ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 3, Issue 8, August 2016

An Advanced Architecture with low Complexity of Partially Parallel Polar Encoder

[1] U.Mahendra Narasimha Raj [2] E.V.Narayana
[1] PG Student [2] Assistant Professor
[1][2] Department of ECE, UCEK, JNTUK, KAKINADA

Abstract: **-- Due to the channel achieving property, the polar code has become one of the most favorable error-correcting codes. As the polar code achieves the property asymptotically, however, it should be long enough to have a good error-correcting performance. Although the previous fully parallel encoder is intuitive and easy to implement, it is not suitable for long polar codes because of the huge hardware complexity required. In this brief, we analyze the encoding process in the viewpoint of very-large-scale integration implementation and propose a new efficient encoder architecture that is adequate for long polar codes and effective in alleviating the hardware complexity. As the proposed encoder allows high-throughput encoding with small hardware complexity, it can be systematically applied to the design of any polar code and to any level of parallelism.**

***Index Terms*—Polar codes, polar encoder, very-large-scale integration (VLSI) optimization**

I. INTRODUCTION

Polar Code is a new class of error-correcting codes that provably achieves the capacity of the underlying channels. In addition, concrete algorithms for constructing, encoding, and decoding the code are all developed [1]–[5]. Due to the channel capacity achieving property, the polar code is now considered as a major breakthrough in coding theory, and the applicability of the polar code is being investigated in many applications, including data storage devices [6], [7].

Although the polar code achieves the underlying channel capacity, the property is asymptotical since a good error- correcting performance is obtained when the code length is sufficiently long. To be close to the channel capacity, the code length should be at least 220 bits, and many literature works [7] – [9] Introduced polar codes ranging from 210 to 215 to achieve good error-correcting performances in practice. In addition, the size of a message protected by an error-correcting code in storage systems is normally 4096 bytes, i.e., 32 768 bits, and is expected to be lengthened to 8192 bytes or 16 384 bytes in the near future. Although the polar code has been regarded as being associated with low complexity, such a long polar code suffers from severe hardware complexity and long latency. Therefore, an architecture that can efficiently deal with long polar codes is necessary to make the very-large-scale integration (VLSI) implementation feasible.

Various theoretic aspects of the polar code, including Code construction and decoding algorithms, have been investigated in previous works [1]–[5], and efficient decoding structures have been studied. Successive cancelation (SC) decoding has been traditionally used in [9]–[11], and advanced decoding algorithms such as belief propagation decoding [12], list de- coding [13], and implied SC [7], [14] have been recently employed.

On the other hand, hardware architectures for polar encoding have rarely been discussed. Among a few manuscripts dealing with hardware implementation, [1] presented a straight- forward encoding architecture that processes all the message bits in a fully parallel manner. The fully parallel architecture is intuitive and easy to implement, but it is not suitable for long polar codes due to excessive hardware complexity. In addition, the partial sum network (PSN) for an SC decoder [7], [8], [11] is regarded as a polar encoder. Due to the nature of successive decoding, however, the number of inputs is severely restricted in the PSN, 1 or 2 bits at a time. Since a polar encoder usually takes the inputs from a buffer or memory of which bit width is much larger, the PSN is not appropriate for designing a general polar encoding architecture. For the first time, this brief analyzes the encoding process in the viewpoint of VLSI implementation and proposes a partially parallel architecture. The proposed encoder is highly attractive in implementing a long polar encoder as it can achieve a high throughput with\ small hardware complexity.

II. POLAR E NCODING

The polar code utilizes the channel polarization phenomenon that each channel approaches either a perfectly

ISSN (Online) 2394-6849
International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 3, Issue 8, August 2016

reliable or a completely noisy channel as the code length goes to infinity over a combined channel constructed with a set of N identical subchannels [1]. As the reliability of each subchannel is known *a priori*, K most reliable subchannels are used to transmit information, and the remaining subchannels are set to predetermined values to construct a polar (N, K) code. Since the polar code belongs to the class of linear block codes, the encoding process can be characterized by the generator matrix. The generator matrix GN for code length N or $2n$ is obtained by applying the nth Kronecker power to the kernel matrix F. Given the generator matrix, the codeword is computed by $x = uGN$, where u and x represent information and codeword vectors, respectively. Throughout this brief, we assume that information vector u is arranged in a natural order, whereas codeword vector x is arranged in a bit-reversed order to simplify the explanation on the encoding process. A straightforward fully parallel encoding architec- ture was presented in [1], which has encoding complexity of $O(N \log N)$ for a polar code of length N and takes n stages when $N = 2n$. For example, a polar code with a length of 16 is implemented with 32 XOR gates and processed with four stages, as depicted in Fig. 1. In the fully parallel encoder, the whole encoding process is completed in a cycle.

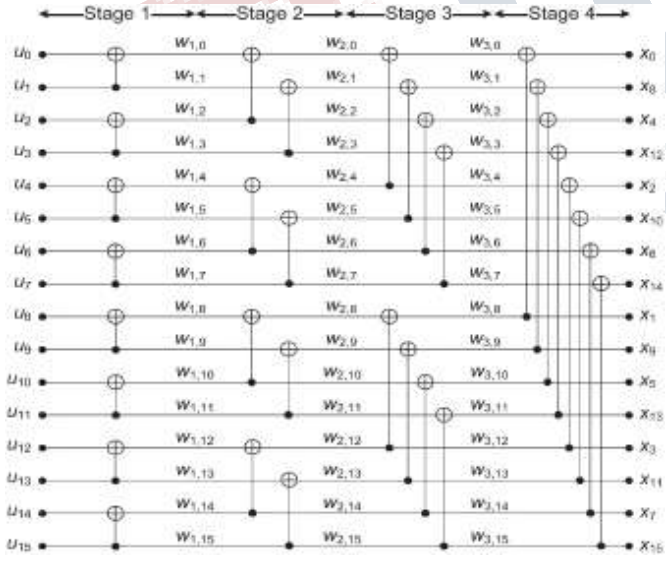
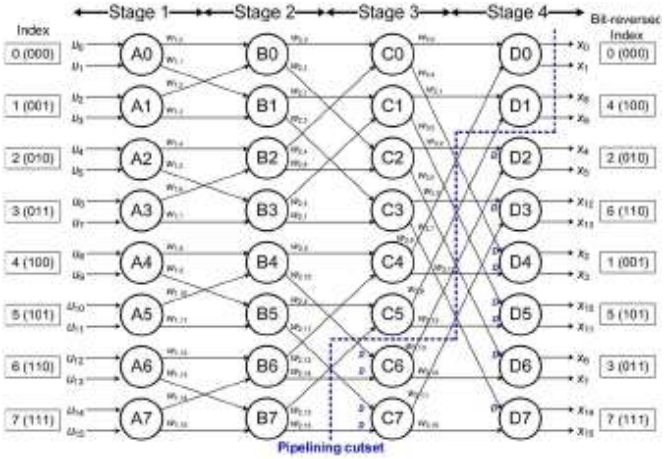


Fig. 1. Fully parallel architecture for encoding a 16-bit polar code



The fully parallel encoder is intuitively designed based on the generator matrix, but implementing such an encoder becomes a significant burden when a long polar code is used to achieve a good error correcting performance. In practical implementations, the memory size and the number of XOR gates increase as the code length increases. None of the previous works has deeply studied how to encode the polar code efficiently, although various tradeoffs are possible between the latency and the hardware complexity.

III. P ROPOSED POLAR E NCODER

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 [15], [18], and then the lifetime analysis [16] and register allocation [17] are applied to the folded architecture. Lastly, the proposed parallel architecture for long polar codes is described

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$D(w_{1,j})$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$D(w_{2,j})$	1	1	2	2	0	0	1	1	1	1	-2	-2	0	0	-3	-3
$D(w_{3,j})$	2	2	-2	-2	-0	-0	-0	-0	0	0	0	0	-2	-2	2	2
$D'(w_{1,j})$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$D'(w_{2,j})$	1	1	2	2	0	0	1	1	1	1	2	2	0	0	1	1
$D'(w_{3,j})$	2	2	2	2	4	4	4	4	0	0	0	0	2	2	2	2

Fig. 3. Original delay requirements D(wij) and

ISSN (Online) 2394-6849
International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 3, Issue 8, August 2016

Recalculated delay require- MENTS D (WIJ)

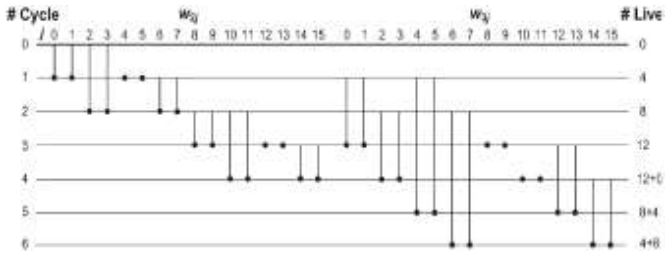


Fig. 4. Linear lifetime chart for w2j and w3j .

A. Folding Transformation

The folding transformation [15], [18] 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. 2, where a node represents the kernel matrix operation F , and wij denotes the j th edge at the ith stage. Note that the DFG of the fully parallel polar encoder is similar to that of the fast Fourier transform [18], [19] 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 [15]. 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 $A2$ and

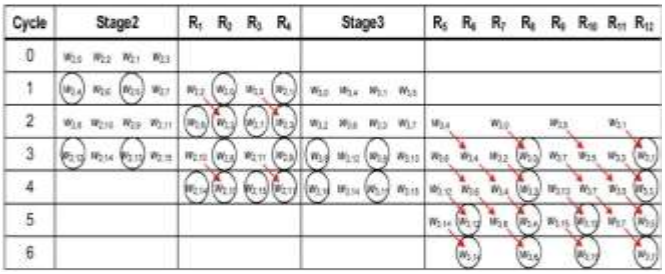


Fig. 5. Register allocation table for w2j and w3j .

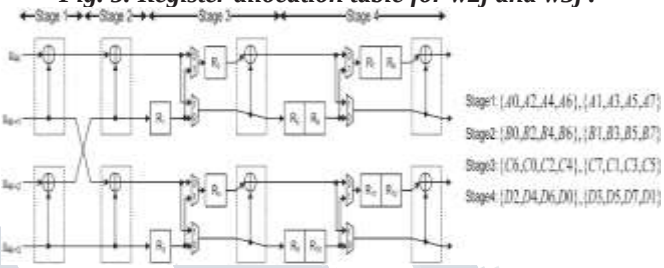


Fig. 6. Proposed 4-parallel folded architecture for encoding the polar (16, K) codes.

$A3$ at the next cycle, and so forth. The folding sets of stage 2 have the same order as those of stage 1, i.e., $\{B\ 0, B\ 2, B\ 4, B\ 6\}$ and $\{B\ 1, B\ 3, B\ 5, B\ 7\}$, since the four-parallel input sequence of stage 2 is equal to that of stage 1. Furthermore, to determine the folding sets of another stage s, the property that the functional unit processes a pair of inputs whose indices differ by $2s-1$ 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, $w2,0$ and $w2,4$ that come from $B\ 0$ and $B\ 2$ are used as the inputs to $C\ 0$. 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 $\{C\ 6, C\ 0, C\ 2, C\ 4\}$ and $\{C\ 7, C\ 1, C\ 3, C\ 5\}$, where $C\ 6$ in the current iteration is overlapped with $A0$ and $B\ 0$ in 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 $\log2\ P$, where P is the level of

IFERP
connecting engineers... developing research

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)

Vol 3, Issue 8, August 2016

parallelism, has the same folding sets determined by evenly interleaving the operations in the consecutive order, and another stage whose index s is larger than log2 P has the folding sets obtained by cyclic shifting the previous folding sets of stage $s - 1$ right by $s - \log 2\ P$.

Now, let us consider the delay elements required in the folded architecture more precisely. When an edge wij from functional unit S to functional unit T has a delay of d, the delay requirements for wij in the F -folded architecture can be calculated as

TABLE I
COMPARISON OF POLAR (N, K) ENCODERS WITH VARIOUS PARALLELISM

Parallelism	XOR gates	Delay elements	Throughput
1	$\log_2 N$	$N-1$	1 bit/cycle
2	$\log_2 N$	$N-2$	2 bits/cycle
P	$\lceil P/2 \rceil \log_2 N$	$N-P$	P bits/cycle
N	$(N/2)\log_2 N$	0	N bits/cycle

$D(wij\ ) = F\ d + t - s$ (1) where t and s denote the position in the folding set corresponding to T and S , respectively. Note that (1) is a simplified delay equation [15] derived with assuming that the kernel functional unit is not pipelined. The delay requirements of the 4-folded architecture, i.e., $D(wij\ )$ for $1 \le i \le 3$ and $0 \le j \le 15$, are summarized in Fig. 3. For instance, $w2,0$ from B 0 to C 0 demands one delay since $d = 0$, $t = 1$, and $s = 0$. Note that some edges indicated by circles have negative delays. For the folded architecture to be feasible, the delay requirements must be larger than or equal to zero for all the edges. Pipelining or retiming techniques can be applied to the fully parallel DFG in order to ensure that its folded hardware has nonnegative delays.

Every edge with a negative delay should be compensated by inserting at least one delay element to make the value of (1) not negative. We have to make sure that the two inputs of an operation pass through the same number of delay elements from the starting points. If they are different, additional delay

TABLE II
SYNTHESIS RESULTS OF POLAR (8192, K) ENCODERS

Parallelism	32	128	512	2048	8192[†]
Critical-Path Delay	2.79 ns	2.81 ns	2.89 ns	2.97 ns	3.86 ns
Latency (Clock cycles)	256	64	16	4	1
Throughput [A]	11.47 Gbps	45.55 Gbps	177.1 Gbps	689.7 Gbps	2.077 Tbps
Gate Count [B] (Logic/Register)	[illegible]	[illegible]	[illegible]	[illegible]	373141 ([illegible])
[B] / [C]	0.27	0.39	0.57	0.56	1.89
[A] / [B] (Mbps/Gate count)	0.12	0.43	1.36	2.46	7.18

TABLE III
GATE COUNTS OF POLAR (N , K) ENCODERS

Code length	1024		2048		4196		8192		16384	
Design	[1]	Proposed*	[1]	Proposed*	[1]	Proposed*	[1]	Proposed*	[1]	Proposed*
Logic	13872	1486 (10.7%)	30812	1651 (5.4%)	67692	1813 (2.7%)	158848	2127 (1.3%)	371776	2490 (0.7%)
Register	7211	7283 (101.0%)	16512	16661(100.9%)	33000	33165(100.5%)	66122	66337 (100.3%)	132812	133121(100.2%)
Total	21083	8769 (41.6%)	47324	18312 (38.7%)	100692	34978 (34.7%)	224970	68464 (30.4%)	504588	135611(26.9%)

*The proposed partially parallel encoder is design with a parallelism of 32.

Elements are inserted to make the paths have the same delay elements. In Fig. 3, for example, four edges with zero delays are specially marked with negative zeros since additional delays are necessary due to the mismatch of the number of delay elements. The DFG is pipelined by inserting delay elements, as shown in Fig. 2, where the dashed line indicates the pipeline cut set associated with 12 delay elements. The delay requirements of the pipelined DFG D (wij) are recalculated based on (1) and shown at the bottom of Fig. 3. As a result, 8 functional units and 48 delay elements in total are enough to implement the 4-parallel 4-folded encoding architecture based on the folding sets.

B. Lifetime Analysis and Register Allocation

Although a folded architecture for 16-bit polar encoding is presented in the previous section, there is room for minimizing the number of delay elements. The lifetime analysis [16] is employed to find the minimum number of delay elements required in implementing the folded architecture. The lifetime of every variable is graphically represented in the linear lifetime chart illustrated in Fig. 4. Since all the edges starting from stage 1 demand no delay elements, only $w2j$ and $w3j$ are presented in Fig. 4. For instance, $w3,0$ is alive for two cycles as it is produced at cycle 1 and consumed at cycle 3. The number of variables alive in each cycle is presented at the right side of the chart. Note that the number of live variables at the fourth or later clock cycles takes into account the next iteration overlapped with the current iteration. Consequently, the maximum number of live variables is 12, which means that the folded

ISSN (Online) 2394-6849
International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 3, Issue 8, August 2016

architecture can be implemented with 12 delay elements instead of 48. Once the minimum number of delay elements has been determined, each variable is allocated to a register. For the above example, the register allocation is tabularized in Fig. 5. In the register allocation table [17], all the 12 registers are shown at the first row, and every row describes how the registers are allocated at the corresponding cycle. With taking into account the 4-parallel processing, variables are carefully allocated to registers in a forward manner. In Fig. 5, an arrow dictates that a variable stored in a register is migrated to another register, and a circle indicates that the variable is consumed at the cycle. For example, $w2,0$ and $w2,4$ are consumed in a functional unit to execute operation C 0 that generates $w3,0$ and $w3,4$. At the same time, $w2,1$ and $w2,5$ are consumed in another functional unit to execute operation C 1 that produces $w3,1$ and $w3,5$. The migration of the other variables can be traced by following the register allocation table.

Finally, the resulting 4-parallel pipelined structure proposed to encode the 16-bit polar code is illustrated in Fig. 6, which consists of 8 functional units and 12 delay elements. A pair of two functional units takes in charge of one stage, and the delay elements are to store variables according to the register allocation table. The hardware structures for stages 1 and 2 can be straightforwardly realized as no delay elements are necessary in those stages, whereas for stages 3 and 4, several multiplexers are placed in front of some functional units to configure the inputs of the functional units. The proposed architecture continuously processes four samples per cycle according to the folding sets and the register allocation table. Note that the proposed encoder takes a pair of inputs in a natural order and generates a pair of outputs in a bit-reversed order, as shown in Fig. 2. As the functional unit in the proposed architecture processes a pair of 2 bits at a time, the proposed architecture maintains the consecutive order at the input side and the bitreversed order at the output side if a pair of consecutive bits is regarded as a single entity.

IV. A NALYSIS AND C OMPARISON

In the proposed architecture, the number of functional units required in the implementation depends on the code length N and the level of parallelism P . Since a functional unit representing the kernel matrix F processes two bits at a time, each stage necessitates $P/2$ functional units and the whole structure requires $P/2 \log2 N$ functional units in total. Moreover, the minimal number of delay elements required in the proposed architecture is $N - P$, as explained below. The stages whose indices s are larger than $\log2 P$ require P delay blocks of length $2s-\log2 P -1$, whereas the other stages can be implemented with no delay elements. In other words, the total number of delay elements is $\log2 N$

$$P (2s-\log2 P -1) = P (1 + 2 + s=\log2 P +1 + 2\log2 N -\log2 P -1) = P (2\log2 N -\log2 P - 1) = N - P. \quad (2)$$

Given the hardware resources, the proposed partially parallel architecture can encode P bits per cycle. To sum up, Table I shows how the hardware complexity and the throughput are dependent on the level of parallelism Furthermore, Table II demonstrates the proposed (8192, K) encoder architecture synthesized in a 130-nm CMOS technology for various parallelism. As the level of parallelism increases, the hardware complexity measured in terms of the gate count is significantly deteriorated due to the complex logic part, whereas the register part in all encoder architectures maintains similar complexity if we take into account a P -bit input buffer needed to hold the data to be read from the memory. On the other hand, the higher parallel architecture has advantages of small latency and high encoding throughput. Therefore, the relationship shown in Table II can be applied to derive the most efficient partially parallel encoder architecture for a given requirement. The throughput per gate is proportional to the level of parallelism as the complexity of the register part is almost independent of the parallelism. Moreover, Table III shows how much the partially parallel encoders save the hardware complexity compared with the fully parallel architecture [1] for various code lengths. For fair comparison, all the encoders designed for the code lengths ranging from 210 to 214 are constrained by a working frequency of 200 MHz to assure a decoding performance over 6.4 Gb/s even for the 32-parallel architecture. Note that the percentage in the parenthesis indicates the ratio of the proposed encoder to the fully parallel encoder. Compared with the fully parallel encoder, the proposed encoder saves the hardware by up to 73%. can save the hardware by up to 73%

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 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. There- fore, the proposed architecture provides a practical solution for encoding a long polar code.

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," in *Proc. 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," in *Proc. 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 hard- ware decoder for polar codes," in *Proc. 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.

[13] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation list decoders for polar codes with multibit decision," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, DOI: 10.1109/TVLSI.2014.2359793, to b published.

[14] C. Zhang and K. K. Parhi, "Latency analysis and architecture design of simplified SC polar decoders," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 2, pp. 115–119, Feb. 2014.

[15] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. Hoboken, NJ, USA: Wiley, 1999.

[16] K. K. Parhi, "Calculation of minimum number of registers in arbitrary life time chart," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 41, no. 6, pp. 434–436, Jun. 1995.

[17] C. Wang and K. K. Parhi, "High-level DSP synthesis using concurrent transformations, scheduling, allocation," *IEEE Trans. Comput.Aided Design Integr. Circuits Syst.*, vol. 14, no. 3, pp. 274–295 Mar. 1995.

[18] M. Ayinala, M. J. Brown, and K. K. Parhi, "Pipelined parallel FFT architectures via folding transformation," *IEEE Trans. Very Large Scale Integr.(VLSI) Syst.*, vol. 20, no. 6, pp. 1068–1081, Jun. 2012.

[19] C. Y. Wang, "MARS: A high-level synthesis tool for digital signal processing architecture design," M.S. thesis, Dept. Elect. Eng., University of Minnesota, Minneapolis, MN, USA, 1992.