Efficient Bit-Channel Reliability Computation for Multi-Mode Polar Code Encoders and Decoders

Carlo Condo, Seyyed Ali Hashemi, Warren J. Gross

Abstract—Polar codes are a family of capacity-achieving error-correcting codes, and they have been selected as part of the next generation wireless communication standard. Each polar code bit-channel is assigned a reliability value, used to determine which bits transmit information and which parity. Relative reliabilities need to be known by both encoders and decoders: in case of multi-mode systems, where multiple code lengths and code rates are supported, the storage of relative reliabilities can lead to high implementation complexity. In this work, observe patterns among code reliabilities. We propose an approximate computation technique to easily represent the reliabilities of multiple codes, through a limited set of variables and update rules. The proposed method allows to tune the trade-off between reliability accuracy and implementation complexity. An approximate computation architecture for encoders and decoders is designed and implemented, showing 50.7% less area occupation than storage-based solutions, with less than 0.05 dB error correction performance degradation.

I. INTRODUCTION

Polar codes are a class of error-correcting codes proposed in [1], that can achieve capacity with a low-complexity encoding and decoding. Their construction exploits the channel polarization effect: this means that some of the channels through which codeword bits are transmitted (called bit-channels) are more reliable than others. Information bits are transmitted through the most reliable bit-channels, while the least reliable are set to a fixed value (frozen bits): the relative order of reliabilities is dependent on the code length and on the signal-to-noise ratio (SNR) for which the code has been constructed.

The number and position of information bits in a polar code-word needs to be known by both the encoder and the decoder. For encoders, information bits need to be correctly interleaved with frozen bits before encoding, and frozen bits need to be re-set halfway through systematic encoding [2]. Decoders targeting any decoding algorithm need to be aware of the bit arrangement as well [1, 3-6]. Hardware implementations of encoders and decoders usually consider the frozen-information bit pattern as an input, and thus do not evaluate its storage or calculation cost. Many implementations of encoders and decoders target a single or a limited number of combinations of code lengths and rates, and a single SNR [3, 7-10]: thus, it is possible to store the frozen-information bit pattern for each supported code. However, practical applications demand the support of a possibly large number of code lengths and rates, and various SNRs. Multi-mode decoders, and thus encoders, need to grant an even higher degree of flexibility than what can currently be achieved [11]. Within this framework, the direct storage of the bit pattern for each supported case can lead to unbearable implementation costs.

A few recent works address the problem of easy construction of polar code relative reliabilities. Partial orders in the reliability of polar code bit-channels were noticed in [12], while a theoretical framework based on $\beta$-expansion for fast polar code construction has been proposed in [13]. While these approaches greatly reduce the computation complexity of the relative reliabilities, the direct implementation cost is still very high.

In this work, we propose an approximate method to compute the relative reliability of polar codes, that can be implemented with considerably lower complexity than the direct storage of all values, along with a flexible architecture that can be used in both encoders and decoders. The trade-off between implementation complexity and degree of approximation can be tuned according to the application constraints.

The rest of the paper is structured as follows. Section II briefly introduces polar codes. Section III describes the observed patterns in reliabilities and proposes the approximate computation method. Section IV details a case study and evaluates the impact of various approximations on the error-correction performance, whereas Section V details an architecture for the implementation of the proposed technique, and provides implementation results. Finally, Section VI draws the conclusions.

II. POLAR CODES

A polar code $P(N,K)$ is a linear block code of length $N = 2^n$ and rate $K/N$. It is constructed as the concatenation of two polar codes of length $N/2$, and can be expressed as the matrix multiplication

$$x = uG^{\otimes n}$$

(1)

where $u = \{u_0, u_1, \ldots, u_{N-1}\}$ is the input vector, $x = \{x_0, x_1, \ldots, x_{N-1}\}$ is the codeword, and the generator matrix $G^{\otimes n}$ is the $n$-th Kronecker product of the polarizing matrix $G = [1 \ 0]$. The polar code structure allows to sort the $N$-bit input vector $u$ according to the reliability of the bit-channels. The reliabilities associated with the bit-channels can be determined either by using the Bhattacharyya parameter [1], or through the direct use of probability function [14]. The $K$ information bits are assigned to the most reliable bit-channels of $u$, while the remaining $N-K$ (frozen bits) are set to a predetermined value, usually 0. Codeword $x$ is transmitted through the channel, and the decoder receives the Logarithmic Likelihood Ratio (LLR) vector $y = \{y_0, y_1, \ldots, y_{N-1}\}$. 

The encoding process in Equation (1) can be represented as in Fig. 1 that shows a polar code encoding example for \( \mathcal{P}(8, 4) \) where the frozen bits set \( \mathcal{F} \) contains \( \{u_0, u_1, u_2, u_4\} \).

Polar codes have been defined in [1] together with the successive cancellation (SC) decoder: SC-based decoding process can be represented as a full binary tree search, in which the tree is explored depth first, with priority to the left branches. Fig. 2 shows an example of SC decoding tree for \( \mathcal{P}(16, 8) \), where nodes at stage \( s \) contain \( 2^s \) bits. White leaf nodes are frozen bits, while black leaf nodes are information bits.

The message passing criteria among tree nodes is detailed in Fig. 3. LLR values \( \alpha \) are sent from parents to children, while \( \beta \) is computed as

\[
\beta_i = \begin{cases} 
\beta_i^l \oplus \beta_i^r, & \text{if } i < 2^{s-1} \\
\beta_i^{s-2}, & \text{otherwise,}
\end{cases}
\]

where \( \oplus \) is the bitwise XOR. Due to data dependencies, SC computations need to follow a particular schedule. Every node receives \( \alpha \) first, then computes \( \alpha^l \), receives \( \beta^l \), computes \( \alpha^l \), receives \( \beta^r \), and finally sends back \( \beta \). At leaf nodes, \( \beta_i \) is set as the estimated bit \( \hat{u}_i \):

\[
\hat{u}_i = \begin{cases} 
0, & \text{if } i \in \mathcal{F} \text{ or } \alpha_i \geq 0, \\
1, & \text{otherwise.}
\end{cases}
\]

SC decoding suffers from mediocre error correction performance when applied to moderate and short code lengths. The SC-list (SCL) algorithm described in [4] improves the error correction performance by storing a set of \( L \) codeword candidates, that gets updated after every bit estimation.

III. APPROXIMATE RELIABILITY COMPUTATION

Given a polar code \( \mathcal{P}(N, K) \), let us define the reliability vector \( p \) as the N-length sequence of elements \( p_i \), where \( 0 \leq i < N \). Each element \( p_i \) represents the reliability of bit-channel \( i \), where \( p_i = N - 1 \) is the least reliable bit and \( p_i = 0 \) is the most reliable bit. Index \( i = 0 \) refers to the leftmost bit on the decoding tree, while \( i = N - 1 \) refers to the rightmost.

It is possible to identify regular patterns in the reliability vectors of polar codes, both within the same \( p \) and among vectors constructed for polar codes with different lengths. These patterns allow to efficiently describe \( p \) through variables, update rules and scaling.

In this section, we describe some of the patterns that we have identified and used to derive a hardware-efficient approximate reliability computation technique. We focused on codes constructed for the AWGN channel with the method used in [13], targeting an SNR of around 6 dB. However, the proposed method can be easily extended to the reliabilities of codes constructed for other SNRs.

We define here some variables that are going to be useful to explain the proposed method. Let us divide the reliability vector \( p \) in two halves, \( p_L \) and \( p_H \), where \( p_L \) contains all \( p_i \) for \( 0 \leq i < N/2 \), and \( p_H \) the other ones. We call a reliability byte \( p^8 \) a series of eight \( p_i \) belonging to \( p_L \), and \( p^8_H \) one belonging to \( p_H \). To identify different reliability bytes, we assign an additional subscript to \( p^8_L \) and \( p^8_H \), as in \( p^8_{B1L} \) and \( p^8_{E3H} \).

A. Intra-code reliability patterns

Observing \( p_L \) from \( i = 0 \), it is possible to see how, generally, \( p_i \) tends to decrease (i.e. becomes more reliable) as \( i \) increases: this is because the first bits tend to be the least reliable of the code. In the same way, in \( p_H \), starting at \( i = N - 1 \), \( p_i \) tends to increase as \( i \) decreases, since the last bits are usually the most reliable.
Both $p_L$ and $p_H$ can be expressed as a series of variables associated to update rules. As an example, let us consider $p$ for $N = 8$:

$$p = \{7, 6, 5, 3, 4, 2, 1, 0\}$$

where $p_L = \{7, 6, 5, 3\}$ and $p_H = \{4, 2, 1, 0\}$. We can write these vectors as, for example:

$$p_L = \{N - 1, \ N - 2, \ N - 3, \ ENDL\},$$

$$p_H = \{\ENDH, \ H + 2, \ H + 1, \ \ H\}.$$

As the code length increases, the regularity of the reliability decreases, and either a higher number of variables are needed to represent $p$ exactly, or more irregular update patterns need to be used. For example, for $N = 16$:

$$p = \{15, 14, 13, 10, 12, 9, 8, 4, 11, 7, 6, 3, 5, 2, 1, 0\}.$$  
A possible representation of $p_L$ and $p_H$ can be:

$$p_L = \{N - 1, \ N - 2, \ N - 3, \ Z, \ N - 4, \ Z - 1, \ Z - 2, \ ENDL\},$$

$\end{equation}

$\$p_H = \{\ENDH, \ I + 2, \ I + 1, \ H + 3, \ I, \ H + 2, \ H + 1, \ \ H\}.$

With larger code lengths, we can derive an approximate $p$ by limiting the number of variables used, and assigning a single, regular update pattern to each variable. As an example, Table I reports the variable allocation within $p_L = \{p_{A^L}, p_{B^L}\}$ and $p_H = \{p_{A^H}, p_{B^H}\}$. An initialization value and a single update rule are selected for each variable: every time that a variable is encountered, the previous value is substituted with the updated one. For example, variable $Z$ is initialized as 25, and its update rule is $-1$. Thus, $p_3 = 25$, $p_5 = 24$, $p_6 = 23$, $p_9 = 22$ and $p_{10} = 21$.

Moreover, the variable sequence within different reliability bytes of the same code can repeat itself, like in case of the second and third byte of $p_H$ for $N \geq 64$.

### TABLE I: Variable sequence in $p$ for $N = 32$.

| Reliability byte | Composition sequence |
|------------------|----------------------|
| $p_{A^L}$        | $NNNZNZZY$           |
| $p_{B^L}$        | $NZZYXYY$ ENDL       |
| $p_{A^H}$        | $ENDH LLIIIH$        |
| $p_{B^H}$        | $LIHHIIIHH$          |

### B. Inter-code reliability patterns

Having defined the code reliability as in Section II-A it is possible to observe the evolution of $p$ from a smaller code to larger codes. A first observation can be made towards the reuse of $p_L$ and $p_H$ of a length-$N$ code as part of $p_L$ and $p_H$ of a length-$2N$ code. Fig. 4 shows that the variable sequence found in $p_L$ for length $N$, can approximate the first $N/2$ positions of $p_L$ for length $2N$. A mirrored observation can be made for $p_H$. Different initialization values and different update rules will be necessary when the code length is changed, but the same variable sequence can be used. The precision with which variable sequences of lower-length codes can approximate part of larger-length code sequences depends on the number of variables. A high number of variables will guarantee very good precision and will allow to reuse a particular variable sequence for much larger codes. On the other hand, a large number of variable results in a higher degree of implementation complexity.

The frequency of occurrence and positioning of a variable within $p$ can often be computed exactly. For example, variables $N$ and $H$ in Table II will be encountered every $\delta_i = 2\delta_{i-1} + 1$ variables. Moreover, the initialization values of many variables can be expressed in function of the code length. Variable $I$ can be initialized as $\log_2(N) + 1$, while $ENDL = \log_2(N)$ and $L \approx 6\log_2(N) - 8$.

We thus exploit the observed intra- and inter-code reliability patterns to propose an efficient way to store code reliabilities in decoder implementations. A single variable sequence $p$ is selected, targeting the maximum code length supported. Shorter code lengths can be derived by considering only some $p^8$, as in Fig. 5. To each variable is assigned an initialization value and an update value for each supported code length. The complete $p$ can be constructed sequentially, starting from $i = 0$ for $p_L$ and from $i = N - 1$ for $p_H$. The code structure derived for a certain code length can be extended to that targeting different SNR points by a limited number of ad hoc $p^8$ substitutions. The selection of variables and update rules can be helped by theoretical construction approaches like [13]. The technique proposed in our work is orthogonal to the construction method.

### IV. Simulations and Performance

As a proof of concept, we provide the full construction method of the approximate reliability for codes of length 8 to 256. Table III lists the variable sequences for the reliability...
Fig. 5: FER for $P(64, 32)$, for original reliabilities and approximated reliabilities with 10 variables.

Fig. 6: FER for $P(256, 128)$, for original reliabilities and approximated reliabilities with 24 and 32 variables.

The error-correction performance of the approximated reliability has been evaluated under both SC and SCL decoding. Fig. 5 shows the frame error rate (FER) for the $P(64, 32)$ polar code, both with the original reliabilities and the approximated ones according to Table II or III. The degradation in FER brought by the approximation is negligible for both SC and SCL ($L = 2$ and $L = 8$); thus, an approximation with 10 variables guarantees sufficient precision. Fig. 6 shows the same type of result for $P(256, 128)$: we can see that the proposed 24-variable approximation causes significant FER degradation, particularly severe with SC. Increasing the number of variables to 32 allows a more precise approximation of $p$: the red curves show a $\approx 0.05$ dB FER degradation with respect to the ideal case. It is worth noting that the codes considered in Fig. 5 or 6 are rate 1/2, and thus more susceptible to the imprecisions of the approximation method. In fact, in general, the lowest and highest reliabilities are easy to represent with a regular structure, while the middle reliabilities are more complex. Thus, with high and low rate codes, the demarcation line between frozen and information bits will be closer to well-approximated reliabilities, and thus less likely to cause degradation.

V. RELIABILITY COMPUTATION ARCHITECTURE

Reliability vectors need to be stored both at the encoder and the decoder side: multiple code lengths and rates can lead to high memory requirements. Frozen bit sequences require less memory to be stored, but a single reliability vector is sufficient for 256 variables.

Table II: Variable sequence in $p$ for $8 \leq N \leq 256$.

| Reliability byte | Composition sequence |
|------------------|----------------------|
| $p^8_{AL}$       | NNNZZZZY             |
| $p^8_{BL}$       | NNZZZYWW             |
| $p^8_{CL}$       | NNZZYYYW             |
| $p^8_{DL}$       | XYYYWWWV             |
| $p^8_{EL}$       | XYYXYYYW             |
| $p^8_{FL}$       | XYYXYYYW             |
| $p^8_{GL}$       | XYYXYYYW             |
| $p^8_{HL}$       | XYYXYYYW             |
| $p^8_{IL}$       | XYYXYYYW             |
| $p^8_{JL}$       | XYYXYYYW             |
| $p^8_{KL}$       | XYYXYYYW             |
| $p^8_{ML}$       | XYYXYYYW             |
| $p^8_{NL}$       | XYYXYYYW             |
| $p^8_{PL}$       | XYYXYYYW             |
| $p^8_{QL}$       | XYYXYYYW             |
| $p^8_{RL}$       | XYYXYYYW             |
| $p^8_{SL}$       | XYYXYYYW             |
| $p^8_{TL}$       | XYYXYYYW             |
| $p^8_{UL}$       | XYYXYYYW             |
| $p^8_{VL}$       | XYYXYYYW             |
| $p^8_{WL}$       | XYYXYYYW             |
| $p^8_{XL}$       | XYYXYYYW             |
| $p^8_{YL}$       | XYYXYYYW             |
| $p^8_{ZL}$       | XYYXYYYW             |
| $p^8_{WL}$       | XYYXYYYW             |
| $p^8_{YL}$       | XYYXYYYW             |
| $p^8_{ZL}$       | XYYXYYYW             |
| $p^8_{WL}$       | XYYXYYYW             |
| $p^8_{YL}$       | XYYXYYYW             |
| $p^8_{ZL}$       | XYYXYYYW             |
| $p^8_{WL}$       | XYYXYYYW             |
| $p^8_{YL}$       | XYYXYYYW             |
| $p^8_{ZL}$       | XYYXYYYW             |
| $p^8_{WL}$       | XYYXYYYW             |
| $p^8_{YL}$       | XYYXYYYW             |
| $p^8_{ZL}$       | XYYXYYYW             |
| $p^8_{WL}$       | XYYXYYYW             |
| $p^8_{YL}$       | XYYXYYYW             |
| $p^8_{ZL}$       | XYYXYYYW             |
| $p^8_{WL}$       | XYYXYYYW             |
| $p^8_{YL}$       | XYYXYYYW             |
| $p^8_{ZL}$       | XYYXYYYW             |
| $p^8_{WL}$       | XYYXYYYW             |
| $p^8_{YL}$       | XYYXYYYW             |
| $p^8_{ZL}$       | XYYXYYYW             |
| $p^8_{WL}$       | XYYXYYYW             |
| $p^8_{YL}$       | XYYXYYYW             |
| $p^8_{ZL}$       | XYYXYYYW             |
| $p^8_{WL}$       | XYYXYYYW             |
| $p^8_{YL}$       | XYYXYYYW             |
| $p^8_{ZL}$       | XYYXYYYW             |
| $p^8_{WL}$       | XYYXYYYW             |
| $p^8_{YL}$       | XYYXYYYW             |
| $p^8_{ZL}$       | XYYXYYYW             |
TABLE III: Variable initial values and update values for $p$, $8 \leq N \leq 256$.

| Variable | $N$ | Initial Value | Update |
|----------|-----|---------------|--------|
| $N$      | 8→256 | $N-1$       | -1     |
| $Z$      | 16→256 | $N-7$      | -1     |
| $Y$      | 32 | 15           | -1     |
|          | 64 | 43           | -1.5   |
|          | 128 | 108         | -2     |
|          | 256 | 233         | -2.5   |
| $Y_2$    | 128 | 95           | -2     |
|          | 256 | 219         | -2.5   |
| $Y_3$    | 256 | 232         | -4     |
| $X$      | 32 | 19           | -2     |
|          | 64 | 50           | -1.5   |
|          | 128 | 113         | -2     |
|          | 256 | 241         | -2     |
| $W$      | 64 | 21           | -2     |
|          | 128 | 70           | -2     |
|          | 256 | 204         | -5     |
| $W_2$    | 256 | 174         | -6     |
| $V$      | 128 | 30           | -2     |
|          | 256 | 140         | -10    |
| $V_2$    | 256 | 113         | -3     |
| $U$      | 256 | 47           | -3     |
| $T$      | 256 | 204         | -3.5   |
| $S$      | 256 | 145         | -3     |
| $H$      | 8→256 | 0           | +1     |
| $I$      | 16→256 | $\log_2(N)+1$ | +1     |
| $L$      | 32 | 16           | +1     |
|          | 64 | 22           | +1     |
|          | 128 | 28           | +1.5   |
|          | 256 | 35           | +1.5   |
| $L_2$    | 256 | 53           | +2     |
| $M$      | 64 | 40           | +2     |
|          | 128 | 61           | +2     |
|          | 256 | 79           | +2     |
| $M_2$    | 256 | 95           | +3.5   |
| $O$      | 128 | 91           | +3     |
|          | 256 | 131         | +5     |
| $O_2$    | 256 | 143         | +4     |
| $Q$      | 256 | 192         | +3     |
| ENDL     | 8→256 | $I-1$    | -      |
| ENDH     | 8 | 4           | -      |
|          | 16 | 11          | -      |
|          | 32 | 26          | -      |
|          | 64 | 56          | -      |
|          | 128 | 116        | -      |
|          | 256 | 238        | -      |

for all code rates, while each frozen bit sequence identifies a single $P(N, K)$. A frozen bit sequence is easily generated by comparing each reliability with a threshold value: if $p_i$ is higher than the threshold, then bit $i$ is among the least reliable bit-channels, and is used as a frozen bit.

The proposed approximated reliability computation method can be efficiently implemented in hardware. Figure 7 depicts the architecture at the encoder or decoder side: two of them are instantiated, one for $p_L$ and one for $p_H$. The Structure memory stores the variable sequence, as shown in Table II. Each variable is represented with 5 bits, sufficient to represent up to $2^5$ variable types. The Initialization memory holds the 8-bit initialization values for all combinations of variables and code lengths in $p_L$, and the Update memory the relative 5-bit update rules. In our case study, these two memories hold 38 values for $p_L$ and 30 values for $p_H$ in the 24-variable case. In the 32-variable case, 43 values are needed for $p_L$ and 35 for $p_H$. A counter from 0 to $\frac{N-1}{2}$ addresses the Structure memory; the output variable type, together with the code length, selects the correct variable and update values. The current variables memory holds the update value of all variable types used in the relative $p$ half, appropriately initialized at the beginning of the computation. All memories can be reconfigured through external inputs.

The Update memory stores fixed point values with four integer bits and one decimal bit, while the current variable values are integers. Before the summation, the selected variable is left-shifted of one position, and the result is right-shifted back before the current variables memory is updated.

Along with the proposed architecture, we have designed a baseline architecture for comparative purposes. If no on-line construction technique is applied, the reliability vector for each supported code must be stored. Thus, the baseline considers a total 504 8-bit memory elements, along with the upload and reading logic. Both architectures output two reliabilities at each clock cycle.

Table IV reports synthesis results on 65 nm CMOS technology for the storage-based baseline architecture, along with the 24-variable and the 32-variable approximated architectures, supporting codes of length 8 to 256. The 24-var and 32-var results are relative to two instances of the architecture shown in Fig. 7 so that both $p_L$ and $p_H$ can be computed. With a target frequency of 500 MHz, the area occupation $A$ of the 24-var architecture is 54.4% less than that of the baseline. The multiplexing logic and adders account for a larger part of the area occupation than the baseline, reducing the percentage of memory elements $A_{mem}$. The 32-var architecture extra memory requirements and logic lead to an area reduction.
TABLE IV: Implementation results for reliability storage and computation in CMOS 65 nm technology.

|                  | Full storage | 24-var | 32-var |
|------------------|--------------|--------|--------|
| $f$ [MHz]        | 500          | 500    | 500    |
| $A$ [$\mu$m$^2$] | 52817        | 24103  | 26039  |
| $A_{\text{mem}}$ [$\mu$m$^2$] | 41249 (78.1%) | 17584 (73%) | 18941 (72.7%) |
| $A_{\text{red}}$ | –            | –4.4%  | –50.7% |

$A_{\text{red}}$ of 50.7%. When the target frequency is increased to 1 GHz, $A_{\text{red}}$ decreases for both approximated architectures. This is due to the longer critical path that the reliability computation entails, that leads to higher synthesis effort and logic duplication.

VI. CONCLUSION

In this work, we have proposed an approximate approach to polar code reliability computation that can efficiently be implemented in hardware encoders and decoders. Regular patterns within the reliability vector of a code and among those of different codes are observed, expressing the reliability vectors as a sequence of variables and update rules. Simulation show how the proposed method can be tuned to strike the desired trade-off between accuracy and ease of implementation. A low-complexity architecture is designed for various degrees of approximation, implemented and compared to a storage-based solution, showing 50.7% less area occupation with $< 0.05$ dB FER loss.

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] G. Sarkis, I. Tal, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, “Flexible and low-complexity encoding and decoding of systematic polar codes,” *IEEE Transactions on Communications*, vol. 64, no. 7, pp. 2732–2745, July 2016.
[3] 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.
[4] I. Tal and A. Vardy, “List decoding of polar codes,” *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.
[5] S. A. Hashemi, C. Condo, and W. J. Gross, “Simplified successive-cancellation list decoding of polar codes,” in *IEEE Int. Symp. on Inform. Theory*, July 2016, pp. 815–819.
[6] ——, “Fast simplified successive-cancellation list decoding of polar codes,” in *IEEE Wireless Commun. and Netw. Conf.*, to appear 2017.
[7] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, “237 Gbit/s unrolled hardware polar decoder,” *Electronics Letters*, vol. 51, no. 10, pp. 762–763, 2015.
[8] ——, “Multi-mode unrolled architectures for polar decoders,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 9, pp. 1443–1453, Sept 2016.
[9] B. Yuan and K. Parhi, “Low-latency successive-cancellation list decoders for polar codes with multibit decision,” *IEEE Trans. VLSI Syst.*, vol. 23, no. 10, pp. 2268–2280, October 2015.
[10] C. Xiong, J. Lin, and Z. Yan, “Symbol-decision successive cancellation list decoder for polar codes,” *IEEE Trans. Signal Process.*, vol. 64, no. 3, pp. 675–687, February 2016.
[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] C. Schurch, “A partial order for the synthesized channels of a polar code,” in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 220–224.
[13] G. He, J.-C. Belfiore, X. Liu, Y. Ge, R. Zhang, I. Land, Y. Chen, R. Li, J. Wang, G. Yang, and W. Tong, “$\beta$-expansion: A Theoretical Framework for Fast and Recursive Construction of Polar Codes,” *ArXiv e-prints*, Apr. 2017.
[14] I. Tal and A. Vardy, “How to construct polar codes,” *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6562–6582, October 2013.
[15] 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.