DESIGN AND COMPARISON OF LPC AND OLSC

N.SUPRIYA HARINI [1] **B.NIHAR[2]**
Nallabharathkumar89@gmail.com[1]

[1]PG Scholar,Dept of ECE, Prasad Engineering College, Jangoan, Warangal, Telangana.

[2]Assistant professor, Dept of ECE, Prasad Engineering College, Jangoan, Warangal, Telangana.

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, proves its correctness, and uses 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.

Key Words—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 standards specify twelve low-density parity-check (LDPC) codes of different rate and length combinations; in addition to convolutional codes. 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. There has been much recent interest in Polar codes, which achieve the symmetric capacity of memory less channels with an explicit construction and are decoded with the low complexity successive-cancellation decoding algorithm. It was also recently shown that polar codes do not exhibit any error floor when transmitted over symmetric binary-input memory less channels. 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.

In this work, we show how this flexibility can be extended to decode and also encode any code of length $n \leq nmax$. Polar codes were initially introduced as non-systematic block codes. Later, systematic polar encoding was described in as a method to ease information extraction and improve bit-error rate without affecting the frame-error rate.

The serial nature of this encoding (O(n · 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 is parallel by nature, and is amenable to very fast hardware implementations. To address this, a new systematic encoding algorithm that is easy to parallelize was first described. 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 general idea for achieving error detection and correction is to add some redundancy which means to add some extra data to a message, which receiver can use to check uniformity of the delivered message, and to pick up data determined to be corrupt. Error-detection and correction scheme may be systematic or it may be non-systematic. In the system of the module non-systematic code, an encoded is achieved by transformation of the message which has least possibility of number of bits present in the message which is being converted. Another classification is the type of systematic module unique data is sent by the transmitter which is attached by a fixed number of parity data like check bits that obtained from the data bits. The receiver applies the same algorithm when only detection of the error is required to the received data bits which is then compared with its output with the received check bits if the values does not match, there we conclude that an error has crept at some point in the process of transmission. Error correcting codes are regularly used in lower-layer communication, as well as for reliable storage in media such as CDs, DVDs, hard disks and RAM.

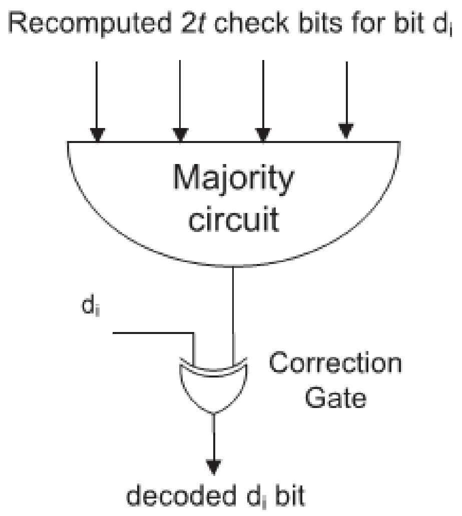


Fig.1. Illustration of OS-MLD decoding for OLS codes

Provision against soft errors that apparent they as the bit-flips in memory is the main motto of error detection and correction. Several techniques are used present to midi gate upsets in memories. For example, the Bose – Chaudhuri– Hocquenghem codes, Reed–Solomon codes, punctured difference set codes, and matrix codes has been used to contact with MCUs in memories. But the above codes mentioned requires more area, power, and delay overheads since the encoding and decoding circuits are more complex in these complicated codes. Reed-Muller code is another protection code that is able to detect and correct additional error besides a Hamming code. But the major drawback of this protection code is the more area it requires and the power penalties.

Reliability is a major issue for advanced electronic circuits. As technology scales, circuits become more vulnerable to error sources such as noise and radiation and also to manufacturing defects and process variations. A number of error mitigation techniques can be used to ensure that errors do not compromise the circuit functionality. Among those, Error Correction Codes (ECCs) are commonly used to protect memories or registers. Traditionally, Single Error

Correction (SEC) codes that can correct one bit error in a word are used as they are simple to implement and require few additional bits. A SEC code requires a minimum Hamming distance between code-words of three. This means that if a double error occurs, the erroneous word can be at distance of one from another valid word. In that case, the decoder will miss-correct the word creating an undetected error. To avoid this issue, Single Error Correction Double Error Detection (SEC-DED) codes can be used. Those codes have a minimum Hamming distance of four. Therefore, a double error can in the worst case cause the word to be at a distance of two of any other valid word so that miss-correction is not possible. More generally, for a code that can correct t errors, it is of interest to also detect t+1 errors. This reduces the probability of undetected errors that can cause Silent Data Corruption (SDC). SDC is especially dangerous as the system continues its operation unaware of the error and this can lead to further data corruption or to an erroneous behavior long after the original error occurred.

II. Literature Survey

This section deals with the existing decoding methodologies used for error detection. In error detection and correction, majority logic decoding is a method to decode repetition codes, based on the assumption that the largest number of occurrences of a symbol was the transmitted symbol. Majority logic decoder is based on a number of parity check equations which are orthogonal to each other. So the majority result of these parity check equations decide the correctness of the current bit under decoding.

A. One Step Majority Logic Decoder

As described in earlier, Majority-logic decoder is a simple and effective decoder capable of correcting multiple bit flips depending on the number of parity checksum equations. It consists of four parts: 1) a cyclic shift register; 2) an XOR matrix; 3) a majority gate; 4) an EXOR gate for error correction, as illustrated in figure 2.

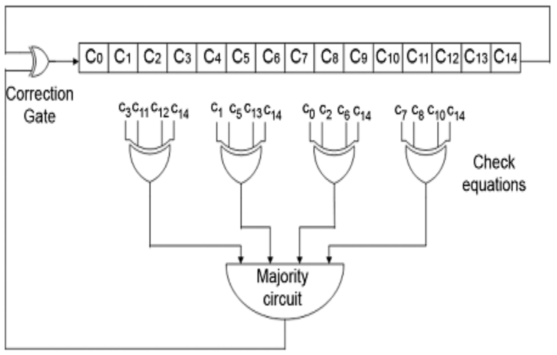


Fig 2: One step Majority Logic Decoder for (15, 7) EG-LDPC Codes

In one step majority logic decoding, initially the code word is loaded into the cyclic shift register. Then the check equations are computed. The resulting sums are then forwarded to the majority gate for evaluating its correctness. If the number of 1‟s received in is greater than the number of 0‟s which means that the current bit under decoding is wrong, and a signal to correct it would be triggered. Otherwise the bit under decoding is correct and no extra operations would be needed on it. In next, the content of the registers are rotated and the above procedure is repeated until codeword bits have been processed. Finally, the parity check sums should be zero if the codeword has been correctly decoded. In this process, each bit may be corrected only once. As a result, the decoding circuitry is simple, but it requires a long decoding time if the code word is large. Thus, by one-step majority-logic decoding, the code is capable of correcting any error pattern with two or fewer errors . For example, for a code word of 15-bits, the decoding would take 15 cycles, which would be excessive for most applications

B. Majority Logic Decoder/Detector (MLDD)

In order to overcome the drawback of MLD method, majority logic decoder/detector was proposed, in which the majority logic decoder itself act as a fault detector. In general, the decoding algorithm is still the same as the majority logic decoder. The difference is that instead of decoding all codeword bits, the MLDD method stops intermediately in the third cycle, which can able to detect up to five bit flips in three decoding cycles. So the number of decoding cycles can be reduced to get improved performance. The schematic of majority logic decoder/detector is illustrated in figure3.

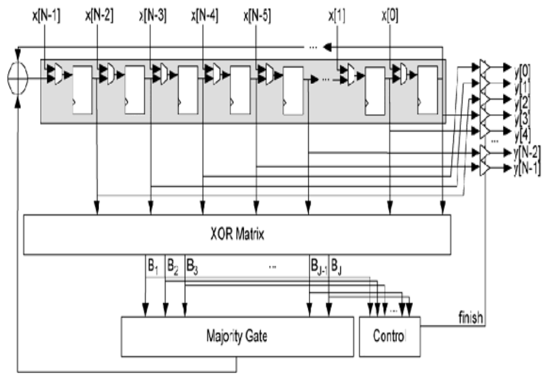


Fig 3: Schematic of Majority Logic Decoder/Detector (MLDD)

Initially the code word is stored into the cyclic shift register and it shifted through all the taps. The intermediate values in each tap are given to the XOR matrix to perform the checksum equations. The resulting sums are then forwarded to the majority gate for evaluating its correctness. If the number of 1‟s received is greater than the number of 0‟s which would mean that the current bit under decoding is wrong, so it mo ve on the decoding process. Otherwise, the bit under decoding would be correct and no extra operations would be needed on it. Decoding process involving the operation of the content of the registers is rotated and the above procedure is repeated and it stops intermediately in the third cycle. If in the first three cycles of the decoding process, the evaluation of the XOR matrix for all is "0," the code word is determined to be error - free and forwarded directly to the output. If the error contains in any of the three cycles at least a "1," it would continue the whole decoding process in order to eliminate the errors. Finally, the parity check sums should be zero if the code word has been correctly decoded. In conclusion the MLDD method is used to detect the five bit errors and correct four bit errors effectively. If the code word contain more than five bit error, it produces the output but it did not show the errors presented in the input. This type of error is called the silent data error. Drawback of this method is did not detecting the silent data error and it consuming the area of the majority gate. The schematic for this memory system is shown in figure 5. It is very similar to the one shown in fig. 1; additionally the control unit was added in the MLDD module to manage the decoding process (to detect the error).

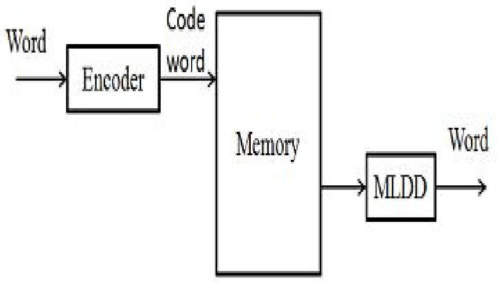


Fig 4: Schematic of memory system with MLDD

Overall operation of the MLDD is illustrated in figure 5

Volume 5, Issue 4 SEP 2016

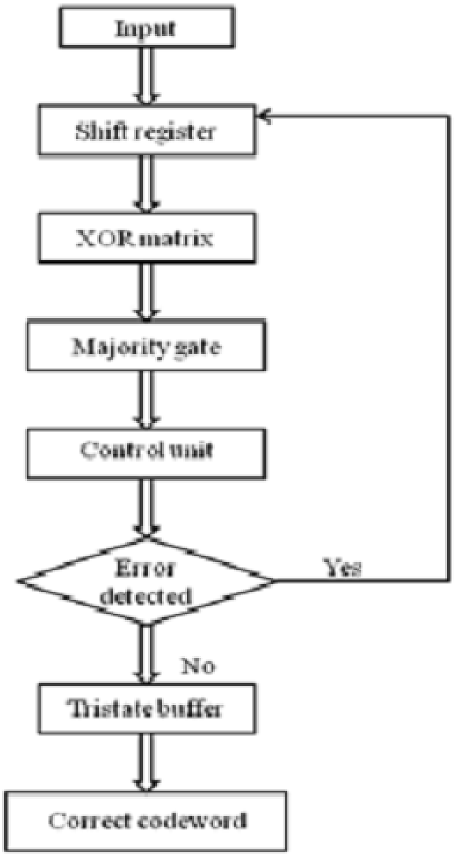


Fig 5: MLDD Algorithm

III. Proposed Polar Encoder

In this section, we propose a partially parallel structure to encode long polar codes efficiently. To clearly show the proposed approach and how to transform the architecture, a 4-parallel encoding architecture for the 16-bit polar code is exemplified in depth. The fully parallel encoding architecture is first transformed to a folded form, and then the lifetime analysis and register allocation are applied to the folded architecture. Lastly, the proposed parallel architecture for long polar codes is described.

A. Folding Transformation

The folding transformation is widely used to save hardware resources by time-multiplexing several operations on a functional unit. A data flow graph (DFG) corresponding to the fully parallel encoding process for 16-bit polar codes is shown in Fig.6, where a node represents the kernel matrix operation F, and wij denotes thejth edge at the ith stage.

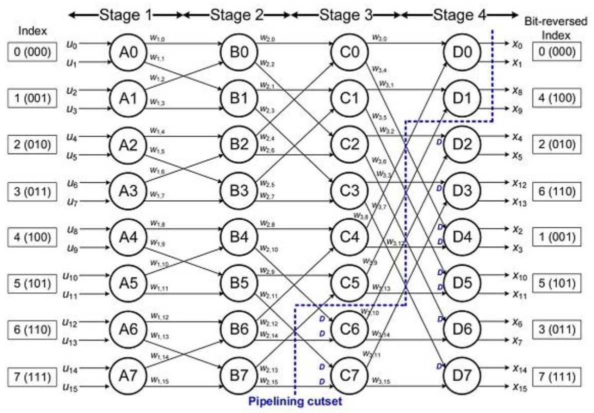
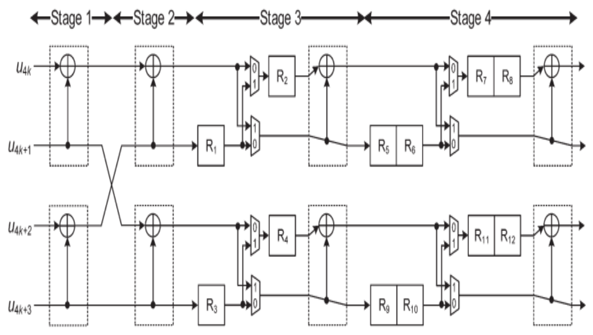


Fig. 6. DFG of 16-bit polar encoding

Note that the DFG of the fully parallel polar encoder is similar to that of the fast Fourier transform except that the polar encoder employs the kernel matrix instead of the butterfly operation. Given the 16-bit DFG, the 4-parallel folded architecture that processes 4 bits at a time can be realized with placing two functional units in each stage since the functional unit computes 2 bits at a time. In the folding transformation, determining a folding set, which represents the order of operations to be executed in a functional unit, is the most important design factor. To construct efficient folding sets, all operations in the fully parallel encoding are first classified as separate folding sets. Since the input is in a natural order, it is reasonable to alternatively distribute the operations in the consecutive order. Thus, each stage consists of two folding sets, each of which contains only odd or even operations to be performed by a separate unit. Considering the four-parallel input sequence in a natural order, stage 1 has two folding sets of{A0,A2,A4,A6}and {A1,A3,A5,A7}. Each folding set contains four elements, and the position of an element represents the operational order in the corresponding functional unit. Two functional units for stage 1 execute A0 and A1 simultaneously at the beginning and A2and A3at the next cycle, and so forth. The folding sets of stage 2 have the same order as those of stage 1, i.e.,{B0,B2,B4,B6} and{B1,B3,B5,B7}, since the four-parallel input sequence of stage 2 is equal to that of stage 1. Furthermore, to determine the folding sets of another stages, the property that the functional unit processes a pair of inputs whose indices differ by2s−1 is exploited. In the case of stage 3, two data whose indices differ by 4 are processed together, which implies that the operational distance of the corresponding data is two as the kernel functional unit computes two data at a time. For instance, w2,0andw2,4that come fromB0andB2are used as the inputs to C0. Since both inputs should be valid to be processed in a functional unit, the operations in stage 3 are aligned to the late input data. Cyclic shifting the folding sets right by one, which can be realized by inserting a delay of one time unit, is to enable full utilization of the functional units by overlapping adjacent iterations. As a result, the folding sets of stage 3 are determined to {C6,C0,C2,C4} and {C7,C1,C3,C5}, whereC6in the current iteration is overlapped with A0and B0in the next iteration. In the same manner, the property that the functional unit processes a pair of inputs whose indices differ by 8 is exploited in stage 4. The folding sets of stage 4 are {D2,D4,D6,D0} and {D3,D5,D7,D1}, which are obtained by cyclic shifting the previous folding sets of stage 3 by two. Generally speaking, a stage whose index s is less than or equal to log2P, where P is the level of parallelism, has the same folding sets determined by evenly interleaving the operations in the consecutive order, and another stage whose index sis larger than log2P has the folding sets obtained by cyclic shifting the previous folding sets of stages−1right by s−log2P.



Stage1: $\{A0,A2,A4,A6\},\{A1,A3,A5,A7\}$

Stage2: $\{B0,B2,B4,B6\},\{B1,B3,B5,B7\}$

Stage3: $\{C6,C0,C2,C4\},\{C7,C1,C3,C5\}$

Stage4: $\{D2,D4,D6,D0\},\{D3,D5,D7,D1\}$

Fig. 7. Proposed 4-parallel folded architecture for encoding the polar (16,K) codes

IV. Orthogonal Latin Squares codes

The concept of Latin squares and their applications are well known [12]. A Latin square of size m is an m * m matrix that has permutations of the digits 0,1,..M-1 in both its rows and columns. For each value of m there can be more than one Latin square. When that is the case, two Latin squares are said to be orthogonal if when they are superimposed every ordered pair of elements appears only once. Orthogonal Latin Squares (OLS) codes are derived from Orthogonal Latin squares [9]. These codes have k=m2 data bits and 2tm check bits where t is the number of errors that the code can correct. For a Double Error Correction (DEC) code t=2 and therefore 4m check bits are used. One advantage of OLS codes is that their construction is modular. This means that to obtain a code that can correct t+1 errors, simply 2m check bits are added to the code that can correct t errors. The modular property enables the selection of the error correction capability for a given word size. As mentioned in the introduction, OLS codes can be decoded using One Step Majority Logic Decoding (OS-MLD) as each data bit participates in exactly 2t check bits and each other bit participates in at most one of those check bits. This enables a simple correction when the number of bits in error is t or less. The 2t check bits are recomputed and a majority vote is taken, if a value of one is obtained, the bit is in error and must be corrected. Otherwise the bit is correct. As long as the number of errors is t or less this ensures the error correction as the remaining t-1 errors can, in the worst case affect t-1 check bits so that still a majority of t+1 triggers the correction of an erroneous bit. For an

OLS code that can correct t errors using OS-MLD, t+1 errors can cause miss-corrections. This occurs for example if the errors affect t+1 parity bits in which bit di participates as this bit will be miss-corrected. The same occurs when the number of errors is larger than t+1. Each of the 2t check bits in which a data bit participates is taken from a group of m parity bits. Those groups are bits 1 to m, m+1 to 2m, 2m+1 to 3m and 3m+1 to 4m.

$$
\begin{array}{c}
M_1 \\ \\ H= \\ \\ M_2
\end{array}
\left[\begin{array}{cccccccccccccccc|cccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&1&0&0&0&0\\
\hline
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1
\end{array}\right]
\; I_{2m}
$$

Fig 2: Parity check matrix for OLS code having k and t as 16&1

The „H" matrix for OLS codes is build from their properties. The matrix is capable of correcting single type error. By the fact that in direction of the modular structure it might be able to correct many errors. They have check bits of number "2tm" in which, „t" stands for numeral of errors such that code corrects. If we wanted to correct a double bit then we have „2" as the value of t and thereby the check bits required are 4m.the H matrix , of Single Error Code „OLS" code is construct as :

$$H = \begin{bmatrix} M_1 & I_{2m} \\ M_2 & \end{bmatrix} \qquad (1)$$

a. In the above, I2m is the identity matrix of size 2 m.
b. M1, M2 are the matrices of given size m × m2.
„"The matrix M1 have m ones in respective rows. For the rth row, the 1"s are at the position (r − 1) × m + 1,(r − 1)× m + 2,………….(r − 1) × m+ m − 1, (r − 1) × m + m". The matrix M2 is structured as:M2 = [Im Im . . . Im] (2)

For the given value 4 for m, the matrices M1 and M2 can be evidently experiential in Fig. H Matrix in the check bits we remove is evidently the G Matrix

$$G = \begin{bmatrix} M_1 \\ M_2 \end{bmatrix} \qquad (3)$$

On concluding the above mentioned, it is evident that the encoder is intriguing m2 data bits and computing 2tm parity check bits by using G matrix . These resulted from the Latin Squares have the below properties:
a. Exactly in 2t parity checks each info bit is involved.
b. Utmost one in parity check bits info bits takes participation.

We use the above properties in the later section to examine our proposed technique. The proposed method is based on the observation that by construction, the groups formed by the mparity bits in each Mi matrix have at most a one in every column of H.For the example in Fig. 2, those groups correspond to bits (or rows) 1–4 (M1), 5–8 (M2), 9–12 (M3), and 13–16 (M4). Therefore, any combination of four bits from one of those groups will at most sharea one with the existing columns inH. For example, the combination formed by bits 1, 2, 3, and 4 shares only bit 1 with columns 1, 2, 3,and 4. This is the condition needed to enable OS-MLD. Therefore, combinations of four bits taken all from one of those groups can be used to add data bit columns to the Hmatrix. For the code with k=16 andt =2 shown in Fig. 2, we have m=4. Hence, one combination can be formed in each group by setting all the positions in the group to one. This is shown in Fig. 3, where the columns added are highlighted. In this case, the data bit block is extended fromk=16 to k=20 bits.

$$\left[\begin{array}{cccccccccccccccc|cccc|cccccccccccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&1&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&1&0&0&1&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{array}\right]$$

Added columns

Fig. 3. Parity check matrix H for the extended OLS code with k=20 and t =2

The proposed method first divides the parity check bits in groups of m bits given by the Mi matrices. Then, the second step is for each group to find the combinations of 2t bits such that any pair of them share at most one bit. This second step can be seen as that of constructing an OS-MLD code with m parity check bits. Obviously, to keep the OS-MLD property for the extended code, the combinations formed for each group have to share at most one bit with the combinations formed in the other 2t −1 groups. This is not an issue as by construction, different groups do not share any bit. When m is small finding, such combinations is easy. For example, in the case considered in Fig. 3, there is only one possible combination per group. When m is larger, several combinations can be formed in each group. This occurs, for example, when m=8. In this case, the OLS code has a data block size k =64. With eight positions in each group, now two combinations of four of them that share at most one position can be formed. This means that the extended code will have eight (4×2) additional data bits. As the size of the OLS code becomes larger, the number of combinations in a group also grows. For the case m=16 and k =256, each group has 16 elements. Interestingly enough, there are 20 combinations of four elements that share at most one element. In fact, those combinations are obtained using the extended OLS code shown in Fig. 3 in each of the groups. Therefore, in this case, 4×20=80 data bits can be added in the extended code. The parameters of the extended codes are shown in Table I, where n−k =2tm is the number of parity bits. The data block size for the original OLS codes (kOLS) is also shown for reference The method can be applied to the general case of an OLS code with k =m2 that can correct t errors. Such a code has 2tm parity bits that as before are divided in groups ofmbits. The code can be extended by selecting combinations of 2t parity bits taken from each of the groups. These combinations can be added to the code as long as any pair of the new combinations share at most one bit. When m is small, a set of such combinations with maximum size can be easily found. However, as m grows, finding such a set is far from trivial (as mentioned before, solving that problem is equivalent to designing an OS-MLD code with m parity bits that can correct t errors). An upper bound on the number of possible combinations can be derived by observing that any pair of bits can appear only in one combination. Because each combination has 2t bits, there are (2t 2) pairs in each combination. The number of possible pairs in each group of m bits is m 2. Therefore, the number of combinations NG in a group of m bits has to be such that

$$\binom{m}{2} \geq \binom{2t}{2} \times N_G \qquad (2)$$

which can be simplified as

$$\frac{m^2 - m}{4t^2 - 2t} \geq N_G . \qquad (3)$$

One particular case for which a simple solution can be found is when m=2t ×l. In this case, an OLS code with a smaller data block size (l2) can be used in each group. One example for t =2 is when m=16 (k=256) for which the OLS code with block size k=4^2 can be used in each group as explained before. Similarly, for t =2, whenk=1024 (m=32) the OLS code of size k $=8^2$ can be used in each group.

V. CONCLUSION

This brief has presented a new partially parallel encoder architecture developed for long polar codes. Many optimization techniques have been applied to derive the proposed architecture. Experimental results show that the proposed architecture can save the hardware by up to 73% compared with that of the fully parallel architecture. Finally, the relationship between the hardware complexity and the throughputs is analyzed to select the most suitable architecture for a given application. Therefore, the proposed architecture provides a practical solution for encoding a long polar code.

The proposed error checking scheme required a significant delay; however, its impact on access time could be minimized. This was achieved by performing the checking in parallel with the writing of the data in the case of the encoder and in parallel with the majority voting and error correction in the case of the decoder.In a general case, the proposed scheme required a much larger overhead as most ECCs did not have the properties of OLS codes. This limited the applicability of the proposed CED scheme to OLS codes. The availability of low overhead error detection techniques for the encoder and syndrome computation is an additional reason to consider the use of OLS codes in high-speed memories and caches.

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, Jul. 2009.

[2] R. Mori and T. Tanaka, “Performance of polar codes with the construction using density evolution,”IEEE Commun. Lett., vol. 13, no. 7, pp. 519– 521, Jul. 2009.

[3] S. B. Korada, E. Sasoglu, and R. Urbanke, “Polar codes: Characterization of exponent, bounds, constructions,”IEEE Trans. Inf. Theory, vol. 56, no. 12, pp. 6253–6264, Dec. 2010.

[4] I. Tal and A. Vardy, “List decoding of polar codes,” inProc. IEEE ISIT, 2011, pp. 1–5.

[5] K. Chen, K. Niu, and J. Lin, “Improved successive cancellation decoding of polar codes,”IEEE Trans. Commun., vol. 61, no. 8, pp. 3100–3107, Aug. 2013.

[6] G. Sarkis and W. J. Gross, “Polar codes for data storage applications,” in Proc. ICNC, 2013, pp. 840–844.

[7] 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.

[8] G. Berhault, C. Leroux, C. Jego, and D. Dallet, “Partial sums generation architecture for successive cancellation decoding of polar codes,” inProc. IEEE Workshop SiPS, Oct. 2013, pp. 407–412.

[9] B. Yuan and K. K. Parhi, “Low-latency successive-cancellation polar decoder architectures using 2-bit decoding,”IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 61, no. 4, pp. 1241–1254, Apr. 2014.

[10] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, “A semi-parallel successive-cancellation decoder for polar codes,” IEEE Trans. Signal Process., vol. 61, no. 2, pp. 289–299, Jan. 2013.

[11] A. J. Raymond and W. J. Gross, “Scalable successive-cancellation hardware decoder for polar codes,” inProc. IEEE GlobalSIP, Dec. 2013, pp. 1282–1285.

[12] U. U. Fayyaz and J. R. Barry, “Low-complexity soft-output decoding of polar codes,”IEEE J. Sel. Areas Commun., vol. 32, no. 5, pp. 958–966, May 2014.