IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 6, Issue 6, Ver. II (Nov. - Dec. 2016), PP 87-95
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Implementation of Optimal Encoder Architecture for Long Polar Codes

K.Supriya, Dr.G.Mamatha
[1]*(ECE, GRIET/ JNTUH,INDIA)*
[2]*(ECE, GRIET/ JNTUH, INDIA)*

Abstract: *The polar encoding is one among the most effective error correcting code attributable to the channel achieving property. It finds its application in communication, information theory. This coding technique proposed by Eardal Arikan is significant because of its zero errors and simple architecture designed for hardware implementation. Although the fully parallel encoder is intuitive and easy to implement, but it is not suitable for long polar codes because of its hardware complexity. In this brief, we analyze the encoding process in the viewpoint of very-large-scale integration and implement a efficient encoder architecture that is suitable for long polar codes and effective in the hardware complexity. It can be applicable to the design of any polar code and at any level of parallelism.*

Keywords: *Partially parallel encoder, long polar codes, VLSI implementation*.

I. Introduction

Polar code is also brand new class of error correcting codes that incontrovertibly achieves the potential of the underlying channels [1]. As a result of the data rate achieving property, the polar code is presently thought of as a significant breakthrough to writing theory, and additionally the relevance of the polar code is being investigated in many applications, as well as information storage devices. The polar code achieves the underlying channel capacity, the property is essential since a good error correcting performance is obtained once the code length is sufficiently long. To be close to the information rate, the code length got to be a minimum of 220 bits, and many of literature works introduced polar codes ranging from 210 to 215 to achieve sensible error-correcting performances in addition, to the dimensions of a message protected by associate error correcting code in storage systems in mostly 4096 bytes i.e, thirty 2768 bits, and is predicted to be long to 8192 bytes or sixteen 384 bytes inside the close to future. Although the polar code has been thought about being with low Complexity, but it suffers from severe hardware complexity and long inactivity. Therefore, architecture that which will efficiently beware of long polar codes is necessary to make the VLSI implementation feasible. Successive cancelation(SC), secret writing has been traditionally used, and advanced secret writing algorithms like belief propagation secret writing, list secret writing, and simplified SC square measure recently utilized. On the other side, hardware architectures for polar coding have rarely been mentioned. Among a several manuscripts explaining the hardware implementation [1] presented a simple encoding architecture that processes all the message bits in a completely parallel manner. The fully parallel architecture is intuitive and simple to implement but it is not acceptable for long polar codes as a result of excessive hardware complexity. Proposed encoder is efficient in implementing a long polar encoder, as it can achieve better throughput with a simple hardware complexity. Hence we present the encoding process in viewpoint of VLSI implementation.

II. Polar Encoding

The polar code utilizes the channel polarization development that each channel approaches either a superbly reliable or a completely screaming channel as the code length goes to eternity over a combined channel constructed with set of N identical sub channel [1]. As the reliability of each sub channel is thought as priori, K most reliable sub channels are used to transmit information and the remaining sub channels are set to predetermined values to construct a polar (N,K)code. Since the polar code belongs to the category of linear block codes, the encoding process can be characterized by the generator matrix. The generator matrix G_N for code length N or 2^n is obtained by applying the *n*th kronecker power to the kernel matrix F=$\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}$ [1].Given the generator matrix, the codeword is computed by *x*=*u* G_N,where *u* represent information vector arranged in a natural order, and *x* represents codeword vector arranged in bit reversed order.

The encoding complexity of 0(NlogN) for a polar code of length N and take n stages when N= 2^n. For instance a polar code with a length of 16 bits is enforced with 32 XOR gates and processed with 4 stages within a total parallel encoder as shown in Fig1. The totally parallel encoder is intuitively designed and supported on the generator matrix, however implementing such associate encoder becomes a significant burden once an extended polar code is employed to achieve a good error correcting performance. The memory size and also range of XOR gates increase as the code length increases in implementations.

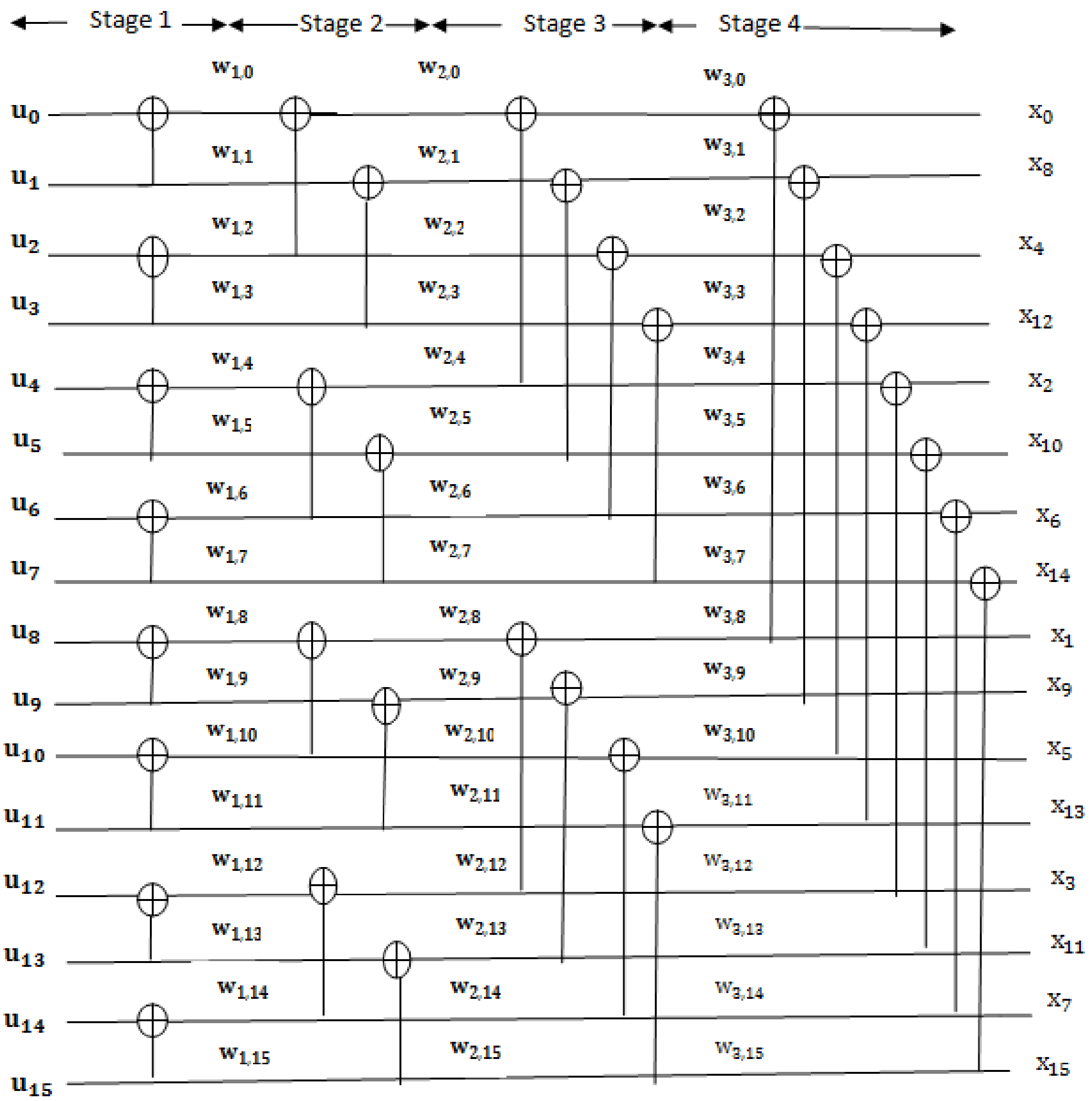


Figure 1. Fully parallel architecture for 16 bit polar code

III. Proposed Polar Encoder

To design partially parallel encoder first the parallel encoding architecture is transformed to a folded form [15],[18], after that the lifetime analysis[16] and register allocation unit[17] are applied to the folded architecture. Lastly, the proposed parallel architecture for long polar codes is potrayed.

3.1. Folding Transformation

The folding transformation is widely used to save hardware resources by time multiplexing various operations on functional unit. A data low graph corresponding to the fully parallel encoding process is shown in Fig 2. Where node represents the kernel matrix operation F, and w_{ij} denotes the jth edge at the ith stage. Data flow graph (DFG) of the all parallel polar encoder is comparable to that of the Fast Fourier transform [18], [19] except that the polar encoder works on the kernel matrix in place of the butterfly operation. Given the 16 bit DFG, the 4-parallel folded architecture that processes 4 bits at a time can be accomplished with two functional units in each stage. Thus the functional unit computes 2 bits at a time. In the folding transformation finding folding sets, that represents the order of operations to be executed in a functional unit, is the most crucial design factor [15]. To construct folding sets, all operations in the fully parallel encoding are initially classified as separate folding sets. Since the input is in natural order, it is reasonable to alternatively distribute the operations in the consecutive order. Hence each stage consists of two folding sets, each of that entirely odd or even operation to be performed by a separate unit.

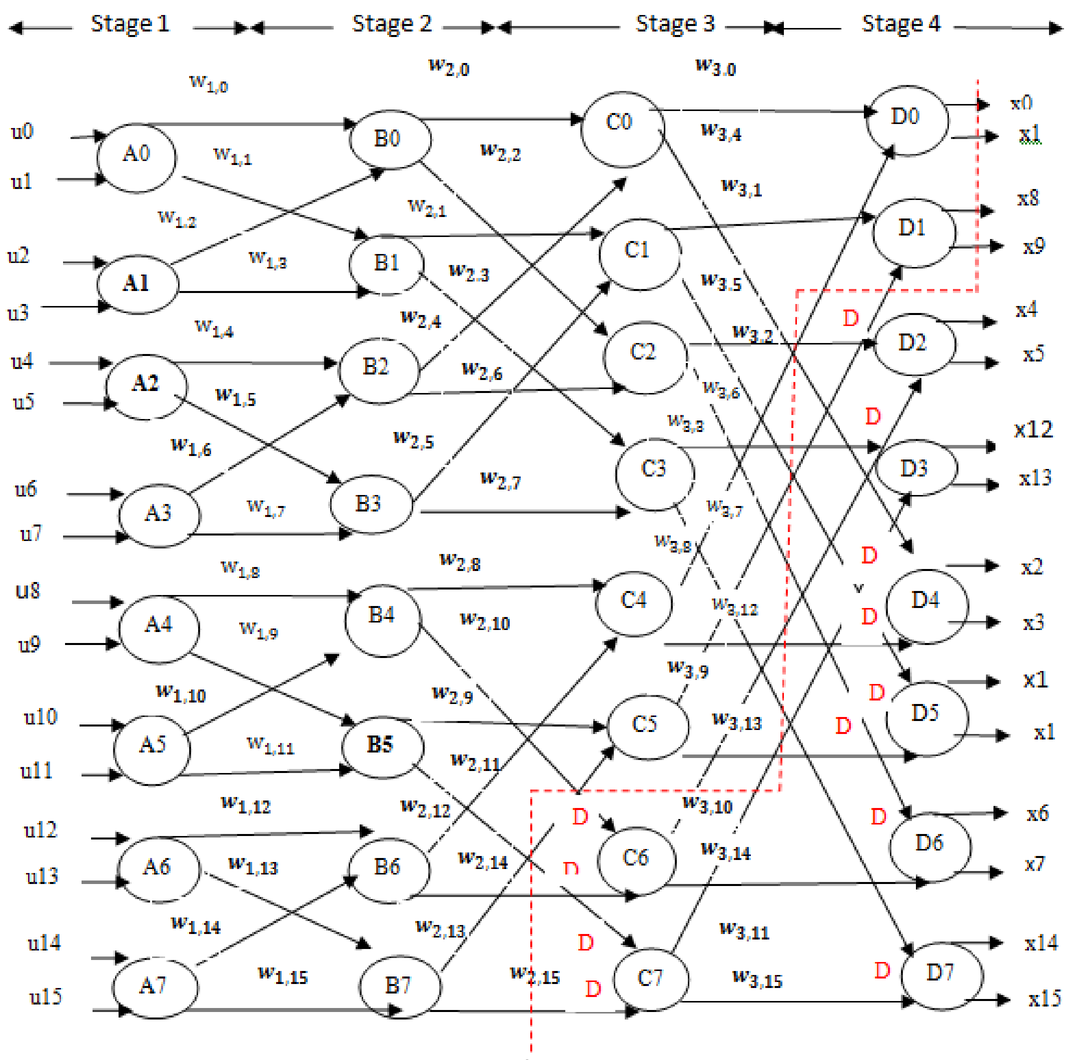


Figure 2. DFG of 16-bitpolar encoding

Taking 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 sets contains four elements, and the position of an element represents the operational order in the respective functional unit. At the beginning two functional units for stage one execute A0 and A1 simultaneously, A2 and A3 next cycle, so forth. The folding sets of stage two have the same order as those of stage 1,i.e.,{B0,B2,B4,B6}and {B1,B3,B5,B5},since the four parallel input sequence of stage two is same as stage one .To determine the folding sets of another stage the functional unit perform on a pair of inputs those indices differ by 2^{s-1}. In case of stage 3 two data inputs whose indices differ by 4 are processed together, which means that operational distance of the corresponding data is two i,e the functional unit computes two data inputs at a time. For instance,$w_{2,0}$ and $w_{2,4}$that come from B0 and B2 are inputs to C0.Both inputs should be valid for functional unit for processing the operations in stage 3 and to be are aligned to the late input data. Cyclic shifting the folding sets right by one can be done by inserting one unit delay time, it gives the full utilization of the functional units by overlapping next iterations .As a result, folding sets{C6,C0,C2,C4}and {C7,C1,C3,C5}, of stage 3 are determined, where C6 is overlapped with A0 and B0 in the next iteration. Likewise, the functional unit processes a pair of inputs whose indices differ by 8 in stage 4. The folding sets of stage 4 are {D2, D4, D6, D0} and {D3, D5, D7, D1}.These are obtained by cyclic shifting the previous stage 3 folding sets by two. Generally speaking, a stage whose index *s* is less than or equal to

log_2P, 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 *s* is larger than log_2P has the folding sets obtained by cyclic shifting the previous folding sets of stage s-1 right by $s - log_2 P$.

Consider the case with accurate delay when an edge w_{ij} from functional unit T to functional unit S has a delay of *d*, the delay requirements for w_{ij} in the *F*-folded architecture can be calculated as

$$D(w_{ij}) = Fd + t - s \qquad (1)$$

Where *t* and *s* denote the position in the folding set corresponding to *T* and *S*, respectively. Equation (1) is a simplified delay equation [15] derived by assuming that the kernel functional unit is not pipelined. For the 4-folded architecture, the delay requirements, i.e., $D(w_{ij})$ for 1≤ i≤ 3 and 0≤ j≤ 15 are review in Fig. 3. For example, $w_{2,0}$ from *B*0 to *C*0 requires one delay so *d* = 0, *t* = 1, and *s* = 0. Some edges indicated by circles have negative delays. For the folded architecture to be practical, the delay must be equal or larger than to zero for all the edges. Retiming or Pipelining techniques can be applied to the fully parallel DFG in order to ensure that its folded hardware has non negative delays. Negative delay of every edge should be adjusted by inserting at least one delay element to make the value of equation (1) non 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 delay elements are different, additional delays 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 so additional delays are required due to the mismatch of the number of delay elements. By inserting delay elements the DFG is pipelined, as shown in Fig. 2, where the dashed line indicates the pipeline cut set associated with 12 delay elements. For the pipelined DFG D'(w_{ij}) the delay requirements are recalculated based on equation (1) and shown at the below Fig. 3. As a result, 8 functional units and 48 delay elements are enough to implement the 4-folded 4-parallel 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

Figure 1. Original delay requirements D(wij) and recalculated delay requirements D'(wij)

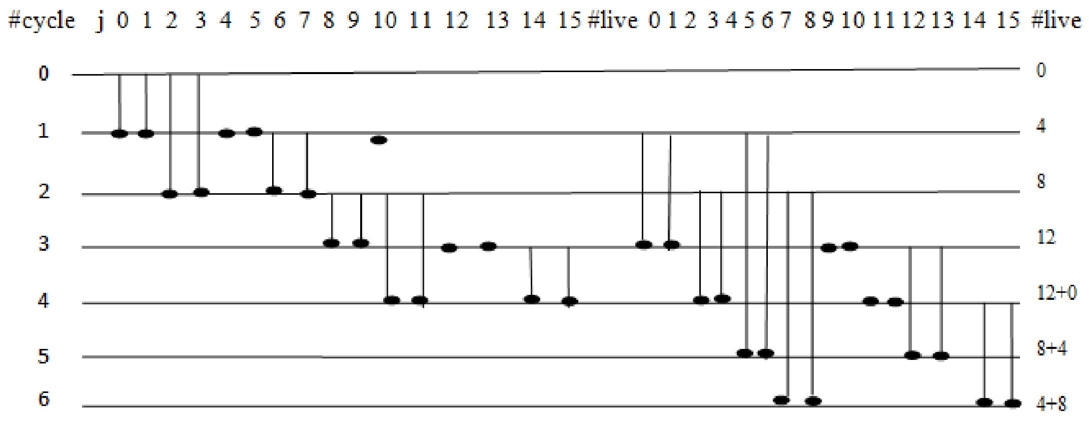
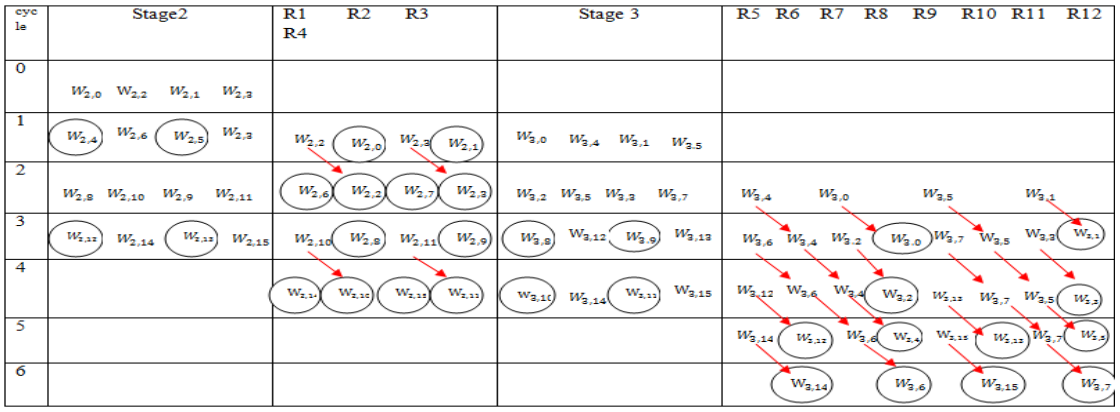


Figure 2. Linear lifetime chart for w2j and w3j .

3.2. 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 applied to find the minimum number of delay elements required to implementing the folded architecture. The lifetime of every variable is represented in the linear lifetime chart illustrated in Fig. 4. The edges in stage 1has no delay elements, only w_{2j} and w_{3j} are presented in Fig. 4. For instance, $w_{3,0}$ will live for two cycles as it is build at cycle 1 and consumed at cycle 3. The number of variables live in each cycle is shown in the chart. According to lifetime chart the number of live variables at the fourth or after clock cycles takes into account the next iteration and the current iterations are overlapped. Thus, the number of live variables is 12, by using this 12 delay elements the folded architecture can be implemented. After determination of the number of delay elements, 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], first row shows the all registers and every row report how the registers are allocated at the corresponding cycle. According to 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 the variable with circle indicates that it remains same cycle. For example, $w_{2,0}$ and $w_{2,4}$ are propagated to functional unit to execute operation C_0 that produces $w_{3,0}$ and $w_{3,4}$. At the same time, $w_{2,1}$ and $w_{2,5}$ are propagated to another functional unit to execute operation C_1 that gives $w_{3,1}$ and $w_{3,5}$. The moving of the other variables can be find by following the register allocation table (Fig5).



cycle	Stage2	R1	R2	R3	R4	Stage 3	R5	R6	R7	R8	R9	R10	R11	R12
0	$W_{2,0}$ $W_{2,2}$ $W_{2,1}$ $W_{2,3}$													
1	$W_{2,4}$ $W_{2,6}$ $W_{2,5}$ $W_{2,3}$	$W_{2,2}$	$W_{2,0}$	$W_{2,3}$	$W_{2,1}$	$W_{3,0}$ $W_{3,4}$ $W_{3,1}$ $W_{3,5}$								
2	$W_{2,8}$ $W_{2,10}$ $W_{2,9}$ $W_{2,11}$	$W_{2,6}$	$W_{2,2}$	$W_{2,7}$	$W_{2,3}$	$W_{3,2}$ $W_{3,5}$ $W_{3,3}$ $W_{3,7}$	$W_{3,4}$		$W_{3,0}$		$W_{3,5}$		$W_{3,1}$	
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								$W_{3,14}$		$W_{3,6}$		$W_{3,15}$		$W_{3,7}$

Figure5. Register allocation table for w2j and w3j.

Finally, the following 4-parallel pipelined structure that consists of 8 functional units and 12 delay elements projected to encrypt the 16-bit polar code is illustrated in Fig.6. A trial of 2 functional units takes in place of one stage and thus the delay elements area unit to store variables in line with the register allocation table. The hardware structures for stages 1 and 2 can be accomplished as no delay elements are present whereas for stages 3 and 4 several multiplexers are placed before the functional units to configure the inputs of the functional units. The present architecture constantly processes four samples per cycle in line with the folding sets and the register allocation table. Note that proposed encoder takes an attempt of inputs in a natural order and generates an attempt of outputs in bit reversed order as shown in Fig2. As a result the functional unit in the proposed architecture processes an attempt of 2 bits at a time and pair of consecutive bits is regarded as a single entity.

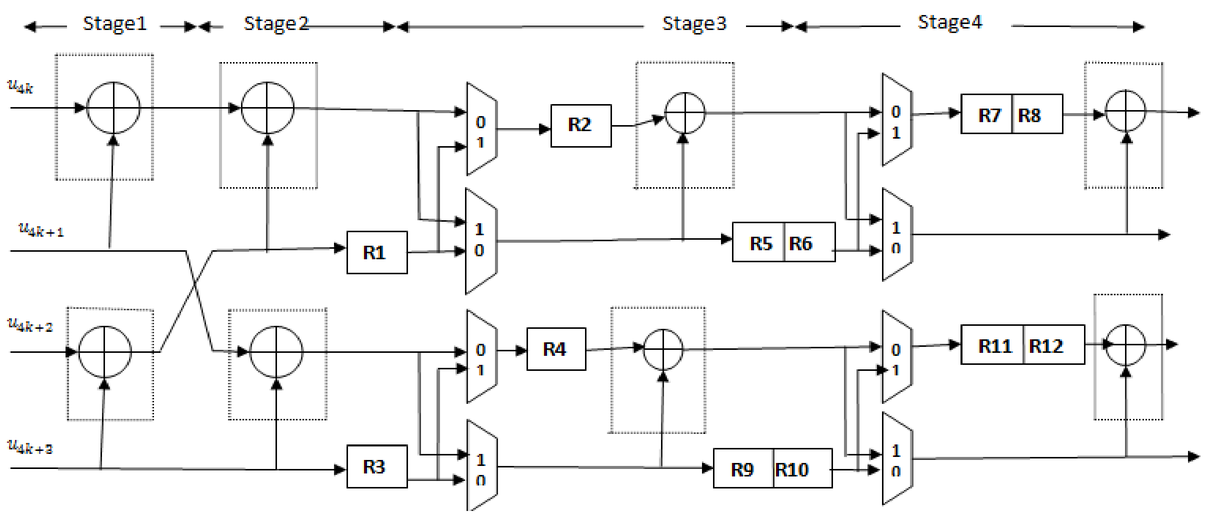


Figure 6. Proposed 4-parallel folded architecture for encoding the polar (16, K) codes.

IV. Analysis And Comparison

In the present architecture, the number of functional units required in the implementation depends on the code length N and 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]*$log_2 N$ functional units in total.

Furthermore, 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 $log_2 P$ require P delay blocks of length $2^{s-\log_2 P-1}$ whereas the other stages can be implemented with no delay elements. The total number of delay elements is

$$\sum_{S=log_2^{P+1}}^{log_2^N} P(2^{s-\log_2 P-1}) = P(1+2+\cdots$$
$$+(2^{\log_2 N-\log_2 P-1})$$
$$= P\,(2^{\log_2 N-\log_2 P-1})$$
$$= N - P \quad (2)$$

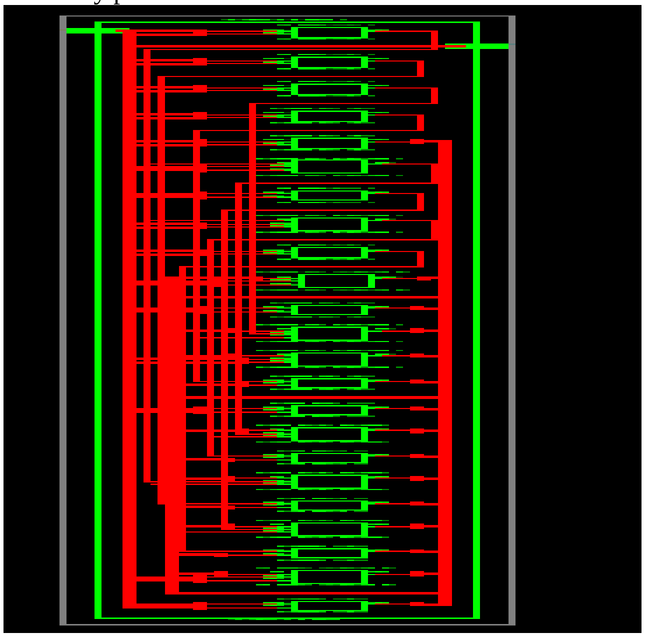
Given the hardware resources, the proposed partially parallel architecture can encode *P* bits per cycle.

Comparison between existing and proposed work

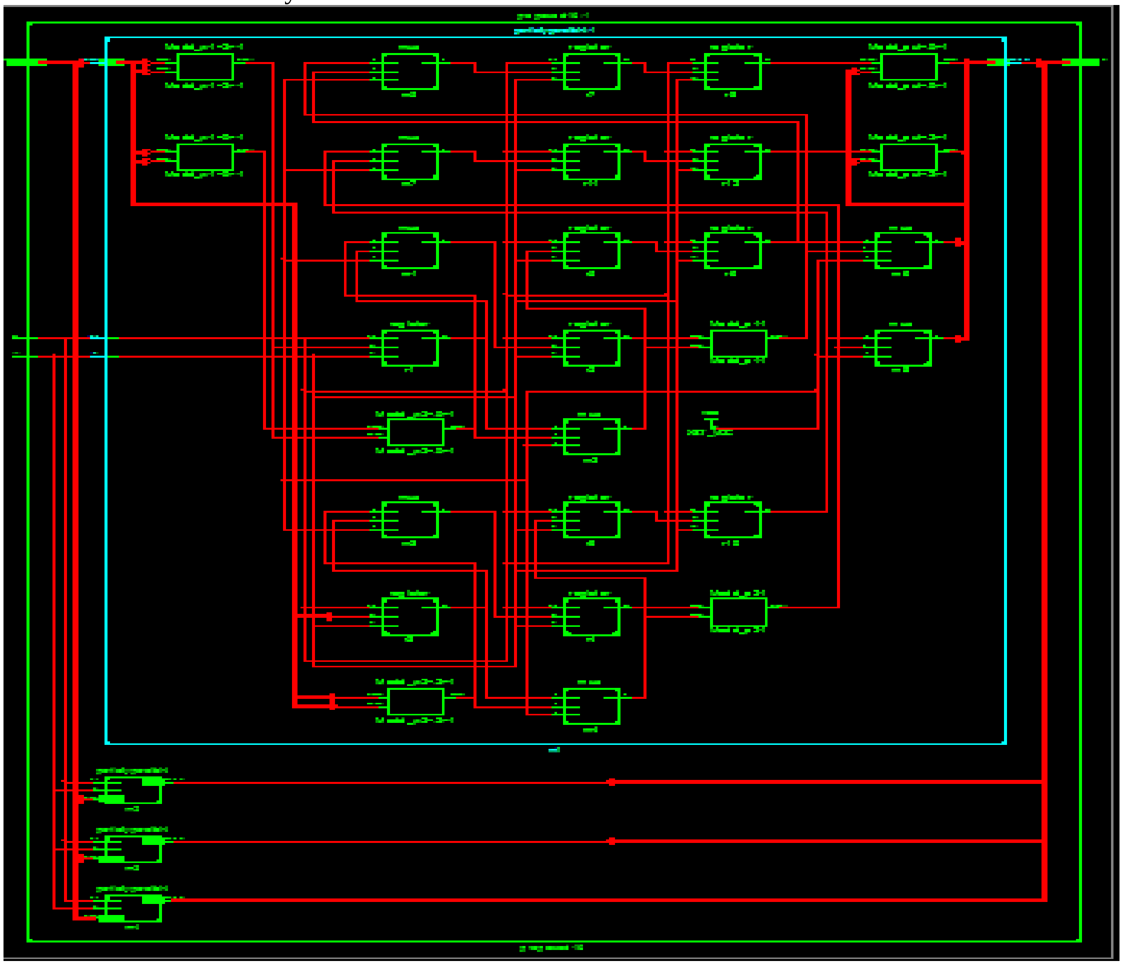
SNO	Parameter	Exisiting work	Proposed work
1	Delay	1.907ns	0.648ns
2	Memory	446776 kb	424312kb
3	Number of Slice LUTs	39	28
4	Number of fully used LUT-FF pairs	0	6
5	Number of bonded IOBs	80	34

V. Results

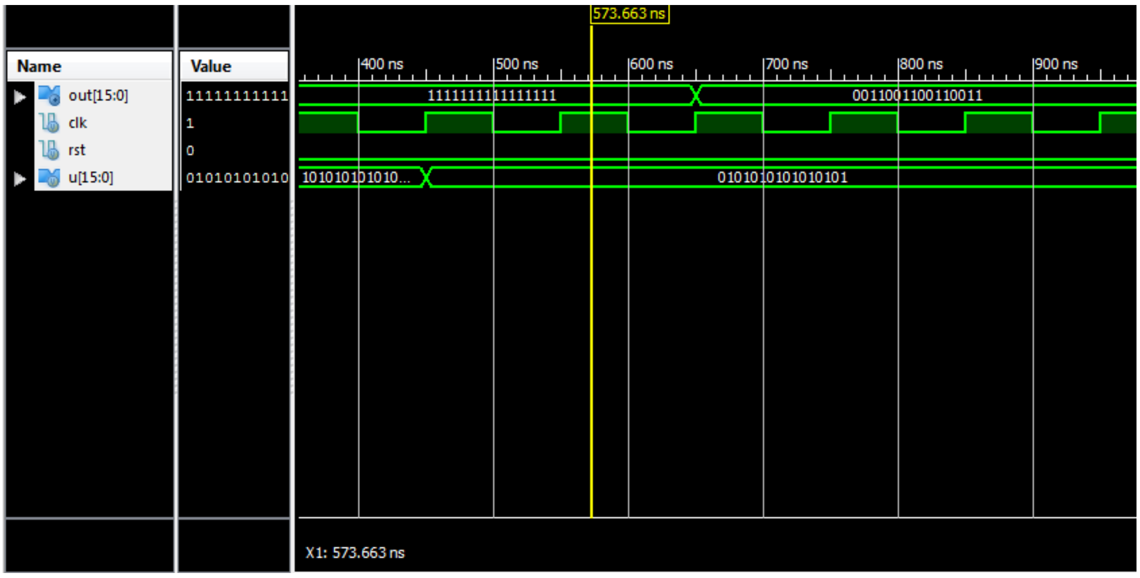
5.1. Techonology schematic of fully parallel encoder:



5.2 RTL schematic for Partially Parallel Encoder:

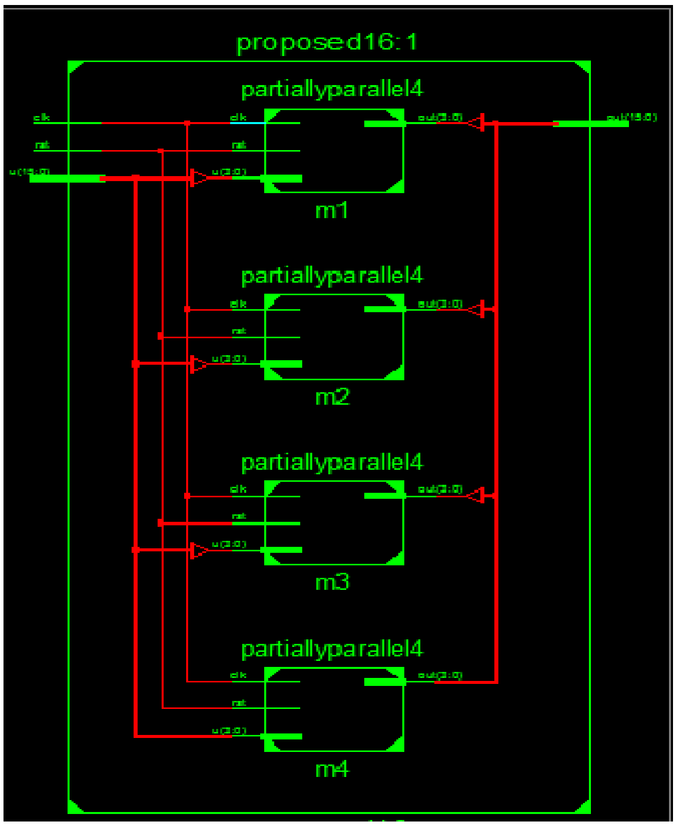


5.3. Simulation results:



Here, u[15:0] is input vector x[15:0]is output vector, clk signal, Input vector takes 6 cycles to produce the output vector based on the register allocation table. Reset signal clears the any pending errors or events.

5.4. RTL schematic:



VI. Conclusion

This brief has conferred a new partially parallel encoder style developed for long polar codes. Many improved techniques have been unit applied to derive the current style. Experimental results show that the current style can save the hardware compared with that of fully parallel style. Therefore, the current style 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," *IEEETrans. 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– , 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. SignalProcess.*, 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. 958–966, May 2014.
[13]. B. Yuan and K. K. Parhi, "Low-latency successive-cancellation list decoders for polar codes with multibit decision," *IEEE Trans. Very LargeScale Integr. (VLSI) Syst.*, DOI: 10.1109/TVLSI.2014.2359793, to be 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.

[20]. Hoyoung and In cheol park, "partially parallel encoder architecture for long polar codes" *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS,* VOL. 62, NO. 3, MARCH 2015.