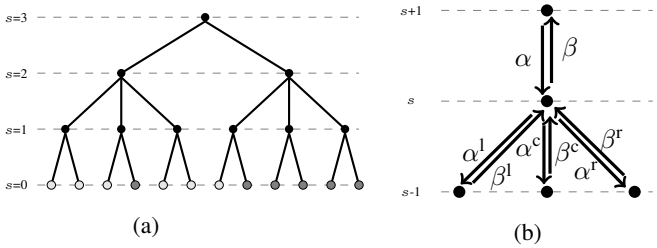


Fig. 3: Tanner graph for a $N = 12$ polar code.



(a) (b)

Fig. 4: (a) Decoding tree for a $\mathcal{P}(12, 6)$ polar code and (b) ternary node message passing.

$$\alpha_i^l = 2\,\mathrm{arctanh}\left(\tanh\frac{\alpha_i}{2}\cdot\tanh\frac{\alpha_{i+\frac{N_s}{3}}}{2}\cdot\tanh\frac{\alpha_{i+\frac{2N_s}{3}}}{2}\right) \simeq \varphi(\alpha_i)\,\varphi\left(\alpha_{i+\frac{N_s}{3}}\right)\varphi\left(\alpha_{i+\frac{2N_s}{3}}\right)\cdot \cdot\min\left(|\alpha_i|,\left|\alpha_{i+\frac{N_s}{3}}\right|,\left|\alpha_{i+\frac{2N_s}{3}}\right|\right) \quad (5)$$

$$\alpha_i^c = \left(1-2\beta_i^l\right)\alpha_i + f^b\left(\alpha_{i+\frac{N_s}{3}},\alpha_{i+\frac{2N_s}{3}}\right), \quad (6)$$

$$\alpha_i^r = \left(1-2\beta_i^l\right)\alpha_{i+\frac{N_s}{3}} + \left(1-2\beta_i^l\oplus\beta_i^c\right)\alpha_{i+\frac{2N_s}{3}}, \quad (7)$$

$$\left[\beta_i,\beta_{i+\frac{N_s}{3}},\beta_{i+\frac{2N_s}{3}}\right] = \left[\beta_i^l\oplus\beta_i^c,\beta_i^l\oplus\beta_i^r,\beta_i^l\oplus\beta_i^c\oplus\beta_i^r\right]. \quad (8)$$

Similarly to the binary kernel case, we define (5), (6), (7) and (8) as f^t, g_1^t, g_2^t and $comb^t$ respectively.

III. Multi-kernel codes

The multi-kernel code construction method proposed in [6] yields substantial error-correction performance gain with respect to puncturing and shortening schemes. Table I reports such gain when two multi-kernel codes are compared to codes obtained with the puncturing method in [18] and the shortening method in [19], for SC decoding and list SC (SCL) [20] with a list size of 8. Depending on the target FER, the gain ranges from 0.1 to 1.1 dB.

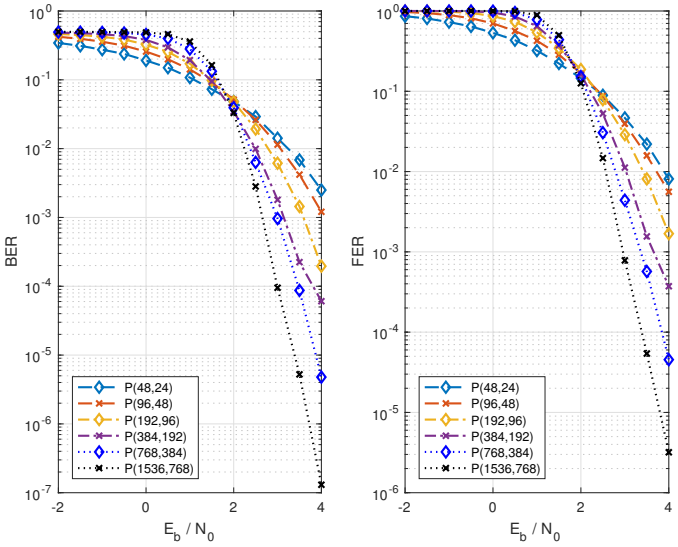


Fig. 5: Error-correction performance of binary-ternary mixed polar codes.

Using the construction method described in [6], multi-kernel codes have been constructed. Their error-correction performance has been simulated through a binary-input additive white Gaussian noise (AWGN) channel with binary phase-shift keying modulation. The bit error rate (BER) and FER curves are shown in Fig. 5, obtained with SC decoding and LLRs represented in double-precision floating-point format. As discussed in [6], [7], the Kronecker product is not commutative, and different kernel orders will results in different codes. However, there is currently no theoretical way to identify the best kernel multiplication order: thus, the different kernel orders need to be simulated to identify the one that gives the best error-correction performance. In the remainder of our work, we considered the following codes and kernel orders, obtained with the method described in [7]:

- $\mathcal{P}(48, 24)$ with $\mathbf{G} = \mathbf{T_3}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}$
- $\mathcal{P}(96, 48)$ with $\mathbf{G} = \mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_3}\otimes\mathbf{T_2}\otimes\mathbf{T_2}$
- $\mathcal{P}(192, 96)$ with $\mathbf{G} = \mathbf{T_3}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}$
- $\mathcal{P}(384, 192)$ with $\mathbf{G} = \mathbf{T_3}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\otimes\mathbf{T_2}$
- $\mathcal{P}(768, 384)$ with $\mathbf{G} = \mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_3}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\otimes\mathbf{T_2}\otimes\mathbf{T_2}$
- $\mathcal{P}(1536, 768)$ with $\mathbf{G} = \mathbf{T_3}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\otimes\mathbf{T_2}\otimes\mathbf{T_2}\otimes\mathbf{T_2}$

IV. Decoder Architecture

We propose a multi-code semi-parallel SC decoder which supports purely-binary, purely-ternary and binary-ternary mixed construction polar codes. The architecture is sized with a maximum code length N_{max}, and can support any code length $N \geq 2$ that can be expressed as a combination of binary and ternary kernels, and any code rate. For mixed polar codes, the architecture can decode codes constructed with any kernel order, without knowledge of the code structure at design time.

TABLE I: Coding gain for multi-kernel codes with respect to shortening and puncturing schemes.

$N = 72$				
	SC		SCL	
FER	10^{-2}	$4 \cdot 10^{-3}$	10^{-2}	$2 \cdot 10^{-3}$
[6] VS puncturing [18]	0.20 dB	0.45 dB	0.20 dB	0.25 dB
[6] VS shortening [19]	0.45 dB	0.70 dB	0.65 dB	1.10 dB
$N = 48$				
	SC		SCL	
FER	10^{-2}	$2 \cdot 10^{-3}$	10^{-2}	$2 \cdot 10^{-3}$
[6] VS puncturing [18]	0.10 dB	0.25 dB	0.35 dB	0.50 dB
[6] VS shortening [19]	0.15 dB	0.35 dB	0.75 dB	0.80 dB

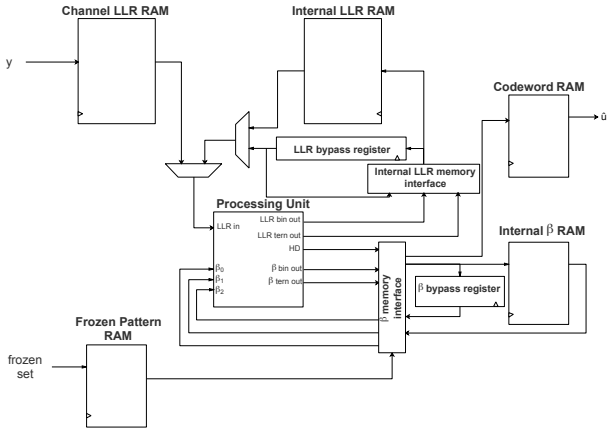


Fig. 6: Datapath of the implemented architecture.

The overall decoder architecture is shown in Figure 6. It relies on P processing elements (PEs) implementing (1)-(8), and dedicated memories for channel and internal LLRs, β values and candidate codeword. Both channel and internal LLRs are represented on Q bits, Q_f of which are assigned to the fractional part.

Together with the code length, the decoder receives as inputs the following parameters:

- information about binary and ternary stages;
- memory address offsets for both LLRs and β values, relative to the current code length;
- number of steps required by each stage to process all inputs given the number P of PEs. This is due to the fact that the decoder has a semi-parallel architecture and, for stages where $N_s > 2P$, the number of PEs is not sufficient to elaborate all data in a single clock cycle.

In order to simplify and reduce both memory accesses and routing, the architecture has been designed for bit-reversed polar codes [8]. This approach allows to dramatically simplify the memory accesses.

A. Data flow

The channel output $\mathbf{y}$ is initially stored in the *Channel LLR RAM*, while the frozen set $\mathcal{F}$ and the code parameters listed in the previous section are uploaded to their dedicated memories, respectively the *Frozen Pattern RAM* and a set of registers. For operations involving soft values, the *Processing Unit* receives as input either the channel or the internal LLRs, according to the current stage of the decoding tree. For $comb$ operations (3)-(8), data read from the *Internal* β *RAM* are used. Results are stored either in the *Internal LLR RAM* or in the *Internal* β *RAM*, according to the performed operation. When a leaf node is reached and a hard decision (HD) is performed to decide the value of a bit (4), the result is stored in the *Codeword RAM*. The decoding phase ends when the bit associated to the rightmost leaf node is estimated: the decoded codeword $\hat{\mathbf{u}}$ is thus output.

B. Processing Unit

The *Processing Unit* (PU) is the computational core of the decoder, where all the operations are performed: f^b (1), g^b (2), $comb^b$ (3), f^t (5), g_1^t (6), g_2^t (7) and $comb^t$ (8). It contains P processing elements (PEs) and P combine blocks (CBs) organized as follows:

- $\frac{2}{3}P = P^{b/t}$ binary-ternary mixed PEs, each of them able to compute any f or g operations, both binary and ternary;
- $\frac{1}{3}P$ binary PEs, which support only f^b and g^b;
- $\frac{2}{3}P = P^{b/t}$ binary-ternary mixed CBs which perform both $comb^b$ and $comb^t$;
- $\frac{1}{3}P$ binary CBs, which support only $comb^b$.

Since it has been observed that between binary and ternary operations there are common computations, mixed PEs are used to increase resource sharing, at the cost of a multiplexing operation; additional purely-binary PEs are used to align the number of used inputs both for binary and ternary operations. Thus the maximum number of elaborated soft inputs is fixed to $2P = 3P^{b/t}$, while the results are either P or $P^{b/t}$ LLRs: in fact it can be noticed that the number of operations simultaneously performed is P in the binary case and $P^{b/t}$ in the ternary one. For binary operations each i-th PE elaborates the $2i$-th and $(2i+1)$-th LLR inputs, while for ternary ones each i-th mixed PE uses LLRs corresponding to indices $3i$, $3i+1$ and $3i+2$. The same holds for CBs. From the last considerations P must be a multiple of 3; an example of PU with $P = 3$ is shown in Fig. 7, where the multiplexers inserted before the PEs are used to align the correct LLRs in case of a binary or ternary stage.

Although there are situations in which not all PEs are performing useful computations, $2P$ inputs are nevertheless elaborated and stored in the corresponding memory. Unnecessary data are subsequently ignored in the final estimation: this happens for stages s where N_s is not a multiple of $2P$. The impact of two different LLR representations on the implementation cost of the PU has been evaluated: we have in fact designed PEs with both 2's complement and sign and magnitude representations. FPGA synthesis results have shown that the sign and magnitude binary PE has 14% lower resource requirements and 20% shorter critical path than the 2's complement one, while the sign and magnitude mixed PE has similar resource requirements and 23% shorter critical path

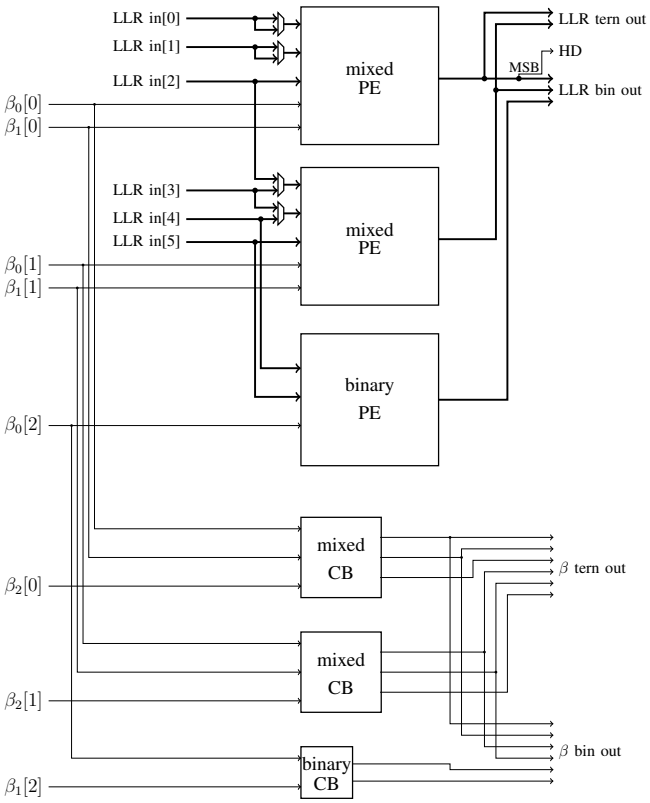


Fig. 7: Example of *Processing Unit* with $P = 3$.

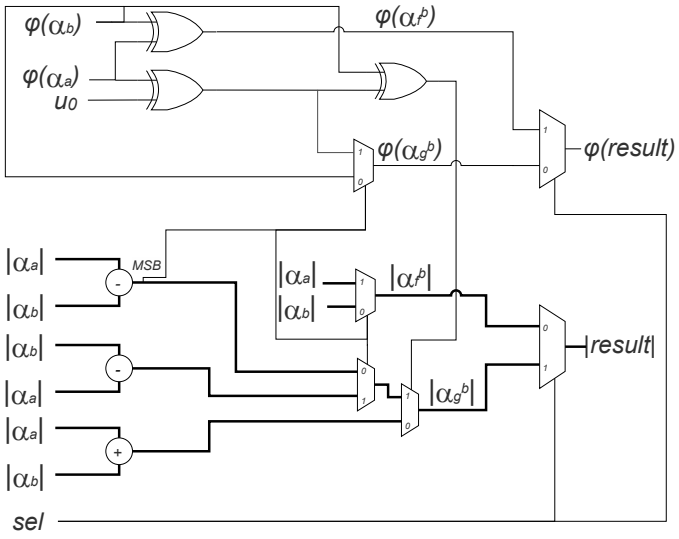


Fig. 8: Datapath of a binary PE.

than the 2's complement one. Thus, all LLRs in the proposed decoder are represented with sign and magnitude.

1) Binary Processing Elements: The architecture of binary PEs is the one proposed in [8]. Let us call α_a and α_b the input LLRs. For the hardware-friendly version of f^b (1) operation the result computation is straightforward:

$$\varphi(\alpha_f^b) = \varphi(\alpha_a) \oplus \varphi(\alpha_b) \ , \tag{9}$$

$$|\alpha_f^b| = \min(|\alpha_a|, |\alpha_b|) \ , \tag{10}$$

where α_f^b is the f^b operation result. Analyzing the complete truth table both for sign $\varphi(\alpha_g^b)$ and magnitude $|\alpha_g^b|$ of g^b (2), its resulting equations are:

$$\varphi(\alpha_g^b) = \overline{\gamma_{ab}} \cdot \varphi(\alpha_b) \ + \ \gamma_{ab} \cdot (u_0 \oplus \varphi(\alpha_a)) \ , \tag{11}$$

$$|\alpha_g^b| = \max(|\alpha_a|, |\alpha_b|) \ + \ (-1)^{\chi} \min(|\alpha_a|, |\alpha_b|) \ , \tag{12}$$

where

$$\gamma_{ab} = \begin{cases} 1 & \text{if } |\alpha_a| > |\alpha_b| \ , \\ 0 & \text{otherwise} \ , \end{cases} \tag{13}$$

$$\chi = u_0 \oplus \varphi(\alpha_a) \oplus \varphi(\alpha_b) \ . \tag{14}$$

This architecture is shown in Figure 8. The rightmost multiplexers select the output values depending on the selected operation, while the internal multiplexers are used to select the correct result of the maximum and minimum identification, of $|\alpha_g^b|$, and of $\varphi(\alpha_g^b)$, based on the comparison between input LLRs and computed partial results. Adders and subtractors saturate their result if outside the available range.

2) Binary-ternary mixed Processing Elements: An analysis analogous to the binary case has been conducted on f^t (5), g_1^t (6) and g_2^t (7). The resulting equations are the following:

$$\varphi(\alpha_f^t) = \varphi(\alpha_a) \oplus \varphi(\alpha_b) \oplus \varphi(\alpha_c) \ , \tag{15}$$

$$|\alpha_f^t| = \min(|\alpha_a|, |\alpha_b|, |\alpha_c|) \ , \tag{16}$$

$$\varphi(\alpha_{g_1}^t) = \overline{\gamma_{g_1}} \cdot (\varphi(\alpha_b) \oplus \varphi(\alpha_c)) \ + \ \gamma_{g_1} \cdot (u_0 \oplus \varphi(\alpha_a)) \ , \tag{17}$$

$$|\alpha_{g_1}^t| = \max(|\alpha_a|, \min(|\alpha_b|, |\alpha_c|)) + \\ + \ (-1)^{\chi_{g1}} \min(|\alpha_a|, \min(|\alpha_b|, \alpha_c|)) \ , \tag{18}$$

$$\varphi(\alpha_{g_2}^t) = \overline{\gamma_{g_2}} \cdot (u_0 \oplus u_1 \oplus \varphi(\alpha_c)) + \gamma_{g_2} \cdot (u_0 \oplus \varphi(\alpha_b)) \ , \tag{19}$$

$$|\alpha_{g_2}^t| = \max(|\alpha_b|, |\alpha_c|) \ + \ (-1)^{\chi_{g2}} \min(|\alpha_b|, |\alpha_c|) \ , \tag{20}$$

where

$$\gamma_{g_1} = \begin{cases} 1 & \text{if } |\alpha_a| > \min(|\alpha_b|, |\alpha_c|) \\ 0 & \text{otherwise} \end{cases} \ , \tag{21}$$

$$\chi_{g1} = u_0 \oplus \varphi(\alpha_a) \oplus \varphi(\alpha_b) \oplus \varphi(\alpha_c) \ , \tag{22}$$

$$\gamma_{g_2} = \begin{cases} 1 & \text{if } |\alpha_b| > |\alpha_c| \\ 0 & \text{otherwise} \end{cases} \ , \tag{23}$$

$$\chi_{g2} = u_1 \oplus \varphi(\alpha_b) \oplus \varphi(\alpha_c) \ . \tag{24}$$

The circuit implementing these operations is shown in Figure 9, where again adders and subtractors can saturate the result. The multiplexers have the same role of those shown in Fig. 8: their number increases due to the higher number of input LLRs, computations to be performed, higher number of results to be computed, and concurrent binary/ternary datapath. The M block is a combination of pruned multiplexers selecting the minimum absolute value according to the already computed selection signals, which correspond to the most significant bits of the output of the subtractors.

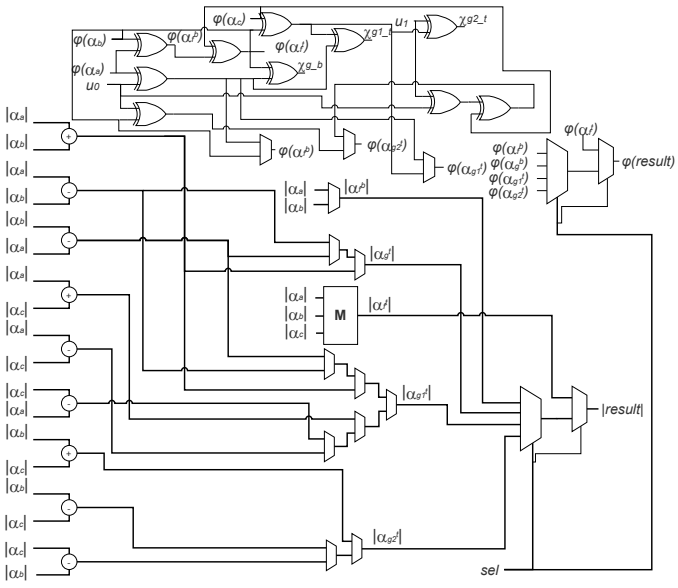


Fig. 9: Datapath of a binary-ternary mixed PE.

Mixed PEs perform both binary and ternary operations, and need to select their input accordingly. Thus, LLR multiplexing logic is inserted at their input. This logic consists of two Q-bit multiplexers for each mixed PE.

3) Combine blocks: Both binary and binary-ternary mixed CBs are composed of XOR gates implementing $comb^b$ and $(comb^b)\overline{sel} + (comb^t)sel$ respectively, where sel is the binary/ternary selector.

C. Memory system

While efficient in terms of resource usage, register-based approaches like [11] lead to excessive area occupation. Thus, this design foresees the usage of SRAM banks. The width of these memories is different from that of memories in a purely-binary decoder design, since they have to accommodate ternary operations and their concurrent input and output volume. Additionally, for *Internal LLR RAM* a three-bank solution has been implemented, since ternary-kernel functions are supported: for purely-binary decoders two banks would have been sufficient.

1) Channel LLR RAM: This memory stores the LLRs coming from the channel. Each memory word is $2P \cdot Q$ long, since for each operation involving LLRs $2P$ of them are required by the PU. Its depth is $D_{LLR\,ch} = \lceil \frac{N_{max}}{2P} \rceil$. This memory uses two separate ports, one for reading and one for writing.

2) Internal LLR RAM: It contains the partial results of f and g operations. Similarly to the *Channel LLR RAM*, the parallelism must be $2P \cdot Q$. The computation of the depth $D_{LLR\,int}$ takes into account that for each decoding stage only one LLR vector must be stored: once the node which took as input the computed LLR has generated its output β, that soft value will be no longer used and can be overwritten. In addition, for stage $s = 0$ it is not needed to memorize the result since the hard decision is performed in the same clock cycle.

The memory depth is computed as:

$$D_{LLR\,int} = \sum_{s=1}^{\log_2(N_{max})-1} \left\lceil \frac{N_{max}}{2^s \cdot 2P} \right\rceil .$$

Also for this memory two separate ports for reading and writing are required.

It is possible to rearrange the *Internal LLR RAM* with a bank structure. However, due to the variable number of data that needs to be written, depending on the stage being binary or ternary, four banks with two different widths should be implemented. This would incur significant control and addressing overhead, with no tangible advantage with respect to the proposed structure. More details on the handling of different result sizes are given in Section IV-D.

3) Internal β RAM: This memory stores all β values computed inside the decoding tree; it is organized in three banks, which share the same input writing bus:

- BANK0 for β_0: it is equal to β^l in both binary and ternary cases;
- BANK1 for β_1: it is equal to β^r for binary stages, while for ternary ones it represents β^c;
- BANK2 for β_2: it corresponds to the ternary stages β^r.

The bank organization is fundamental for parallel data reading in g_2^t, $comb^b$ and $comb^t$ operations. Each bank has a width of $2P$ since results of $comb$ operations are on $2P$ bits, while their depths $D_{\beta\,int}$ are equal to:

$$D_{\beta\,int} = \sum_{s=0}^{\log_2(N_{max})-1} \left\lceil \frac{N_{max}}{2^s \cdot 2P} \right\rceil .$$

4) Codeword RAM: It is used to store the decoder output $\hat{\mathbf{u}}$, composed by the HDs performed at the leaf nodes. Its width W_{cod} is a design choice independent from all other parameters, while the depth is

$$D_{cod} = \left\lceil \frac{N_{max}}{W_{cod}} \right\rceil .$$

5) Frozen Pattern RAM: It stores the frozen set, where each of N_{max} bits identifies if the corresponding bit-channel is frozen or not. The memory width W_{frozen} is an independent design choice, while the depth can be expressed as

$$D_{frozen} = \left\lceil \frac{N_{max}}{W_{frozen}} \right\rceil .$$

Table II reports the breakdown of the memory requirements for the proposed decoder with various N_{max}, P and Q combinations. To correctly evaluate the memory overhead brought by the multi-kernel approach, the memory sizes for purely binary polar decoders with similar parameters have been detailed as well. It can be seen that most of the additional memory bits can be found in the internal β memory.

D. Memory interfaces

Two interfacing modules are required to adapt the inherent parallelism of the memories to that of the PU.

TABLE II: Memory requirements for various decoder parameters, considering both a multi-kernel (MK) and a purely binary (PB) approach.

	MK	PB	MK	PB	MK	PB
N_{max}	4096	4096	1024	1024	256	256
P	120	128	60	64	18	16
Q	7	7	6	6	5	5
	[bit]	[bit]	[bit]	[bit]	[bit]	[bit]
Channel LLR RAM	30240	28672	6480	6144	1440	1280
Internal LLR RAM	43680	39424	11520	9984	1980	1760
Internal β RAM	31680	19456	9000	5376	2052	1216
Codeword RAM	4096	4096	1024	1024	256	256
Frozen Pattern RAM	4096	4096	1024	1024	256	256
Total	113792	95744	29048	23552	5984	4768

1) Internal LLR memory interface: Fig. 10 shows the interface circuit. It is tasked with choosing, during write operations, which part of the memorized word has to be overwritten. In fact, the results of f and g operations are P or $P^{b/t}$ LLR, for binary and ternary cases respectively, while the width of the LLR memories is $2P = 3P^{b/t}$. Each memory location takes two or three clock cycles to be overwritten with useful data. So, at tree stages where $N_s > 2P$ and the PU takes more than one clock cycle to process them, the following steps are performed:

- For binary stages:
  1) The $2i$-th operation result (P LLRs) is stored in the memory together with QP^b appended zeros;
  2) The $(2i+1)$-th operation result is stored after the P most significant bits of the previously written word, so that the padding zeros are overwritten and the new stored word contains the P results of both the $2i$-th and $(2i+1)$-th operations.
- For ternary stages:
  1) The $3i$-th operation result ($P^{b/t}$ LLRs) is stored in the memory together with $2QP^{b/t}$ appended zeros.
  2) The $(3i+1)$-th operation result is stored after the $QP^{b/t}$ most significant bits of the previously written word. The new word contains the $P^{b/t}$ results of both the $3i$-th and $(3i+1)$-th operations;
  3) The $(3i+2)$-th operation result is stored after the previously written $2QP^{b/t}$ bits, completing the $3P^{b/t} = 2P$ LLR word.

To overwrite only parts of the previously written word, the bypass buffer output is used. When $N_s \leq 2P$, the results are stored in the first part of the word as usual; the remaining bits are not considered in subsequent operations.

2) β memory interface: Figure 11 shows the interface architecture. It is used both for reading and writing from the *Internal β RAM*:

- Reading: operations involving β values need either P or $P^{b/t}$ bits per bank as input, while each word is composed of $2P$ bits. Thus, the relevant word parts are selected according to the actual number of elaborated LLRs for that node.

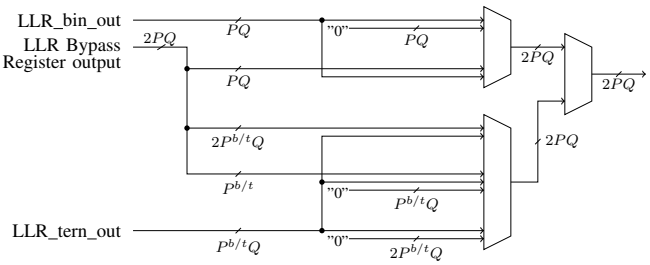


Fig. 10: *Internal LLR RAM* interface circuit.

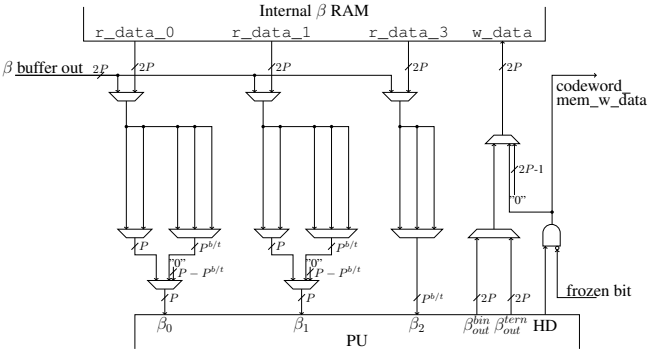


Fig. 11: β memory interface circuit, where `r_data_0`, `r_data_1` and `r_data_2` are the outputs of `bank0`, `bank1` and `bank2` respectively.

- Writing: the data is selected between the CB results and the HD for the leaf nodes.

E. Bypass registers

Two bypass registers must be used since the memory system is RAM-based and, if a result is computed and ready to be stored at the j-th clock cycle, it can be correctly read only from the $(j+2)$-th cycle onwards, to avoid incurring conflicts. So, for all the nodes at stage $s \leq \log_2 2P$, bypass registers allow reading newly computed data already at the following clock cycle. A $2QP$-bit register is used for the *Internal LLR RAM*, while a second $2P$-bit register is necessary for the *Internal β RAM*.

F. Control Unit

The *Control Unit* provides all the memory addresses to the memories and control signals to the datapath. It has been designed as several hierarchically controlled finite state machines. The decoding process follows the same approach of the tree exploration by means of different counters, which keep track of the status and of the number of visited leafs. The decoding process ends when a number of leafs equal to the code length has been visited.

G. Multi-code support

Memories are sized for a maximum code length N_{max}, but any code length $N \leq N_{max}$, with N a multiple of 2 or 3 is supported. Memory requirements are upper bounded by the largest combination of $\mathbf{T_2}$ kernels leading to N_{max}, since a

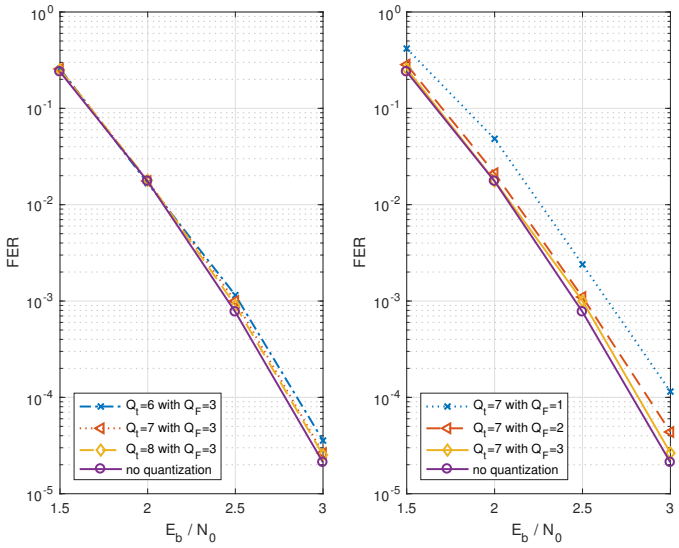


Fig. 12: Error-correction performance of a $\mathcal{P}(4096, 2048)$ with various Q and Q_f values.

higher number of stages are present in the decoding tree than in a mixed-kernel polar code with similar code length. The input code parameters allow to know when the leaf node stage is reached, and thus when the tree ascension has to start. The status counter in the CU uses foreknowledge of the number of kernels and their dimension to schedule the right operation at each stage: thus, any code rate and kernel order can be decoded without any change to the hardware. The total amount of bits required to store the code parameters for a code of length N is $\lceil \log_2 N \rceil + s_m \left(2 + 2 \left\lceil \log_2 \frac{N}{2P} \right\rceil\right)$, where s_m is the number of kernel composing the code. The PU has been designed independently of the code length.

V. Implementation Results

The decoder architecture illustrated in the previous Section has been described in VHDL, verified with ModelSim, and synthesized with Cadence RTL Compiler on TSMC 65nm CMOS technology node.

The choice of the number of LLR quantization bits Q influences a substantial part of the computational hardware and memory width. In Figure 12 the error-correcting performance of a $\mathcal{P}(4096, 2048)$ polar code is shown: between $Q = 7$ and $Q = 8$ curves there is not a significant difference, while choosing $Q = 6$ leads to larger error figures with respect to floating point precision. Although the number of fractional bits Q_f does not influence the hardware architecture, a high Q_f requires a higher Q. In Figure 12 we can notice that $Q_f = 3$ yields only minor FER degradation. Thus, for $N_{max} = 4096$ we chose $Q = 7$ and $Q_f = 3$. Similar studies were performed in case of $N_{max} = 1024$ and $N_{max} = 256$, leading to $Q = 6$, $Q_f = 3$ in the first case and to $Q = 5$, $Q_f = 2$ in the second.

Table III reports synthesis results for three sets of decoder parameters. Along with the parameters, the number of supported code lengths $\mathcal{N}$ and the maximum achievable frequency f_{max} are shown. All implementations can run at more than one GHz. The A^{reg} is the area occupation when all memories are synthesized as registers, while in A^{RAM} all the memories are implemented as SRAM. For both estimations the logic and memory cells area percentages are shown.

TABLE III: ASIC implementation results for TSMC 65nm CMOS technology, with number of supported code lengths $\mathcal{N}$, maximum frequency f_{max}, register-based area occupation A^{reg} and RAM-based area occupation A^{RAM}.

N_{max}		4096	1024	256
P		120	60	18
Q		7	6	5
$\mathcal{N}$		55	40	27
f_{max}	[GHz]	1.06	1.11	1.23
A^{reg}	[mm^2]	2.63	0.62	0.14
Combinational	[%]	45.0	38.9	40.3
Sequential	[%]	55.0	61.1	59.7
A^{RAM}	[mm^2]	2.01	0.46	0.11
Combinational	[%]	58.9	56.7	55.2
Registers	[%]	28.6	28.9	31.2
RAM	[%]	12.5	14.4	13.6

The latency of the decoding phase depends on the number P of PEs, on the number of kernels s_m, on the kernels dimension and their order.

The decoding latency, measured in clock cycles (CCs) can be computed as:

$$\mathcal{L} = \sum_{s=1}^{s_m} \left\lceil \frac{N_s}{2P} \right\rceil \left((n_s + 1) \frac{N}{N_s} - 1 \right) . \tag{25}$$

In Table IV some polar code timing performance are shown, where $\mathcal{L}$ is the decoding latency, f is the achievable frequency, and T the coded throughput. They consider a wide range of code parameters over three different decoder implementations. Since the kernel order impacts the decoding latency, dimension of each kernel has been reported, from left to right as in the Kronecker product. It is possible to see that the achievable frequency is consistently above 1 GHz, and that the coded throughput ranges from 350 to 615 Mbps.

In Table V the implementation results of the proposed decoder have been compared to rate-flexible purely binary decoders in the state of the art, since to the best of our knowledge this is the first multi-kernel decoder in literature. All decoders have been implemented with 65 nm CMOS technology, and target a code with $N = 1024$, that for our work corresponds to N_{max} as well. Both in [8] and [10] semi-parallel architectures are proposed, supporting the SC algorithm and a single fixed code length. The reported results for [10] refer to their best devised architecture, called folded high performance partial sum network. It limits the number of processing elements by folding highly parallel operations and performing them in several clock cycles, thus increasing hardware utilization. Observing the bit-per-cycle (bpc) throughput in Table V, it can be noticed that both [8] and [10] outperform the proposed decoder for the considered purely binary codes. The reason can be found in the additional clock cycles required for *comb* operations in our architecture:

TABLE IV: Latency $\mathcal{L}$ and coded throughput T of various polar codes with three different decoder implementations.

CODE PARAMETERS	DECODER PARAMETERS	$\mathcal{L}$ [CCs]	f [GHz]	T [bpc]	T [Mbps]
{2,3,2,2,2,3,3,3,3} $N = 3888$	$N_{max} = 4096$ P=120 Q=7	7965	1.06	0.49	519.4
{2,3,3,2,3,3,3,3,3} $N = 2916$	$N_{max} = 4096$ P=120 Q=7	5953	1.06	0.49	519.4
{2,2,2,2,2,2,3,3,3} $N = 1728$	$N_{max} = 4096$ P=120 Q=7	3548	1.06	0.49	519.4
{3,2,2,2,2,2,2,2,2,2} $N = 1536$	$N_{max} = 4096$ P=120 Q=7	4663	1.06	0.33	350.6
{2,2,3,2,2,2,2,2,2} $N = 768$	$N_{max} = 1024$ P=60 Q=6	2326	1.11	0.33	366.5
{2,2,2,2,2,2,3,3} $N = 576$	$N_{max} = 1024$ P=60 Q=6	1234	1.11	0.47	521.7
{3,2,2,2,2,2,2,2,2} $N = 384$	$N_{max} = 1024$ P=60 Q=6	1156	1.11	0.33	368.7
{2,2,3,3,3,3,3} $N = 324$	$N_{max} = 1024$ P=60 Q=6	652	1.11	0.50	555.0
{3,3,3,3,3} $N = 243$	$N_{max} = 256$ P=18 Q=5	519	1.23	0.47	578.1
{3,2,2,2,2,2,2} $N = 192$	$N_{max} = 256$ P=18 Q=5	587	1.23	0.32	402.3
{2,2,2,3,2,2} $N = 96$	$N_{max} = 256$ P=18 Q=5	272	1.23	0.35	434.1
{3,3,3,3} $N = 81$	$N_{max} = 256$ P=18 Q=5	162	1.23	0.50	615.0
{3,2,2,2,2} $N = 48$	$N_{max} = 256$ P=18 Q=5	137	1.23	0.35	430.9

since different kernel orders are supported, the sequence of (3) and (8) is not always the same. Thus, it is not possible to hardwire an XOR tree to compute the $comb$ at all stages in one clock cycle, like in decoders supporting only binary kernels: separate clock cycles are spent to perform the $comb$ operations according to the correct kernel order. On the other hand, [8] and [10] consider only binary kernels and, implementing a tree of $comb$ operations and eventually selecting a partial result, β values are computed in the same clock cycle immediately after the g. This is not affecting the critical path in a significant way since only few XOR gates are added. As shown in Table IV, codes constructed with higher-dimension kernels yield a higher throughput. When decoding a ternary node, due to the higher utilization factor of the PEs and the higher number of useful computations in each clock cycle, the number of clock cycles needed to decode a codeword is lower. Moreover, latency-reduction techniques like the ones presented in [9], [21] can be easily adapted to the proposed architecture.

The proposed decoder yields a higher area occupation than both [8] and [10]. This is mainly due to the higher quantization parameter Q and to the support to ternary functions. Mixed PEs require $\times 2.57$ LUTs on FPGA and $\times 2.10$ area occupation with respect to the purely binary ones. However, our decoder is completely code-length flexible and supports multiple kernel sizes, any code rate and any kernel order. Moreover, it can achieve the highest frequency among the considered works, and a higher throughput in Mbps than [8].

Semi-parallel SC-based decoders in literature, while supporting only binary kernels and often being designed targeting a single code, share the basic multi-PE structure of our work. For the sake of completeness, in Table V we consider also [13] and [17]. These architectures are very different from semi-parallel decoders, but guarantee a certain degree of flexibility. The decoder in [17] can decode a fixed set of combinations of code lengths and code rates, while the architecture proposed in [13] is rate-flexible. Both architectures are able to achieve a higher throughput than the proposed decoder, at the cost of larger area occupation and a lower degree of flexibility.

TABLE V: Comparison with the state of the art, $N = 1024$ polar codes, coded throughput T, area A.

DECODER	P	Q	f [GHz]	T [bpc]	T	A [mm^2]
This work	60	6	1.11	0.33	361.98 Mbps	0.46
[8]	64	5	0.50	0.49	246.10 Mbps	0.31
[10]	64	5	1.01	0.49	497.28 Mbps	0.07
[13]	–	5	0.0025	1418	3.54 Gbps	1.68
[17]	–	5	0.65	39.4	25.60 Gbps	1.44

VI. Conclusion

In this work, we have proposed the first polar code decoder architecture supporting kernels of different sizes. It implements the successive cancellation algorithm, and can support any code rate, any sequence of binary and ternary kernels and any code length $N \leq N_{max}$ that can be expressed as a combination of binary and ternary kernels. The decoder can achieve a frequency of more than a GHz in 65 nm CMOS technology, and a throughput of 615 Mb/s. The area occupation ranges between 0.11 mm^2 for $N_{max} = 256$ and 2.01 mm^2 for $N_{max} = 4096$. Implementation results show an unprecedented degree of flexibility: with $N_{max} = 4096$, up to 55 code lengths can be decoded with the same hardware, along with any kernel sequence and code rate.

References

[1] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, July 2009.

[2] "Final report of 3GPP TSG RAN WG1 #87 v1.0.0," http://www.3gpp.org/ftp/tsg_ran/WG1_RL1/TSGR1_87/Report/Final_Minutes_report_RAN1%2387_v100.zip, Reno, USA, November 2016.

[3] L. Chandesris, V. Savin, and D. Declercq, "On puncturing strategies for polar codes," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, May 2017, pp. 766–771.

[4] V. Bioglio, F. Gabry, and I. Land, "Low-complexity puncturing and shortening of polar codes," *CoRR*, vol. abs/1701.06458, 2017. [Online]. Available: http://arxiv.org/abs/1701.06458

[5] N. Presman, O. Shapira, S. Litsyn, T. Etzion, and A. Vardy, "Binary polarization kernels from code decompositions," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2227–2239, May 2015.

[6] F. Gabry, V. Bioglio, I. Land, and J. C. Belfiore, "Multi-kernel construction of polar codes," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, May 2017, pp. 761–765.

[7] V. Bioglio, F. Gabry, I. Land, and J. Belfiore, "Minimum-distance based construction of multi-kernel polar codes," *CoRR*, vol. abs/1701.07616, 2017. [Online]. Available: http://arxiv.org/abs/1701.07616

[8] C. Leroux, A. Raymond, G. Sarkis, and W. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Trans. Signal Process.*, vol. 61, no. 2, pp. 289–299, January 2013.

[9] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation polar decoder architectures using 2-bit decoding," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 4, pp. 1241–1254, April 2014.

[10] Y. Fan and C. y. Tsui, "An efficient partial-sum network architecture for semi-parallel polar codes decoder implementation," *IEEE Transactions on Signal Processing*, vol. 62, no. 12, pp. 3165–3179, June 2014.

[11] S. A. Hashemi, C. Condo, and W. J. Gross, "A fast polar code list decoder architecture based on sphere decoding," *IEEE Trans. Circuits Syst. I*, vol. 63, no. 12, pp. 2368–2380, December 2016.

[12] ——, "Fast and flexible successive-cancellation list decoders for polar codes," *IEEE Transactions on Signal Processing*, vol. PP, no. 99, pp. 1–14, 2017.

[13] O. Dizdar and E. Arikan, "A hihg-throughput energy-efficient implementation of successive cancellation decoder for polar codes using combinational logic," *IEEE Trans. Circuits Syst. I*, vol. 63, no. 3, pp. 436–447, March 2016.

[14] T. Che, J. Xu, and G. Choi, "Tc: Throughput centric successive cancellation decoder hardware implementation for polar codes," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2016, pp. 991–995.

[15] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 946–957, May 2014.

[16] F. Ercan, C. Condo, and W. J. Gross, "Reduced-memory high-throughput fast-SSC polar code decoder architecture," in *2017 IEEE International Workshop on Signal Processing Systems (SiPS 2017)*, to appear 2017.

[17] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "Multi-mode unrolled architectures for polar decoders," *IEEE Trans. Circuits Syst. I*, vol. 63, no. 9, pp. 1443–1453, September 2016.

[18] K. Niu, K. Chen, and J. R. Lin, "Beyond turbo codes: Rate-compatible punctured polar codes," in *2013 IEEE International Conference on Communications (ICC)*, June 2013, pp. 3423–3427.

[19] R. Wang and R. Liu, "A novel puncturing scheme for polar codes," *IEEE Communications Letters*, vol. 18, no. 12, pp. 2081–2084, Dec 2014.

[20] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.

[21] C. Zhang, B. Yuan, and K. K. Parhi, "Reduced-latency SC polar decoder architectures," in *2012 IEEE International Conference on Communications (ICC)*, June 2012, pp. 3471–3475.