AN EFFICIENT DESIGN OF PARALLEL PIPELINED ENCODER ARCHITECTURE FOR LONG POLAR CODES

MONISHA.D[1], ARUL KARTHICK.V.J[2]

[1]PG scholar, Department of ECE, SNS college of technology ,coimbatore , India,

[2]Assistant professor, Department of ECE, SNS college of technology ,coimbatore , India.

---***---

Abstract - *Polar codes represent an emerging class of error-correcting codes with power to approach the capacity of a discrete memory less channel.The main objective is to perform error correction and detection. The proposed new efficient 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.The delay elements can be reduced by new parallel pipelined architecture.This particular architecture uses folding transformation technique as well as register minimization.Pipelining and parallel processing is used to reduce the power consumption.*

***Key Words*:** **Polar codes, polar encoder,polar decoder,pipelining,very-large-scale integration (VLSI)optimization.**

1.INTRODUCTION

Approaching capacity with a practical en/decoding complexity is a central challenge in coding theory."turbo-like" code families,such as turbo codes and low-density parity-check(LDPC) codes, have been found to achieve this goal. The key issue is how to practically implement the ideas used in the proof of the channel coding theorem. Coding randomness is introduced by interleavers in turbo codes or by pseudo-random connections between the variable and check nodes in LDPC codes. Among a few manuscripts dealing with hardware implementation, presented a straightforward 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. 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.polar code is a linear block error correcting code developed by Erdal Arıkan. It is the first code to provably achieve the channel capacity for symmetric binary-input, discrete, memoryless channels.polar codes were constructed using a generator matrix created using the Kronecker power of the base matrix .This paper is organised into V sections where section III represents the proposed folding transformation IV section represents register allocation.

2.EXISTING METHOD

Arıkan showed that SC decoding can be efficiently implemented by the factor graph of the code which has a structure that of the Fast Fourier transform. Fast Fourier Transform (FFT) is a commonly used technique for the computation of Discrete Fourier Transform (DFT). DFT computations are required in the fields like filtering,spectral analysis etc. to calculate the frequency spectrum or to identify a system's frequency response from its impulse response and vice versa. FFT is used in digital video broadcasting and OFDM systems. Much research has been carried out to design pipelined architectures for computation of FFT. The folding sets are designed in a way to reduce the number of storage elements and also the latency.The prior FFT architectures had no systematic way of approach. This architecture simplifies the design of FFT and is a systematic approach towards the design of FFT with arbitrary level of parallelism.

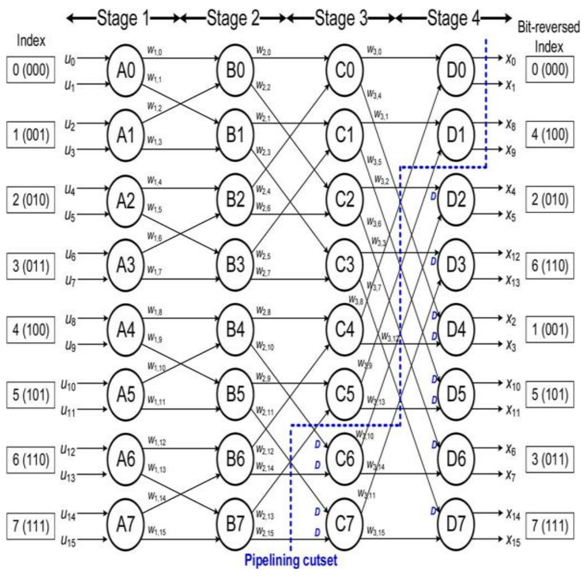


Fig- 1: DFG of 16-bit polar encoding

Fully parallel architecture for encoding a 16-bit polar code.The fully parallel encoder is 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.

3.PROPOSED METHOD

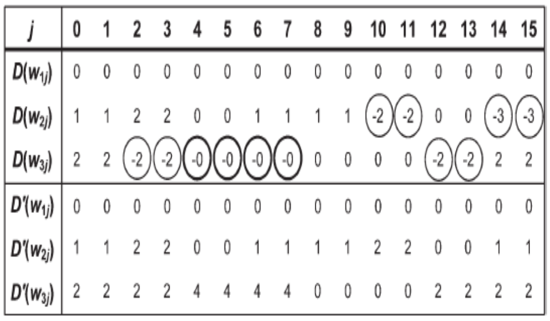
The folding transformation is widely used to save hardware resources by time-multiplexing several operations on a functional unit. A data flow graph (DFG) corresponding to the fully parallel encoding process for 16-bit polar codes is shown in Fig. 1 where a node represents the kernel matrix operation F, and wij denotes the jth edge at the ith stage. 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. 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 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 B0 and B2 are 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},where C6 in the current iteration is overlapped with A0 andB0 in the next iteration.

4.LIFETIME ANALYSIS AND REGISTER ALLOCATION

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

$$D(wij) = Fd + t - s$$

where t and s denote the position in the folding set corresponding to T and S, respectively. 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. For instance, w2,0 from B0 to C0 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. The two inputs of an operation pass through the same number of delay elements from the starting points. If they are different, additional delay elements are inserted to make the paths have the same delay elements. Four edges with zero delaysare specially marked with negative zeros since additional delays are necessary due to the mismatch of the number of delay elements. The delay requirements are recalculated based on units and 48 delay elements in total are enough to implement the 4-parallel 4-folded encoding architecture based on the folding sets.



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

Fig- 2: Delay Requirement calculation for 16bit DFG

The lifetime analysis is employed to find the minimum number of delay elements required in implementing the folded architecture[13]. The lifetime of every variable is graphically represented in the linear lifetime chart illustrated in Fig.3

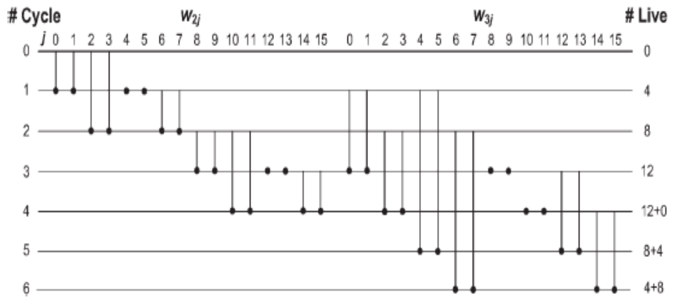
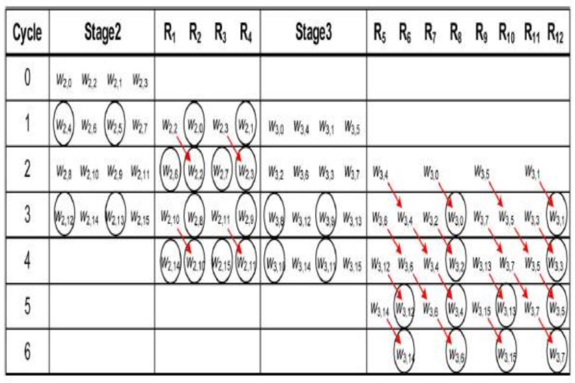


Fig- 3: Lifetime chart analysis.

Since all the edges starting from stage 1 demand no delay elements, only $w2j$ and $w3j$ are presented in 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.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 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.



Cycle	Stage2	R_1	R_2	R_3	R_4	Stage3	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}
0	$w_{2,0}$ $w_{2,2}$ $w_{2,1}$ $w_{2,3}$													
1	$w_{2,4}$ $w_{2,6}$ $w_{2,5}$ $w_{2,7}$	$w_{2,2}$	$w_{2,0}$	$w_{2,3}$	$w_{2,1}$	$w_{3,0}$ $w_{3,4}$ $w_{3,1}$ $w_{3,5}$								
2	$w_{2,8}$ $w_{2,10}$ $w_{2,9}$ $w_{2,11}$	$w_{2,6}$	$w_{2,2}$	$w_{2,7}$	$w_{2,3}$	$w_{3,2}$ $w_{3,6}$ $w_{3,3}$ $w_{3,7}$	$w_{3,4}$		$w_{3,0}$		$w_{3,5}$		$w_{3,1}$	
3	$w_{2,12}$ $w_{2,14}$ $w_{2,13}$ $w_{2,15}$	$w_{2,10}$	$w_{2,8}$	$w_{2,11}$	$w_{2,9}$	$w_{3,8}$ $w_{3,12}$ $w_{3,9}$ $w_{3,13}$	$w_{3,6}$	$w_{3,4}$	$w_{3,2}$	$w_{3,0}$	$w_{3,7}$	$w_{3,5}$	$w_{3,3}$	$w_{3,1}$
4		$w_{2,14}$	$w_{2,10}$	$w_{2,15}$	$w_{2,11}$	$w_{3,10}$ $w_{3,14}$ $w_{3,11}$ $w_{3,15}$	$w_{3,12}$	$w_{3,6}$	$w_{3,4}$	$w_{3,2}$	$w_{3,13}$	$w_{3,7}$	$w_{3,5}$	$w_{3,3}$
5							$w_{3,14}$	$w_{3,12}$	$w_{3,6}$	$w_{3,4}$	$w_{3,15}$	$w_{3,13}$	$w_{3,7}$	$w_{3,5}$
6								$w_{3,14}$		$w_{3,6}$		$w_{3,15}$		$w_{3,7}$

Fig- 4: Register allocation.

The register allocation is tabularized in Fig.4. In the register allocation table 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. An arrow indicates 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 $C0$ 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 $C1$ 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 Figure 5 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 the proposed encoder takes a pair of inputs in a natural order and generates a pair of outputs in a bit-reversed order.

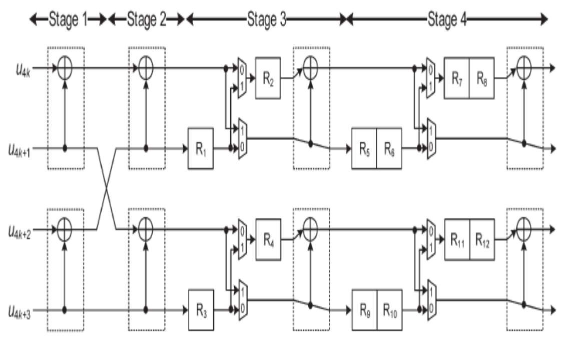


Fig- 5:4 parallel folded architecture

Input	Stage 1	Stage 2	Stage 3	Stage 4	output
1	0	0	0	1	1
1	1	1	1	0	0
0	0	0	0	1	1
0	0	0	0	1	1
0	0	0	0	1	1
0	0	0	0	1	1
0	0	0	0	1	1
0	0	0	0	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	1	1	1	1

0	0	1	1	1	1
0	1	1	1	1	1
1	1	1	1	1	1

Table -1: 16 Bit polar encoded bits

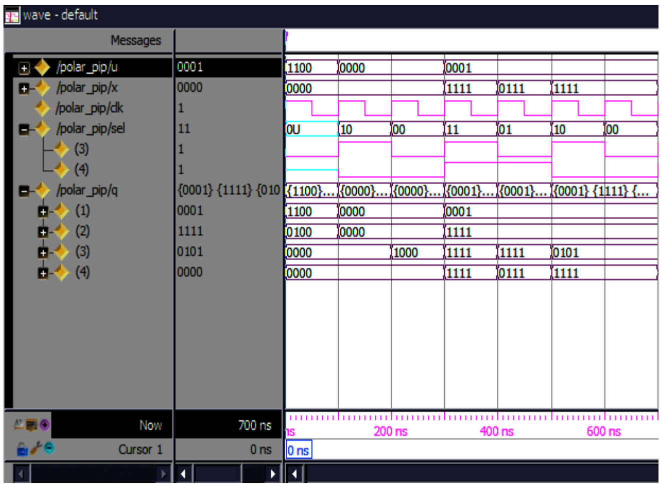


Fig- 6: Simulated output

5.CONCLUSION

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 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.

REFERENCES

[1]Hoyoung yoo and in-cheol park,"partially parallel encoder architcture for long polar codes,"IEEE transactions on circuits.,vol.62,no.3,mar.2015.
[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 hardware
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. 958–9664.
[13] K. K. Parhi, VLSI Digital Signal Processing Systems: Design and Implementation.Hoboken, NJ, USA: Wiley, 1999.