Partial sums generation architecture for successive cancellation decoding of polar codes

Guillaume Berhault, Camille Leroux, Christophe Jego, Dominique Dallet

PARTIAL SUMS GENERATION ARCHITECTURE FOR SUCCESSIVE CANCELLATION DECODING OF POLAR CODES

Guillaume Berhault, Camille Leroux, Christophe Jego, Dominique Dallet

IMS laboratory, University of Bordeaux, Institut Polytechnique de Bordeaux
Talence 33400, France
e-mail: firstname.lastname@ims-bordeaux.fr

ABSTRACT

Polar codes are a new family of error correction codes for which efficient hardware architectures have to be defined for the encoder and the decoder. Polar codes are decoded using the successive cancellation decoding algorithm that includes partial sums computations. We take advantage of the recursive structure of polar codes to introduce an efficient partial sums computation unit that can also implements the encoder. The proposed architecture is synthesized for several code-lengths in 65nm ASIC technology. The area of the resulting design is reduced up to 26% and the maximum working frequency is improved by 25%.

***Index Terms*—** FEC, polar codes, hardware architecture, successive cancellation decoding

1. INTRODUCTION

Polar codes [1] are a new class of error correction codes. These linear block codes are proven to achieve the capacity of any symmetric memoryless channel under successive cancellation (SC) decoding [2]. Moreover, for a code of length N, encoding and decoding computational complexities are $O(N \log_2(N))$. Despite these desirable properties, polar codes require a very large code length in order to actually approach the channel capacity ($N > 2^{20}$). Consequently, the practical interest of polar codes highly depends on the possibility to design efficient encoders and decoders for large N values.

Since polar codes invention, several hardware architectures were proposed. In [1], Arıkan suggests to use a fast Fourier transform structure to efficiently reuse computations. This first architecture requires $N \log_2(N)$ processing elements (PEs) and as many memory elements (MEs).

Some works then focused on reducing the number of PEs and MEs in SC decoders [3]. In [4], a *line architecture* is implemented. It only uses $(N-1)$ PEs and as many MEs without affecting the decoding performance and the throughput. In [5], it is shown that the number of PEs can be further reduced (64 PEs) with a negligible impact on throughput. This SC decoder was fabricated in 180nm ASIC technology [6].

Since SC decoding has a low intrinsic parallelism, complementary works focused on increasing the throughput of SC decoders. In [7] and [8], lookahead techniques are used to reduce the decoding latency while using limited extra hardware resources. In [9], a simplification of SC decoding is proposed in order to reduce the number of computations without altering error correction performance. Extra latency reduction technique is investigated in [10] where maximum likelihood decoding is used to further speedup the decoding process. However, these low latency decoders have not been implemented yet.

As shown in [5] and [6], the hardware implementation of SC decoders is constrained by the partial sums computation unit which occupies a significant part of the area and limits the maximum working frequency, especially as N grows. In [8], an alternative method to compute partial sums is proposed but was not implemented. In this paper, we show that the partial sums computation unit can be implemented with a shift register structure, lowering hardware complexity and increasing maximum clock frequency. We also show that the proposed architecture can be used as a sequential polar code encoder.

The remainder of the paper is organized as follows: in section 2, the polar code construction, encoding and SC decoding processes are briefly reviewed. In the following section, the partial sums computation is introduced and a hardware implementation is proposed. Finally, in section 4, this partial sums unit is compared with existing implementations in terms of area and maximum working frequency in ASIC 65nm technology.

2. POLAR CODES

2.1. Definition and construction

Polar codes are linear block codes of size $N = 2^n$, n being a positive integer. In [1], Arıkan defined a construction based on a 2×2 binary matrix, denoted as the *kernel* of the code: $\kappa = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. The generator matrix of the code is a submatrix of the n^{th} Kronecker power of κ, denoted $\kappa^{\otimes n}$. Thus, for $n = 3$ ($N = 8$),

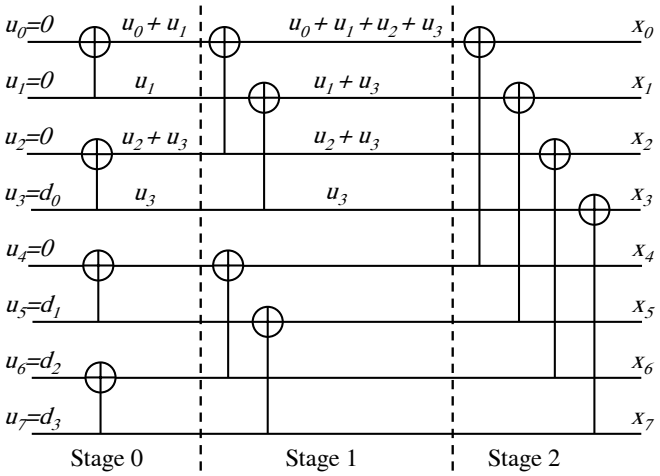


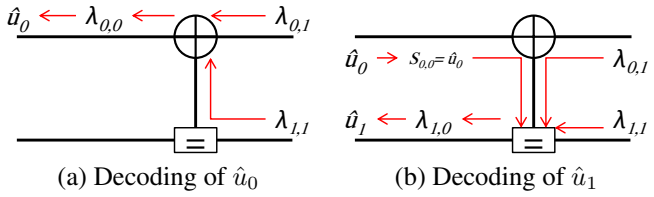
Fig. 1. $N = 8$ polar code encoder graph.

$$\kappa^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \quad (1)$$

A polar code with dimension K and codelength N ($K \leq N$), is denoted as PC(N,K) whose code rate is $R = \frac{K}{N}$. Assuming a particular successive cancellation decoding algorithm on the receiver side, the K rows are selected according to the reliability of some equivalent channels. In [1], Arıkan described a method to select these K rows for a binary erasure channel and a binary symetric channel. The construction was later extended to the more general binary input memoryless channels [2]. The reader may refer to [1] for more explanations on the polarization phenomenon and polar codes definition.

2.2. Encoding Process

As any linear block code, polar code codewords are obtained by multiplying a K-bit information vector, $D = [d_0, d_1, ..., d_{K-1}]$, with the $(K \times N)$-bit generator matrix of the code. An alternative encoding process is to build an *extended information vector* U which contains the K information bits and $(N - K)$ *frozen bits* (all set to 0). This extended information vector is built in such a way that information bits are located on the most reliable positions corresponding to the K selected rows of $\kappa^{\otimes n}$. The corresponding codeword X can then be constructed by calculating $X = U \times \kappa^{\otimes n}$.
A polar code encoder may also be represented graphically as shown in Fig. 1 for $n = 3$. It consists of n stages of $\frac{N}{2}$ XORs each. The input vector U, on the left hand side, is propagated into the graph in order to get X, on the right. In Fig. 1, a $R = 0.5$ polar code is considered; it means that half of the bits in vector U are frozen bits (set to 0) while the rest of them are information bits.



(a) Decoding of $\hat{u}_0$ (b) Decoding of $\hat{u}_1$

Fig. 2. $N = 2$ polar code decoding example.

2.3. Successive Cancellation Decoding

After being sent over the transmission channel, the noisy version Y of the codeword X is received. Each sample y_i is converted into log likelihood ratio (LLR) format. These LLRs are denoted λ_i, with $0 \leq i \leq N - 1$. The decoder successively estimates every bit u_i based on the channel observation vector (λ_0^{N-1})[a] and the previously estimated bits $(\hat{u}_0^{i-1})$[b]. In order to estimate each bit u_i, the decoder computes the following LLR value:

$$\lambda_{i,0} = \log \frac{\Pr(y_0^{N-1}, \hat{u}_0^{i-1} | u_i = 0)}{\Pr(y_0^{N-1}, \hat{u}_0^{i-1} | u_i = 1)}. \quad (2)$$

The estimated bit $\hat{u}_i$ is calculated based on the following rule:

$$\hat{u}_i = \begin{cases} 0 & \text{if } \lambda_{i,0} > 0 \\ 1 & \text{otherwise.} \end{cases} \quad (3)$$

Since the decoder knows which bits are frozen, if u_i is a frozen bit, then $\hat{u}_i = 0$ regardless of $\lambda_{i,0}$ value.
As proposed by Arıkan in [1], the factor graph representation of polar codes can be used to efficiently compute the $\lambda_{i,0}$. SC decoding can be seen as an instance of belief propagation decoding where LLRs are propagated on the factor graph of the code with a particular scheduling. In SC decoding, bits $\hat{u}_i$ are processed sequentially and the decision is then fed back into the graph for the decoding of subsequent bits. In Fig. 2, the factor graph of a simple $N = 2$ polar code is represented. It is composed of a check node (CN or $\oplus$) and a variable node (VN or $\boxed{=}$). In the LLR domain, the VN function is a simple addition and the CN function uses product of transcendental functions. In the perspective of decoder implementation, the simplified versions of the VN and CN functions are used [5]:

$$\begin{cases} a \oplus b & = sgn(a) \times sgn(b) \times \min(|a|, |b|) \\ a \boxed{=} b & = a + b. \end{cases} \quad (4)$$

[a] $\lambda_0^{N-1} = [\lambda_0 ... \lambda_{N-1}]$
[b] $\hat{u}_0^{i-1} = [\hat{u}_0 ... \hat{u}_{i-1}]$

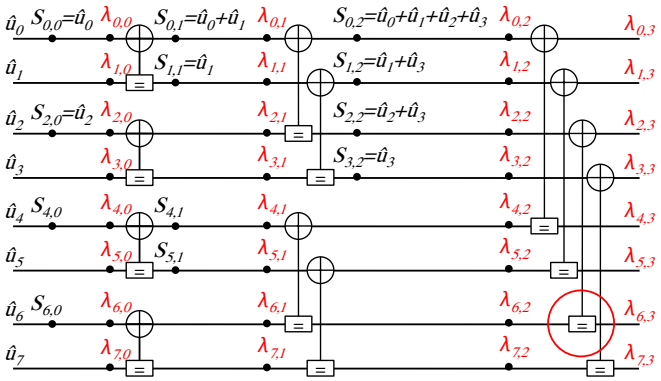


Fig. 3. Factor graph for $N = 8$ polar code.

The decoding process of a $N = 2$ polar code can be summarized as follows:

$$\begin{cases} f(\lambda_{0,1}, \lambda_{1,1}) = sgn(\lambda_{0,1}.\lambda_{1,1}).\min(|\lambda_{0,1}|, |\lambda_{1,1}|) \\ g(\lambda_{0,1}, \lambda_{1,1}, \hat{u}_0) = (-1)^{\hat{u}_0}\lambda_{0,1} + \lambda_{1,1}, \end{cases} \quad (5)$$

where $\hat{u}_0$ and $\hat{u}_1$ are determined according to equation (3). The g function equation shows that the decoding process of a polar code depends on LLRs propagating from right to left $(\lambda_{0,1}; \lambda_{1,1})$ and also on hard decision $(\hat{u}_0)$ propagating from left to right. The factor graph of a $N = 8$ polar code is detailed in Fig. 3. The decoder successively estimates the bits $\hat{u}_i$ from the computation of LLRs of the indexed edges. The LLR of edge (i, j) is computed such as:

$$\lambda_{i,j} = \begin{cases} f\left(\lambda_{i,j+1}, \lambda_{i+2^j,j+1}\right) & \text{if } B(i,j) = 0 \\ g\left(\lambda_{i-2^j,j+1}, \lambda_{i,j+1}, S_{i-2^j,j}\right) & \text{if } B(i,j) = 1, \end{cases} \quad (6)$$

where $B(i,j) \equiv \dfrac{i}{2^j} \bmod 2$, $0 \leq i < N$ and $0 \leq j < n$. $S_{i,j}$ represents the *partial sum* which corresponds to the propagation of decisions back into the factor graph. For instance, in Fig. 3, $S_{1,2} = \hat{u}_1 + \hat{u}_3$ (modulo-2 sum).

3. PARTIAL SUM COMPUTATION

When implemented in hardware, an SC decoder is composed of three main units: the *processing unit* (PU), the *memory unit* (MU) and the *partial sums unit* (PSU). The PU consists of several processing elements (PEs) used to compute f and g functions. The MU stores the computed LLRs $(\lambda_{i,j})$ in register banks during the decoding process. The third unit, the PSU, computes the partial sums required by PEs to calculate the g functions.

In [5], a semi-parallel SC decoder was synthesized for several N values. Synthesis results show that the MU takes about 75% of total decoder area while the rest of the design cost is mainly due to the PSU. In fact, the PU area becomes negligible (<1%) as N grows ($N > 2^{13}$). As stated in [5], the MU cost can be drastically reduced by using RAM blocks instead of register banks. Consequently, the most complex part is then the PSU. Furthermore, in [5] and [6], it is noticed that the critical path of the SC decoder is in the PSU and the maximum working frequency decreases as N increases. Therefore, having an efficient implementation of the PSU would benefit to the SC decoder area and clock frequency.

3.1. Existing partial sums implementations

As depicted in Fig. 3, there are $\dfrac{N}{2}\log_2(N)$ partial sums to be computed. When a bit $\hat{u}_i$ is obtained, the PSU should update all $S_{i,j}$ that include this current bit. For example, in Fig. 3, when $\hat{u}_2$ is available, the partial sums $\{S_{2,0}; S_{0,2}; S_{2,2}\}$ have to be updated by "XORing" their current values with $\hat{u}_2$. All the remaining partial sums should however keep their current values.

It was shown in [5] that some partial sums can share the same D-Flip-Flop (DFF) thus reducing the required storage space from $\dfrac{N}{2}\log_2(N)$ to $(N-1)$ DFFs. In this work, an *Indicator Function* (IF) is defined in order to indicate whether each DFF should be updated with the current $\hat{u}_i$ or not. The IF is implemented by some combinational logic that generates $(N-1)$ bits necessary to control the accumulation in the $(N-1)$ DFFs. As reported in [5], the hardware complexity of the IF-PSU increases linearly with N. Moreover, the number of logic gate stages in the critical path also increases with N. This translates into a reduction of the maximum frequency as N grows.

In [8], a recursive construction of a PSU called the *feedback part* (FB-PSU) is proposed. To the best of our knowledge, this architecture has not been implemented. However, from the description of the structure one can observe that the FB-PSU is composed of $(n-1)$ stages. Each one of them contains $D_l = \left(\frac{N}{2^{\log_2(N)-l+1}} + \frac{N}{2^{\log_2(N)-l+2}} \times (2^{l-2}-2)\right)$ DFFs. Therefore the total number of DFFs is $\sum_{l=2}^{n}(D_l)$. Thus $\left(\frac{N^2-4}{12}\right)$ DFFs are necessary to implement the FB-PSU along with $\left(\dfrac{N}{2}-1\right)$ XOR gates and $(N-2)$ multiplexers. Finally, the authors reported that the critical path goes through $(\log_2(N)-1)$ XOR gates and $(\log_2(N)-2)$ multiplexers meaning that the maximum clock frequency is affected as N grows.

In this paper, a reduced complexity PSU architecture is described. The critical path includes few logic gates which enables to reach a high working frequency. Moreover, the proposed structure can also be used as a sequential polar code encoder.

3.2. Shift-register-based partial sums computation unit (SR-PSU)

During SC decoding process, a maximum of $\frac{N}{2}$ g functions can be performed in parallel. Therefore, we propose to store

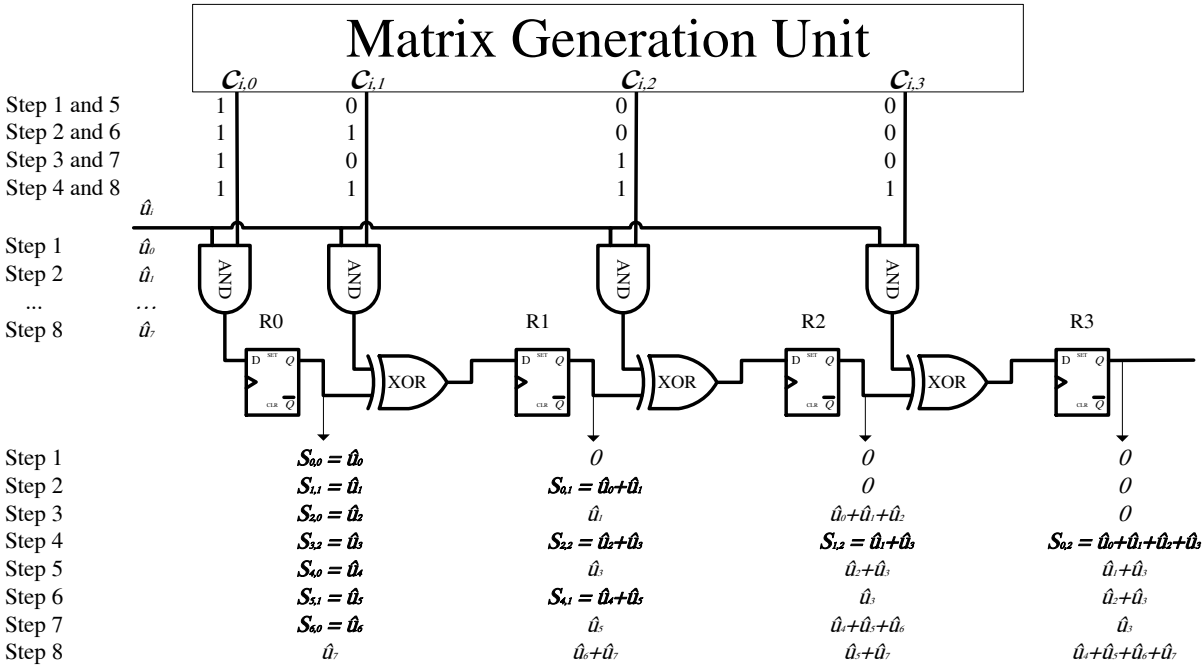


Fig. 4. SR-PSU example for $N = 8$.

the partial sums in an $\frac{N}{2}$ bit register. In addition to the $\frac{N}{2}$ DFFs, it consists of $\frac{N}{2}$ XOR gates, $\frac{N}{2}$ AND gates and a *matrix generation unit* whose structure is detailed in the next section. Fig. 4 details the proposed partial sums computation structure for $N = 8$.

In this architecture, each DFF R_k receives the value of R_{k-1} which is first added (XOR) with the current decoded bit $\hat{u}_i$ if the control bit $c_{i,k} = 1$. This architecture can be devised for any code length N according to the following rule:

$$\begin{cases} R_0 & \Leftarrow & \hat{u}_i \text{ AND } c_{i,0} \\ R_k & \Leftarrow & R_{k-1} \text{ XOR } (\hat{u}_i \text{ AND } c_{i,k}) \quad \text{if } k > 0 \end{cases} \tag{7}$$

In Fig. 4, the current value of the shift-register is given for each step. A step corresponds to the generation of a new bit $\hat{u}_i$. This structure generates all the required partial sums (bold values in Fig. 4). This shift-register structure was selected so that all partial sums required by a PE are all generated in the same DFF. It means that this SR-PSU can be included in a line SC decoder by simply connecting the $\frac{N}{2}$ PEs to a single DFF R_k. This avoids any extra multiplexing logic to route the partial sums to the PEs. This PSU can then be used as such for a tree or a line SC decoder. For the semi-parallel architecture, some multiplexing logic is required, exactly like in [5].

Although this architecture produces partial sums, it also encodes an $\frac{N}{2}$-bit vector. In Fig. 4, at step 4, each DFF R_k contains the bit $x_{\frac{N}{2}-1-k}$ such that $X = U \times \kappa^{\otimes 2}$. A polar code encoder for a code length of N can then be devised with N DFFs, N AND gates, $(N-1)$ XORs and a matrix generation unit. After bits u_i are sequentially shifted during N clock cycles, the codeword X is contained in the register. To the best of our knowledge, this is the first reported sequential encoder for polar codes.

3.3. Matrix generation unit

Let us define the control matrix as $C = \begin{bmatrix} \kappa^{\otimes n-1} \\ \kappa^{\otimes n-1} \end{bmatrix}$ whose element $c_{i,k}$ is the k^{th} control bit generated at step i. In order to implement the generation of the matrix, a naive approach is to store C in a ROM of size $N \times \frac{N}{2}$. This would be very complex to implement especially for large N values. For this reason, a solution based on a linear feedback shift register (LFSR) is proposed to generate the matrix $\kappa^{\otimes n-1}$. As shown in Fig. 4, the matrix generation unit has to produce the rows of C sequentially. We propose to use an LFSR of size $\frac{N}{2}$ to generate this sequence. In such a structure, the state of the LFSR at step i corresponds to the i^{th} row of C. By observing the matrix C, one can verify that:

$$\begin{cases} c_{i,0} & = & 1 & 0 \leq i \leq N-1 \\ c_{i+1,k} & = & c_{i,k-1} \text{ XOR } c_{i,k} & 0 \leq i < N-1 \\ & & & 1 \leq k \leq \frac{N}{2} - 1 \end{cases} \tag{8}$$

Let us denote the DFF that generates $c_{i,k}$ as M_k. From Equation 8, one can deduce that matrix C is generated by applying the following mapping rule to the LFSR:

$$\begin{cases} M_0 & \Leftarrow 1 \\ M_k & \Leftarrow M_k \text{ XOR } M_{k-1} \quad \text{if } k > 0 \end{cases} \tag{9}$$

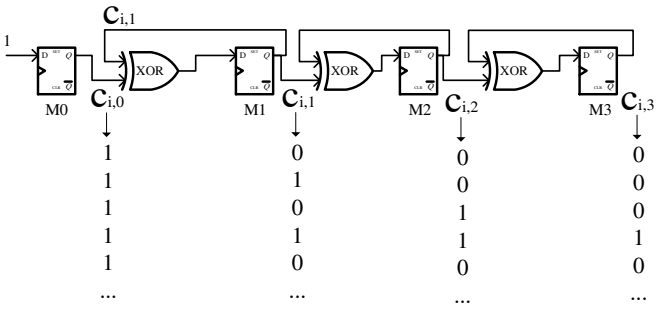


Fig. 5. Matrix generation unit.

This is illustrated in Fig. 5 where the matrix $\kappa^{\otimes 2}$ is generated twice which correspond to the matrix C for $N = 8$.

4. IMPLEMENTATION RESULTS AND COMPARISONS

To the best of our knowledge IF-PSU [4] is the only implemented partial sums unit in the literature. In this section, SR-PSU logic synthesis results are provided and compared with IF-PSU in terms of area and maximum working frequency. All syntheses were performed using the low power ST Microelectronics 65nm standard cell library with supply voltage 1.0V and nominal temperature 25°C. The FB-PSU architecture, introduced in [8], was not implemented. However, the authors give some insights on the hardware complexity and the critical path depth. These architectural estimations are used to perform the comparison with the proposed SR-PSU.

4.1. Functional verification and implementation methodology

The PSU architecture introduced in section 3 was included in a tree SC decoder described in VHDL. A set of tree SC decoders was generated for different codelengths ($2^3 < N < 2^{10}$)[a]. The resulting designs were synthesized in ASIC technology and validated by post-synthesis simulations with more than 2500 test vectors. These test vectors were obtained by a software SC decoder reference simulator that includes an AWGN channel model. Noisy codewords were generated at 7 different SNR values ranging from 0dB to 3dB. The behavior of IF-PSU and SR-PSU can be considered identical since in both architectures, $\hat{u}_i$ is shifted in sequentially and partial sums are generated in parallel. In order to perform a fair comparison, IF-FSU and SR-FSU were synthesized with the same technology and constraints.

4.2. Hardware complexity

In order to have a fair comparison between the two architectures in terms of area, the clock frequency was set to a relatively low value ($F_{CK} = 500$MHz) for the ST 65nm technology, so that the synthesis tool does not insert extra buffers or large cells in the design. Thus, the reported area corresponds to the minimum achievable hardware complexity. IF-PSU and SR-PSU designs were synthesized for different codelengths, $2^{10} \leq N \leq 2^{14}$. For both architectures, the reported area is linear with N. In average, for all investigated codelengths, the SR-PSU architecture is 12% smaller than the corresponding IF-FSU architecture.

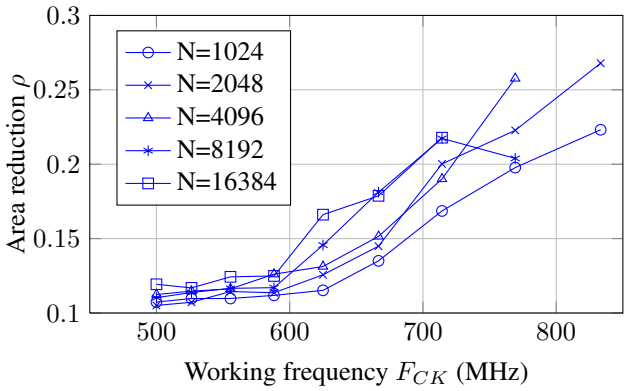


Fig. 6. Area reduction *vs* working frequency.

The same designs were synthesized for higher frequency values. In such a case, the synthesis tool optimizes the design resulting in an increased area. Fig. 6 shows the area reduction provided by the SR-PSU architecture for a range of target clock frequencies. The area reduction is calculated as $\rho = (\text{Area}_{\text{IF-PSU}} - \text{Area}_{\text{SR-PSU}})/\text{Area}_{\text{IF-PSU}}$. The area reduction significantly increases with the clock frequency and the codelength. It reaches 26% for $N = 4096$ and $F_{CK} = 769$MHz. For each curve, the highest reported frequency corresponds to the maximum achievable frequency for the IF-PSU. In the case of SR-PSU, frequency can be pushed further as seen in the following section. The different curves show that a significant area reduction is achieved.

The hardware complexity estimations of FB-PSU are reported in Table 1 in terms of DFFs, XORs and MUXes as a function of N. These values are compared with the SR-PSU hardware complexity. The estimated gate count is obtained by replacing each logic operator (DFF, XOR, ...) with its equivalent NAND gate count provided by the datasheet of the ST 65nm ASIC library. The IF-PSU gate count is estimated after the area synthesis results. For a small code length, $N = 1024$, FB-PSU gate count is roughly 440,000 gates while the SR-PSU architecture only consists of 7,680 gates for the same codelength. In fact, the very high complexity of the FB-PSU is due to the number of DFFs that grows with N^2, making this architecture non realistic for large codelengths. Since these estimations are carried out at a low frequency (500MHz, no optimization), the SR-PSU and IF-PSU seem to be equivalent. Nevertheless, as seen in Fig. 6, the IF-PSU area increases

[a]Larger codelengths were not verified due to very long post-synthesis simulation runtime

	FB-PSU	SR-PSU	IF-PSU
DFF	$\frac{N^2-4}{12}$	N	no data
XOR	$\frac{N}{2}-1$	$N-2$	no data
MUX	$N-2$	-	no data
AND	-	$\frac{N}{2}-1$	no data
NAND equivalent	$\boldsymbol{\frac{5N^2}{12}+3N}$	$\boldsymbol{\frac{15}{2}N}$	$\boldsymbol{\frac{17}{2}N}$

Table 1. Estimated NAND gate count comparison.

more than the SR-PSU.

4.3. Working frequency

In order to estimate the maximum working frequency of IF-PSU and SR-PSU architectures, all designs were synthesized under increasing timing constraints until the synthesis tool fails at meeting the timing constraint. Fig. 7 shows that for both architectures, the maximum frequency decreases with N. For each N value, the SR-PSU reaches a higher maximum frequency than the IF-PSU. This confirms that the proposed SR-PSU has a shorter critical path than the IF-PSU. A detailed analysis of the synthesized SR-PSU designs show that the critical path starts from input $\hat{u}_i$ and ends on each of the N DFFs. This means that input $\hat{u}_i$ drives N AND gates resulting in a high fanout net. The synthesis tool has to insert extra buffers on this path in order to meet the timing constraint. This explains that in spite of the constant logic gate number included in the critical path, the SR-PSU maximum working frequency decreases with N.
As mentioned in section 3.1, FB-PSU critical path is composed of $(\log_2(N)-1)$ XOR gates and $(\log_2(N)-2)$ MUXes while the SR-PSU consists of only 1 AND gate and 1 XOR gate. This should result in a lower clock frequency. In the FB-PSU, despite the longer critical path, the input $\hat{u}_i$ has a smaller fanout which may result in a reasonable frequency. However the high hardware complexity of the FB-PSU design makes the routing phase critical and consequently affects the maximum working frequency.

5. CONCLUSION AND PERSPECTIVES

The hardware implementation of polar code decoders is a decisive step towards their potential inclusion in future digital telecommunication standards. Recent works paved the way for the definition of efficient decoder architectures. In current state of the art polar code successive cancellation decoders, the limiting element is the partial sums computation unit. In this paper, we propose a new partial sums computation architecture with improved working frequency and reduced hardware complexity. The resulting design was verified and synthesized using ASIC 65nm technology and favorably compares with state of the art partial sums units. Moreover, we also showed how this structure can be used as a sequential polar encoder. This new computation method opens the way for several interesting research topics such as the extension of this architecture to higher kernels or the enhancement of parallelism in this structure.

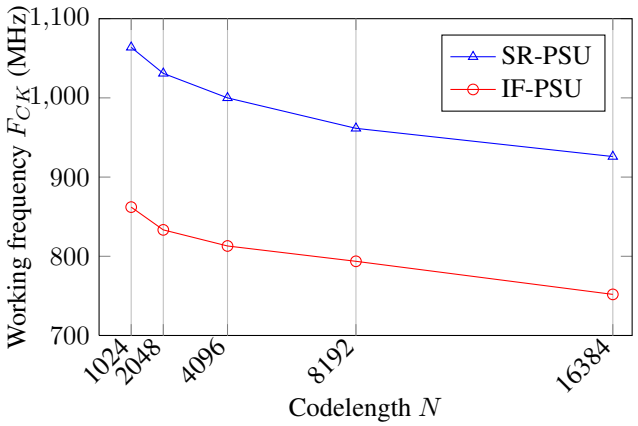


Fig. 7. Maximum working frequency *vs* codelength.

6. REFERENCES

[1] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *arXiv:0807.3917*, July 2008.

[2] E. Sasoglu, E. Telatar, and E. Arikan, "Polarization for arbitrary discrete memoryless channels," *arXiv:0908.0302*, 2009.

[3] C. Leroux, I. Tal, A. Vardy, and W.J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," in *2011 IEEE ICASSP*, May 2011.

[4] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J. Gross, "Hardware implementation of successive-cancellation decoders for polar codes," *Journ. of Sig. Proc. Syst.*, Dec. 2012.

[5] C. Leroux, A. Raymond, G. Sarkis, and W. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *Signal Processing, IEEE Transactions on*, 2012.

[6] A. Mishra, A. J. Raymond, L. G. Amaru, G. Sarkis, C. Leroux, P. Meinerzhagen, A. Burg, and W.J. Gross, "A successive cancellation decoder ASIC for a 1024-bit polar code in 180nm CMOS," in *Asian Solid-State Circuits Conference*, Nov. 2012.

[7] C. Zhang, B. Yuan, and K. K. Parhi, "Low-latency SC decoder architectures for polar codes," *arXiv:1111.0705*, Nov. 2011.

[8] C. Zhang and K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Transactions on Signal Processing*, 2013.

[9] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Communications Letters*, Dec. 2011.

[10] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders," *IEEE Communications Letters*, 2013.