Recursive Descriptions of Decoding Algorithms and Hardware Architectures for Polar Codes

Noam Presman and Simon Litsyn*

Abstract

Polar codes are recursive general concatenated codes. This property motivates a recursive formalization of the known decoding algorithms: Successive Cancellation, Successive Cancellation with Lists and Belief Propagation. This description allows an easy development of the first two algorithms for arbitrary polarizing kernels. Hardware architectures for these decoding algorithms are also described in a recursive way, both for Arikan’s standard polar codes and for arbitrary polarizing kernels.

1 Introduction

Polar codes were introduced by Arikan [1] and provided a scheme for achieving the symmetric capacity of binary memoryless channels (B-MC) with polynomial encoding and decoding complexities. Arikan used the so-called \((u+v,v)\) construction, which is based on the following linear kernel

\[ G_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}. \]

In this scheme, a \(2^n \times 2^n\) matrix, \(G_2^\otimes n\), is generated by performing the Kronecker power on \(G_2\). An input vector \(u\) of length \(N = 2^n\) is transformed to an \(N\) length vector \(x\) by multiplying a certain permutation of the vector \(u\) by \(G_2^\otimes n\). The vector \(x\) is transmitted through \(N\) independent copies of the memoryless channel, \(W\). This results in new \(N\) (dependent) channels between the individual components of \(u\) and the outputs of the channels. Arikan showed that these channels exhibit the phenomenon of polarization under Successive Cancellation (SC) decoding. This means that as \(n\) grows, there is a proportion of \(I(W)\) (the symmetric channel capacity) of the channels that become clean channels (i.e. having the capacity approaching 1) and the rest of the channels become completely noisy (i.e. with the capacity approaching 0). Arikan showed that the SC decoding algorithm has an algorithmic time and space complexity which is \(O(N \cdot \log(N))\) (The same complexity holds also for the encoding algorithm). Furthermore, it was shown [2] that asymptotically in the block length \(N\), the block error probability of this scheme decays to zero like \(O(2^{-\sqrt{N}})\).

Generalizations of Arikan’s code structures were soon to follow. Korada et al. considered binary and linear kernels [3]. They showed that a binary linear kernel is polarizing if and only if its corresponding generating matrix is upper-triangular, and analyzed their rate of polarization, by introducing the notion of kernel exponent. Mori and Tanaka considered the general case of a mapping \(g(\cdot)\), which is not necessarily linear and binary, as a basis for channel polarization constructions [4]. They gave sufficient conditions for polarization and generalized the exponent for these cases. They further showed examples of linear and non-binary Reed-Solomon codes and Algebraic Geometry with exponents that are far better than the exponents of the known binary kernels [5]. The authors of this correspondence gave examples of binary but non-linear kernels having the optimal exponent per their kernel dimensions [6]. All of these structures

*Noam Presman and Simon Litsyn are with the the School of Electrical Engineering, Tel Aviv University, Ramat Aviv 69978 Israel. (e-mails: {presmann, litsyn}@eng.tau.ac.il).
were having homogenous kernels, meaning that the alphabet of their inputs and their outputs were the same. The authors of this correspondence considered the case that some of the inputs of a kernel may have different alphabet than the rest of the inputs \[7\]. This results in the so-called mixed kernel structure, that have demonstrated good performance for finite length codes in many cases. A further generalization of the polar code structure was suggested by Trifonov \[8\], in which the outer polar codes were replaced by suitable codes along with their appropriate decoding algorithms. We note here, that the representation of polar codes as instances of general concatenated codes (GCC) is fundamental to this correspondence, and we elaborate on it in the sequel.

Generalizations and alternatives to SC as the decoding algorithm were also studied. Tal and Vardy introduced the Successive Cancellation List (SCL) decoder \[9, 10\]. In this algorithm, the decoder considers up to \(M\) concurrent decoding paths at each one of its stages, where \(M\) is the size of the list. At the final stage of the algorithm, the most likely result is selected from the list. The asymptotic time and space complexity of this decoder are the same of those of the standard SC algorithm, multiplied by \(M\). Furthermore, an introduction of a cyclic redundancy check code (CRC) as an outer code, results in a scheme with an excellent error-correcting performance, which is sometimes compatible with state of the art schemes (see e.g. \[10\] Section V). Bonik et al. suggested using a separate CRC and a different list size for each outer code, in the GCC structure of the polar code. This approach seems to give better results, comparing it to standard list approach with the same average list size. Finally, Li et al. \[11\], suggested an iterative SCL with CRC algorithm in which the decoder increases the list size by a multiplicative factor of 2 and restart the algorithm, if at the end of the SCL algorithm there doesn’t exist a result that satisfies the CRC. Here again, the excellent performance is achieved with limited average list size and outperforms Tal and Vardy’s original approach. Note, however, that here the average time and space complexity (rather than the worst case complexity) is the basis for comparison between the approaches.

Belief-Propagation is an alternative to the SC decoding algorithm. This is a message passing iterative decoding algorithm that operates on the normal factor graph representation of the code. It is known to outperform SC over the Binary Erasure Channel (BEC) \[12\], and seems to have good performance on other channels as well \[12, 13\].

Leroux et al. considered efficient hardware implementations for the SC decoder for the \((u + v, v)\) polar code \[14, 15\]. They gave an explicit design of a ”line decoder” with \(N/2\) processing elements and \(O(N)\) memory elements. Their work, contains an efficient approximate min-sum decoder, and a discussion on a fixed point implementation. Their design is verified by an ASIC synthesis. Pamuk considered a hardware design of BP decoder tailored for an FPGA implementation \[16\].

The goal of this paper is to emphasize the formalization of polar codes as recursive GCCs and the implication of this structure on the decoding algorithms. The main contributions of this correspondence are as follows: 1) Formalizing Tal and Vardy’s SCL as a recursive algorithm, and thereby generalizing it to arbitrary kernels. 2) Formalizing Leroux et al. SC line decoder and generalizing it to arbitrary kernels. 3) Defining a BP decoder with GCC schedule, and suggesting a BP line architecture for it.

The paper is organized as follows. In Section 2 we describe polar codes kernels as the generating building blocks of polar codes. We then elaborate on the fact that polar codes are examples of recursive GCC structures. This fundamental notion, is the motivation for formalizing the decoding algorithms in a recursive fashion in Section 3. Specifically, we do this for the standard SC, the SCL (both for Arikan’s kernels and arbitrary ones) and BP (for Arikan’s kernel using the GCC schedule). These formalizations lay the ground for hardware architectures of the decoding algorithms in Section 4. Specifically, we restate Leroux et al. SC pipeline and SC line decoders, and introduce a line decoder for the GCC schedule of the BP algorithm. Finally, in Section 5 we consider generalizations of these architectures for arbitrary kernels.

2 Preliminaries

Throughout, we use the following notations. Vectors are denoted by bold letters, for example \(\mathbf{u}\). For \(i \geq j\), let \(\mathbf{u}_j^i = (u_j, ..., u_i)\) be the sub-vector of a vector \(\mathbf{u}\) of length \(i - j + 1\) (if \(i < j\) we say the \(\mathbf{u}_j^i = ()\),
the empty vector, and its length is 0).

In this paper we consider kernels that are based on bijective transformations over a field $F$. A channel polarization kernel of dimension $\ell$, denoted by $g(\cdot)$, is a mapping
\[
g : F^\ell \to F^\ell.
\]
This means that $g(u) = x$, $u, x \in F^\ell$. Denote the output components of the transformation by
\[
g_i(u) = x_i \quad 0 \leq i \leq \ell - 1,
\]
We note that this type of kernel is referred to as homogeneous kernel, because its $\ell$ input coordinates and $\ell$ output coordinate are from the same alphabet $F$.

The homogenous kernel $g(\cdot)$ may generate a polar code of length $\ell^m$ by inducing a larger mapping from it, in the following way [3,7].

**Definition 1** Given a transformation $g(\cdot)$ of dimension $\ell$, we construct a mapping $g^{(m)}(\cdot)$ of dimension $\ell^m$ (i.e. $g^{(m)}(\cdot) : \{0,1\}^{\ell^m} \to \{0,1\}^{\ell^m}$) in the following recursive fashion.

\[
g^{(1)}(u_0^{\ell-1}) = g(u_0^{\ell-1});
\]
\[
g^{(m)} = \left[ g^{(1)}(\gamma_0,0, \gamma_1,0, \gamma_2,0, \ldots, \gamma_{\ell-1},0) ;
\right.
\]
\[
\left.\ldots
g^{(1)}(\gamma_0,1, \gamma_1,1, \gamma_2,1, \ldots, \gamma_{\ell-1},1) ;
\right.
\]
\[
\left.\ldots
g^{(1)}(\gamma_0,\ell^m-1, \gamma_1,\ell^m-1, \gamma_2,\ell^m-1, \ldots, \gamma_{\ell-1},\ell^m-1) \right],
\]
where
\[
\gamma_{i,j} = g_j^{(m-1)}(u_{i,\ell^{m-1}+j}) \quad 0 \leq i \leq \ell^m-1 \quad 0 \leq j \leq \ell - 1.
\]

General Concatenated Codes (GCC) are error correcting codes that are generated by a construction technique, which was introduced by Blokh and Zyablov [18] and Zinoviev [19]. In this construction, we have $\ell$ outer codes $\{C_r\}_{r=0}^{\ell-1}$, where $C_r$ is an $N_{out}$ length code of size $M_r$ over alphabet $F_r$. We also have an inner code of length $N_{in}$ and size $\prod_{r=0}^{\ell-1} |F_r|$ over alphabet $F$, with a nested encoding function $\phi : F_0 \times F_1 \times \ldots \times F_{\ell-1} \to F^{N_{out}}$. The GCC that is generated by these components is a code of length $N_{out} \cdot N_{in}$ and of size $\prod_{r=0}^{\ell-1} M_r$. It is created by taking an $\ell \times N_{out}$ matrix, in which the $r$th row is a codeword from $C_r$, and applying the inner mapping $\phi$ on each of the $N_{out}$ columns of the matrix. As Dumer describes in his survey [20], the GCCs can give good code parameters for short length codes due to a good combination of outer codes and a nested inner code. In fact, some of them give the best parameters known. Moreover, it is common that the decoding algorithms associated with them, utilize their structure by performing local decoding steps on the (short) outer-codes and exchanging decisions via the inner code decoding.

As Arikan already noted, polar codes are examples of recursive GCCs [9, Section I.D]. This observation is useful as it allows to formalize the construction of large length polar code as a concatenation of several smaller length polar codes (outer codes) by using a kernel mapping (an inner code). Therefore, applying this notion to Definition [9] we see that a polar code of length $\ell^m$, may be regarded as a collection of $\ell$ outer polar codes of length $\ell^{m-1}$. These codes are then joined together by applying an inner code (defined by the mapping $g^{(1)}(\cdot)$) on the outputs of these mappings. This idea is illustrated in Figure [a]. In this figure, we see the $\ell$ outer codewords of length $\ell^{m-1}$ organized in $\ell$ rows of the matrix. The inner codeword mapping is depicted as the vertical rectangle that is located on top of them. This is appropriate, as this mapping operates on columns of the of the matrix which rows are the outer codewords. Note, that for brevity we only drew one instance of the inner mapping, but there should be $\ell^{m-1}$ instances of it, one for each column of this matrix. In the homogenous case, the outer codes themselves are constructed in the
Example 1 (Arikan’s Construction) Let $u$ be an $N = 2^m$ length binary vector. The vector $u$ is transformed to an $N$ length vector $x$ by using a bijective mapping $g(\cdot) : \{0, 1\}^N \rightarrow \{0, 1\}^N$. The transformation is defined recursively as

$$
\begin{align*}
\text{for } n = 1 & \quad g^{(1)}(u) = [u_0 + u_1, u_1] \\
\text{for } n > 1 & \quad g^{(m)}(u) = [v_0, w_0, v_1, w_1, ..., v_{N/2-1}, w_{N/2-1}] = x
\end{align*}
$$

where $v_{0}^{N/2-1} = g^{(m-1)}(u_{N/2}^{N/2-1}) + g^{(m-1)}(u_{0}^{N-1})$ and $w_{0}^{N/2-1} = g^{(m-1)}(u_{N/2}^{N-1})$. See also Figure 2.

In a mixed kernel constructions the outer codes are not necessarily from the same family of polar codes. For example, if we take the first kernel $g_1(u_0, u_1, 2, u_3) = x_0^3 \in \{0, 1\}^4$ and define the RS kernel.

---

Figure 1: GCC representations of homogenous kernel according to Definition[1]

Figure 2: Arikan’s Construction
as \( g_2(u_{0,1}, u_{2,3}, u_{4,5}, u_{6,7}) = x_0^3 \in \{(0,1)^2\}^4 \), then the general concatenated construction is given in Figure 3. Now, note that using \( g_2^{(m)} \) mapping over a binary channel is like using a concatenated scheme in which the inner code is the standard binary full space mapping. It can be observed, that the mapping in Figure 3 has more potential in transforming between the used alphabets. This concept may be further generalized, by replacing some of the outer polar codes, with other types of codes (see e.g. Trifonov’s proposal [8]).

The recursive GCC structure of polar codes calls for recursive formalization of the algorithms associated with them. These algorithms enjoy from a simple and clear description, which may lead to an elegant analysis. Furthermore, in some cases it allows reuse of resources and indicates which operations may be done in parallel. The recursive encoding algorithm has already been described in Definition 1. The recursive decoding algorithms are described in the next section.

3 Recursive Descriptions for Decoding Algorithms of Polar Codes

In this section, we describe decoding algorithms for polar codes in a recursive framework that is induced from their recursive GCC structure. Roughly speaking, all the algorithms we consider here have a similar format. Consider the GCC structure of Definition 1. This means that we have a length \( N \) code, that is composed of \( \ell \) layers of outer codes, denoted by \( \{C_r\}_{r=0}^{\ell-1} \), each one of length \( N/\ell \). The decoding algorithms, we consider here, are composed of \( \ell \) pairs of steps. Pair number \( r \) is dedicated to decoding \( C_{r-1} \), in the following way.

\[
\begin{align*}
\text{STEP} & \quad 2 \cdot r - 1 \\
& \quad \text{Using the previous steps, prepare the inputs to the decoder of code } C_i. \\
\text{STEP} & \quad 2 \cdot r \\
& \quad \text{Call the decoder of code } C_i \text{ on the input you’ve prepared.} \\
& \quad \text{Process the output of this decoder, together with the outputs of the previous steps.}
\end{align*}
\]

Typically, the codes \( \{C_r\}_{r=0}^{\ell-1} \) are polar codes of length \( N/\ell \), thereby creating the recursive structure of the decoding algorithm.

It should be noted that the above decoding format is quite common for decoding algorithms of GCCs. As an example, see the decoding algorithms in Dumer’s survey on GCCs [20]. In addition, the recursive decoding algorithms for Reed-Muller (RM) codes, utilizing their Plotkin \((u+v, v)\) recursive GCC structure were extensively studied by Dumer [21] [22] and are closely related to the algorithms we present here. Actually, Dumer’s simplified decoding algorithm for RM codes [22, Section IV] is the SC decoding for the Arikan’s structure, we describe in Subsection 3.1.

The algorithms we describe in a recursive fashion are the SC (Subsection 3.1), Tal and Vardy’s SCL (Subsection 3.2) and BP (Subsection 3.3). For all of these algorithms, we first consider Arikan’s \((u+v, v)\)
code. For the first two algorithms we also provide generalizations to other kernels, both homogenous and mixed. We note, that when possible, we prefer that the inputs to the algorithm and the internal computations are interpreted as log likelihood ratios (llrs). Thus, the SC algorithms and the BP are described in such manner, but in SCL decoding, we use likelihoods instead of llrs. Furthermore, in our discussion we do not consider how to efficiently compute these quantities. In some cases, especially with large kernels or with large alphabet size, these computations pose a computational challenge. Approaches to adhere this challenge, are efficient decoding algorithms (such as variants of Viterbi algorithms) or approximations of the computations (for example, the min-sum approximation that Leroux et al. used [15] or the near Maximum Likelihood (ML) decoding algorithms that were used by Trifonov [8]).

### 3.1 A Recursive Description of the SC Algorithm

We begin by considering the SC decoder for Arikan’s \((u + v, v)\) construction, and then generalize it to arbitrary kernels. First, let us describe the decoding algorithm for length \(N = 2\) code, i.e. for the basic kernel \(g^{(1)}(u, v) = (u + v, v) \equiv (a, b)\). We get as input \([\lambda_a, \lambda_b]\) which are the log likelihood ratios (llrs) of the output of the channel (\(\lambda_a\) corresponds to the first output of the channel and \(\lambda_b\) corresponds to the second output). The algorithm has four steps.

| **STEP I** | Compute the llr of \(u\), \(L_u = 2 \tanh^{-1}(\tanh(\lambda_a/2) \tanh(\lambda_b/2))\). |
| --- | --- |
| **STEP II** | Decide on \(u\), (denote it by \(\hat{u}\)). |
| **STEP III** | Compute the llr of \(v\), (given the estimate of \(\hat{u}\)): \(L_v = (-1)^{\hat{u}} \cdot \lambda_a + \lambda_b\). |
| **STEP IV** | Decide on \(v\), (denote it by \(\hat{v}\)). |

It should be noted that steps II and IV, may be done based on the llrs computed on steps I and III (i.e. by their sign), or by using an additional side information (for example, if \(u\) is frozen, then the decision is based on its known value).

Now, for describing a SC decoder of length \(N = 2^n\), let us assume that we already developed a SC decoder for length \(N/2\) polar code. We assume that the \(N\) length decoder gets as input \(N\) channel output llrs, \(\{\lambda_i\}_{i=0}^{N-1}\), and the frozen bits indices. The decoder outputs the estimation of the information (unfrozen) bits and the estimation of the codeword that was sent on the channel. For convenience, we assume that the estimation of the information word is an \(N\) length vector (denoted by \(u\)) which includes also the values for the frozen bits. A decoder for length \(N\) polar code contains the following steps.

| **STEP I** | Partition the llr vector into pairs of consecutive llr values \(\{(\lambda_{2i}, \lambda_{2i+1})\}_{i=0}^{N/2-1}\). Compute the llr input vector, \(L_0^{N/2-1}\), for the first outer code such that \(L_i = 2 \tanh^{-1}(\tanh(\lambda_{2i}/2) \tanh(\lambda_{2i+1}/2))\), \(0 \leq i \leq N/2 - 1\). |
| --- | --- |
| **STEP II** | Give the vector \(L_0^{N/2-1}\) as an input to the polar code decoder of length \(N/2\). Also provide to the decoding algorithm, the indices of the frozen bits from the first half of the codeword (corresponding to the first outer code). Assume that the decoder outputs \(u^{(0)}\) as the estimation of the information word, and |
\( x^{(0)} \) as the estimation of the first outer polar codeword of length \( N/2 \). Both of them are vectors of length \( N/2 \). Then, we can output \( u^{(0)} \) as \( u_0^{N/2-1} \) (the first half of the estimated information word).

**STEP III**

Using, again, the input llr pairs and \( x^{(0)} \) as the estimation of the first outer polar codeword, prepare the llr input vector for the second outer code, \( L_0^{N/2-1} \), such that

\[
L_i = (-1)^{x_i^{(0)}} \lambda_{2i} + \lambda_{2i+1}, \quad 0 \leq i \leq N/2 - 1.
\]

**STEP IV**

Give the vector \( L_0^{N/2-1} \) as an input to the polar code decoder of length \( N/2 \) and the indices of the frozen bits from the second half of the codeword (corresponding to the second outer code). Assume that the decoder outputs \( u^{(1)} \) as the estimation of the information word, and \( x^{(1)} \) as the estimation of the second outer polar codeword of length \( N/2 \). Then, we can output \( u^{(1)} \) as \( u_{N/2}^{N-1} \) (the second half of the estimated information word).

Construct the estimation of the codeword as follows:

\[
x = \left\{ x_i^{(0)} + x_i^{(1)} \right\}_{i=0}^{N/2-1}.
\]

Let us now generalize this decoding algorithm for the GCC scheme with a general kernel. In this case for length \( N \) code, we have an \( \ell \) length mapping \( g(u) = x \) over an alphabet \( F \), i.e. \( g(\cdot) : F^\ell \to F^\ell \). We also have at most \( \ell \) outer codes \( \{C_i\} \), each one of length \( N/\ell \). We may have less than \( \ell \) outer codes, in case some of the inputs are glued (which results in a mixed kernel case). In this case, the outer code corresponding to the glued inputs is considered to be over a larger size input alphabet. We assume that each outer code has a decoding algorithm associated with it. This decoding algorithm is assumed to get as input the "channel" observations on the outer code symbols (usually manifested as probabilities matrices, or llr vectors). If the outer code is a polar code, then this algorithm should also get the indices of the frozen bits of the outer code. We require that the algorithm will output its estimation on the information vector and its corresponding outer code codeword. Assuming that we know input symbols \( u_0^\ell \), computing the llr vector \( L(\cdot) \) corresponding to input number \( k+1 \) of the transformation is done according to the following rule.

\[
L(t) = \ln \left( \frac{\sum_{u_{k+1}^\ell \in F^{t-k+1}} R_g \left( u_0^\ell, 0, u_{k+1}^\ell \right)}{\sum_{u_0^\ell \in F^{t-k+1}} R_g \left( u_0^\ell, t, u_{k+1}^\ell \right)} \right),
\]

where

\[
R_g(u_0^{t-1}) = \exp \left( \sum_{i=0}^{t-1} \lambda_i \left( g_i(u) \right) \right),
\]

which is the likelihood ratio associated with input \( u \) to the kernel \( g(\cdot) \), and \( \lambda_i(\cdot) \) is the llr associated with the \( i^{th} \) output of the kernel, \( x_i \). Because \( F \) may be non-binary, \( \lambda(\cdot) \) and \( L(\cdot) \) are assumed to be functions of llrs, that is \( \lambda_i(t) = \log \left( \frac{Pr(y|x=0)}{Pr(y|x=t)} \right) \), for \( t \in F \), where \( y \) is assumed to be the vector of the observations.

We now describe the SC decoding algorithm. As we already mentioned, because of structure of the code, the decoding algorithm is composed of pairs of steps, such that pair \( r \) deals with outer code \( r-1 \), where \( 1 \leq r \leq \ell \). As a preparation step, we partition the decoder’s \( N \) length input llr vector \( \lambda(\cdot) \) to \( N/\ell \) length vectors, each of length \( \ell \), denoted by \( \lambda^{(m)}(\cdot) \), such that

\[
\lambda^{(m)}_i(\cdot) = \lambda_{m, \ell+i}(\cdot), \quad 0 \leq m \leq N/\ell - 1, \quad 0 \leq i \leq \ell - 1.
\]

The \( \ell \) length vector \( \lambda^{(m)}(\cdot) \) is associated with the output symbols corresponding to the \( m^{th} \) symbol of the outer codes (transformed by kernel \( g(\cdot) \)). We denote the information word that was given by the decoder of the \( m^{th} \) outer code by \( u^{(m)} \) and its corresponding codeword by \( x^{(m)} \), both of them are of length \( N/\ell \).

We have the following pair of steps of the algorithm \( 1 \leq r \leq \ell \).
STEP 2 · \( r - 1 \)
Using the results on the outer-codewords of the previous steps i.e. \( x^{(m)} \), for \( 0 \leq m \leq r - 2 \), prepare the \( N/\ell \) length llr input vector \( L(\cdot) \) for outer code number \( r - 1 \). To do that, for \( 0 \leq j \leq N/\ell - 1 \), compute \( L_j(\cdot) \) using (2) with \( \left\{ x_j^{(m)} \right\}_{0 \leq m \leq r - 2} \) as the estimated inputs to the transformation.

STEP 2 · \( r \)
Give the llr vector \( L(\cdot) \) as an input to the decoder of outer code number \( r - 1 \). If this is a polar code decoder of length \( N/\ell \), then also supply the indices of the frozen symbols in the range \( [(r - 1) \cdot N/\ell, r \cdot N/\ell - 1] \). The decoder outputs \( u^{(r-1)} \), as the estimation of the information word and \( x^{(r-1)} \) as the estimation of the outer codeword. Both of these vectors are of length \( N/\ell \) symbols.

After the step 2 · \( \ell \), the decoder outputs its estimation for the information word, by concatenating the information parts generated by all the outer code decoders, i.e. \( u = [u^{(0)}, u^{(1)}, ..., u^{(\ell-1)}] \). The estimation of the codeword, \( x \), is done by applying the transformation \( g(\cdot) \) on the column of the matrix, which rows are \( \left\{ x^{(m)} \right\}_{m=0}^{r-1} \), that is

\[
x_{\ell,i}^{(i+1)-1} = g \left( x_i^{(0)}, x_i^{(1)}, ..., x_i^{(\ell-1)} \right), \quad 0 \leq i \leq N/\ell - 1.
\]

The base case of the recursion, i.e. the decoder for \( N = \ell \) length polar code, is a simple generalization of the SC decoder for length \( N = 2 \) code of Arikan. The idea is to successively estimate the input bits to the transformation \( g(\cdot) \), using (2). We decide on the symbol \( u_i \) using the llr generated by (2) (in which our previous decisions are taken as known values). If \( u_i \) is frozen, we skip the calculation of (2), and decide on its known value.

In case we have a mixed kernel construction, the generalization is very easy. Let us assume that we have glued the symbols \( u_1 \) and \( u_2 \) to a new symbol \( u_{1,2} \in \mathbb{F}^2 \). In this case, we treat these two symbols as a one entity, and consider the outer code associated with them, denoted as \( C_{1,2} \), as \( N/\ell \) length code over the alphabet \( \mathbb{F}^2 \). The only change we have in the decoding algorithm is for the pair of steps corresponding to this "glued" outer code. For the first step in the pair, we need to compute the \( N/\ell \) length llr vector \( L(\cdot, \cdot) \), that serves as an input to the the decoder of \( C_{1,2} \). In this case, we need that each llr function in the vector, will be a function of both \( u_1 \) and \( u_2 \). Equation (2) is therefore updated accordingly.

\[
L(t_1, t_2) = \ln \left( \frac{\sum_{u_{1,2} \in \mathbb{F}_{t_1-3}} R_g(u_0, 0, 0, u_{1,2})}{\sum_{u_{1,2} \in \mathbb{F}_{t_1-3}} R_g(u_0, t_1, t_2, u_{1,2})} \right).
\]

The second step of the pair is remained unchanged.

3.2 A Recursive Description of the SCL Algorithm
Tal and Vardy introduced an efficient SCL decoder [10]. We give here a recursive description of this algorithm. In the algorithm, there is a requirement to compare between the likelihoods of different decoding possibilities. Therefore, we need to assume that inputs to the algorithm as well as its internal computations are interpreted as likelihoods, instead of llrs. Note, that if the decoding list is of size 1, then the formulation we give below is of the SC decoder we described in the previous subsection (with the only difference that we use likelihoods instead of llrs to describe the computations).

We also note, that here we only describe the algorithm to generate the list. At the end of the algorithm, the most likely element of the list should be given as output. If there is an outer CRC, only outputs that agree with the CRC should be considered. The notion of likelihoods normalization that was considered by Tal and Vardy [10] Algorithm 14] to avoid floating-point underflow is also applicable here. These two issues and their generalization are not further discussed in this paper.
The algorithm of SCL decoding of $N$ length polar code with list of size $M$ gets as input the following structures.

- Two likelihood matrices $\Pi^{(0)}$ and $\Pi^{(1)}$ of $M \times N$ dimension, which represent $M$ arrays of conditional probability values (each array of length $N$) - each one corresponds to an input option, that the decoder should consider. We refer to these input options as *models*. The plurality of the models exists, because at any given point, in the list decoding algorithm we allow $M$ options for past decisions on the symbols of the information word (these options form the list). Each one of these options induces a different statistical model, in which we assume that the information sub-vector, which is associated with it, is the one that was transmitted. We have $\Pi_{i,j}^{(b)} = \Pr(Y^{(i)}_j = y^{(i)}_j | V_j = b)$, where $Y^{(i)}_j$ is the measurement of the $j$th channel $V_j \rightarrow Y_j$ of the $i$th option in the list and $b \in \{0, 1\}$.

- A marker $\rho_{in}$ indicating how many rows in $\Pi^{(0)}$ and $\Pi^{(1)}$ are occupied. The algorithm supports decoding of $\rho_{in} \in [1, M]$ input models.

- The vector of the indices of the frozen bits.

The algorithm outputs the following structures.

- A matrix $U$ of $M \times N$ dimension, which represents $M$ arrays of information values (each array of length $N$) - this is the list of the possible information words that the decoder estimated.

- A matrix $X$ of $M \times N$ dimension, which represents $M$ arrays of codewords (each array of length $N$) - this is the list of codewords that correspond to the information words in $U$.

- An indicator vector $s_{0}^{M-1}$, that indicates for each row in $U$ and $X$ to which row in the input $\Pi^{(0)}$ and $\Pi^{(1)}$ it has originated from (i.e. it refers to the statical model that was assumed when estimating this row).

- A marker $\rho_{out}$ indicating how many rows in $U$ or $X$ are occupied.

For the basic $N = 2$ length case the algorithm operates as follows.

**STEP I**

For each of the $\rho_{in}$ occupied rows of $\Pi^{(0)}$ and $\Pi^{(1)}$ compute

$$P_i^{(0)} = \frac{1}{2} \left( \Pi_{i,0}^{(0)} \cdot \Pi_{i,1}^{(0)} + \Pi_{i,0}^{(1)} \cdot \Pi_{i,1}^{(1)} \right)$$

and

$$P_i^{(1)} = \frac{1}{2} \left( \Pi_{i,0}^{(0)} \cdot \Pi_{i,1}^{(1)} + \Pi_{i,0}^{(1)} \cdot \Pi_{i,1}^{(0)} \right), \text{ for } 0 \leq i \leq \rho_{in} - 1.$$

**STEP II**

Concatenate the two vectors to one $2 \cdot \rho_{in}$ length vector, $P = [P^{(0)}, P^{(1)}]$.

Let $\hat{P}$ be a vector that contains the $\rho = \min\{2 \cdot \rho_{in}, M\}$ largest values of $P$. Let $s^{(0)}, u^{(0)}$, be $\rho$ length column vectors corresponding to $\hat{P}$, such that the $i^{th}$ element of $\hat{P}$ is element number $s_{i}^{(0)}$ in the vector $P^{(u_{i}^{(0)})}$. This element was originated from model number $s_{i}^{(0)}$, which means that it was computed assuming that row number $s_{i}^{(0)}$ of $\Pi^{(0)}$ and $\Pi^{(1)}$ was the statistical model.

**REMARK:** If $u$ is frozen (without loss of generality assume that it is set to the 0 value), then steps I and II should be skipped and $s^{(0)} = [0, 1, ..., \rho_{in} - 1], u^{(0)} = 0$.

**STEP III**

Generate two $\rho$ length vectors, $P^{(0)}$ and $P^{(1)}$. For each of the $\rho$ occupied rows of $s^{(0)}$, $U^{(0)}$
compute \((i \in [0, \rho - 1])\).

\[
P_i^{(0)} = \frac{1}{2} \left\{ \begin{array}{ll}
\Pi_{i,0}^{(0)} \cdot \Pi_{s_i,1}^{(0)}, & x_i^{(0)} = 0; \\
\Pi_{i,0}^{(1)} \cdot \Pi_{s_i,1}^{(0)}, & x_i^{(0)} = 1.
\end{array} \right.
\]  

(7)

\[
P_i^{(1)} = \frac{1}{2} \left\{ \begin{array}{ll}
\Pi_{i,0}^{(1)} \cdot \Pi_{s_i,1}^{(0)}, & x_i^{(0)} = 0; \\
\Pi_{i,0}^{(1)} \cdot \Pi_{s_i,1}^{(1)}, & x_i^{(0)} = 1.
\end{array} \right.
\]  

(8)

**STEP IV**

Concatenate the two vectors to one \(2 \cdot \rho\) length vector, \(\mathbf{P} = [\mathbf{P}^{(0)}, \mathbf{P}^{(1)}]\).

Let \(\mathbf{P}\) be a vector that contains the \(\rho_{out} = \min\{2 \cdot \rho, M\}\) largest values of \(\mathbf{P}\). Let \(s^{(1)}, u^{(1)}\), be \(\rho_{out}\) length column vectors corresponding to \(\mathbf{P}\), such that the \(i^{th}\) element of \(\mathbf{P}\) is element number \(s_i^{(1)}\) of the vector \(\mathbf{P}^{(u^{(1)})}\).

**REMARK:** If the second bit is frozen (without loss of generality assume that it is set to the 0 value), then steps III and IV should be skipped and \(s^{(1)} = [0, 1, ..., r - 1]\), \(u^{(1)} = 0\), \(\rho_{out} = r\).

Output:

\[
\begin{align*}
\rho_{out} \\
s_i = s_\sigma & \quad s.t \quad \sigma = s_i^{(1)} & i \in [0, \rho_{out} - 1] \\
\mathbf{U} = [u^{(0)}; u^{(1)}] \\
\mathbf{X} = [u^{(0)} + u^{(1)}; u^{(1)}]
\end{align*}
\]

Now, for describing a SC decoder for length \(N = 2^n\) polar code, let us assume that we already developed a SCL decoder for length \(N/2\) polar code. Therefore, a decoder for length \(N\) polar code contains the following steps.

**STEP I**

Prepare the probability transition matrices for the first polar outer code decoder. Specifically, generate two matrices \(\mathbf{P}^{(b)}\) of dimension \(M \times N/2\), \(b \in \{0, 1\}\), such that for \(0 \leq i \leq \rho_{in} - 1\) \(0 \leq j \leq N/2 - 1\)

\[
P_{i,j}^{(0)} = \frac{1}{2} \left( \Pi_{i,2,j}^{(0)} \cdot \Pi_{i,2,j+1}^{(0)} + \Pi_{i,2,j}^{(1)} \cdot \Pi_{i,2,j+1}^{(1)} \right)
\]  

(9)

and

\[
P_{i,j}^{(1)} = \frac{1}{2} \left( \Pi_{i,2,j}^{(0)} \cdot \Pi_{i,2,j+1}^{(1)} + \Pi_{i,2,j}^{(1)} \cdot \Pi_{i,2,j+1}^{(0)} \right)
\]  

(10)

**STEP II**

Give the \(M \times N/2\) matrices \(\mathbf{P}^{(0)}\) and \(\mathbf{P}^{(1)}\), the frozen bits from the first half of the codeword and \(\rho_{in}\) as the number of elements in the list as inputs to the polar code decoder of length \(N/2\). Assume that the decoder outputs \(\mathbf{U}^{(0)}\) and \(\mathbf{X}^{(0)}\) as the list of estimations of the information word and the outer polar codeword of length \(N/2\), respectively. Both of these structures are matrices of dimension \(M \times N/2\). The decoder also outputs \(s^{(0)}\) as the source indicator vector (of length \(M\)), and \(\rho\) as the size of the list.

**STEP III**

Prepare the input matrices for the decoder of the second outer polar code of length \(N/2\).
Specifically, generate two matrices $P^{(b)}$ of dimension $M \times N/2$, $b \in \{0, 1\}$, such that for $0 \leq i \leq \rho - 1$, $0 \leq j \leq N/2 - 1$

$$P_{i,j}^{(0)} = \frac{1}{2} \left\{ \begin{array}{ll} \Pi_{s_i^{(0)},2,j}^{(0)} \cdot \Pi_{s_i^{(0)},2,j+1}^{(0)} & X_{s_i^{(0)},j}^{(0)} = 0; \\ \Pi_{s_i^{(0)},2,j}^{(1)} \cdot \Pi_{s_i^{(0)},2,j+1}^{(1)} & X_{s_i^{(0)},j}^{(1)} = 1, \end{array} \right. \quad (11)$$

and

$$P_{i,j}^{(1)} = \frac{1}{2} \left\{ \begin{array}{ll} \Pi_{s_i^{(1)},2,j}^{(1)} \cdot \Pi_{s_i^{(1)},2,j+1}^{(1)} & X_{s_i^{(1)},j}^{(0)} = 0; \\ \Pi_{s_i^{(1)},2,j}^{(0)} \cdot \Pi_{s_i^{(1)},2,j+1}^{(0)} & X_{s_i^{(1)},j}^{(1)} = 1. \end{array} \right. \quad (12)$$

**STEP IV**

Give these matrices $P^{(0)}$ and $P^{(1)}$, the vector of indices of the frozen bits from the second half of the codeword and $\rho$ (as the number of elements in the list) as inputs to the decoder of the second outer polar code of length $N/2$.

Assume that the decoder outputs $U^{(1)}$ and $X^{(1)}$ as the list of estimations of the information words and their corresponding outer polar codeword of length $N/2$. Both of these structures are matrices of dimension $M \times N/2$. The decoder also outputs $s^{(1)}$ as the source indicator vector (of length $M$) and $\rho_{out}$ as the size of the output list.

Now, generate the outputs of the decoder ($i \in [0, \rho_{out} - 1]$):

- $s_i = \tilde{s}_{\sigma(i)}^{(0)}$, where $\sigma(i) = s_i^{(1)}$.
- $U_{\rightarrow i} = [U_{s_i}^{(0)}, U_{s_i}^{(1)}]$.
- $X_{i,\text{even}} = X_{\rightarrow s_i}^{(0)} + X_{\rightarrow s_i}^{(1)}$.
- $X_{i,\text{odd}} = X_{\rightarrow s_i}^{(1)}$.

where $X_{i,\text{even}}$ ($X_{i,\text{odd}}$) are the vectors of the even (odd) indices columns of row number $i$ in the matrix $X$, and for a matrix $A$, the $i^{th}$ row is denoted by $A_{\rightarrow i}$.

Let $T(n)$ be the decoding time complexity, for length $N = 2^n$ polar code. Then $T(n) = 2 \cdot T(n - 1) + O(M \cdot N)$, and $T(1) = O(M)$, which results in $T(n) = O(M \cdot N \cdot \log_2 N)$. Similarly, the space complexity of the algorithm can be shown to be $O(M \cdot N)$.

The generalization of the decoding algorithm for a homogenous kernel of dimension $\ell$ with alphabet $F$ is quite simple. Here we emphasize the principal changes, from the $(u + v, v)$ algorithm. First, the only change in the input is that we should have $|F|$ input channel matrices, $\mathbf{P}^{(b)}$, one for each $b \in F$. In the decoding algorithm, we have $\ell$ pairs of steps, such that each one is dedicated to a different outer codes. Before step $2 \cdot r - 1$, we have decoded outer-codes $C_i$, where $0 \leq i \leq r - 2$. We assume, that we have temporary lists $X^{(i)}$ and $U^{(i)}$ of the estimated codewords and their corresponding information words, which are represented by matrices of size $M \times N/\ell$. The $i^{th}$ matrix corresponds to the decoding of $C_i$, $0 \leq i \leq r - 2$. We maintain a temporary indicator vector $s^{(0)}_i$ of length $M$, such that $X_{i}^{(0)}$ and $U_{i}^{(0)}$ were estimated assuming model $s^{(0)}_i$. We also have $\rho$ as the number of occupied elements in the list so far (on the initialization, $\rho = \rho_{in}$).

**STEP 2 \cdot r - 1**

Using the decoding results of the outer-codewords from the previous steps i.e. $X^{(m)}$, for $0 \leq m \leq r - 2$, prepare the $N/\ell$ length likelihood lists, $\{P^{(b)}\}_{b \in F}$. Each list is an $M \times N/\ell$ matrix, and all of them will serve as an input to the decoder of the $N/\ell$ length outer code number $r - 1$. For the computation of row $i$ of $P^{(b)}$, use the input statistical model $s^{(0)}_i$.
that is the likelihoods in rows $\{\Pi_{b \rightarrow s}^{(b)}\}_{b \in F}$. Also, as the estimated codewords of the previous outer codes, we need to use rows $\{X_{s}^{(m)}\}_{0 \leq m \leq r-2}$. To prepare $\{P_{b}^{(b)}\}_{b \in F}$ we do computations on likelihoods (instead of llrs), which are the equivalent to step $2 \cdot r - 1$ in the description of the general SC decoding (Subsection 3.1).

**STEP 2 : $r$**

Give the matrices $\{P_{b}^{(b)}\}_{b \in F}$, the vector of indices of the frozen bits from the second half of the codeword and $\rho$ (as the number of elements in the list) as inputs to the decoder of outer polar code number $r - 1$.

Assume that the decoder outputs $U_{\rightarrow s}^{(r-1)}$ and $X_{\rightarrow s}^{(r-1)}$ as the list of estimations of the information word and their corresponding estimations of the transmitted codeword of the outer code number $r - 1$, respectively. Both of these structures are matrices of dimension $M \times N/\ell$. The decoder also outputs $s^{(1)}$ as the model indicator vector (of length $M$) and $\rho$ as the number of occupied elements in the list.

Allocate $s$, a temporary vector of size $M$, and temporary matrices $\tilde{X}^{(i)}$, $\tilde{U}^{(i)}$ of size $M \times N/\ell$, where $0 \leq i \leq r - 2$.

- $s_{i} = s_{\sigma(i)}^{(0)}$, where $\sigma(i) = s_{1}^{(1)}$ and $0 \leq i \leq r - 2$.
- $\tilde{X}_{\rightarrow j}^{(i)} = X_{\rightarrow \sigma(j)}^{(i)}$, $\leq i \leq r - 2, \ 0 \leq j \leq \rho - 1$
- $\tilde{U}_{\rightarrow j}^{(i)} = U_{\rightarrow \sigma(j)}^{(i)}$, $\leq i \leq r - 2, \ 0 \leq j \leq \rho - 1$

Copy these matrices to the internal data structures.

- $s^{(0)} = s$.
- $X^{(i)} = \tilde{X}^{(i)}$, $\leq i \leq r - 2$
- $U^{(i)} = \tilde{U}^{(i)}$, $\leq i \leq r - 2$

If this is step $2 \cdot \ell$ (the last step), then prepare the output.

- $\rho_{\text{out}} = \rho$.
- $s$.
- $U = [U^{(0)}; U^{(1)}; \ldots; U^{(\ell-1)}]$.
- $X_{i, \ell \cdot m : \ell \cdot (m-1)+1} = g(X_{i, m}^{(0)}; X_{i, m}^{(1)}; \ldots; X_{i, m}^{(\ell-1)})$, $0 \leq m \leq N/\ell - 1$.

Where for a matrix $A$, the subvector that is composed of the columns $n_{1}$ to $n_{2}$ of the $i^{th}$ row is denoted by $A_{i, n_{1}:n_{2}}$.

The decoder for the basic $N = \ell$ length code, also contains $\ell$ pairs of steps. The decoding is similar to the above, with the exception that instead of delivering the likelihood matrices $\{P_{b}^{(b)}\}_{b \in F}$ (here these matrices are actually column vectors) to a decoder, we concatenate them to a vector $\tilde{P}$ and choose the $\rho = \min\{M, 2 \cdot \rho\}$ maximum elements from it, and generate the indicator vector $s^{(1)}$ and the information symbols list $u^{(r-1)}$, similarly to the case of the $N = 2$ length decoder of the $(u + v, v)$ construction.

In case the kernel is mixed, the generalization is also quite easy. Let us consider the mixed example, from the end of Subsection 3.1. The only changes we have in the decoding algorithm, are for the pair of steps associated with the glued outer code $C_{1, 2}$. In step 3 (the preparation step for this outer-code), we prepare $|F|^{2}$ input matrices $P_{b_{1}, b_{2}}^{(b_{1}, b_{2})}$, for $(b_{1}, b_{2}) \in F^{2}$. For this, we use the equivalent of equation 14 for likelihoods (instead of llrs). The decoder of $C_{1, 2}$ is supposed to return a list of estimations of the information words, their corresponding codewords and the model indicator vectors. These outputs
and the temporary structures are re-organized, as is done in step 2 of the decoding algorithm of the homogenous kernel polar code. Note, however, that at the end of step 4, there are three information words lists \( \mathbf{U}^{(0)} \), \( \mathbf{U}^{(1)} \) and \( \mathbf{U}^{(2)} \) along with their corresponding three outer codewords lists. This is because we have decoded the glued outer code \( \mathcal{C}_{1,2} \) simultaneously, which contributed \( \mathbf{U}^{(1)} \), \( \mathbf{U}^{(2)} \), \( \mathbf{C}^{(1)} \) and \( \mathbf{C}^{(2)} \) in the same decoding step.

### 3.3 A Recursive Description of the BP Algorithm

BP is an alternative to SC decoding \([1]\). It is an iterative message-passing algorithm, which messages are defined using Forney’s normal factor graph \([23]\). There is no evidence which algorithm is better for general channels, except for the BEC, in which BP is shown to outperform SC \([12]\). However, simulations indicate that BP outperforms SC in many cases. The order of sending the messages on the graph is called the schedule of the algorithm. Hussami et al. suggested to use a "Z shape schedule" for transferring the messages \([12\) Section II.A.\]. Here we prefer, to present a serial schedule which is induced from the GCC structure of the code.

We begin by describing the type of messages that are computed during the algorithm. Figure 4 depicts the normal factor graph representation of Arikan’s kernel. We have 4 symbol half edges denoted by \( u, v, x_0 \) and \( x_1 \). These symbols have the following functional dependencies among them \( x_0 = u + v \) and \( x_1 = v \). The messages and the inputs that may be sent on the graph are assumed to be llrs, and their values are taken from \( \mathbb{R} \cup \{ \pm \infty \} \). The \( \infty \) and \( -\infty \) are special types of llrs, that indicate known values of 0 and 1, respectively. They are used to support the existence of the frozen bits of the polar code.

For the symbol half edges, we assume that we have 4 input llr messages. These messages may be generated by the output of the channel, by known values associated with frozen bits, or by computations that were done in this iteration or previous ones. We denote these messages by \( \mu_v^{(in)} \), \( \mu_u^{(in)} \), \( \mu_{x_0}^{(in)} \) and \( \mu_{x_1}^{(in)} \). The algorithm computes (in due time) 4 output llr messages, \( \mu_u^{(out)} \), \( \mu_v^{(out)} \), \( \mu_{x_0}^{(out)} \) and \( \mu_{x_1}^{(out)} \), indicating the estimations of \( u, v, x_0 \) and \( x_1 \), respectively, by the decoding algorithm. The messages are computed using the extrinsic information principle, i.e. each message that is sent from a node on an adjacent edge is a function of all the messages that were previously sent to the node, except the message that was received over the particular edge. The nodes of the graphs are denoted by \( a_0 \) (the adder functional) and \( e_1 \) (the equality functional). Using the ideas mentioned above we have the following computation rules.

\[
\begin{align*}
\mu_{e_1 \rightarrow a_0} &= f_{(1)}(\mu_v^{(in)}, \mu_{x_1}^{(in)}), \\
\mu_{a_0 \rightarrow e_1} &= f_{(+)}(\mu_u^{(in)}, \mu_{x_0}^{(in)}), \\
\mu_u^{(out)} &= f_{(+)}(\mu_{x_0}^{(in)}, \mu_{e_1 \rightarrow a_0}), \\
\mu_v^{(out)} &= f_{(1)}(\mu_{a_0 \rightarrow e_1}, \mu_{x_1}^{(in)}), \\
\mu_{x_0}^{(out)} &= f_{(+)}(\mu_{e_1 \rightarrow a_0}, \mu_u^{(in)}), \\
\mu_{x_1}^{(out)} &= f_{(1)}(\mu_{a_0 \rightarrow e_1}, \mu_v^{(in)}),
\end{align*}
\]

![Figure 4: Normal Factor Graph Representation of Arikan’s Construction](image-url)
where $f(z_0, z_1) = z_0 + z_1$ and $f(z_0, z_1) = 2 \tanh^{-1}(\tanh(z_0/2) \cdot \tanh(z_1/2))$. Note that, $\mu_{\alpha \rightarrow \beta}$ where $\alpha, \beta \in \{e_1, a_0\}$ is the message which is sent from node $\alpha$ to node $\beta$. $\mu^{(\text{out})}_{u,i}$ and $\mu^{(\text{out})}_{x_0}$ are sent from $a_0$ over the half edges corresponding to symbols $u$ and $x_0$, respectively. $\mu^{(\text{in})}_{v,i}$ and $\mu^{(\text{out})}_{x_1}$ are sent from $e_1$ over the half edges corresponding to symbols $v$ and $x_1$, respectively.

We, now, turn to give a recursive description of an iteration of the algorithm. The factor graph of the length $N$ code, has $\log_2 N$ layers. In each layer, there exist $N/2$ copies of the normal factor graph, that we depicted in Figure 4. Their organization can be implied from the recursive description in Figure 2. Therefore, for each layer, we have $N/2$ realizations of the input messages, output messages and inner messages (each one is corresponding to a different set of symbols and interconnect). To denote the frozen bits) is set to 0 before the first iteration. It is assumed that the inner messages are preserved over the half edges corresponding to symbols $u$ and $x_0$, respectively.

STEP I
Partition the llr vector into pairs of consecutive llr values \[
\{\left(\mu^{(\text{in})}_{x_0,i}, \mu^{(\text{in})}_{x_1,i}\right)\}_{i=0}^{N/2-1}.
\]
Compute the messages $\{\mu_{e_1 \rightarrow a_0,i}\}_{i=0}^{N/2-1}$ using (13). Compute the messages $\{\mu_{a_0,i}\}_{i=0}^{N/2-1}$ using (15) (Note that the two computations in this step can be combined to one computation).

STEP II
Give the vector $\{\mu_{u,i}\}_{i=0}^{N/2-1}$ as an input to the polar code BP iterative decoder of length $N/2$. Also provide the indices of the frozen bits from the first half of the codeword. Assume that the decoder outputs $\{\mu_{u,i}\}_{i=0}^{N/2-1}$ and the estimation of the information word.

STEP III
Compute the messages $\{\mu_{a_0 \rightarrow e_1,i}\}_{i=0}^{N/2-1}$ using (14). Compute the messages $\{\mu^{(\text{out})}_{v,i}\}_{i=0}^{N/2-1}$ using (16) (Note that the two computations in this step can be combined to one computation).

STEP IV
Give the vector $\{\mu^{(\text{out})}_{v,i}\}_{i=0}^{N/2-1}$ as an input to the polar code decoder of length $N/2$. Also provide to this decoder the indices of the frozen bits from the second half of the codeword. Assume that the decoder outputs $\{\mu^{(\text{in})}_{v,i}\}_{i=0}^{N/2-1}$ and the estimation of the information word of the second outer polar codeword of length $N/2$.

The information part may be concatenated to the information part of step II, to generate the decision on the information word after this iteration.

Compute the messages $\{\mu_{e_1 \rightarrow a_0,i}\}_{i=0}^{N/2-1}$ using (13). Compute the messages $\{\mu^{(\text{out})}_{x_0,i}\}_{i=0}^{N/2-1}$ and $\{\mu^{(\text{out})}_{x_1,i}\}_{i=0}^{N/2-1}$ using (17) and (18), respectively.

Any input message or inner message, unless given (by the channel output or by a prior knowledge on the frozen bits) is set to 0 before the first iteration. It is assumed that the inner messages are preserved between the iterations (and see a further discussion in the sequel).

To complete the recursive description of the algorithm, we need to consider the case of the length $N = 2$ code. Assume, that we get $\mu^{(\text{in})}_{x_0}, \mu^{(\text{in})}_{x_1}$ as the input values. Also, before the first iteration initialize \[
\mu^{(\text{out})}_{x_0,0} = \alpha \text{ and } \mu^{(\text{out})}_{x_1,0} = \beta \text{.}
\]
\[ w \in \{ u, v \} \]
\[ \mu^{(in)}_w = \begin{cases} 0, & \text{w is not frozen;} \\ (-1)^b \cdot \infty, & \text{u is frozen and equals b.} \end{cases} \] (19)

**STEP I**
Compute \( \mu_{e_1 \rightarrow a_0} \) according to (13).

**STEP II**
If \( u \) is not frozen, compute \( \mu^{(out)}_u \) according to (19), and make a hard decision on this bit, based on its sign.

**STEP III**
Compute \( \mu_{a_0 \rightarrow e_1} \) according to (14).

**STEP IV**
If \( v \) is not frozen, compute \( \mu^{(out)}_v \) according to (16), and make a hard decision on it, based on its sign.

Compute \( \mu^{(out)}_{x_0}, \mu^{(out)}_{x_1} \) according to (17), (18).

We should note that
\[ f(\pm \infty, z_1) = f(\pm \infty, \pm \infty) = \pm \infty \] (20)
\[ f(\pm \infty, z_1) = \pm z_1, \quad f(\pm \infty, \pm \infty) = \pm z_0. \] (21)

We further note, that for \( N = 2 \) length code, steps I and II can be combined to one operation, and similarly steps III and IV can be combined to one operation. Both of these combined steps are independent, so they may be performed in any order, or in parallel.

In this implementation, we assumed that there is a memory for storing messages of type \( \mu^{(in)}_u, \mu^{(in)}_v, \mu^{(in)}_{x_0}, \mu^{(in)}_{x_1}, \mu_{a_0 \rightarrow e_1} \), that were previously computed. This memory is dedicated for each realization of such messages, specifically, for each layer of the graph and for each \((u + v, v)\) normal subgraph, as in Figure 4. Actually, for this particular schedule, excluding \( \mu^{(in)}_v \), we do not need to save any message beyond the iteration boundary (this observation reduces the required memory consumption as we’ll see in the hardware implementation). The memory consumption of the algorithm is \( \Theta (N \cdot \log(N)) \). The running time is also \( \Theta (N \cdot \log(N)) \), assuming no parallelism is allowed.

In each iteration, we send one instance for each of the possible messages and for each \((u + v, v)\) block realization in the code, except for the \( \mu_{e_1 \rightarrow a_0} \) for which we send two messages (for all the layers, besides the last one). The full implementation may contain several iterations. The number of iterations may be fixed or set adaptively, which means that the algorithm continues until some consistency criteria are fulfilled. An example for such a criterion, is that the signs of the llr estimations for all the frozen bits agree with their know values (i.e. if all the frozen bits are set to zero, then \( \text{sign} \left( \mu^{(out)}_\gamma \right) > 0 \) of all the frozen bits, \( \gamma \)). In this case, one can stop an iteration in the middle by holding a counter in a similar way to the method that is usually used in BP decoding of LDPC codes using the check-node based serial schedules (see e.g. [24]). We note, however, that in the LDPC case, the consistency is manifested in the fact that all the parity check equations are satisfied.

In the next section we describe hardware architectures for the decoding algorithms we covered so far.

## 4 Recursive Descriptions of Hardware Architectures of Decoders for Arikan’s Construction

We now turn to study hardware architectures, that are inspired by the recursive decoding algorithms, which we presented in Section 3. This section covers hardware architectures for Arikan’s \((u + v, v)\)
construction. A generalization of this discussion to other kernels is presented in Section 5. We begin by the simple SC pipeline decoder (Subsection 4.1), and then progress to the more efficient SC line decoder (Subsection 4.2). Both of these designs were presented by Leroux et al. [14, 15] in a non recursive fashion. We finish by considering a BP line decoder (Subsection 4.3).

It is important to note that throughout the hardware discussion, our presentation is relatively abstract, emphasizing the important concepts and features of the recursive designs without dwelling into all the details. As such, the figures representing the block diagrams should not be considered as full detailed specifications of the implementation, but rather as illustrations that aim to aid the reader in the task of designing the decoder.

We usually prefer to use the same notation for signals array or registers arrays. Let \( u(0 : N - 1) \) be an \( N \) length signals array, then its \( i^{th} \) value is denoted by \( u(i) \). If \( v \) is a two dimensional array of \( M \) rows and \( N \) columns, we denote it by \( v(0 : M - 1, 0 : N - 1) \). Naturally, the \( i^{th} \) row of this array is denoted by \( v(i, 0 : N - 1) \), and it is a one dimensional array of \( N \) elements, of which the \( j^{th} \) element is denoted by \( v(i, j) \).

### 4.1 The SC Pipeline Decoder

A block diagram of the SC pipeline decoder for Arikan’s construction is depicted in Figure 5. The main ingredients of the diagram are listed below.

1. Processing Element (PE) - This is the basic computation unit of the decoder. It gets as input two channel llrs, an estimate of the \( u \) input for the \( (u + v, v) \) mapping and a control signal, \( c_u \), indicating whether to compute the llr of \( u (c_u = 0) \) or \( v (c_u = 1) \). Note, that the estimate of \( u \) is only needed in the latter case.

2. \( \lambda(0 : N - 1) \) - An array of \( N \) registers holding the llrs from the channels.

3. SC decoding unit of polar code of length \( N/2 \) - This unit has the following inputs: \( N/2 \) length signals array of input llrs and a binary signals array containing the indices of the frozen bits of the code. Its outputs are \( \tilde{u}(0 : N/2 - 1) \), which is the estimation of the transmitted information word (including the frozen bits), and \( \tilde{x}(0 : N/2 - 1) \), which is the estimation of the transmitted codeword.
4. A register for the estimated information word \( u(0 : N - 1) \).

5. Encoding unit for generating the estimated codeword, it includes a register for the codeword \( x(0 : N - 1) \) and \( N/2 \) bitwise xor circuits for generating the codeword based on the output of the \( N/2 \) length decoder.

We note that a basic \( N = 2 \) length decoder has only one PE, and operates according to the algorithm described in Section 2. The algorithm for \( N > 2 \) is based on the notion of the recursion, as we describe below.

| STEP I | Using the processing elements \( PE_0, PE_1, \ldots, PE_{N/2-1} \) with \( c_u = 0 \), prepare the llr input for the decoder of the first \( N/2 \) length outer code and output it on the signals array \( L(0 : N/2 - 1) \), such that 
\[
L(k) = 2 \tanh^{-1}\left( \tanh(\lambda(2k)/2) \tanh(\lambda(2k+1)/2) \right), \quad 0 \leq k \leq N/2 - 1.
\]

| STEP II | Give the signals array \( L(0 : N/2 - 1) \) and the list of indices corresponding to the first half of the codeword (i.e. the first outer code) as inputs to the polar code decoder of length \( N/2 \).

Call the decoder of length \( N/2 \) polar code on these inputs (decoding the first outer polar code).

Store \( u(0 : N/2 - 1) = \hat{u}(0 : N/2 - 1), x_{even}(0 : N/2 - 1) = \hat{x}(0 : N/2 - 1) \).

| STEP III | Using the signals array \( \hat{x}(0 : N/2 - 1) \) as the vector of estimations of \( u \) from the \( (u + v, v) \) pair, operate the processing elements \( PE_0, PE_1, \ldots, PE_{N/2-1} \) with \( c_u = 1 \). This will prepare the llr input for the second outer code, and output it on signals array \( L(0 : N/2 - 1) \), such that 
\[
L(k) = (-1)^{\hat{x}(k)} \lambda(2k + \lambda(2k+1)), \quad 0 \leq k \leq N/2 - 1.
\]

| STEP IV | Give the signals array \( L(0 : N/2 - 1) \) as an input to the polar code decoder of length \( N/2 \). Also provide the indices of the frozen bits corresponding to the second half of the codeword (i.e. the second outer code).

Call the decoder of the length \( N/2 \) polar code on these inputs (which means that we decode the second outer polar code).

Store \( u(N/2 : N - 1) = \hat{u}(0 : N/2 - 1), x_{even}(0 : N/2 - 1) = x_{even}(0 : N/2 - 1) + \hat{x}(0 : N/2 - 1), x_{odd}(0 : N/2 - 1) = \hat{x}(0 : N/2 - 1) \).

Here, for an array \( x \), we denote by \( x_{even} \) and \( x_{odd} \) the 2-decimated arrays containing \( x \)'s even indices samples and odd indices samples, respectively. Note, that to avoid any delays due to sampling by a register, it is important that the codeword estimation (which is one of the outputs of the decoder) will be the output of the encoding layer and not the register following it. This issue and further timing concerns are considered in the next subsection.

Let us consider the complexity of this circuit. We assume that a call to a PE finishes in one clock cycle. Denote by \( T(n) \) the time (in terms of the number of clock cycles) that is required to complete the decoding of \( N = 2^n \) length polar code. Then, \( T(n) = 2^2 \cdot T(n - 1) \) \( n > 1 \) and \( T(1) = 2 \). This recursion yields \( T(n) = 2^N - 2 \). Denote by \( P(n) \) the number of PEs for a decoder of length \( N = 2^n \) polar code, we have \( P(n) = 2^n - 1 = N - 1 \).
unit is of \(2 \cdot \sum_{i=1}^{n} 2^i = 4 \cdot (N-1)\) bits registers, and \(\sum_{i=0}^{n-1} 2^i = N - 1\) xor circuits. We should have \(R(n)\) registers for holding llr values, so \(R(n) = 2^n + R(n-1)\) \(n > 1\) and \(R(1) = 2\), so \(R(n) = 2 \cdot P(n) = 2N - 2\). Note, that in this design, we assume that the re-encoding unit is a combinatorial circuit.

### 4.2 The SC Line Decoder

In the pipeline design of the decoder of length \(N\), the \(N/2\) processing elements \(\{PE_k\}_{k=0}^{N/2-1}\), are only used during steps I and III of the algorithm. During the other steps (that ideally consumes \(2 \cdot T(n-1) = 2N - 4\) clock cycles of the total \(2N - 2\)), these processors are idle, and this results in an inefficient design. To improve this, we observe that the maximum number of operations that can be done in parallel by the PEs in the SC decoding algorithm is \(N/2\). So, in order to allow this maximum level of parallelism, a design must have at least \(N/2\) processors. The line decoder\(^2\) that we describe in this subsection, achieves this lower-bound. In order to support this, we need to redefine the decoder of length \(N\) polar code.

First, The line decoder has two operation modes.

**Standard Mode (S-Mode)**

In this mode, the decoder gets as inputs llrs and the indices of the frozen bits, and outputs the hard decision on the information word and its corresponding codeword (this is the operation mode we assumed so far).

**PE-Array Mode (P-Mode)**

In this mode, the decoder gets as input a signals array of llrs \(\lambda(0 : N-1)\), a control signal \(c_u\), and a binary array of length \(N/2\), \(z(0 : N/2 - 1)\). The output is a signals array \(L(0 : N - 1)\) of llrs, where for \(0 \leq k \leq N/2 - 1\)

\[
L(k) = \begin{cases} 
2 \cdot \tanh^{-1}\left(\tanh\left(\lambda(2k)/2\right) \cdot \tanh\left(\lambda(2k+1)/2\right)\right), & c_u = 0; \\
(-1)^{z(k)} \cdot \lambda(2k) + \lambda(2k+1), & c_u = 1.
\end{cases}
\]

In Figure 6, we give a block diagram of this decoder. Note, that in order to maintain the maximum level of parallelism, the length \(N\) polar code decoder has \(N/2\) processors. Thus, in order to build the length \(N\) polar code decoder using an embedded \(N/2\) length polar code decoder (already having \(N/4\) processors), we use an additional array of \(N/4\) PEs, which is referred to as the auxiliary array. The input signal \(modeIn\) indicates wether the decoder is used in S-Mode or in P-Mode. The \(mode\) signal is an internal signal that controls wether the \(N/2\) length embedded decoder is in P-Mode.

The algorithm for the S-Mode is described below.

#### STEP I

Simultaneously,

- At the multiplexers array (MUX array), at the input of the embedded decoder of length \(N/2\) polar code, set the control signal \(c_m = 0\), which means that the array \(\lambda(0 : N/2 - 1)\) is selected as an input to this unit. Set \(c_u = 0\) and use the decoder of length \(N/2\) polar code in P-Mode, which causes this unit to output the signals array \((0 \leq k \leq N/4 - 1)\)

\[
L(k) = 2 \cdot \tanh^{-1}\left(\tanh\left(\lambda(2k)/2\right) \cdot \tanh\left(\lambda(2k+1)/2\right)\right).
\]

Store this array in the registers array \(R(0 : N/4 - 1)\).

---

\(^2\) We note, that the original line decoder, which was presented by Leroux et al. [15, Section 3.3] is not precisely the same design, which we discuss here. The differences, appear to be minor (especially in the area of the routing between the llr registers and the PEs), so we preferred not to distinguish it from Leroux’s design by giving it another name.
Figure 6: Block Diagram for the SC Line Decoder
• Use the auxiliary array of processors and compute for $N/4 \leq k \leq N/2 - 1$

\[ L(k) = 2 \cdot \tanh^{-1} (\tanh (\lambda(2k)/2) \cdot \tanh (\lambda(2k+1)/2)) \]

and store them in the registers array $R(N/4 : N/2 - 1)$.

**STEP II**

• At the MUX array, at the input of the decoder of the length $N/2$ polar code, set the control signal $c_m = 1$, which means that content of the registers array $R(0 : N/2 - 1)$ is selected as an input to this unit.

• Provide the vector of indices, corresponding to the frozen bits from the first half of the codeword to the $N/2$ length decoder.

Call the decoder of the length $N/2$ polar code in S-Mode on these inputs (decoding of the first outer polar code).

Store $u(0 : N/2 - 1) = \hat{u}(0 : N/2 - 1)$, $x_{even}(0 : N/2 - 1) = \hat{x}(0 : N/2 - 1)$.

**STEP III**

Simultaneously,

• At the MUX array, at the input of the embedded decoder of length $N/2$ polar code, set the control signal $c_m = 0$, which means that the array $\lambda(0 : N/2 - 1)$ is selected as an input to this unit.

Set $c_u = 1$ and use the decoder of length $N/2$ polar code in P-Mode, which causes this unit to output the signals array $(0 \leq k \leq N/4 - 1)$

\[ L(k) = (-1)^{\hat{x}(k)} \cdot \lambda(2k) + \lambda(2k+1). \]

Store this signals array in the registers array $R(0 : N/4 - 1)$. Note, that we use $\hat{x}(0 : N/4 - 1)$, the estimation of the first half of the codeword, that the embedded decoder gave as output in step II, as an input to this unit.

• Use the auxiliary array and compute for $N/4 \leq k \leq N/2 - 1$

\[ L(k) = (-1)^{\hat{x}(k)} \cdot \lambda(2k) + \lambda(2k+1) \]

and store them in the registers array $R(N/4 : N/2 - 1)$.

**STEP IV**

At the MUX array, at the input of the decoder of the length $N/2$ polar code, set the control signal $c_m = 1$, which means that the array of registers $R(0 : N/2 - 1)$ is selected as an input to this unit.

Provide to the $N/2$ length decoder, the vector of indices, corresponding to the frozen bits of the second half of the codeword.

Call the decoder of the length $N/2$ polar code in S-Mode on these inputs (decoding of the second outer polar code).

Store $u(N/2 : N - 1) = \hat{u}(0 : N/2 - 1)$, $x_{even}(0 : N/2 - 1) = x_{even}(0 : N/2 - 1) + \hat{x}(0 : N/2 - 1)$, $x_{odd}(0 : N/2 - 1) = \hat{x}(0 : N/2 - 1)$.

The P-Mode operation of the decoder is quite simple. Use $c_m = 0$, which means that the channel llrs are given as an input to the line decoder of the $N/2$ length polar code. Also provide as input the vector $x_{in}(0 : N/2 - 1)$, that will serve here as estimations of the $u$ bits from the $(u + v, v)$ pairs. Set $c_u = c_{u,in}$, and operate simultaneously the auxiliary array of processors and the line-decoder of length $N/2$ in P-Mode. Return the llr output of the line-decoder of length $N/2$ and the auxiliary array, i.e. the
signals array $L(0:N/2-1)$.

We now analyze the complexity of the decoder. Let $P(n)$ be the number of processors of the $N = 2^n$ decoder. Then, $P(n) = 2^{n-2} + P(n-1)$ $P(1) = 1$, so $P(n) = 2^{n-1} = N/2$. The number of registers, we use in the design for the llrs (not including registers for the input and the encoding registers) are $R(n) = 2^{n-1} + R(n-1)$, $R(1) = 1$, so we have $R(n) = 2^n - 1 = N - 1$. The number of multiplexers for the llrs is denoted by $M(n) = 2^{n-1} + M(n-1)$ $M(1) = 0$, so $M(n) = N - 2$.

We want to make a remark about the efficiency of the design we propose here. The recursive design has a potential advantage of being a clearer reflection of the underlined algorithm. It also has the potential advantage of emphasizing the parts of the system that may be reused. However, it may have a disadvantage when considering the routing of signals in the circuit. Because we want to use the decoder of the $N/2$ length polar code as a closed box, we route all the signals from it and to it, using its interface. This may result in some signals traversing a long path before reaching their target processor. These paths may be too long for the circuit to have a good clock frequency, thereby resulting in degradation of the achievable throughput. It is therefore advised to optimize the circuit by "opening" the recursive units and making the paths shorter, after completing the design of the circuit in a recursive manner. It will also be a good idea, that when building a decoder for a $2N$ length code, the designer will use this "optimized" design of the $N$ length decoder in the $2N$ length design, enjoying the benefits of the recursion. We give here two examples of these long paths hazards, that we believe that are likely to pose a problem along with their possible solutions.

1. The multiplexers layer at the input of the embedded line decoder of the length $N/2$ code is required because of the introduction of the P-Mode. A closer look of our design, reveals that some of the signals have long paths before reaching their target PE. For example, the inputs $\lambda_0$ and $\lambda_1$ need to traverse $\log_2(N) - 1$ multiplexer layers before reaching their processor. Since the P-Mode needs to be accomplished in one time-unit, this long path may be prohibitive. By "opening" the $N/2$ length decoder box, the designer is able to control the lengths of the paths by a proper routing.

2. The "Encoding Layer" also suffers from long routing. We assumed, in our analysis, that the encoding procedure is combinatorial, and therefore can be done within the clock cycle. This may be a problem when several encoding circuits are operated one after the other. This is, for example, the case of step IV of the decoder of length $N/2^i$ code, that occurs within the step IV of the decoder of length $N/2^{i-1}$ code for $1 \leq i \leq \log_2 N - 2$. In this case, $O(\log N)$ operations need to occur in a sequential manner in one clock cycle. For large $N$ and high clock frequency circuit, this may not be feasible. The idea of Leroux et al. was to use flip-flops for saving the partial encoding for each code bit in the different layers of the decoding circuit. Each such flip-flop, is connected using a xor circuit to the signal line of the estimated information bit. As such, whenever the SC decoder decides on an information bit, the flip-flops corresponding to the code bits that are dependent on this information bit are updated accordingly. These flip-flops need to be reset whenever we start decoding their corresponding outer-code. For example, when we start using the embedded $N/2$ length decoder (on step II and step IV) its flip-flops of partial encoding need to be erased (as they correspond to new outer code).

It should be noted, that this idea may also be described recursively, by changing the specification of the length $N$ polar code decoder in S-mode, and requiring it to output the estimated information bits as soon as they’re ready. The decoder should also have an $N$ length binary indicator vector, that indicates which code bits is dependent on the currently estimated information bit. It is easy to see that using the indicator vector of the length $N/2$ decoder, it is possible to calculate the $N$ length indicator vector, by using the $(u + v, v)$ mapping. This, however, generates again a computation path of length $\Theta(\log N)$. This problem, can be addressed, by having a fixed indicator circuit for each partially encoded-bit flip-flop. This circuit will indicate which information bit should be accumulated depending on the ordinal number of this bit. For example, for the decoder of the code of length $N$, we should have an array of $N/2$ flip-flops, each one corresponds to a bit of the codeword of the $N/2$ length first outer code. Each one of these flip-flops, should have an indicator circuit, that
it is possible to have parallel codewords simultaneously. Because the auxiliary array is only used on steps I and III, and is idle on the other steps. This might motivate us to consider two variations on this design. The first one, adds hardware and use these arrays to increase the throughput, while the second one decreases the throughput and thereby reduces the required hardware.

4.2.1 Parallel Decoding of Multiple Codewords

There are cases that it is required to increase the throughput of the decoder, by allowing parallel decoding of multiple codewords. A simple solution is to introduce $p$ decoders when there is a need for decoding $p$ codewords simultaneously. Because the auxiliary array of processors is idle most of the time, it seems like a good idea to "share" this array among several decoders. By appropriately scheduling the commands to the processors, it is possible to have an implementation of a decoder for $p$ parallel codewords which is less expensive than just duplicating the decoders (the naive solution).

Since the array is idle during steps II and IV, in which the decoder of the length $N/2$ code is active, it is possible to have $p \leq T(n-1) + 1 = N - 1$ decoders sharing the same auxiliary array. The decoding of each one of them is issued in a delay of one clock cycle from each other. Assuming the that $p = N - 1$, we have a decoding time $T(n) + N - 2 = 3N - 4$ for $N - 1$ codewords while having $p \cdot P(n - 1) + N/4 = (N - 1)(N - 2) + N/4$ processors, which is about half of the number of processors in the naive solution.

This notion can be developed further. For the decoder of the length $N/2$ code that is embedded in the $N$ length decoder, there is a an auxiliary array of $N/8$ processors. This auxiliary array is used on steps I and III of the decoder of length $N$ and length $N/2$. Therefore, it is idle most of the time, and we can share it among the $p$ decoders of length $N/2$. Assuming that $p = N - 1$, we may allocate 3 auxiliary arrays that will be shared among the decoders, each one is dedicated for one of these different steps: one array for step I (and III) of the $N$ length decoder, one array for step I of the $N/2$ length decoder and one array for step III of the $N/2$ length decoder. For each of the decoded codewords the number of clock cycles between these steps is at least $p$, therefore there will be no contention on these resources and the throughput will not suffer because of this hardware reduction.

In general, for $p = N - 1$, the auxiliary array within the embedded decoder of length $N/2$ polar decoder ($i \in [1, \log_2(N) - 2]$), can be shared among the $p$ decoders, provided that we allocate an instance of the array for each of the decoding steps it is used in, during the first half of the decoding algorithm for the length $N$ code (i.e. for the time of steps I and II). Thus, for this specific array, we have 1 call in step I of the $N$ length decoder, 1 call for step I and 1 call for step III of the $N/2$ length decoder, 2 calls for step I and 2 calls for step III of the $N/2$ length algorithm, ..., $2^i$ calls for step I and $2^i$ calls for step III for the length $N/2^i$ decoder. In summary, we need $\sum_{i=0}^{\log_2(N)} 2^i = 2^{i+1} - 1$ auxiliary arrays of processors, each one contains $N/2^i$ PEs. In particular, we need $N - 1$ PEs for the 2 length decoder (each PE is allocated to a
specific decoder), and \( \frac{N}{2} \cdot \sum_{i \geq 0} \log_2(N)^{-2} 2^{i+1} \approx \frac{N}{2} (\log_2(N) - 1) \) PEs for the other decoders lengths. This adds up to approximately \( \frac{N}{2} (1 + \log_2(N)) \) PEs. We conclude that this solution allows an increase of the throughput in a multiplicative factor of \( N \), while the PEs hardware is only increased by an approximately \( \log_2(N) \) factor. Note, that the number of registers should increase by a multiplicative factor of \( p \).

A closer look at the above hardware design, reveals that we actually allocated for each sub-step of steps I and II of the \( N \) length decoder a different array of processors. The decoding operation of the \( p \) codewords will go through these units in a sequential order. However, each decoder should have its own set of registers saving the state of the decoding algorithm. Another observation is that when we finish decoding the first codeword (i.e. the one we started decoding in time 0), we can start decoding codeword number \( N \) in the next time slot (and then codeword number \( N+1 \), etc.), in a pipelined fashion. It should be noted that Leroux et al. considered a similar idea, and referred to it as the vector-overlapping structure [13].

### 4.2.2 Limited Parallelism Decoding

Another approach for addressing the problem of low utilization of the auxiliary arrays is to limit the number of processing elements that may be allowed to operate simultaneously. This is a very practical consideration, as typically, a system designer has a parallelism limitation which is due to power consumption and silicon area. The limited parallelism, inevitably results in an increase of the decoding time, and thereby a decrease of the throughput. The line decoder of the code of length \( N \) has a PE parallelism of \( \frac{N}{2} \), because it may simultaneously compute at most \( \frac{N}{2} \) llrs using the \( \frac{N}{2} \) PEs.

We consider a line decoder of length \( N \) code with limited parallelism of \( \frac{N}{2^i} \), where \( i \in [1, \log_2 N] \). This means, that the decoder has exactly \( \frac{N}{2^i} \) PEs. If \( i = 1 \) then the decoder is actually, the standard line-decoder. If \( i > 1 \) then the decoder’s block diagram will be similar to the one shown in Figure 6, with the following changes.

- There will be no auxiliary PEs array.
- The embedded line decoder of the \( N/2 \) length code will be replaced by a limited parallelism line decoder, with parallelism factor of \( N/2^i \).
- The signals array \( L(0 : N/4 - 1) \) at the output of the embedded line decoder will also be connected to the registers array \( R(N/4 : N/2 - 1) \).
- The multiplexers array at the the input of the \( N/2 \) length line decoder, will change to also include the input array \( \lambda(N/2 : N - 1) \). This means, that we should have an array of \( 3 \rightarrow 1 \) multiplexers (instead of \( 2 \rightarrow 1 \)), in which the \( k^{th} \) multiplexer selects between inputs \( \lambda(k), \lambda(k+N/2) \) and \( R(k) \).
- There will be an additional array of multiplexers at the input of the line-decoder for selecting between \( x(0 : N/4 - 1) \) and \( x(N/4 : N/2 - 1) \), to support the use of both parts of the decided codeword. Similarly, for the P-Mode, we should have an array of multiplexers to select between the two parts of the \( x_{in}(0 : N/2 - 1) \) array.

The S-mode decoding algorithm will have 4 steps as before, however steps I and III are modified as follows.

#### STEP I

Sequentially,

- **STEP I-a**: At the MUX array, at the input of the (limited parallelism) decoder of the length \( N/2 \) polar code, set the control signal \( c_m = 0 \), which means that \( \lambda(0 : N/2 - 1) \) is selected as an input to this unit.

Set \( c_u = 0 \) and use the \( N/2 \) length polar code decoder in P-Mode. Store the output
array of signals $L(0 : N/4 - 1)$, corresponding to

$$L(k) = 2 \cdot \tanh^{-1} \left( \tanh \left( \frac{\lambda(2k)}{2} \right) \cdot \tanh \left( \frac{\lambda(2k + 1)}{2} \right) \right) , \quad 0 \leq k \leq N/4 - 1$$

in the registers array $R(N/4 : N/2 - 1)$.

- **STEP I-b**: Set the control signal of the MUX array to $c_m = 1$, which means that $\lambda(N/2 : N - 1)$ is selected as an input to this unit. Set $c_u = 0$ and use the decoder of the polar code of length $N/2$ in P-Mode. Store the output signals array

$$L(k - N/4) = 2 \cdot \tanh^{-1} \left( \tanh \left( \frac{\lambda(2k)}{2} \right) \cdot \tanh \left( \frac{\lambda(2k + 1)}{2} \right) \right) , \quad N/4 \leq k \leq N/2 - 1$$

in registers array $R(N/4 : N/2 - 1)$.

### STEP III

Sequentially,

- **STEP III-a**: At the MUX array, at the input of the (limited parallelism) decoder of length $N/2$ polar code, set the control signal $c_m = 0$, which means that $\lambda(0 : N/2 - 1)$ is selected as an input to this unit.

Set $c_u = 0$ and use the $N/2$ length polar code decoder in P-Mode. Store the output array of signals $L(0 : N/4 - 1)$, corresponding to

$$L(k) = (-1)^{\tilde{x}(k)} \cdot \lambda(2k) + \lambda(2k + 1) , \quad 0 \leq k \leq N/4 - 1$$

in the registers array $R(N/4 : N/2 - 1)$. Note that we use $\tilde{x}(0 : N/4 - 1)$, the first half of the output from step III, as an input to the $N/2$ length decoder.

- **STEP III-b**: Set the control signal of the MUX array to $c_m = 1$, which means that $\lambda(N/2 : N - 1)$ is selected as an input to this unit. Set $c_u = 0$ and use the decoder of the polar code of length $N/2$ in P-Mode. Store the output signals array

$$L(k - N/4) = (-1)^{\tilde{x}(k)} \cdot \lambda(2k) + \lambda(2k + 1) , \quad N/4 \leq k \leq N/2 - 1$$

in registers array $R(N/4 : N/2 - 1)$. Note that we, now, use $\tilde{x}(N/4 : N/2 - 1)$, the second half of the output from step III, as an input to the $N/2$ length decoder.

The P-Mode operation of the decoder is also changed, and now contains two steps.

### STEP I

At the MUX array, at the input of the (limited parallelism) polar code decoder of length $N/2$ set the control signal $c_m = 0$, which means that $\lambda(0 : N/2 - 1)$ is selected as an input to this unit. Set $c_u = c_{u,\text{in}}$ and use the $N/2$ length polar code decoder in P-Mode. Store the output signals array $L(0 : N/4 - 1)$ in the registers array $R(0 : N/4 - 1)$. If $c_{u,\text{in}} = 1$, use the first half of the input signals array $x_{in}$ (i.e. $x_{in}(0 : N/4 - 1)$) as an input to the $N/2$ length decoder (otherwise this input is ignored).

### STEP II

Set the control signal of the MUX array to $c_m = 1$, which means that $\lambda(N/2 : N - 1)$ is selected as an input to this unit. Set $c_u = c_{u,\text{in}}$ and use the $N/2$ length polar code decoder in P-Mode. Store the output signals array $L(0 : N/4 - 1)$ in the registers array $R(N/4 : N/2 - 1)$. If $c_{u,\text{in}} = 1$, use the second half of the input signals array $x_{in}$ (i.e. $x_{in}(N/4 : N/2 - 1)$) as an input to the $N/2$ length decoder (otherwise this input is ignored).
The output of the decoder is the array of signals corresponding to the array of registers $R(0 : N/2 - 1)$.

Let’s analyze the time complexity of this algorithm. We denote the S-Mode running time (in terms of clock cycles) for length $N = 2^n$ polar code with limited parallelism of $N/2^i = 2^{n-i}$, by $T(n, n-i)$. We note that $T(n, n-i) = T(n)$, where $T(n) = 2N - 2$ is the running time of the standard line decoder. The recursion formula is

$$T(n, n-i) = 2 \cdot T(n-1, n-i) + 4 \cdot T_p(n-1, n-i), \quad (23)$$

where $T_p(n, m)$ is the running time of the $N = 2^n$ length decoder with $2^m$ limited parallelism in P-Mode.

$$T_p(n, m) = \begin{cases} 
1, & n - m \leq 1; \\
2 \cdot T_p(n-1, m), & \text{otherwise}. 
\end{cases} \quad (24)$$

Therefore,

$$T_p(n, m) = \begin{cases} 
1, & n - m \leq 1; \\
2^{n-m-1}, & \text{otherwise}. 
\end{cases} \quad (25)$$

It can be shown that

$$T(n, n-i) = 2 \cdot N + (i-2) \cdot 2^i, \quad i \geq 1. \quad (26)$$

Equation (26) reveals the tradeoff between the number of PEs and the running time of the algorithm. For example, decreasing the number of processors by a multiplicative factor of 8, compared to the standard case (i.e. $i = 4$), results in an increase of only 34 clock cycles in the decoding time. We note however, that to build such a decoder, additional control hardware (e.g. multiplexers layers) should be designed.

It seems that for a limited list size, the Successive Cancellation List decoder may also be implemented by a line decoder. This requires to duplicate the hardware by the size of the list, $M$, and to introduce the appropriate logic (i.e. comparators and multiplexer layers). It is possible to provide an implementation with $O(f(M) \cdot N)$ time complexity, where $f(\cdot)$ is a polynomially bounded function, that is dependent on the efficiency of algorithms for selection of $M$ most likely paths (in the $N = 2$ decoder). Furthermore, the normalization of the likelihoods should be considered carefully, and also should have its impact on the precise (i.e. non asymptotic) time complexity.

### 4.3 The BP Line Decoder

As we saw in Subsection 3.3, BP is an iterative algorithm, in which messages are sent on the normal factor graph representing the code. In this subsection, we consider an implementation of the BP decoder with the GCC serial schedule. The proposed decoder structure is a variation on the recursive structure of the SC line decoder. Figure 7 depicts a block diagram for this design. The main changes, in respect to the GCC serial schedule. The proposed decoder structure is a variation on the recursive structure of the graph representing the code. In this subsection, we consider an implementation of the BP decoder with

$$4.3 \quad \text{The BP Line Decoder}$$

As we saw in Subsection 3.3, BP is an iterative algorithm, in which messages are sent on the normal factor graph representing the code. In this subsection, we consider an implementation of the BP decoder with the GCC serial schedule. The proposed decoder structure is a variation on the recursive structure of the SC line decoder. Figure 7 depicts a block diagram for this design. The main changes, in respect to the SC decoder, are in the memory resources and the processor structure.

The memory plays a fundamental role in the design, as it helps keeping computed messages within the iteration boundary and beyond it. The basic requirement is that each "butterfly" realization of the $(u + v, v)$ factor graph, should have memory resources to store its messages. To allow messages to be kept within the iteration boundary, it is only required to have one registers array for each length of outer code and for each message type. However, the need for keeping a message beyond the iteration boundary requires a dedicated memory array for each instance of the outer code.

In the case of the $(u + v, v)$ code and the GCC schedule, only messages of type $\mu_v^{(in)}$ need to be kept beyond the iteration boundary. We suggest to address this requirement, in the following way. For the decoder of length $N$, we associate a registers matrix $\mu_v^{(in)}(0 : \#_r(N) - 1, 0 : N/2 - 1)$. Here, $\#_r(N)$ is the number of realizations of factor graphs corresponding to outer codes of size $N$ that exist in our code. For the code of length $N$, there is only one factor graph of this size (i.e. the entire graph), and therefore for this decoder $\#_r(N) = 1$.

Consider, now, the $N/2$ length decoder that is embedded within the $N$ length decoder. We see in Figure 7 that this decoder has its number of realizations as $2 \cdot \#_r(N)$, i.e. for the $N$ length decoder we have
Figure 7: Block Diagram for the BP Line Decoder
\#_r(N/2) = 2. This is because we have two outer codes of length \(N/2\) in the \(N\) length code. Therefore, the memory matrix associated with it has two rows and \(N/4\) columns. The first row is dedicated for the first realization of the outer code and the second row is dedicated for the second realization. Within this \(N/2\) length decoder, there is an embedded \(N/4\) length decoder with \(2 \cdot \#_r(N/2)\) realizations, so in this case \(\#_r(N/4) = 4\). As a result, it has a registers matrix with 4 rows and \(N/8\) columns (each row is dedicated to one of the 4 outer codes of length \(N/4\) in this GCC scheme). This development continues, until we reach the embedded decoder of length 2, which, by induction, has \(\#_r(2) = N/2\) realizations for the \(N\) length decoder, so it requires a registers matrix with \(N/2\) rows and one column.

For a correct operation of the decoder, it is required to inform the embedded decoders to which realization of the outer code’s factor graph they are currently referring to. This is the role of the signals realizationID\(_{N/2}\), realizationID\(_N\) and OuterCodeID, that indicate the specific realization as follows. The signal realizationID\(_N\) notifies the decoder of length \(N\), on which realization of the factor graph of the code of length \(N\) it is working on. Note, that because we describe here a decoder for a code of length \(N\), we have only one realization of this graph, therefore this signal is fixed to 0. However, if this was an embedded decoding unit within a larger length code decoder, then this signal should indicate the ordinal number of the outer code we’re decoding, ranging from 0 to \(\#_r(N) - 1\). The signal realizationID\(_{N/2}\) gives the identification of the realization of the outer polar code of length \(N/2\). It is computed as \(2 \cdot \) realizationID\(_N\) + OuterCodeID, where OuterCodeID equals 0 on step II, and equals 1 on step IV of the iteration for the \(N\) length decoder.

We also need to have registers arrays for the messages of type \(\mu_{e_1 \to a_0}, \mu_{a_0 \to e_1}, \mu_u^{(in)}, \mu_u^{(out)}\) and \(\mu_v^{(out)}\), each one of them of length \(N/2\). We denote them by \(\mu_{e_1 \to a_0}(0 : N/2 - 1), \mu_{a_0 \to e_1}(0 : N/2 - 1), \mu_u^{(in)}(0 : N/2 - 1), \mu_u^{(out)}(0 : N/2 - 1)\) and \(\mu_v^{(out)}(0 : N/2 - 1)\). Note, that as opposed to the memory structure for the \(\mu_u^{(in)}\) messages, these arrays do not need to be available beyond the iteration boundary, therefore it suffices to have them as arrays and not matrices. Furthermore, the arrays for messages \(\mu_{e_1 \to a_0}, \mu_u^{(out)}\) and \(\mu_v^{(out)}\), can be replaced by one temporary array. However, in the description of the hardware structure, we chose not to do this, in order to keep the discussion more comprehensible.

Figure 8, depicts the processing element \(BP\_PE\) that is considered here. This unit has two inputs for message \(llrs\), and depending on the control signal \(c_{BP\_PE}\) it performs either the \(f_{(+)}(\cdot, \cdot)\) function or...
the \( f(m) \). Because it has to implement the functionalities of equations (13)-(18), we introduce routing layers for the inputs (OP-MUX) and the outputs (OP-De-MUX) that ensure that the proper inputs will be given to the processor and that its output is stored in the appropriate array, depending on the computation schedule of the iteration.

Besides the messages that serve as inputs or outputs to the processor, we allocate two additional message inputs, denoted by \( ext_a \) and \( ext_b \), and one additional output message, denoted by \( ext\text{out} \). These inputs and output are used during the P-Mode of the decoder. We note, that in Figure 7, we preferred, for brevity, not to specify these routing units for each processor, but rather to group them into routing arrays. The inputs and outputs to these routing arrays are arrays of inputs and outputs corresponding to the types of inputs and outputs that appear in Figure 8. The convention is that in these routing arrays, the \( i^{th} \) output corresponds to the \( i^{th} \) input from each signals array (the signals array is selected by the control signal of the routing array). Moreover, the \( i^{th} \) output of the OP-MUX array corresponds to the consecutive \( i^{th} \) processor from the array of processors it serves. Similarly, the \( i^{th} \) input of the OP-De-MUX array corresponds to the \( i^{th} \) consecutive processor from the array of processors it serves.

As in the SC case, the BP line decoder has two operation modes.

- **S-Mode** - The decoder gets as input \( \mu^{(in)} \) type of messages referring to its inputs. It outputs, \( \mu^{(out)} \) type of messages, corresponding to its inputs (i.e. messages, that are sent from the subgraph which realization the decoder is operating on, to its neighbors) and estimation of the information word vector (denoted by \( \text{infoEst} \)).

- **P-Mode** - The decoder serves as an array of \( N/2 \) processors and performs simultaneously the computation of the type of message indicated by \( C_{PPPE,\text{external}} \) using signals \( ext_a \) and \( ext_b \) as the inputs and \( ext\text{out} \) as the output.

The S-Mode decoding algorithm operates as follows.

**STEP I**

Simultaneously,

- At the MUX array, at the input of the decoder of the code of length \( N/2 \), set the control signal \( c_m = 0 \), which means that the OP-MUX array is selected as the input to the decoder. Set \( c_{\text{opMUX}} \) such that \( \mu_v^{(in)}(0 : N/4 - 1) \) and \( \mu_{x_1}^{(in)}(0 : N/4 - 1) \) will be selected as the first input and the second input, respectively of this unit.
- Set \( c_{BPPE,\text{internal}} \) to correspond to the computation of (13) and use the \( N/2 \) length polar code decoder in P-Mode. Set the OP-De-MUX array to direct the output to \( \mu_{e_1 \rightarrow a_0}(0 : N/4 - 1) \).
- Having the same values for \( c_{\text{opMUX}} \) and \( c_{BPPE,\text{internal}} \), use the auxiliary array of processors to operate on the inputs \( \mu_v^{(in)}(N/4 : N/2 - 1) \) and \( \mu_{x_1}^{(in)}(N/4 : N/2 - 1) \) and have the output directed to \( \mu_{e_1 \rightarrow a_0}(N/4 : N/2 - 1) \).

Simultaneously,

- Keep \( c_m = 0 \). Set \( c_{\text{opMUX}} \) such that \( \mu_v^{(in)}(0 : N/4 - 1) \) and \( \mu_{e_1 \rightarrow a_0}(0 : N/4 - 1) \) will be selected as the first input and the second input, respectively to the \( N/2 \) length decoder. Set \( c_{BPPE,\text{internal}} \) to correspond to the computation of (15) and use the decoder of length \( N/2 \) in P-Mode. Set the OP-De-MUX array to direct the output to \( \mu_u^{(out)}(0 : N/4 - 1) \).
- Having the same values for \( c_{\text{opMUX}} \) and \( c_{BPPE,\text{internal}} \), use the auxiliary array of processors to operate on the inputs \( \mu_{x_0}^{(in)}(N/4 : N/2 - 1) \) and \( \mu_{e_1 \rightarrow a_0}(N/4 : N/2 - 1) \) and have the output directed to \( \mu_u^{(out)}(N/4 : N/2 - 1) \).
STEP II

- At the MUX array, at the input of the BP decoder of the $N/2$ length polar code, set the control signal $c_m = 1$, which means that the input from the second multiplexer is selected as input to this unit. Specifically, since $OuterCodeID = 0$ it means that $\mu_u^{(\text{out})}(0 : N/2 - 1)$ is the input to this decoder.

- Provide the indices of the the frozen bits from the first half of the codeword to the $N/2$ length decoder, and operate it in S-Mode. Store the estimation of the information word (output signals array $\text{infoSet}$) to the bits array $u(0 : N/2 - 1)$. Direct the output messages to be saved in $\mu_v^{(\text{in})}(0 : N/2 - 1)$, using the de-mux that is connected to the $\text{outMessages}$ signals array, at the output of the $N/2$ length decoder.

STEP III

Simultaneously,

- At the MUX array, at the input of the BP decoder of the $N/2$ length polar code set the control signal $c_m = 0$, which means that the input from the second multiplexer is selected as input to this unit. Set $c_{\text{opMUX}}$ such that $\mu_{a_0 \rightarrow e_1}(0 : N/4 - 1)$ and $\mu_u^{(\text{in})}(0 : N/4 - 1)$ will be selected as the first input and the second input, respectively to this unit. Set $c_{BP\text{PE, internal}}$ to correspond to the computation of (14), and use the $N/2$ length decoder in P-Mode. Set the OP-De-MUX array to direct the output to the array $\mu_{a_0 \rightarrow e_1}(N/4 : N/2 - 1)$.

- Having the same values for $c_{opMUX}$ and $c_{BPPE, internal}$, use the auxiliary array of processors to operate on the inputs $\mu_{x_0}^{(\text{in})}(N/4 : N/2 - 1)$ and $\mu_u^{(\text{in})}(N/4 : N/2 - 1)$ and have the output directed to $\mu_{a_0 \rightarrow e_1}(N/4 : N/2 - 1)$.

Simultaneously,

- Keep $c_m = 0$ and change $c_{opMUX}$ such that $\mu_{x_1}^{(\text{in})}(0 : N/4 - 1)$ and $\mu_{a_0 \rightarrow e_1}(0 : N/4 - 1)$ will be the first input and the second input, respectively to the $N/2$ length decoder. Set $c_{BPPE, internal}$ to correspond to the computation of (15) and use the $N/2$ length decoder in P-Mode. Set the OP-De-MUX array to direct its output to the array $\mu_v^{(\text{out})}(0 : N/4 - 1)$.

- Having the same values for $c_{opMUX}$ and $c_{BPPE, internal}$, use the auxiliary array of processors to operate on the inputs $\mu_{x_0}^{(\text{in})}(N/4 : N/2 - 1)$ and $\mu_{a_0 \rightarrow e_1}(N/4 : N/2 - 1)$ and have the output directed to $\mu_v^{(\text{out})}(N/4 : N/2 - 1)$.

STEP IV

- At the MUX array, at the input of the decoder of the code of length $N/2$, set the control signal $c_m = 1$, which means that the input from the second multiplexer is selected as an input to this unit. Also set $OuterCodeID = 1$, which means that $\mu_v^{(\text{out})}(0 : N/2 - 1)$ is the input to this decoder.

- Provide the indices of the the frozen bits from the second half of the codeword to the $N/2$ length decoder, and operate it in S-Mode. Perform the decoding of the second outer polar code of length $N/2$. Save the estimation of the information word (output signals array $\text{infoSet}$) to the bits array $u(N/2 : N - 1)$. Direct the output messages to be stored in $\mu_v^{(\text{in})}(0 : N/2 - 1)$, using the de-mux that is connected to the $\text{outMessages}$ signals array, at the output of the $N/2$ length decoder.

Simultaneously,
At the MUX array, at the input of the decoder of the code of length \( N/2 \), set the control signal \( c_m = 0 \). Set \( \mu_{OPMUX}^{(in)}(0 : N/4 - 1) \) and \( \mu_{OPMUX}^{(in)}(0 : N/4 - 1) \) will be selected as the first input and the second input, respectively of this unit. Set \( c_{BPPE,\text{internal}} \) to correspond to the computation of (13) and use the \( N/2 \) length polar code decoder in P-Mode. Set the OP-De-MUX array to direct the output to \( \mu_{e_1 \rightarrow a_0}(0 : N/4 - 1) \).

Having the same values for \( \mu_{OPMUX} \) and \( c_{BPPE,\text{internal}} \), use the auxiliary array of processors to operate on the inputs \( \mu_{c}^{(in)}(N/4 : N/2 - 1) \) and \( \mu_{c}^{(in)}(N/4 : N/2 - 1) \) and have the output directed to \( \mu_{e_1 \rightarrow a_0}(N/4 : N/2 - 1) \).

Simultaneously,

- Keep \( c_m = 0 \) and change \( \mu_{OPMUX}^{(in)}(0 : N/4 - 1) \) and \( \mu_{e_1 \rightarrow a_0}(0 : N/4 - 1) \) will be the first input and the second input, respectively to the \( N/2 \) length decoder. Use the polar code decoder of length \( N/2 \) in P-Mode, and set \( c_{BPPE,\text{internal}} \) to correspond to the computation of (13), and use the \( N/2 \) length polar code decoder in P-Mode. Set the OP-De-MUX array to direct the output to \( \mu_{e_1}(0 : N/4 - 1) \).

Having the same values for \( \mu_{OPMUX} \) and \( c_{BPPE,\text{internal}} \), use the auxiliary array of processors to operate on the inputs \( \mu_{c}^{(in)}(N/4 : N/2 - 1) \) and \( \mu_{c}^{(in)}(N/4 : N/2 - 1) \) and have the output directed to \( \mu_{e_1}(N/4 : N/2 - 1) \).

Simultaneously,

- Keep \( c_m = 0 \) and change \( \mu_{OPMUX}^{(in)}(0 : N/4 - 1) \) and \( \mu_{a_0 \rightarrow e_1}(0 : N/4 - 1) \) will be the first input and the second input, respectively to the \( N/2 \) length decoder. Set \( c_{BPPE,\text{internal}} \) to correspond to the computation of (13), and use the \( N/2 \) length polar code decoder in P-Mode. Set the OP-De-MUX array to direct the output to \( \mu_{c}^{(out)}(0 : N/4 - 1) \).

Having the same values for \( \mu_{OPMUX} \) and \( c_{BPPE,\text{internal}} \), use the auxiliary array of processors to operate on the inputs \( \mu_{c}^{(in)}(N/4 : N/2 - 1) \) and \( \mu_{a_0 \rightarrow e_1}(N/4 : N/2 - 1) \) and have the output directed to \( \mu_{c}^{(out)}(N/4 : N/2 - 1) \).

The output of the decoder, in S-mode, will be the array \( u(0 : N - 1) \) and the two arrays \( \mu_{x_0}(0 : N/2 - 1) \) and \( \mu_{x_1}(0 : N/2 - 1) \) interleaved. This means, that the output message vector is an array in which the entries with even indices are from \( \mu_{x_0}(0 : N/2 - 1) \) and the entries with odd indices are from \( \mu_{x_1}(0 : N/2 - 1) \).

In the P-Mode, the decoder serves as an array of \( N/2 \) processors that operate in parallel, the control signal \( C_{BPPE,\text{external}} \) indicates which operation is performed on all the processors. The inputs to the processor are denoted by the signals arrays \( ext_a(0 : N/2 - 1) \) (the first input) and \( ext_b(0 : N/2 - 1) \) (the second input). The output is directed to the signals array \( ext_{out}(0 : N/2 - 1) \). The P-Mode decoding algorithm operates as follows.

Simultaneously,

- At the MUX array, at the input of the BP-decoder of the polar code of length \( N/2 \), set the control signal \( c_m = 0 \), which means that the OP-MUX array is the input of the decoder. Set \( \mu_{OPMUX} \) such that \( ext_a(0 : N/4 - 1) \) and \( ext_b(0 : N/4 - 1) \) will be the first input and the second input, respectively. Use the polar code decoder of length \( N/2 \) in P-Mode, and set \( c_{BPPE,\text{internal}} \) to be equal to \( C_{BPPE,\text{external}} \). Have the OP-De-MUX array to direct the output to \( ext_{out}(0 : N/4 - 1) \).

Having the same values for \( \mu_{OPMUX} \) and \( c_{BPPE,\text{internal}} \), use the auxiliary array of
processors to operate on the inputs $ext_a(N/4 : N/2 − 1)$ and $ext_b(N/4 : N/2 − 1)$ and have the output directed to $ext_{out}(N/4 : N/2 − 1)$.

Let us, now, consider the time complexity (in terms of the number of clock cycles consumed by an iteration) of this design. As before, let $T(n)$ be the time complexity of the decoder of the polar code of length $N = 2^n$. We assume that each call to a PE requires one clock cycle. In our design, we therefore have

$$T(n) = 2 \cdot T(n − 1) + 7,$$  \hspace{1cm} (27)$$

and $T(1) = 4$, so $T(n) = 5.5 \cdot N − 7 = \Theta(N)$. The memory consumption, however is $\Theta(N \cdot \log N)$, because of the memory matrices for the $\mu_v^{(out)}$ type of messages. The number of processing elements in this design is $N/2$. It should be noted, that the suggested processor can be further improved to support some operations to occur in parallel. For example, if the PE could run one operation of $f_+(\cdot)$ and one operation of $f_−(\cdot)$ in parallel, we could have the two last operations in step IV to be done in one clock cycle, therefore reducing the free addend in (27) to 6. Further reduction would be achieved if one could perform $f_+(\cdot)$ and direct its output to $f_−(\cdot)$ in one clock cycle. This will result in joining the two operations in step III, into one operation. Allowing the computation of $f_−(\cdot)$ and directing its output to $f_+(\cdot)$ in the same clock cycle, will results in consolidation of the two operations of step I into one operation (actually, the latter change may also allow to consolidate the second and third computation in step IV, making the first change redundant). These changes will result in 4, as the free addend in (27) and $T(2) = 2$, so $T(n) = 3 \cdot N − 4$. Naturally, these changes require the appropriate amendments in the routing units, that we described before.

We want to note here that the remarks, which we made on the SC line decoder at the end of Subsection 4.2 also apply here. Specifically, the long paths hazards, requiring a more efficient designs by opening the recursive boxes is also relevant for the BP decoder, specifically for the routing layers in P-Mode. Furthermore, the issue of idle clock cycles for the PEs is also a problem of this design and the solution of Subsections 4.2.1 and 4.2.2 may be adapted to this decoder too. However, while in the SC decoder, the existence of inactive PEs is due to the properties of the SC algorithm, which dictates the scheduling of the message computation, in the BP case, this is due to the scheduling we choose and not a mandatory property of the algorithm. Other types of scheduling do exist, and currently there is no evidence which scheduling is better (for example, in terms of the achieved error rate or in terms of the average number of iterations required for convergence). Hussami et al. [12] proposed to use the Z-shape schedule, which description suggests a constant level of parallelism of $N$ PEs (of the type we considered here) operating all the time. This seems to give the Z-shape schedule an advantage over the GCC schedule if the number of processors is not limited (unless the technique of Subsection 4.2.1 is applied). It is an interesting question to find which schedule is better, when the number of processors is limited. This is a matter for further research.

5 Hardware Architectures for General Kernels

So far, we described algorithms for decoding of polar codes in a recursive way. This notion has enabled us to restate the hardware implementation for SC for Arikan’s construction, that were proposed by Leroux et al. [15]. In addition, we suggested an implementation for BP decoding for the GCC schedule. In this section, we would like to generalize these constructions for other types of kernels. Because we already covered the implementation for Arikan’s codes in some details, we will be more brief in this section, mainly emphasizing the principle differences from the designs in Section 4.

5.1 Recursive Description for the SC Line Decoder for General Kernels

Figure 9 depicts a block diagram for a SC line decoder for a general linear kernel of dimension $\ell$, over alphabet $F$. This kernel has an $\ell \times \ell$ generating matrix, $G$ associated with it. We assume, that this
Figure 9: Block Diagram for the SC Line Decoder for a General Linear Kernel of Dimension $\ell$
decoder has the same requirements for the inputs and outputs, that were given for the $(u + v, v)$ line decoder in Subsection 4.2.

The basic processing element of this design (denoted by PE), gets $\ell$ llr functions (each function is of $|F| - 1$ values), and the coset vector that reflects the previous stages decisions. The control signal $c_u$ indicates which type of llr function the processor should output. There are $\ell$ types of computations that the processor should support according to the different stages of decoding, as (2) implies. Since we consider here a linear kernel, when decoding outer code number $k$, the assumption on $u^{k-1}_0$ (the information sub-vector input to the kernel) is manifested by the coset vector, which this sub-vector induces. This coset vector is generated by $u^{k-1}_0 \cdot G_{\rightarrow (0 : k-1)}$, where $G_{\rightarrow (0 : k-1)}$ is a matrix containing only the first $k$ rows of $G$. This coset vector is gradually computed and maintained in the registers array $x(0 : N - 1)$, as we explain in the sequel. We note, that if the kernel is not linear, then each processor should get the previously decided bits associated with it, i.e. the estimated sub-vector $u^{k-1}_0$, in order to perform (2).

The way the llr computations of (2) are done is an important question, that we do not elaborate on here. For example, it may be beneficial to consider trellis implementation, of the decoding stages, or even consider using approximations of it, such as min-sum rule [13], or near ML decoding variants, such as order statistics or box and match [25].

Since the outer codes in this design are of length $N/\ell$, the processors in the preparatory steps of the SC algorithm (i.e. steps $2 \cdot r - 1$, as defined in Section 4) should generate $N/\ell$ llr functions, serving as inputs to the decoder of the outer code. Therefore, to have the maximum level of parallelism we use $N/\ell$ PEs in the decoder. The embedded $N/\ell$ length recursive decoder is able to contribute only $N/\ell^2$ processors, so the auxiliary array of processors needs to supply the rest of the processors, i.e. it should have $N/\ell - N/\ell^2$ additional processors.

The encoding unit gets the decisions on the codewords of the outer codes from the $N/\ell$ length decoder. Using these decisions, it computes the estimated coset vectors of the inner codes. To support this, we use the signal outerCodeID, that identifies which outer-code is currently decoded. At the end of step $2 \cdot r$, we have outerCodeID $= r - 1$, because we just finished decoding outer code number $r - 1$, by the $N/\ell$ length decoder. This decoder outputs the estimation of the codeword using the signals vector $\tilde{x}(0 : N/\ell - 1)$. Now, the encoding layer performs the following operation, for $0 \leq i \leq N/\ell - 1$,

$$x(\ell \cdot i : \ell \cdot (i + 1) - 1) = x(\ell \cdot i : \ell \cdot (i + 1) - 1) + \tilde{x}(i) \cdot G_{\rightarrow r-1}. \tag{28}$$

This means, that we add row number $r - 1$ of $G$, multiplied by the symbols of the recently estimated outer codeword, to the previously estimated coset vectors (Note, that we have $N/\ell$ coset vectors, such that $x(\ell \cdot i : \ell \cdot (i + 1) - 1)$, corresponds to the $i^{th}$ inner code, $0 \leq i \leq N/\ell - 1$). At the end of step $2 \cdot \ell$, the output of the encoding layer is the estimation of the codeword.

As in the $(u + v, v)$ line decoder, we have two operation modes. The first one is S-Mode, in which the decoder gets llr functions and the indices of the frozen symbols, and outputs the hard decisions on the information word and its corresponding codeword. The second one is P-Mode, in which the decoder operates as an array of processors and performs the same type of operation according to the signal $c_u$.

In S-Mode, we have $\ell$ pairs of computation steps, as described below ($1 \leq r \leq \ell$).

**STEP 2 $\cdot r - 1$**

Simultaneously,

- At the MUX array, at the input of the decoder of the polar code of length $N/\ell$, set the control signal $c_m = 0$, which means that the array $\lambda(0 : N/\ell - 1)$ is selected as an input to this unit. Set $c_u = r - 1$ and supply the coset vectors $x(0 : N/\ell - 1)$ to the unit (the latter is achieved because modeIn $= 0$). Use the decoder of the polar code of length $N/\ell$ in P-Mode. This means that the processors will perform the computation of the llrs of type $r - 1$ according to (2), where $k = r$. The values of the computations are stored in the registers array $R(0 : N/\ell^2 - 1)$. 

33
• Use the auxiliary array of processors, and perform the same computations given the rest of the \( x \) array, \( \lambda(N/\ell : N - 1) \), and the rest of the cosets vector, \( x(N/\ell : N - 1) \). The outputs of the computations are stored in the registers array \( R(N/\ell^2 : N/\ell - 1) \).

**STEP 2 : \( r \)**

• At the MUX array, at the input of the decoder of the polar code of length \( N/\ell \), set the control signal \( c_m = 1 \), which means that the values of the registers array \( R(0 : N/\ell - 1) \) are inputs to this unit.

• Provide to the \( N/\ell \) length polar code decoder, the indices of the frozen symbols from the range \( [(r - 1) \cdot N/\ell, r \cdot N/\ell - 1] \). Operate the \( N/\ell \) length polar code decoder in S-Mode, which results in decoding of the outer code number \( r - 1 \). Store the estimated information word in the following way \( u((r - 1) \cdot N/\ell : r \cdot N/\ell - 1) = \hat{u}(0 : N/\ell - 1) \). Perform the computation of the coset vector as defined in (28).

If this is the last step (i.e. \( r = \ell \)), then we give as output the content of \( u(0 : N - 1) \) and the content of \( x(0 : N - 1) \) (To avoid the sampling delay due to the registers, we prefer to give as output \( [u(0 : N - \ell + 1) \ \ 
\hat{u}(0 : N/\ell - 1)] \) instead of \( u(0 : N - 1) \), and the output of the encoding layer block instead of \( x(0 : N - 1) \)).

The P-Mode operation is quite straightforward. We have the signal \( modeIn = 1 \), which indicates that the \( N \) length decoder operates in P-Mode. This causes the input cosets vectors (denoted by the signals array \( x_{in}(0 : N - 1) \)) to be routed to the processors (instead of the internal cosets vector \( x(0 : N - 1) \)). The embedded \( N/\ell \) length decoder operates in P-Mode (i.e. \( mode = 1 \), as well). As a result, both the auxiliary array of processors and the embedded \( N/\ell \) length decoder computes the operation that is indicated by the signal \( c_{u,in} \), and output the computations results using the signals array \( L(0 : N/\ell - 1) \).

The complexity analysis is also quite simple. As an example, if we assume that the processor requires \( c \) clock cycles to complete the computation of each of its \( \ell \) stages, then we have for \( N = \ell^n \) length code, \( T(n) = \ell \cdot T(n - 1) + \ell \cdot c \) and \( T(1) = \ell \cdot c \), so \( T(n) = c \cdot (N + \ell \cdot (\log_2 N - 1)) \) clock cycles. The number of \( R \) registers for holding the llr functions (each function contains \( |F| - 1 \) values) can be shown to be \( N \sum_{i=1}^{\log_2 N} \ell^{i-1} = N \cdot \frac{1 - \ell^{\log_2 N}}{1 - \ell} \). The long routing path hazard that we raised in the context of the \( (u + v, v) \) decoder, may also be of concern here. Therefore, our suggestion to open the recursion boxes and to optimize them accordingly, may be relevant here as well. The ideas of sharing the auxiliary array of processors for increasing the throughput, or decreasing the parallelism studied in Subsections 4.2.1 and 4.2.2 respectively, are also applicable here with the obvious adaptations.

### 5.2 About Decoders for Mixed Kernels and General Concatenated Codes

So far, we considered decoders for homogenous kernels that may be non-binary. These codes have the nice property, that the outer codes in their GCC structure are themselves polar codes from the same family (but shorter ones). Therefore, we were able to use a single embedded decoder of a code of length \( N/\ell \) within the decoder of the code of length \( N \). This embedded decoder is used \( \ell \) times, each time with different inputs (i.e. indices of the frozen symbols and the input messages). This property no longer applies when mixed kernels are employed.

Consider, for example, the \( \ell = 4 \) dimension mixed kernel that we presented in one of our previous papers [2]. In the decoder of the mixed code of length \( N = 4^n \), we should have an embedded decoder of the mixed code of length \( N/4 \), and an additional embedded decoder for the \( RS(4) \) polar code of length \( N/4 \). It should be noted, however, that even here, a reuse of hardware is still possible, as the decoder for the \( RS(4) \) of length \( N/4 \), requires an embedded decoder for the \( RS(4) \) of length \( N/16 \) within it. The latter decoder (and its embedded decoders) can be shared with the decoder for mixed code of length \( N/4 \) (that requires an embedded \( RS(4) \) decoder of the same length).

A further step in generalization of this structure, is the general concatenated structure, in which the outer codes are not required to be polar codes. This means, that other types of codes may be used with
their corresponding decoding algorithms. Examples of such structures using BCH codes and near ML decoding algorithms, were recently described by Trifonov [8]. In these types of constructions, we need to have a separate decoding unit for each outer code. As in the cases of the mixed kernels, if the outer codes share structure and decoding algorithm these resources may be reused, thereby enabling a more efficient design.

Summary and Conclusions

We considered the recursive GCC structures of polar codes which led to recursive description of their decoding algorithms. Specifically, known algorithms (SC and SCL) were formalized in a recursive fashion, and then were generalized for arbitrary kernels. The BP decoding algorithm the with the GCC schedule was also depicted. Then, recursive hardware architectures for these algorithms were considered. We restated known architectures, and generalized them for arbitrary kernels.

In our discussion, we preferred for brevity, to give somewhat abstract descriptions of the subjects, emphasizing the main properties while neglecting some of the technical details. However, a complete hardware design requires a full treatment of all of these details (as was done by Leroux et al. for the $(u + v, v)$ case [15]). We intend to verify this design for arbitrary kernels in a further work.

Another issue, that needs a more careful attention, is the BP decoder, and specifically the proposed GCC schedule. A comparison between it and other proposed schedules (e.g. the $Z$ shaped schedule) is an interesting question, which is also a subject for further research. The usage of BP decoder for arbitrary kernels is another interesting problem, that also worth further studying. For these kernels, the way to compute the messages is well understood. However, the question of an appropriate schedule that enables the convergence of the algorithm, is not clear. We note however, that for a specific kernel, if such a schedule exists it may be beneficial to try to define it in a recursive manner, thereby enabling the utilization of the approach in this paper to construct a decoding hardware for it.

References

[1] E. Arikan, “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, 2009.

[2] E. Arikan and E. Telatar, “On the rate of channel polarization,” Jul. 2008. [Online]. Available: http://arxiv.com/abs/0807.3806

[3] S. B. Korada, E. Sasoglu, and R. Urbanke, “Polar codes: Characterization of exponent, bounds, and constructions,” Jan. 2009. [Online]. Available: http://arxiv.com/abs/0901.0536

[4] R. Mori and T. Tanaka, “Channel polarization on q-ary discrete memoryless channels by arbitrary kernels,” Jan. 2010. [Online]. Available: http://arxiv.org/abs/1001.2662

[5] ——, “Non-binary polar codes using reed-solomon codes and algebraic geometry codes,” Jul. 2010. [Online]. Available: http://arxiv.org/abs/1007.3661

[6] N. Presman, O. Shapira, and S. Litsyn, “Binary polar code kernels from code decompositions,” Jan. 2011. [Online]. Available: http://arxiv.org/abs/1101.0764

[7] ——, “Polar codes with mixed kernels,” Jul. 2011. [Online]. Available: http://arxiv.org/abs/1107.0478

[8] P. Trifonov, “Efficient design and decoding of polar codes.” [Online]. Available: http://dcn.infos.ru/~petert/

[9] I. Tal and A. Vardy, “List decoding of polar codes,” 2011.
[10] ——, “List decoding of polar codes,” Jun. 2012. [Online]. Available: http://arxiv.org/abs/1206.0050

[11] B. Li, H. Shen, and D. Tse, “An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check,” Aug. 2012. [Online]. Available: http://arxiv.org/abs/1208.3091

[12] N. Hussami, S. B. Korada, and R. Urbanke, “Performance of polar codes for channel and source coding,” Jan. 2009.

[13] E. Arkan, “A performance comparison of polar codes and reed-muller codes,” IEEE Commun. Lett., vol. 12, no. 6, pp. 447–449, 2008.

[14] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, “Hardware architectures for successive cancellation decoding of polar codes,” Nov. 2010. [Online]. Available: http://arxiv.org/abs/1011.2919

[15] C. Leroux, A. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. Gross, “Hardware implementation of successive-cancellation decoders for polar codes,” Journal of Signal Processing Systems, vol. 69, pp. 305–315, 2012, 10.1007/s11265-012-0685-3. [Online]. Available: http://dx.doi.org/10.1007/s11265-012-0685-3

[16] A. Pamuk, “An fpga implementation architecture for decoding of polar codes,” in Wireless Communication Systems (ISWCS), 2011 8th International Symposium on, nov. 2011, pp. 437 –441.

[17] J. Forney, G. D., Concatenated Codes. Cambridge, MA: M.I.T. Press, 1966.

[18] E. Blokh and V. Zyabolov, “Coding of generalized concatenated codes,” Probl. Peredachi. Inform., vol. 10, no. 3, pp. 45–50, 1974.

[19] V. Zinoviev, “Generalized concatenated codes,” Probl. Peredachi. Inform., vol. 12, no. 1, pp. 5–15, 1976.

[20] I. Dumer, Handbook of Coding Theory. Eds., Elsevier, The Netherlands, 1998, ch. Concatenated Codes and Their Multilevel Generalizations. [Online]. Available: http://www.ee.ucr.edu/~dumer/con-fi.pdf

[21] I. Dumer and K. Shabunov, “Soft-decision decoding of reed-muller codes: recursive lists,” IEEE Trans. Inf. Theory, vol. 52, no. 3, pp. 1260 –1266, march 2006.

[22] I. Dumer, “Soft-decision decoding of reed-muller codes: a simplified algorithm,” IEEE Trans. Inf. Theory, vol. 52, no. 3, pp. 954 –963, march 2006.

[23] J. Forney, G. D., “Codes on graphs: normal realizations,” IEEE Trans. Inf. Theory, vol. 47, no. 2, pp. 520–548, 2001.

[24] E. Sharon, S. Litsyn, and J. Goldberger, “Efficient serial message-passing schedules for ldpc decoding,” IEEE Trans. Inf. Theory, vol. 53, no. 11, pp. 4076 –4091, nov. 2007.

[25] V. Miloslavskaya and P. Trifonov, “Design of binary polar codes with arbitrary kernel.” IEEE Information Theory Workshop, 2012. [Online]. Available: http://dcn.infos.ru/~petert/papers/polardesignkernel.pdf