Flexible and Low-Complexity Encoding and Decoding of Systematic Polar Codes

Gabi Sarkis, Ido Tal *Member, IEEE*, Pascal Giard *Student Member, IEEE*, Alexander Vardy *Fellow, IEEE*, Claude Thibeault *Senior Member, IEEE*, and Warren J. Gross *Senior Member, IEEE*

***Abstract*—The capacity-achieving property of polar codes has garnered much recent research attention resulting in low-complexity and high-throughput hardware and software decoders. It would be desirable to implement *flexible* hardware for polar encoders and decoders that can implement polar codes of different lengths and rates, however this topic has not been studied in depth yet. Flexibility is of significant importance as it enables the communications system to adapt to varying channel conditions and is mandated in most communication standards. In this work, we describe a low-complexity and flexible systematic-encoding algorithm, prove its correctness, and use it as basis for encoder implementations capable of encoding any polar code up to a maximum length. We also investigate hardware and software implementations of decoders, describing how to implement flexible decoders that can decode any polar code up to a given length with little overhead and minor impact on decoding latency compared to code-specific versions. We then demonstrate the application of the proposed decoder in a quantum key distribution setting, in conjunction with a new sum-product approximation to improve performance.**

***Index Terms*—polar codes, systematic encoding, multi-code encoders, multi-code decoders.**

I. Introduction

Modern communication systems must cope with varying channel conditions and differing throughput constraints. The 802.11-2012 wireless communication standard specifies twelve low-density parity-check (LDPC) codes of different rate and length combinations; in addition to convolutional codes [1]. The overhead of building a flexible LDPC decoder capable of decoding different codes is significant, and creating flexible LDPC decoders is an active area of research [2], [3].

There has been much recent interest in Polar codes, which achieve the symmetric capacity of memoryless channels with an explicit construction and are decoded with the low complexity successive-cancellation decoding algorithm [4]. It was also recently shown that polar codes do not exhibit any error floor when transmitted over symmetric binary-input memoryless channels [5]. There have been several implementations of polar decoders in the literature, some of which are capable of decoding polar codes of different rates given a fixed code length [6], [7]. In this work, we show how this flexibility can be extended to decode and also encode any code of length $n \leq n_{\max}$.

Polar codes were initially introduced as non-systematic block codes [4]. Later, systematic polar encoding was described in [8] as a method to ease information extraction and improve bit-error rate without affecting the frame-error rate. In addition to the error rate improvement, systematic polar codes were shown to be well suited for use with the fast decoding algorithm introduced in [6].

The systematic encoding scheme originally proposed in [8] is serial by nature, and seems non-trivial to parallelize, unless restricted to a single polar code of fixed rate and length. The serial nature of this encoding ($O(n \cdot \log n)$ time-complexity) places a speed limit on the encoding process which gets worse with increasing code length. In contrast, the non-systematic encoder presented in [4] is parallel by nature, and is amenable to very fast hardware implementations [9]. To address this, a new systematic encoding algorithm that is easy to parallelize was first described in [6]. This new encoding algorithm offers the best of both worlds: on one hand, it is systematic, and thus gains all the advantages described above. On the other hand, it is essentially equivalent to running the non-systematic encoder twice. Thus, the prior art (and future advances) used to implement fast non-systematic encoders can be used as is to implement a fast systematic encoder. We further highlight that the systematic encoder in [6] is very flexible: it can encode any polar code of a given length by simply updating bit masks stored in memory, without any other modifications to the implementation.

The systematic encoder presented in [6] is known *not to work* (i.e. not produce valid polar codewords) for certain choices of frozen index sets. However, it was observed in [6] that such bad choices of frozen sets do not occur in practice. It was speculated in [6] that this phenomenon is true in general, but no proof was given. A key result of this paper is such a proof. We show that if ties are broken in a specific way during the polar code construction phase, then the systematic encoder in [6] will always be correct and produce valid codewords. The result follows from Theorems 1 and 6 below.

This paper contains two conceptual parts addressing flexible encoding and decoding, respectively. The first one starts with Section II where we define some preliminary notation and contrast the implementation of the original systematic encoder presented in [8] to the more efficient systematic encoder pre-

G. Sarkis, P. Giard, and W. J. Gross are with the Department of Electrical and Computer Engineering, McGill University, Montréal, QC H3A 0E9, Canada (e-mails: {gabi.sarkis, pascal.giard}@mail.mcgill.ca, warren.gross@mcgill.ca).

I. Tal is with the Technion—Israel Institute of Technology, Haifa 32000, Israel. (e-mail: idotal@ee.technion.ac.il).

A. Vardy is with the Department of Electrical and Computer Engineering and the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093, USA (e-mail: avardy@ucsd.edu).

C. Thibeault is with the Department of Electrical Engineering, École de technologie supérieure, Montréal, QC H3C 1K3, Canada (e-mail: claude.thibeault@etsmtl.ca).

sented in [6]. Note that reading [8] or [6] *is not* a prerequisite to reading the current paper. Section III is mainly about setting notation and casting the various operations needed in matrix form. In Section IV, we define the property of domination contiguity, and prove that our algorithms work if this property is satisfied. The fact that domination contiguity indeed holds for polar codes is proved in Section V.

The second conceptual part of this paper deals with flexibility of encoders and decoders with respect to codeword length. Section VI details how the systematic decoder of [6] can be adapted to work at various codeword lengths. Sections VII and VIII discuss such flexibility with respect to hardware and software versions of the simplified successive-cancellation (SSC) decoder, respectively. Lastly, Section IX shows how such a flexible software decoder can be used in the quantum key distribution setting, and introduces a new sum-product approximation that improves error-correction performance compared to the min-sum algorithm.

II. Background

We start by defining what we mean by a "systematic encoder", with respect to a general linear code. For integers $0 < k \leq n$, let $G = G_{k\times n}$ denote a $k \times n$ binary matrix with rank k. The notation G is used to denote a generator matrix. Namely, the code under consideration is

$$\operatorname{span}(G) = \left\{\mathbf{v} \cdot G \mid \mathbf{v} \in \mathrm{GF}(2)^k\right\} \ .$$

An encoder

$$\mathcal{E}\colon \mathrm{GF}(2)^k \to \operatorname{span}(G)$$

is a one-to-one function mapping an *information bit vector*

$$\mathbf{u} = (u_0, u_1, \ldots, u_{k-1}) \in \mathrm{GF}(2)^k$$

to a *codeword*

$$\mathbf{x} = (x_0, x_1, \ldots, x_{n-1}) \in \operatorname{span}(G) \ .$$

All the encoders discussed in this paper are linear. Namely,

$$\mathcal{E}(\mathbf{u}) = \mathbf{u} \cdot \Pi \cdot G \ ,$$

where $\Pi = \Pi_{k\times k}$ is an invertible matrix defined over $\mathrm{GF}(2)$.

The encoder $\mathcal{E}$ is systematic if there exists a set of k *systematic indexes*

$$S = \{s_j\}_{j=0}^{k-1} \ , \quad 0 \leq s_0 < s_1 < \cdots < s_{k-1} \leq n-1 \ , \quad (1)$$

such that restricting $\mathcal{E}(\mathbf{u})$ to the indexes S yields $\mathbf{u}$. Specifically, position s_i of $\mathbf{x}$ must contain u_i. We stress that since the s_i are in increasing order in (1), a restriction operation is all that is needed in order to recover $\mathbf{u}$ form $\mathbf{x}$. Namely, the restriction *need not* be followed by a permutation.

Since G has rank k, there exist k linearly independent columns in G. Thus, we might naively take Π as the inverse of these columns, take S as the indexes corresponding to these columns, and state that we are done. Of course, the point of [8] and [6] is to show that the calculations involved can be carried out efficiently with respect to the computational model.

The model considered in [8] is the standard serial model. By making use of the recursive structure of polar codes, it is shown that both the operation of multiplying $\mathbf{u}$ by Π and the operation of multiplying the resulting vector by G can be carried out in time $O(n \cdot \log n)$.

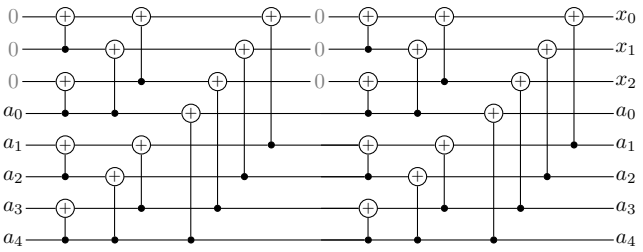

Fig. 1. The systematic encoder of [6] for an (8, 5) polar code.

We are interested in the VLSI model. For this model, it is known that the operation of multiplying by G can be implemented very efficiently and can be made to run very fast. See [9] for current details. In contrast, the algorithm presented in [8] for calculating $\mathbf{u} \cdot \Pi$ seems inherently serial: a computation is carried out on the first half of the codeword. We wait for the computation to finish, and use the results in order to perform a very similar operation on the second half of the codeword. In fact, since the algorithm is similar to successive-cancellation (SC) decoding, the many methods used in order to parallelize SC decoding [6], [10] can be used in this setting as well. However, even with these refinements, multiplying by G will still be much simpler and much faster.

The systematic encoding algorithm presented in [6] essentially involves multiplying by G twice and setting frozen bits locations to '0' in between. Fig. 1 illustrates this process for a non-reversed (8, 5) polar code using a_i and x_i to denote information and parity bits, respectively. As far as we understand, this is rather different from the method in [8]. Specifically, [8] works for any set of frozen indices while [6] does not. It was observed in [6] that a bad set of frozen indices does not occur when constructing polar codes, and this was assumed to always hold. In this paper, we prove that this assumption is indeed true. We analyze two variants of systematic encoders, one for the codes originally presented in [4] in which a bit-reversing operation is carried out on the codeword, and one in which no bit-reversing operation is carried out (this was the version presented in [6]). The bit-reversed version is more natural in hardware implementations, where it simplifies routing; while the non-reversed version is more natural for software implementations, enabling the use of vectorized single-instruction multiple-data (SIMD) instructions.

III. Systematic Encoding of Polar Codes

We start this section by recasting the concepts and operation presented in the previous section into matrix terminology. Recalling the definition of S as the set of systematic indices, define the *restriction matrix* $R = R_{n\times k}$ corresponding to S as

$$R = (R_{i,j})_{i=0\ j=0}^{n-1\ k-1} \ , \quad \text{where} \quad R_{i,j} = \begin{cases} 1 & \text{if } i = s_j \ , \\ 0 & \text{otherwise} \ . \end{cases} \quad (2)$$

With this definition at hand, we require that $\mathcal{E}(\mathbf{u}) \cdot R = \mathbf{u}$, or equivalently that

$$\Pi \cdot G \cdot R = I \ , \tag{3}$$

where I above denotes the $k \times k$ identity matrix. Our proofs will center on showing that (3) holds.

We will shortly introduce the two variants of polar codes for which our two encoders are tailored. Essentially, the difference between the two codes is a bit reversal operation. Thus, as a first step, we define the concept of bit reversal.

From this point forward, we adopt the shorthand

$$m \triangleq \log_2 n \ .$$

For an integer $0 \leq i < n$, denote the binary representation of i as

$$\langle i \rangle_2 = (i_0, i_1, \ldots, i_{m-1}) \ , \quad \text{where } i = \sum_{j=0}^{m-1} i_j 2^j \text{ and } i_j \in \{0,1\} \ . \tag{4}$$

For i as above, we define $\overleftarrow{i}$ as the integer with reversed binary representation. That is,

$$\langle \overleftarrow{i} \rangle_2 = (i_{m-1}, i_{m-2}, \ldots, i_0) \ , \quad \overleftarrow{i} = \sum_{j=0}^{m-1} i_j 2^{m-1-j} \ .$$

As in [4], we denote the $n \times n$ bit reversal matrix as B_n. Recall that B_n is a permutation matrix. Specifically, multiplying a matrix from the left (right) by B_n results in a matrix in which row (column) i equals row (column) $\overleftarrow{i}$ of the original matrix.

A key concept in the definition of polar codes is the kernel matrix. Our encoders assume the kernel matrix presented in the seminal paper [4]. Namely, let $F = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$. The m-th Kronecker product of F is denoted $F^{\otimes m}$ and is defined recursively as

$$F^{\otimes m} = \begin{bmatrix} F^{\otimes(m-1)} & 0 \\ F^{\otimes(m-1)} & F^{\otimes(m-1)} \end{bmatrix} \ , \quad \text{where } F^{\otimes 1} = F \ . \tag{5}$$

We denote the generator matrices corresponding to our two code variants as G_{rv} and G_{nrv}, where the subscripts denote "reversed" and "non-reversed", respectively. We first discuss G_{rv}, the version presented in [4].

The matrix G_{rv} is obtained by selecting a subset of k rows from the $n \times n$ matrix $B_n F^{\otimes n}$. Thus, the name "reversed" highlights the fact that a bit reversing operation is applied to the Kronecker product of the kernel matrix. The rows selected correspond to the k "best" synthetic channels, as discussed in [4]. We denote the set of rows selected, termed the *active rows* as

$$A = \{\alpha_j\}_{j=0}^{k-1} \ , \quad 0 \leq \alpha_0 < \alpha_1 < \cdots < \alpha_{k-1} \leq n-1 \ . \tag{6}$$

As before, we recast the above in matrix terms. Thus, define the matrix $E = E_{k \times n}$ as

$$E = (E_{i,j})_{i=0\ j=0}^{k-1\ n-1} \ , \quad \text{where} \quad E_{i,j} = \begin{cases} 1 & \text{if } j = \alpha_i \ , \\ 0 & \text{otherwise} \ . \end{cases} \tag{7}$$

By this definition, we have

$$G_{\mathrm{rv}} = E \cdot B_n \cdot F^{\otimes m} \ . \tag{8}$$

As explained, applying E to the left of $B_n \cdot F^{\otimes m}$ results in a subset of rows of $B_n \cdot F^{\otimes m}$. However, in our context, it is often more natural to think of E as a matrix which transforms an information vector $\mathbf{u}$ of length k into a vector $\mathbf{u} \cdot E$ of length n. The vector $\mathbf{u} \cdot E$ contains u_i at position α_i, for $0 \leq i \leq k-1$, and 0 in all other positions. Thus, E is termed an *expanding matrix*.

The generator matrix G_{nrv} for our second polar code variant is defined as

$$G_{\mathrm{nrv}} = E \cdot F^{\otimes m} \ . \tag{9}$$

By [4, Proposition 16], we know that $B_n \cdot F^{\otimes m} = F^{\otimes m} \cdot B_n$. Thus, $G_{\mathrm{rv}} = G_{\mathrm{nrv}} \cdot B_n$. Namely, G_{rv} and G_{nrv} span the same code, up to a bit-reversing permutation of indexes.

With the above notation at hand, the non-reversing encoder can be succinctly described as

$$\mathcal{E}_{\mathrm{nrv}}(\mathbf{u}) = \mathbf{u} \cdot \underbrace{E \cdot F^{\otimes m} \cdot E^T}_{\Pi} \cdot \underbrace{E \cdot F^{\otimes m}}_{G_{\mathrm{nrv}}} \ . \tag{10}$$

Note that multiplying a vector $\mathbf{v}$ of length n by E^T results in a vector of length k with entry i equal to entry α_i of $\mathbf{v}$. Put another way, E^T equals the restriction matrix R, if the restriction set S equals the set of active rows A. For the case of $\mathcal{E}_{\mathrm{nrv}}$, this will indeed be the case. Simply put, u_i will appear at position α_i of the codeword[1]. Thus, for the non-reversed case, our aim is to show that

$$E \cdot F^{\otimes m} \cdot E^T \cdot E \cdot F^{\otimes m} \cdot E^T = I \ . \tag{11}$$

Showing this will further imply that the corresponding Π in (10) is indeed invertible.

We now shift to describing our encoder for the bit-reversed case, $\mathcal{E}_{\mathrm{rv}}$. As a first step, we define the set of bit-reversed active rows, $\overleftarrow{A}$, gotten from the set of active rows A by applying the bit-reverse operation on each element α_i. As before, we order the elements of $\overleftarrow{A}$ in increasing order and denote

$$\overleftarrow{A} = \{\beta_j\}_{j=0}^{k-1} \ , \quad 0 \leq \beta_0 < \beta_1 < \cdots < \beta_{k-1} \leq n-1 \ . \tag{12}$$

Recall that the expansion matrix E was defined using A. We now define $\overleftarrow{E} = \overleftarrow{E}_{k \times n}$ according to $\overleftarrow{A}$ in exactly the same way. That is,

$$\overleftarrow{E} = (\overleftarrow{E}_{i,j})_{i=0\ j=0}^{k-1\ n-1} \ , \quad \text{where} \quad \overleftarrow{E}_{i,j} = \begin{cases} 1 & \text{if } j = \beta_i \ , \\ 0 & \text{otherwise} \ . \end{cases} \tag{13}$$

Note that $E \cdot B$ and $\overleftarrow{E}$ are the same, up to a permutation of rows (for i fixed, the reverse of α_i does not generally equal β_i, hence the need for a permutation). Thus, by (8),

$$G'_{\mathrm{rv}} = \overleftarrow{E} F^{\otimes m} \tag{14}$$

is a generator matrix spanning the same code as G_{rv}. Analogously to (10), our encoder for the reversed code is given by

$$\mathcal{E}_{\mathrm{rv}}(\mathbf{u}) = \mathbf{u} \cdot \underbrace{\overleftarrow{E} \cdot F^{\otimes m} \cdot (\overleftarrow{E})^T}_{\Pi} \cdot \underbrace{\overleftarrow{E} \cdot F^{\otimes m}}_{G'_{\mathrm{rv}}} \ . \tag{15}$$

[1]Since the encoder presented in [8] has this property as well, it must be the case that for a given information word, both our encoder and [8] produce the same codeword (if this were not the case, then neither encoder would span the whole code).

For the reversed encoder, the set of systematic indices is $\overleftarrow{A}$. Thus, our aim will be to prove that

$$\overleftarrow{E} \cdot F^{\otimes m} \cdot (\overleftarrow{E})^T \cdot \overleftarrow{E} \cdot F^{\otimes m} \cdot (\overleftarrow{E})^T = I \ . \tag{16}$$

IV. Domination contiguity implies involution

In this section we prove that our encoders are valid by proving that (11) and (16) indeed hold. A square matrix is called an *involution* if multiplying the matrix by itself yields the identity matrix. With this terminology at hand, we must prove that both $E \cdot F^{\otimes m} \cdot E^T$ and $\overleftarrow{E} \cdot F^{\otimes m} \cdot (\overleftarrow{E})^T$ are involutions.

Interestingly, and in contrast with the original systematic encoder presented in [8], the proof of correctness centers on the structure of A. That is, in [8], any set of k active (non-frozen) channels has a corresponding systematic encoder. In contrast, consider as an example the case in which $n = 4$ and $A = \{0, 1, 3\}$. By our definitions,

$$E = \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \end{bmatrix} \ , \quad E^T = \begin{bmatrix} 1&0&0 \\ 0&1&0 \\ 0&0&0 \\ 0&0&1 \end{bmatrix} \ , \quad \text{and} \quad F^{\otimes 2} = \begin{bmatrix} 1&0&0&0 \\ 1&1&0&0 \\ 1&0&1&0 \\ 1&1&1&1 \end{bmatrix} \ .$$

Thus,

$$E \cdot F^{\otimes 2} \cdot E^T = \begin{bmatrix} 1&0&0 \\ 1&1&0 \\ 1&1&1 \end{bmatrix} \ , \quad \text{and}$$
$$(E \cdot F^{\otimes 2} \cdot E^T) \cdot (E \cdot F^{\otimes 2} \cdot E^T) = \begin{bmatrix} 1&0&0 \\ 0&1&0 \\ 1&0&1 \end{bmatrix} \ .$$

Note that the rightmost matrix above is *not* an identity matrix. A similar calculation shows that $\overleftarrow{E} \cdot F^{\otimes 2} \cdot (\overleftarrow{E})^T$ is not an involution either. The apparent contradiction to the correctness of our algorithms is rectified by noting that $A = \{0, 1, 3\}$ cannot correspond to a polar code (as will be formalized shortly). Specifically, the above A implies that W^{+-} is frozen while W^{--} is unfrozen, a case which will never occur [11].

We now characterize the A for which (11) and (16) hold. Recall our notation for binary representation given in (4). For $0 \leq i, j \leq n$, denote

$$\langle i \rangle_2 = (i_0, i_1, \ldots, i_{m-1}) \ , \quad \langle j \rangle_2 = (j_0, j_1, \ldots, j_{m-1}) \ .$$

We define the *binary domination* relation, denoted $\succeq$, as follows.

$$i \succeq j \quad \text{iff for all } 0 \leq t < m, \text{ we have } i_t \geq j_t \ .$$

Namely, $i \succeq j$ iff the support of $\langle i \rangle_2$ (the indexes t for which $i_t = 1$) contains the support of $\langle j \rangle_2$.

We say that a set of indexes $A \subseteq \{0, 1, \ldots, n-1\}$ is *domination contiguous* if for all $h, j \in A$ and for all $0 \leq i < n$ such that $h \succeq i$ and $i \succeq j$, it holds that $i \in A$. For easy reference:

$$(h, j \in A \quad \text{and} \quad h \succeq i \succeq j) \Longrightarrow i \in A \ . \tag{17}$$

Theorem 1. *Let the active rows set $A \subseteq \{0, 1, \ldots, n-1\}$ be domination contiguous, as defined in (17). Let E and $\overleftarrow{E}$ be defined according to (6), (7), (12), and (13). Then, $E \cdot F^{\otimes m} \cdot E^T$ and $\overleftarrow{E} \cdot F^{\otimes m} \cdot (\overleftarrow{E})^T$ are involutions. That is, (11) and (16) hold.*

Proof. We first note that for $0 \leq i, j < n$, we have that $i \succeq j$ iff $\overleftarrow{i} \succeq \overleftarrow{j}$. Thus, if A is domination contiguous then so is $\overleftarrow{A}$. As a consequence, proving that $E \cdot F^{\otimes m} \cdot E^T$ is an involution will immediately imply that $\overleftarrow{E} \cdot F^{\otimes m} \cdot (\overleftarrow{E})^T$ is an involution as well. Let us prove the former — that is, let us prove (11).

We start by noting a simple characterization of $F^{\otimes m}$. Namely, the entry at row i and column j of $F^{\otimes m}$ is easily calculated:

$$(F^{\otimes m})_{i,j} = \begin{cases} 1 & i \succeq j \ , \\ 0 & \text{otherwise} \ . \end{cases} \tag{18}$$

To see this, consider the recursive definition of $F^{\otimes m}$ given in (5). Obviously, $(F^{\otimes m})_{i,j}$ equals 0 if we are at the upper right $(n/2) \times (n/2)$ block. That is, if i_{m-1} (the MSB of i) equals 0 and j_{m-1} equals 1. If this is not the case, we continue recursively in much the same fashion, with respect to $i \mod 2^{m-1}$ and $j \mod 2^{m-1}$. Namely, we continue recursively with "truncated by one position" versions of the binary vectors representing i and j.

Recalling (6) and the notation $|A| = k$, we adopt the following shorthand: for $0 \leq p, q, r < k$ given, let

$$h = \alpha_p \ , \quad i = \alpha_q \ , \quad j = \alpha_r \ .$$

By the above, a straightforward derivation yields that

$$(E \cdot F^{\otimes m} \cdot E^T)_{p,q} = (F^{\otimes m})_{h,i}$$
$$\text{and} \quad (E \cdot F^{\otimes m} \cdot E^T)_{q,r} = (F^{\otimes m})_{i,j} \ .$$

Thus,

$$\Big((E \cdot F^{\otimes m} \cdot E^T) \cdot (E \cdot F^{\otimes m} \cdot E^T) \Big)_{p,r}$$
$$= \sum_{q=0}^{k-1} (E \cdot F^{\otimes m} \cdot E^T)_{p,q} \cdot (E \cdot F^{\otimes m} \cdot E^T)_{q,r}$$
$$= \sum_{i \in A} (F^{\otimes m})_{h,i} \cdot (F^{\otimes m})_{i,j} \ . \tag{19}$$

Proving (11) is now equivalent to proving that the RHS of (19) equals 1 iff h equals j. Recalling (18), this is equivalent to showing that if $h \neq j$, then there is an even number of $i \in A$ for which

$$h \succeq i \quad \text{and} \quad i \succeq j \ , \tag{20}$$

while if $h = j$, then there is an odd number of such i.

We distinguish between 3 cases.

1) If $h = j$, then there is a single $0 \leq i < n$ for which (20) holds. Namely, $i = h = j$. Since $h, j \in A$, we have that $i \in A$ as well. Since 1 is odd, we are finished with the first case.
2) If $h \neq j$ and $h \not\succeq j$, then there can be no i for which (20) holds. Since 0 is an even integer, we are done with this case as well.
3) If $h \neq j$ and $h \succeq j$, then the support of the binary vector $\langle j \rangle_2 = (j_0, j_1, \ldots, j_{m-1})$ is contained in and distinct from the support of the binary vector $\langle h \rangle_2 = (h_0, h_1, \ldots, h_{m-1})$. A moment of thought revels that the number of $0 \leq i < n$ for which (20) holds is equal to $2^{w(h)-w(j)}$, where $w(h)$ and $w(j)$ represent the support size of $\langle h \rangle_2$ and $\langle j \rangle_2$, respectively. Since $h \neq j$ and

$h \succeq j$, we have that $w(h) - w(j) > 0$. Thus, $2^{w(h)-w(j)}$ is even. Since $h, j \in A$ and A is domination contiguous, all of the above mentioned i are members of A. To sum up, an even number of $i \in A$ satisfy (20), as required.

Recall [4, Section X] that an (r, m) Reed-Muller code has length $n = 2^m$ and is formed by taking the set A to contain all indices i such that the support of $\langle i \rangle_2$ has size at least r. Clearly, such an A is domination contiguous, as defined in (17). Hence, the following is an immediate corollary of Theorem 1, and states that our decoders are valid for Reed-Muller codes.

Corollary 2. *Let the active row set A correspond to an (r, m) Reed-Muller code. Let E and $\overleftarrow{E}$ be defined according to (6), (7), (12), and (13), where $n = 2^m$. Then, $E \cdot F^{\otimes m} \cdot E^T$ and $\overleftarrow{E} \cdot F^{\otimes m} \cdot (\overleftarrow{E})^T$ are involutions. That is, (11) and (16) hold and thus our two encoders are valid.*

V. Polar codes satisfy domination contiguity

The previous section concluded with proving that our encoders are valid for Reed-Muller codes. Our aim in this section is to prove that our encoders are valid for polar codes. In order to do so, we first define the concept of a (stochastically) upgraded channel.

A channel W with input alphabet $\mathcal{X}$ and output alphabet $\mathcal{Y}$ is denoted $W : \mathcal{X} \to \mathcal{Y}$. The probability of receiving $y \in \mathcal{Y}$ given that $x \in \mathcal{X}$ was transmitted is denoted $W(y|x)$. Our channels will be binary input, memoryless, and output symmetric (BMS). Binary: the channel input alphabet will be denoted as $\mathcal{X} = \{0, 1\}$. Memoryless: the probability of receiving the vector $(y_i)_{i=0}^{n-1}$ given that the vector $(x_i)_{i=0}^{n-1}$ was transmitted is $\prod_{i=0}^{n-1} W(y_i|x_i)$. Symmetric: there exists a permutation $\pi : \mathcal{Y} \to \mathcal{Y}$ such that that for all $y \in \mathcal{Y}$, $\pi(\pi(y)) = y$ and $W(y|0) = W(\pi(y)|1)$.

We say that a channel $W : \mathcal{X} \to \mathcal{Y}$ is upgraded with respect to a channel $Q : \mathcal{X} \to \mathcal{Z}$ if there exists a channel $\Phi : \mathcal{Y} \to \mathcal{Z}$ such that concatenating Φ to W results in Q. Formally, for all $x \in \mathcal{X}$ and $z \in \mathcal{Z}$,

$$Q(z|x) = \sum_{y \in \mathcal{Y}} W(y|x) \cdot \Phi(z|y) \ .$$

We denote W being upgraded with respect to Q as $W \succeq Q$. As we will soon see, using the same notation for upgraded channels and binary domination is helpful.

Let $W : \mathcal{X} \to \mathcal{Y}$ be a binary memoryless symmetric (BMS) channel. Let $W^- : \mathcal{X} \to \mathcal{Y}^2$ and $W^+ : \mathcal{X} \to \mathcal{Y}^2 \times \mathcal{X}$ be the "minus" and "plus" transform as defined in [4]. That is,

$$W^-(y_0, y_1|u_0) = \frac{1}{2} \sum_{u_1 \in \{0,1\}} W(y_0|u_0 + u_1) \cdot W(y_1|u_1) \ ,$$

$$W^+(y_0, y_1, u_0|u_1) = \frac{1}{2} W(y_0|u_0 + u_1) \cdot W(y_1|u_1) \ .$$

The claim in the following lemma seems to be well known in the community, and is very easy to prove. Still, since we have not found a place in which the proof is stated explicitly, we supply it as well.

Lemma 3. *Let $W : \mathcal{X} \to \mathcal{Y}$ be a binary memoryless symmetric (BMS) channel. Then, W^+ is upgraded with respect to W^-,*

$$W^+ \succeq W^- \ . \tag{21}$$

Proof. We prove that $W^+ \succeq W$ and $W \succeq W^-$. Since "$\succeq$" is easily seen to be a transitive relation, the proof follows. To show that $W^+ \succeq W$, take $\Phi : \mathcal{Y}^2 \times \mathcal{X} \to \mathcal{Y}$ as the channel which maps (y_0, y_1, u_0) to y_1 with probability 1. We now show that $W \succeq W^-$. Recalling that W is a BMS, we denote the corresponding permutation as π. We also denote by $\delta()$ a function taking as an argument a condition. δ equals 1 if the condition is satisfied and 0 otherwise. With these definitions at hand, we take

$$\Phi(y_0, y_1|y) = \frac{1}{2}\big[W(y_1|0) \cdot \delta(y_0 = y) + W(y_1|1) \cdot \delta(y_0 = \pi(y)\big] \ .$$

The following lemma claims that both polar transformations preserve the upgradation relation. It is a restatement of [12, Lemma 4.7].

Lemma 4. *Let $W : \mathcal{X} \to \mathcal{Y}$ and $Q : \mathcal{X} \to \mathcal{Z}$ be two BMS channels such that $W \succeq Q$. Then,*

$$W^- \succeq Q^- \qquad \text{and} \qquad W^+ \succeq Q^+ \tag{22}$$

For a BMS channel W and $0 \leq i < n$, denote by $W_i^{(m)}$ the channel which is denoted "$W_n^{(i+1)}$" in [4]. By [4, Proposition 13], the channel $W_i^{(m)}$ is symmetric. The following lemma ties the two definitions of the $\succeq$ relation.

Lemma 5. *Let $W : \mathcal{X} \to \mathcal{Y}$ be a BMS channel. Let the indexes $0 \leq i, j < n$ be given. Then, binary domination implies upgradation. That is,*

$$i \succeq j \quad \Longrightarrow \quad W_i^{(m)} \succeq W_j^{(m)} \ . \tag{23}$$

Proof. We prove the claim by induction on m. For $m = 1$, the claim follows from either (21), or the fact that a channel is upgraded with respect to itself, depending on the case. For $m > 1$, we have by induction that

$$W_{\lfloor i/2 \rfloor}^{(m-1)} \succeq W_{\lfloor j/2 \rfloor}^{(m-1)} \ .$$

Now, if the least significant bits of i and j are the same we use (22), while if they differ we use (21) and the transitivity of the "$\succeq$" relation.

We are now ready to prove our second main result.

Theorem 6. *Let A be the active rows set corresponding to a polar code. Then, A is domination contiguous.*

Proof. We must first state exactly what we mean by a "polar code". Let the code dimension k be specified. In [4], A equals the indices corresponding to the k channels $W_i^{(m)}$ with smallest Bhattacharyya parameter, where $0 \leq i < n$. Other definitions are possible and will be discussed shortly. However, for now, let us use the above definition.

Denote the Bhattacharyya parameter of a channel W by $Z(W)$. As is well known, if W and Q are two BMS channels, then

$$W \succeq Q \quad \Longrightarrow \quad Z(W) \leq Z(Q) \ . \tag{24}$$

For a proof of this fact, see [12, Lemma 1.8].

We deduce from (23) and (24) that if $i \succeq j$, then $Z(W_i^{(m)}) \leq Z(W_i^{(m)})$. Assume for a moment that the inequality is always strict when $i \succeq j$ and $i \neq j$. Under this assumption, $j \in A$ must imply $i \in A$. This is a stronger claim then (17), which is the definition of A being domination contiguous. Thus, under this assumption we are done.

The previous assumption is in fact true for all relevant cases, but somewhat misleading: The set A is constructed by algorithms calculating with finite precision. It could be the case that $i \neq j$, $i \succeq j$, but $Z(W_i^{(m)})$ and $Z(W_i^{(m)})$ are approximated by the same number (a tie), or by two close numbers, but in the wrong order. Thus, it might conceptually be the case that j is a member of A while i is not (in practice, we have never observed this to happen). These cases are easy to check and fix, simply by removing j from A and inserting i instead. Note that each such operation enlarges the total Hamming weight of the vectors $\langle t \rangle_2$ corresponding to elements t of A. Thus, such a swap operation will terminate in at most a finite number of steps. When the process terminates, we have by definition that if $j \in A$ and $i \succeq j$, then $i \in A$. Thus, A is dominations contiguous.

Instead of taking the Bhattacharyya parameter as the figure of merit, we could have instead used the channel misdecoding probability. That is, the probability of an incorrect maximum-likelihood estimation of the input to the channel given the channel output, assuming a uniform input distribution. Yet another figure of merit we could have taken is the channel capacity. The important point in the proof was that an upgraded channel has a figure of merit value that is no worse. This holds true for the other two options discussed in this paragraph. See [11, Lemma 3] for details.

VI. Flexible Hardware Encoders

The most efficient non-systematic polar encoder implementation is presented in [9]. It is a pipelined, semi-parallel design with a throughput of $\mathcal{P}$ bit/Hz, where $\mathcal{P}$ corresponds to the level of parallelism, and is capable of encoding any polar code of length n when correct frozen bits are set to zero at its input. Fig. 2, derived from [9, Fig. 6], shows the architecture of an encoder for $n = 8$ and $\mathcal{P} = 4$, where D denotes a delay element and the multiplexers alternate between their inputs starting with input '0'. In this section, we show how this decoder can be used as the basis for our proposed systematic encoder.

Adapting this architecture to encode any code of length $n \leq n_{\max}$ requires extracting data from different locations along the pipeline. These locations are indicated with dashed lines in Fig. 2, where the output for a code of length n can be extracted from location $S_{\log n}$. Selecting different sets of output is accomplished using $\mathcal{P}$ instances of a $\log n_{\max} \times 1$ multiplexer. In a practical system, it is unlikely that the minimum length of polar codes of interest will be 2; therefore the required multiplexers will be narrower than $\log n_{\max}$ bits.

The encoder of [9] can be used as the component polar encoder when implementing the algorithm proposed in Section III in two ways: the first targeting high throughput, the second, low implementation complexity. In the high throughput arrangement, two instances of the component encoder are used, with the output of the first modified to set the frozen bits to zero before being sent to the second instance. This requires $\mathcal{P}$ 'AND' gates applying masks with frozen bit locations set to zero and an $n_{\max}/\mathcal{P} \times \mathcal{P}$ memory to store said masks. Alternatively, to save implementation resources at the cost of halving the throughput, one instance of the component can be used in two passes: the output from pass is masked and then routed to the input of the encoder where it is encoded again before being presented as the encoder output. In both cases, encoding a code of length $n < n_{\max}$ can be accomplished by setting the mask locations corresponding to bits with indexes greater than n to '0', without any changes to the component non-systematic encoder. This requires $\log_2 n_{\max} - \log_2 n$ extra cycles for the data to reach the encoder output. The extra latency can be eliminated if the component non-systematic encoder is made flexible as described previously.

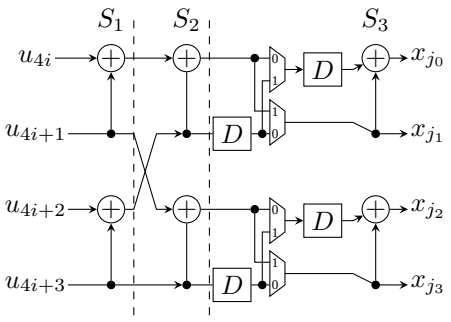

Fig. 2. Architecture of a semi-parallel polar encoder with $n = 8$ and $\mathcal{P} = 4$ created according to [9].

The systematic encoder of [8] can be used in a configuration similar to the proposed high-throughput one. However, it requires multiplication by matrices that change when the frozen bits are changed. Therefore, its implementation requires a configurable parallel matrix multiplier that is significantly more complex than the component non-systematic encoder used in this work. In addition, since G_{AB}^{-1} is different from the encoding matrix G that is used in the second step in [8], separate circuitry is required to implement the operations in a parallelized manner, eliminating the possibility of reusing the component encoder.

VII. Flexible Hardware Decoders

After describing flexible encoders in the previous section, we present flexible hardware decoder in this section and flexible software decoders in the next.

The original fast simplified successive cancellation (Fast-SSC) decoder was capable of decoding all polar codes of a given length: it resembled a processor where the polar code is loaded as a set of instructions [6]. By decoupling a stage's size from its index and updating the control logic, we obtain a flexible Fast-SSC decoder capable of decoding any polar code up to a maximum length $n_{\max}$. In this section, we describe the necessary modifications to the Fast-SSC decoder architecture and analyze the resulting implementation.

TABLE I
IMPLEMENTATION OF A FLEXIBLE POLAR DECODER COMPARED TO THAT OF [6] FOR $n_{\text{MAX}} = 32768$ ON THE ALTERA STRATIX IV EP4SGX530KH40C2.

Decoder	LUTs	FF	RAM (bits)	f (MHz)
[6]	24,066	7,231	536,136	102
Proposed	23,583	7,207	536,136	102

A. Stage Indexes and Sizes

The Fast-SSC decoder is organized into stages with a stage S_i corresponding to a constituent polar code of length 2^i. In the proposed flexible decoder, we modify these two values, so that the aforementioned relationship only holds when the code length $n = n_{\text{max}}$. When $n_{\text{max}}/n = r > 1$, a stage S_i corresponds to a constituent code of length $2^i/r$. The memory allocated for a stage S_i is always calculated assuming $n = n_{\text{max}}$.

The decoder always starts from $S_{\log_2 n_{\text{max}}}$, corresponding to a polar code of length $n \leq n_{\text{max}}$, and proceeds until it encounters a constituent code whose output can be estimated according to the rules of the Fast-SSC algorithm.

B. Implementation Results

Since memory is accessed as words containing multiple LLR or bit-estimate values, the limits used to determine the number of memory words per stage must be changed to accommodate the new n value. The rest of the decoder implementation remains unchanged from [6]. These limits are now provided as inputs to the decoder.

Table I compares the proposed flexible decoder ($n_{\text{max}} = 32768$) with the Fast-SSC decoder of [6] ($n = 32768$) when both are implemented using the Altera Stratix IV EP4SGX530KH40C2 field-programmable gate-array (FPGA). It can be observed that the change in resource utilization is negligible as a result of the localized change in limit calculations. The operating frequency was not affected either. As a results the two decoders have the same throughput and latency. When decoding a code of length $n < n_{\text{max}}$, the flexible decoder has the same latency (in clock cycles) as the Fast-SSC decoder for a code of length n.

VIII. FLEXIBLE SOFTWARE DECODERS

High-throughput software decoders require vectorization using single-instruction multiple-data (SIMD) instructions in addition to a reduction in the number of branches. However, these two considerations significantly limit the flexibility of the decoder to the point that the lowest latency decoders in literature are compiled for a single polar code [13]. In this section, we present a software Fast-SSC decoder balancing flexibility and decoding latency. The proposed decoder has 30% higher latency than a fully specialized decoder, but can decode any polar code of length $n \leq n_{\text{max}}$. As will discussed later in this section, there are two additional advantages to the proposed flexible software decoder: the resulting executable size is an order of magnitude smaller, and it can be used to decode very long polar codes for which an unrolled decoder cannot be compiled.

A. Memory

Unlike in hardware decoders, it is simple to access an arbitrary memory location in software decoders. The LLR memory in the proposed software decoder is arranged into stages according to constituent code sizes. When a code of length $n \leq n_{\text{max}}$ is to be decoded, the channel LLRs are loaded into stage $S_{\log_2 n}$, bypassing any stages with a larger index.

The bit-estimate memory is arranged into a flat structure of length n_{max} bits. Such a layout was found to decrease decoding latency by eliminating superfluous copy operations [13]. For a decoder of length $n \leq n_{\text{max}}$, the decoder writes starting from bit index 0. Once decoding is completed, the estimated codeword will occupy the first n bits of the bit-estimate memory, which are provided as the decoder output.

B. Vectorization

The unrolled software decoder [13] specifies input sizes for each command at compile time. This enables SIMD vectorization without any loops, but limits the decoder to a specific polar code. To efficiently utilize SIMD instructions while minimizing the number of loops and conditionals, we employ dynamic dispatch in the proposed decoder. Each decoder operation in implemented, using SIMD instructions and C++ templates, for all stage sizes up to n_{max}. These differently sized implementations are stored in array indexed by the logarithm of the stage size. Therefore two branch operations are used: the first to look up the decoding operation, and the second to look up the correct size of that operation. This is significantly more efficient than using loops over the SIMD word size.

C. Results

We compare the latency of the proposed vectorized flexible decoder with a non-vectorized version and with the fully unrolled decoder of [13] using floating-point values.

Table II compares the proposed flexible, vectorized decoder with a flexible, non-explicitly-vectorized decoder (denoted by 'Scalar') and a fully unrolled (denoted by 'Unrolled') one running on an Intel Core 2 Quad Q9550 with SSE4 extensions. All decoders were decoding a (32768, 29492) polar code using the Fast-SSC algorithm, floating-point values, and the min-sum approximation. The flexible decoders had $n_{\text{max}} = 32768$. From the results in the table, it can be seen that the vectorized decoder has 41% the latency (or 2.4 times the throughput) of the non-vectorized version. Compared to the code-specific unrolled decoder, the proposed decoder has 130% the latency (or 76% the throughput). In addition to the two layers of indirection in the proposed decoder, the lack of inlining contributes to this increase in latency. In the unrolled decoder, the entire decoding flow is known at compile time, allowing the compiler to inline function calls, especially those related to smaller stages. This information is not available to the flexible decoder.

Results for $n < n_{\text{max}}$ are shown in Table III where $n_{\text{max}} = 32768$ for the flexible decoders and the code used was a (2048, 1723) polar code. The advantage the vectorized decoder has over the non-vectorized one remains similar to the $n = n_{\text{max}}$ case at 48% the latency. The gap between the

TABLE II
SPEED OF THE PROPOSED VECTORIZED DECODER COMPARED WITH THAT OF NON-VECTORIZED AND FULLY-UNROLLED DECODERS WHEN $n = n_{\text{MAX}} = 32768$ AND $k = 29492$.

Decoder	Latency (μs)	Info. Throughput (Mbps)
Scalar Fast-SSC	606.6	48
Unrolled Fast-SSC [13]	188.7	156
Proposed Fast-SSC	247.5	119

TABLE III
SPEED OF THE PROPOSED VECTORIZED DECODER COMPARED WITH THAT OF NON-VECTORIZED AND FULLY-UNROLLED DECODERS FOR A (2048, 1723) CODE AND $n_{\text{MAX}} = 32768$.

Decoder	Latency (μs)	Info. Throughput (Mbps)
Scalar Fast-SSC	36.7	47
Unrolled Fast-SSC [13]	9.8	176
Proposed Fast-SSC	17.6	98

proposed decoder and the unrolled one increases to 1.8 times the latency, as a result of using a shorter code where a smaller proportion of stage operations are inlined in the former.

In addition to decoding different codes, the proposed flexible decoder has an advantage over the fully unrolled one in terms of resulting executable size and the maximum length of the polar code to be decoded. The size of the executable corresponding to the proposed decoder with $n_{\text{max}} = 32768$ was 0.44 MB with 3 kB to store the polar code instruction in an uncompressed textual representation; whereas that of the unrolled decoder was 3 MB. In terms of polar code length, the GNU C++ compiler was unable to compile an unrolled decoder for a code of length 2^{24} even with 32 GB of RAM; while the proposed decoder did not exhibit any such issues.

IX. APPLICATION TO QUANTUM KEY DISTRIBUTION

Quantum key distribution (QKD) is a method that exploits quantum mechanics to provide guaranteed security when transmitting information. QKD occurs over two channels: a quantum one used to transmit the secret information and a classical one used for protocol overhead. The quantum channel is modeled as a binary symmetric channel (BSC) for discrete-value (DV) QKD, or an additive white Gaussian noise (AWGN) channel for continuous-value (CV) distribution. Moreover, it suffers from high noise levels, requiring powerful error-correcting codes of rates close to the channel capacity to correctly and securely transmit information.

It was shown in [14] that long polar codes ($n \geq 2^{24}$) provide very high efficiency when used for QKD, where the efficiency factor is defined as the ratio of the code rate to the channel capacity, i.e. $\beta = R/C$. However, the decoder used the successive-cancellation algorithm and therefore yielded a throughput of only 8.3 Mbps; while state of the art DVQKD systems already exceed 15 Mbps [15], [16].

When using the min-sum algorithm to decode polar codes of length 2^{24} transmitted over the BSC, we observed that the resulting frame error rate (FER) was two to three times that of a significantly slower SPA-based decoder. While the FER can be improved by lowering the code rate; in QKD systems, it is desirable to have rates that are at least 95% of the channel capacity [14]. To resolve this issue, we present a new approximation for the sum-product algorithm (SPA) that suffered no error-rate degradation in our simulation, yet offered $\sim 85\%$ the throughput of the min-sum decoder. Finally we show how the proposed software decoder can be used to decode polar codes of length $n = 2^{24}$ at eight times the throughput of [14].

A. SPA Approximation

Polar decoders use the same parity check update as the $\boxplus$ operation in SPA, which is defined for two input LLRs as

$$a \boxplus b = 2\tanh^{-1}(\tanh(a/2)\tanh(b/2)). \tag{25}$$

Using the Jacobi logarithm, this can be rewritten as [17], [18]

$$\begin{aligned} a \boxplus b = {} & \text{sgn}(a)\text{sgn}(b)\min(|a|,|b|) \\ & + f_+(|a|+|b|) - f_+(|a|-|b|). \end{aligned} \tag{26}$$

The correction function f_+ is defined as

$$f_+(x) = \log(1 + e^{-x}) \tag{27}$$

The first part of (26) is the well-known min-sum approximation that is well suited for hardware and software implementations. The correction function however, has high complexity and is often omitted or approximated using look-up tables in hardware decoders [17]. A good approximation in a software decoder should only use functions with direct mapping to processor instructions, i.e. it cannot use logarithms and exponentiation. Furthermore, it should minimize operations that cannot be implemented using SIMD instructions. A degree-three polynomial, with a $\max()$ function to ensure that the result decays to zero, meets those conditions and provides a very good approximation with a coefficient of determination $R^2 = 0.999$. The proposed approximation is

$$\tilde{f}_+(x) = \max(0, -0.0076x^3 + 0.1010x^2 - 0.4463x + 0.6759); \tag{28}$$

where the operations used—$\max()$, multiplication, and addition—all have SIMD implementations on modern processors.

B. Results

Fig. 3 shows the efficiency of the proposed SPA approximation compared to that of a min-sum decoder for codes of length 2^{24} and different rates. The channel used was a BSC with a probably of crossover $p \in [0.02, 0.10]$ and the rates of the codes were chosen so that the FER was in $[0.08, 0.09]$. We observe that the efficiency gap between the two decoding algorithms grows as p increases. For $p \geq 0.08$, the efficiency of the min-sum decoder drops below 95%, whereas that of the approximate SPA decoder remain $\geq 95\%$ until $p = 0.10$.

In terms of decoding speed, the approximate decoder is 15% slower than the min-sum-based decoder. Table IV lists the latency and information throughput of the proposed approximate SPA decoder, the min-sum decoder, and the SPA decoder of [14]; where the first two are implemented using the flexible software decoder architecture described in Section VIII. The

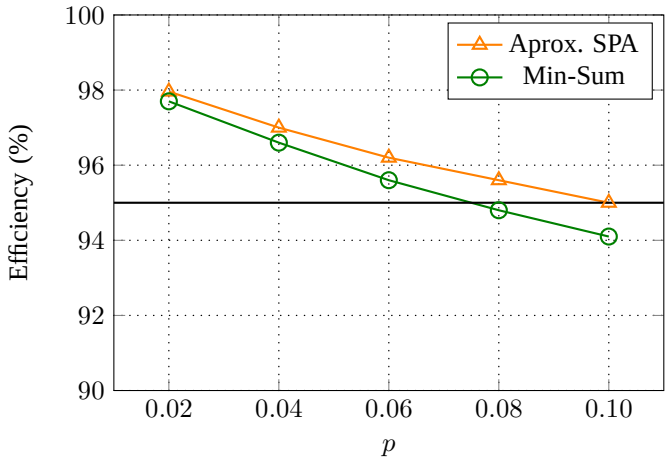

Fig. 3. Efficiency relative to the capacity of the BSC(p) using the approximate SPA and min-sum decoders for codes of length 2^{24}.

TABLE IV
SPEED COMPARISON BETWEEN THE PROPOSED APPROXIMATE SPA DECODER, THE MIN-SUM DECODER, AND THE SPA DECODER USED IN [14].

Decoder	Processor	Latency (ms)	Info. T/P (Mbps)
[14]	i5-670	N/A	8.3
Min-sum	i7-2700	188	74.8
Aprox. SPA	i7-2700	220	64.1
Aprox. SPA	Core2 Q9550	558	25.2

latency numbers include the time required to copy data in and out of the decoder. From the table, it can be seen that the approximate SPA decoder is three to 7.7 times as fast as that of [14], depending on which processor is used. We present the approximate SPA results using two processors: one slower and one faster than the Intel i5-670 used in [14] as we did not have access to the last processor. The min-sum decoder is $\sim 15\%$ faster than the approximate SPA decoder. Therefore, the min-sum decoder is suitable when the channel conditions are good since it is faster; whereas, the approximate SPA decoder should be used when the channel conditions worsen as it has better error-correction performance.

X. CONCLUSION

In this work, we studied flexible implementations of polar encoders and decoder, proving the correctness of a flexible, low-complexity systematic polar encoding algorithm. We also presented hardware and software decoders that can decode any polar code up to a maximum length. Finally, we proposed a new approximation for the SPA and used it in conjunction with the flexible software decoder to provide a decoder for QKD that is 3–8 times as fast as the state of the art.

ACKNOWLEDGMENT

The authors would like to thank Dr. Paul Jouguet of SeQureNet for helpful discussions.

REFERENCES

[1] "IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–2793, Mar 2012.

[2] J.-Y. Lee and H.-J. Ryu, "A 1-Gb/s flexible LDPC decoder supporting multiple code rates and block lengths," *IEEE Trans. Consum. Electron.*, vol. 54, no. 2, pp. 417–424, May 2008.

[3] C. Condo, M. Martina, and G. Masera, "VLSI implementation of a multi-mode turbo/LDPC decoder architecture," *IEEE Trans. Circuits Syst. I*, vol. 60, no. 6, pp. 1441–1454, June 2013.

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

[5] R. U. Marco Mondelli, S. Hamed Hassani, "Unified scaling of polar codes: Error exponent, scaling exponent, moderate deviations, and error floors," *CoRR*, vol. abs/1501.02444, 2015. [Online]. Available: http://arxiv.org/abs/1501.02444

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

[7] Y. S. Park, Y. Tao, S. Sun, and Z. Zhang, "A 4.68Gb/s belief propagation polar decoder with bit-splitting register file," in *Symp. on VLSI Circuits Dig. of Tech. Papers*, Jun 2014, pp. 1–2.

[8] E. Arıkan, "Systematic polar coding," *IEEE Commun. Lett.*, vol. 15, no. 8, pp. 860–862, 2011.

[9] H. Yoo and I.-C. Park, "Partially parallel encoder architecture for long polar codes," *IEEE Trans. Circuits Syst. II*, vol. 62, no. 3, pp. 306–310, March 2015.

[10] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1378–1380, 2011.

[11] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6562–6582, Oct 2013.

[12] S. B. Korada, "Polar codes for channel and source coding," Ph.D. dissertation, EPFL, 2009.

[13] P. Giard, G. Sarkis, C. Leroux, C. Thibeault, and W. J. Gross, "Low-latency software polar decoders," *IEEE Trans. Signal Process.*, 2015, *submitted*. [Online]. Available: http://arxiv.org/abs/1504.00353

[14] P. Jouguet and S. Kunz-Jacques, "High performance error correction for quantum key distribution using polar codes," *Quantum Inf. & Computation*, vol. 14, no. 3-4, pp. 329–338, 2014.

[15] P. Jouguet, D. Elkouss, and S. Kunz-Jacques, "High-bit-rate continuous-variable quantum key distribution," *Physical Review A*, vol. 90, no. 4, p. 042329, 2014.

[16] L. Comandar, B. Fröhlich, M. Lucamarini, K. Patel, A. Sharpe, J. Dynes, Z. Yuan, R. Penty, and A. Shields, "Room temperature single-photon detectors for high bit rate quantum key distribution," *Applied Physics Letters*, vol. 104, no. 2, p. 021101, 2014.

[17] T. Clevon and P. Vary, "Low-complexity belief propagation decoding by approximations with look-up tables," in *Int. ITG Conf. on Source and Channel Coding*, 2004, pp. 211–215.

[18] J. Erfanian, S. Pasupathy, and P. Gulak, "Reduced complexity symbol detectors with parallel structure for ISI channels," *IEEE Trans. Commun.*, vol. 42, no. 234, pp. 1661–1671, Feb 1994.