Circuits and Systems, 2016, 7, 551-561
Published Online April 2016 in SciRes. http://www.scirp.org/journal/cs
http://dx.doi.org/10.4236/cs.2016.75047

Architectural Design of 32 Bit Polar Encoder

G. Indumathi[1], V. P. M. B. Aarthi Alias Ananthakirupa[1*], M. Ramesh[2]

[1]Department of Electronics and Communication Engineering, Mepco Schlenk Engineering College, Sivakasi, India
[2]Department of Electronics and Communication Engineering, Kamaraj College of Engineering and Technology, Virudhunagar, India
Email: *vpmb2aarthi@gmail.com

Received 18 February 2016; accepted 26 April 2016; published 29 April 2016

Abstract

The rapid development in the digital circuit design enhances the applications on very large scale integration era. Encoders are one among the digital circuits found in all communication systems. The polar encoding is mainly meant for its channel achieving property. It finds its application in communications, sensing and information theory. This coding proposed by Erdal Arikan is significant because of its zero error floors and simple architecture for hardware implementation. In this paper, a folded polar encoder is designed to start from the fully parallel architecture and proceeds with its data flow graph, delay requirement calculation, lifetime analysis and register allocation, which results in a very large scale integration architecture with minimum hardware utilization. The results are simulated for 4 and 8 parallel folded 32-bit polar encoder using Xilinx 14.6 ISIM and implemented in Virtex 5 field programmable gate array. A comparison is made on fully parallel and various folding techniques based on their resource utilization.

Keywords

Polar Encoder, Folding, Very Large Scale Integration (VLSI) Architecture, Field Programmable Gate Array (FPGA)

1. Introduction

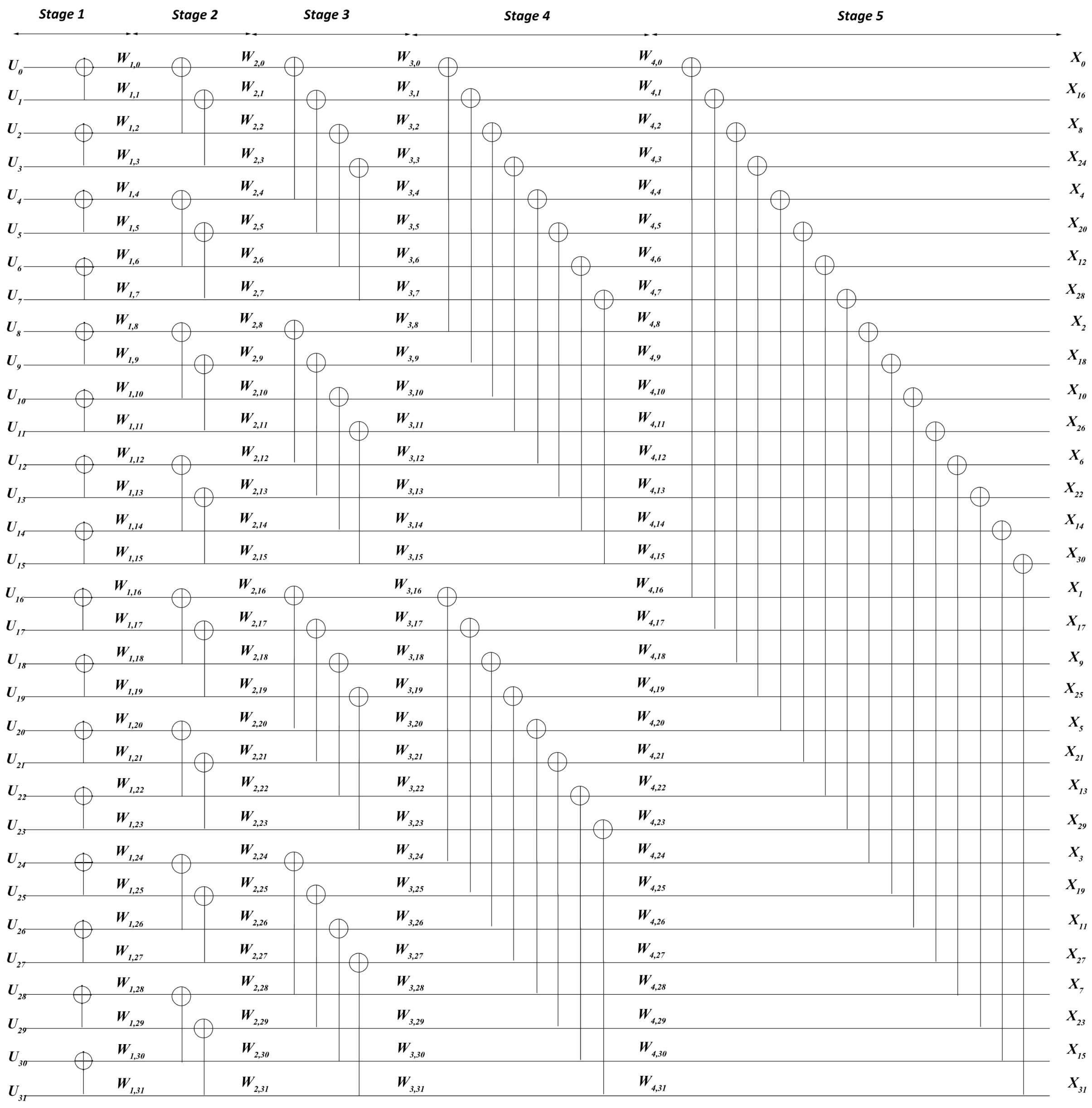
The polar code belongs to the class of linear block codes. The encoding process can be characterized by the generator matrix. The generator matrix G_N for code length N or 2 is obtained by applying the n^{th} kronecker power of the kernel matrix [1].

*Corresponding author.

$$F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{1}$$

Given the generator matrix, the code word x is computed by $x = u \cdot G_N$, where u denotes the information. The information vector u is arranged in natural order, whereas the code vector x is in a bit reversed order. The encoding complexity of straight forward fully parallel encoder architecture is in the order of ($N\log N$) for the polar code of length N and takes n stages. When $N = 2^n$, polar code with the length of 32 bit is implemented with 80 ex-or gates and processed in five stages as shown in **Figure 1**.

In VLSI architecture, reduction focuses on the minimization of the size of the components. Many techniques are involved in the minimization process. Some of the addressable techniques are *k*-map based Boolean expression method and block optimization method. In general, pipelining can be used in the context of architecture design [2]-[4]. The pipelining transformation leads to a reduction in the critical path, which can be exploited to



Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
U_0 … U_{31}
$W_{1,0}$ … $W_{1,31}$
$W_{2,0}$ … $W_{2,31}$
$W_{3,0}$ … $W_{3,31}$
$W_{4,0}$ … $W_{4,31}$
X_0, X_{16}, X_8, X_{24}, X_4, X_{20}, X_{12}, X_{28}, X_2, X_{18}, X_{10}, X_{26}, X_6, X_{22}, X_{14}, X_{30}, X_1, X_{17}, X_9, X_{25}, X_5, X_{21}, X_{13}, X_{29}, X_3, X_{19}, X_{11}, X_{27}, X_7, X_{23}, X_{15}, X_{31}

Figure 1. Fully parallel architecture of 32 bit polar encoder.

either increase the clock speed or sample speed or to reduce the power consumption at the same speed. This in turn reduces the effective critical path by introducing pipelined latches along the data path. The pipelined technique can be broadly classified as feed forward and feedback path. The feed forward pipelined encoder structure consists of 2D commutator followed by ex-or and pass gates for achieving high throughput [5].

The feedback pipelined polar encoder favors for high hardware efficiency rather than high throughput. The number of ex-or gate is equal to the number of processing stages, whereas the number of delay elements gets reduced. In parallel processing, multiple outputs are computed in parallel in a clock period. Therefore, the effective sampling speed is increased by the level of parallelism [6]. This increases the sampling rate by replicating the hardware so that the several inputs can be processed in parallel and several outputs can be produced at the same time.

The polar code architecture is discussed [7] using channel combining phase and channel splitting phase. It incorporates punctured encoding to shorten the length of polar codes. It is observed that reduction of the memory constraints can be achieved during practical applications. The application of polar codes and polarization phenomenon for various problems like wire tap channels [8], multiple access channels [9] [10], data compression [11], and broadcast channels [12] were successful. In addition to the capacity achieving capability, polar codes have interesting properties like good error floor performance [13]. This suggests that the combination of polar coding with other coding schemes could eliminate the shortcomings of both, resulting in a powerful coding paradigm. Furthermore, there are many applications for concatenated codes like deep-space communications, optical transport systems, and magnetic recording channels [14].

In this paper, the parallelism and pipelining have been combined to achieve an effective encoder structure with minimum registers. This implementation proceeds from the conventional fully parallel 32 bit architecture and transforming as a data flow graph (DFG), delay requirement table, linear lifetime chart and register allocation [15]. The flow of this paper proceeds as follows. Section 2 describes the design of four folded 32 bit polar encoder architecture with each step in detail. Section 3 discusses the architectural design of eight folded polar encoder. Finally the comparative results for fully parallel, four and eight folded architectures with respect to resource utilization are discussed.

2. Four Parallel Folded 32 Bit Polar Encoder

The polar encoder relies on the principle of channel polarization. It is a recursive method used to define the polar codes. A class of codes that can provably achieve the capacity of several classes of channels. It comes under linear codes. The phenomenon of channel polarization includes channel combining and channel splitting. The channel W_N can be measured up with two parameters namely mutual information which defines the information capacity and Bhattacharya parameter measures the reliability of the channel.

In synthesizing DSP architectures, it is important to minimize the silicon area of the integrated circuits, which is achieved by reducing the number of functional units, multiplexers, interconnection wires. This in turn may lead to an architecture that uses a large number of registers. To avoid this, various techniques can be used to minimize the number of registers. Folding transformation reduces the hardware utilization by time multiplexing several operations of the functional unit [16]. The DFG of the 32 bit fully parallel polar encoder can be given as shown in **Figure 2**.

3. Folding Transformation

The DFG of the 32 bit polar code is similar to Fast Fourier Transform (FFT), and it uses the kernel matrix instead of butterfly operation. The 4-parallel folded architecture can be realized by placing 2 functional units in each stage, since each of the functional units compute two bits at a time. Let us consider the four parallel input sequences in natural order. The initial folding sets can be given as: For stage 1: {P_0, P_2, P_4, P_6, P_8, P_{10}, P_{12}, P_{14}}, {P_1, P_3, P_5, P_7, P_9, P_{11}, P_{13}, P_{15}}. In this, the two functional units of stage 1 namely P_0 and P_1 execute simultaneously at the beginning and P_2 and P_3 at the next cycle. The stage whose index s is less than or equal to $\log_2 P$, where P is the level of parallelism and has the same folding set as that of the previous one. The stage 2 has the same order as those of stage 1, since it performs the operation within the same four inputs. At later stages, the folding sets are computed by, the property that the functional unit that process a pair of inputs whose indices differ by $2^{(s-1)}$ is exploited [15]. Thus the folding set of stage 2 can be given as {Q_0, Q_2, Q_4, Q_6, Q_8, Q_{10}, Q_{12}, Q_{14}}, {Q_1, Q_3, Q_5, Q_7, Q_9, Q_{11}, Q_{13}, Q_{15}}. In the stage 3, the indices of the two data differ by a factor of four,

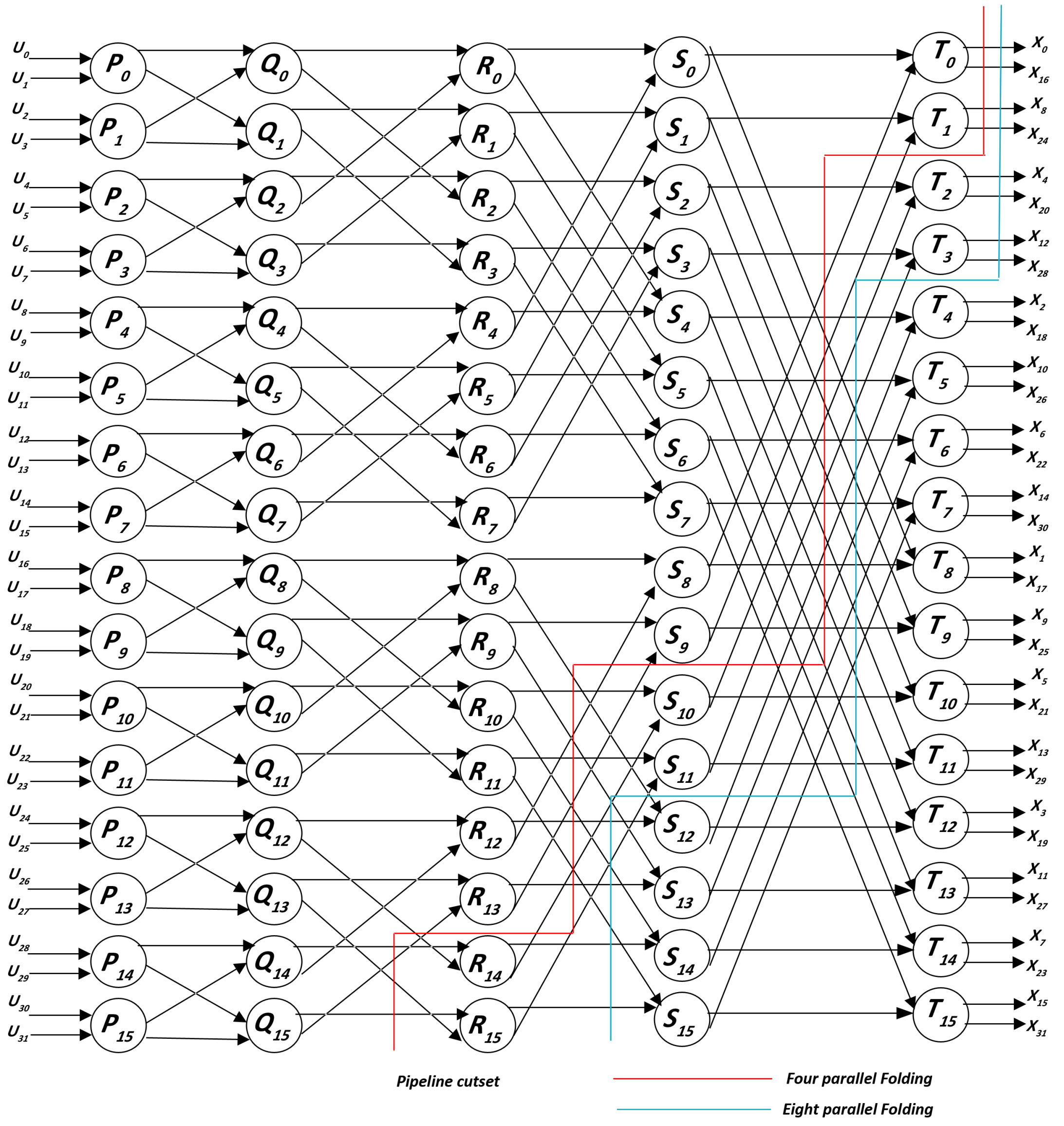


Figure 2. Data flow graph of 32 bit polar encoder.

thus cyclic shifting of four bits right by one can be done by inserting a delay of one time unit. Thus the folding sets of stage 3 are given by {R_{14}, R_0, R_2, R_4, R_6, R_8, R_{10}, R_{12}}, {R_{15}, R_1, R_3, R_5, R_7, R_9, R_{11}, R_{13}}. The folding sets of stage 4 and stage 5 can be obtained by cyclic shifting of stage 3 by two in order to enable full utilization of functional units with adjacent iterations. The folding sets of stage 4 and stage 5 can be given as {S_{10}, S_{12}, S_{14}, S_0, S_2, S_4, S_6, S_8}, {S_{11}, S_{13}, S_{15}, S_1, S_3, S_5, S_7, S_9} and {T_2, T_4, T_6, T_8, T_{10}, T_{12}, T_{14}, T_0}, {T_3, T_5, T_7, T_9, T_{11}, T_{13}, T_{15}, T_1} respectively.

4. Delay Requirement Calculation

The number of delay element required in the folded architecture [3] can be computed

$$D\left(W_{ij}\right) = Fd + t - s \tag{2}$$

where W_{ij} is an edge from the functional unit S to the functional unit T, having the delay d where t and s denote the position in the folding set corresponding to T and S respectively. The delay requirement of four folded 32 bit polar encoder can be given as shown in **Table 1**.

Table 1. Original delay requirement $D(W_{ij})$ for 4-parallel folded Polar encoder.

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$D(W_{1j})$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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	1	1	2	2	0	0	1	1	1	1	−6	−6	0	0	−7	−7
$D(W_{3j})$	2	2	2	2	4	4	4	4	0	0	0	0	2	2	2	2	2	2	−6	−6	−4	−4	−4	−4	0	0	0	0	−6	−6	2	2
$D(\mathrm{W}_{4j})$	4	4	−4	−4	−4	−4	−4	−4	−0	−0	−0	−0	−0	−0	−0	−0	0	0	0	0	0	0	0	0	4	4	4	4	4	4	4	4

In **Table 1**, some edges have negative delays. To have a feasible folded architecture the delay should be greater than or equal to zero for all the edges. Thus we prefer re-timing or pipelining methods for the fully parallel structure to ensure non negative delays. In this, the negative delay should be compensated by inserting at least one delay element to make the value of Equation (2) non negative. The polar encoder utilizes the ex-or operation, and the two inputs should pass through the same number of delay elements. If they are different, additional delays are included to match them. The DFG is pipelined by inserting delay elements, and the red line indicates the pipeline cut-set associated with 4-folded architecture. Thus the recalculated delay is shown in **Table 2**.

The 32 bit polar encoder with four parallel folded structure can be implemented with 10 functional units and 28 delay elements.

5. Lifetime Analysis

The number of delay elements can be reduced by implementing lifetime analysis for the folded architecture [16]. The linear variable chart represents the lifetime of every variable as in **Figure 3**. In this, all the edges starting from stage 1 have zero delay. Therefore W_{2j}, W_{3j} and W_{4j} are presented. Here $\mathrm{W}_{3,0}$ is alive for two cycles. Hence, they are produced at stage 1 and consumed in stage 3. The W_{2j} starts at time 0 and proceeds so on. It is taken in the same order as that in DFG. The W_{3j} starts at the next cycle at time 1 and proceeds as $j = 0, 1, 4, 5$, etc. The next stage W_{4j} starts at time 2 and proceeds as $j = 0, 1, 8, 9$, etc. in the forward manner. The number of variables alive in each cycle is given at the right side of the chart. Thus the maximum number of live variables is 28, which implies that the four folded 32 bit polar encoder can be implemented with 28 delay elements.

6. Register Allocation

In computing the minimum number of registers required, each variable is allocated to a register. The register allocation table is utilized [17] to verify the allocation in all the 28 registers, and every row describes how registers are allocated at each cycle. The indication in bold font implies that the variable gets consumed at the particular stage. The register allocation for 32 bit four parallel folded polar encoder is shown in **Table 3**.

7. Proposed Architecture

The four folded parallel pipelined structure for 32 bit polar encoder is shown in **Figure 4**. It consists of 10 functional units and 28 delay elements. Each stage has two functional units. Stages 1 and 2 include no delay elements. Stages 3, 4 and 5 have several multiplexers placed in front of each functional unit to configure the inputs of the functional units. The proposed architecture continuously processes four samples/cycle, according to folding sets and register allocation table. In this, the inputs are in the natural order and the outputs are in the bit reversed order.

8. Eight Parallel Folded 32 Bit Polar Encoder

The design of eight parallelism considers eight inputs at a time. Hence the stages are split up into four folding sets. The same procedure is applied for eight folded parallelism with the stages depicted below.

Stage 1: $\{P_0, P_4, P_8, P_{12}\}$ $\{P_1, P_5, P_9, P_{13}\}$ $\{P_2, P_6, P_{10}, P_{14}\}$ $\{P_3, P_7, P_{11}, P_{15}\}$

Stage 2: $\{Q_0, Q_4, Q_8, Q_{12}\}$ $\{Q_1, Q_5, Q_9, Q_{13}\}$ $\{Q_2, Q_6, Q_{10}, Q_{14}\}$ $\{Q_3, Q_7, Q_{11}, Q_{15}\}$

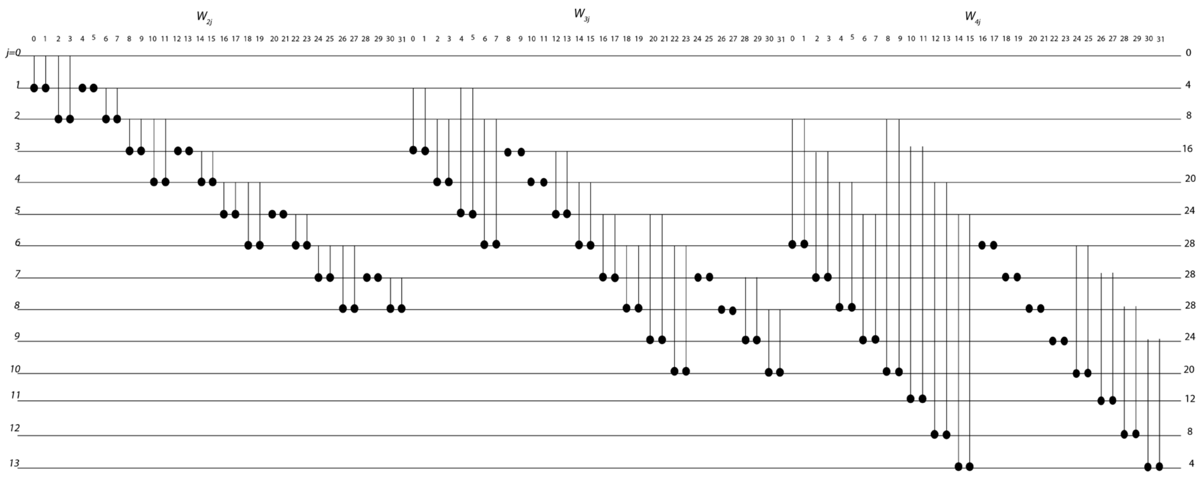


Figure 3. Linear lifetime chart for W_{2j}, W_{3j} and W_{4j}.

Table 2. Recalculated delay requirement $D'(W_{ij})$ for 4-parallel folded polar encoder.

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$D'(W_{1j})$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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	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	2	2	2	2	4	4	4	4	0	0	0	0	2	2	2	2
$D'(W_{4j})$	4	4	4	4	4	4	4	4	8	8	8	8	8	8	8	8	0	0	0	0	0	0	0	0	4	4	4	4	4	4	4	4

Table 3. Register allocation table for W_{2j}, W_{3j} and W_{4j}.

Cycle	Stage 2				R1	R2	R3	R4	Stage 3				R5	R6	R7	R8	R9	R10	R11	R12	Stage 4				R13	R14	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28
0	$W_{2,0}$	$W_{2,2}$	$W_{2,1}$	$W_{2,3}$																																				
1	$\boldsymbol{W_{2,4}}$	$W_{2,6}$	$\boldsymbol{W_{2,5}}$	$W_{2,7}$	$W_{2,2}$	$\boldsymbol{W_{2,0}}$	$W_{2,3}$	$\boldsymbol{W_{2,1}}$	$W_{3,0}$	$W_{3,4}$	$W_{3,1}$	$W_{3,5}$																												
2	$W_{2,8}$	$W_{2,10}$	$W_{2,9}$	$W_{2,11}$	$\boldsymbol{W_{2,6}}$	$\boldsymbol{W_{2,2}}$	$\boldsymbol{W_{2,7}}$	$\boldsymbol{W_{2,3}}$	$W_{3,2}$	$W_{3,6}$	$W_{3,3}$	$W_{3,7}$	$W_{3,4}$		$W_{3,0}$		$W_{3,5}$		$W_{3,1}$		$W_{4,0}$	$W_{4,8}$	$W_{4,1}$	$W_{4,9}$																
3	$\boldsymbol{W_{2,12}}$	$W_{2,14}$	$\boldsymbol{W_{2,13}}$	$W_{2,15}$	$W_{2,10}$	$\boldsymbol{W_{2,8}}$	$W_{2,11}$	$\boldsymbol{W_{2,9}}$	$\boldsymbol{W_{3,8}}$	$W_{3,12}$	$\boldsymbol{W_{3,9}}$	$W_{3,13}$	$W_{3,6}$	$W_{3,4}$	$W_{3,2}$	$\boldsymbol{W_{3,0}}$	$W_{3,7}$	$W_{3,5}$	$W_{3,3}$	$\boldsymbol{W_{3,1}}$	$W_{4,2}$	$W_{4,10}$	$W_{4,3}$	$W_{4,11}$	$W_{4,8}$				$W_{4,0}$				$W_{4,9}$				$W_{4,1}$			
4	$W_{2,16}$	$W_{2,18}$	$W_{2,17}$	$W_{2,19}$	$\boldsymbol{W_{2,14}}$	$\boldsymbol{W_{2,10}}$	$\boldsymbol{W_{2,15}}$	$\boldsymbol{W_{2,11}}$	$\boldsymbol{W_{3,10}}$	$W_{3,14}$	$\boldsymbol{W_{3,11}}$	$W_{3,15}$	$W_{3,12}$	$W_{3,6}$	$W_{3,4}$	$\boldsymbol{W_{3,2}}$	$W_{3,13}$	$W_{3,7}$	$W_{3,5}$	$\boldsymbol{W_{3,3}}$	$W_{4,4}$	$W_{4,12}$	$W_{4,5}$	$W_{4,13}$	$W_{4,10}$	$W_{4,8}$			$W_{4,2}$	$W_{4,0}$			$W_{4,11}$	$W_{4,9}$			$W_{4,3}$	$W_{4,1}$		
5	$\boldsymbol{W_{2,20}}$	$W_{2,22}$	$\boldsymbol{W_{2,21}}$	$W_{2,23}$	$W_{2,18}$	$\boldsymbol{W_{2,16}}$	$W_{2,19}$	$\boldsymbol{W_{2,17}}$	$W_{3,16}$	$W_{3,20}$	$W_{3,17}$	$W_{3,21}$	$W_{3,14}$	$\boldsymbol{W_{3,12}}$	$W_{3,6}$	$\boldsymbol{W_{3,4}}$	$W_{3,15}$	$\boldsymbol{W_{3,13}}$	$W_{3,7}$	$\boldsymbol{W_{3,5}}$	$W_{4,6}$	$W_{4,14}$	$W_{4,7}$	$W_{4,15}$	$W_{4,12}$	$W_{4,10}$	$W_{4,8}$		$W_{4,4}$	$W_{4,2}$	$W_{4,0}$		$W_{4,13}$	$W_{4,11}$	$W_{4,9}$		$W_{4,5}$	$W_{4,3}$	$W_{4,1}$	
6	$W_{2,24}$	$W_{2,26}$	$W_{2,25}$	$W_{2,27}$	$\boldsymbol{W_{2,22}}$	$\boldsymbol{W_{2,18}}$	$\boldsymbol{W_{2,23}}$	$\boldsymbol{W_{2,19}}$	$W_{3,18}$	$W_{3,22}$	$W_{3,19}$	$W_{3,23}$	$W_{3,20}$	$\boldsymbol{W_{3,14}}$	$W_{3,16}$	$\boldsymbol{W_{3,6}}$	$W_{3,21}$	$\boldsymbol{W_{3,15}}$	$W_{3,17}$	$\boldsymbol{W_{3,7}}$	$\boldsymbol{W_{4,16}}$	$W_{4,24}$	$\boldsymbol{W_{4,17}}$	$W_{4,25}$	$W_{4,14}$	$W_{4,12}$	$W_{4,10}$	$W_{4,8}$	$W_{4,6}$	$W_{4,4}$	$W_{4,2}$	$\boldsymbol{W_{4,0}}$	$W_{4,15}$	$W_{4,13}$	$W_{4,11}$	$W_{4,9}$	$W_{4,7}$	$W_{4,5}$	$W_{4,3}$	$\boldsymbol{W_{4,1}}$
7	$\boldsymbol{W_{2,28}}$	$W_{2,30}$	$\boldsymbol{W_{2,29}}$	$W_{2,31}$	$W_{2,26}$	$\boldsymbol{W_{2,24}}$	$W_{2,27}$	$\boldsymbol{W_{2,25}}$	$\boldsymbol{W_{3,24}}$	$W_{3,28}$	$\boldsymbol{W_{3,25}}$	$W_{3,29}$	$W_{3,22}$	$W_{3,20}$	$W_{3,18}$	$\boldsymbol{W_{3,16}}$	$W_{3,23}$	$W_{3,21}$	$W_{3,19}$	$\boldsymbol{W_{3,17}}$	$\boldsymbol{W_{4,18}}$	$W_{4,26}$	$\boldsymbol{W_{4,19}}$	$W_{4,27}$	$W_{4,24}$	$W_{4,14}$	$W_{4,12}$	$W_{4,10}$	$W_{4,8}$	$W_{4,6}$	$W_{4,4}$	$\boldsymbol{W_{4,2}}$	$W_{4,25}$	$W_{4,15}$	$W_{4,13}$	$W_{4,11}$	$W_{4,9}$	$W_{4,7}$	$W_{4,5}$	$\boldsymbol{W_{4,3}}$
8					$\boldsymbol{W_{2,30}}$	$\boldsymbol{W_{2,26}}$	$\boldsymbol{W_{2,31}}$	$\boldsymbol{W_{2,27}}$	$\boldsymbol{W_{3,26}}$	$W_{3,30}$	$\boldsymbol{W_{3,27}}$	$W_{3,31}$	$W_{3,28}$	$W_{3,22}$	$W_{3,20}$	$\boldsymbol{W_{3,18}}$	$W_{3,29}$	$W_{3,23}$	$W_{3,21}$	$\boldsymbol{W_{3,19}}$	$\boldsymbol{W_{4,20}}$	$W_{4,28}$	$\boldsymbol{W_{4,21}}$	$W_{4,29}$	$W_{4,26}$	$W_{4,24}$	$W_{4,14}$	$W_{4,12}$	$W_{4,10}$	$W_{4,8}$	$W_{4,6}$	$\boldsymbol{W_{4,4}}$	$W_{4,27}$	$W_{4,25}$	$W_{4,15}$	$W_{4,13}$	$W_{4,11}$	$W_{4,9}$	$W_{4,7}$	$\boldsymbol{W_{4,5}}$
9													$W_{3,30}$	$\boldsymbol{W_{3,28}}$	$W_{3,22}$	$\boldsymbol{W_{3,20}}$	$W_{3,31}$	$\boldsymbol{W_{3,29}}$	$W_{3,23}$	$\boldsymbol{W_{3,21}}$	$\boldsymbol{W_{4,22}}$	$W_{4,30}$	$\boldsymbol{W_{4,23}}$	$W_{4,31}$	$W_{4,28}$	$W_{4,26}$	$W_{4,24}$	$W_{4,14}$	$W_{4,12}$	$W_{4,10}$	$W_{4,8}$	$\boldsymbol{W_{4,6}}$	$W_{4,29}$	$W_{4,27}$	$W_{4,25}$	$W_{4,15}$	$W_{4,13}$	$W_{4,11}$	$W_{4,9}$	$\boldsymbol{W_{4,7}}$
10														$\boldsymbol{W_{3,30}}$		$\boldsymbol{W_{3,22}}$		$\boldsymbol{W_{3,31}}$		$\boldsymbol{W_{3,23}}$					$W_{4,30}$	$W_{4,28}$	$W_{4,26}$	$\boldsymbol{W_{4,24}}$	$W_{4,14}$	$W_{4,12}$	$W_{4,10}$	$\boldsymbol{W_{4,8}}$	$W_{4,31}$	$W_{4,29}$	$W_{4,27}$	$\boldsymbol{W_{4,25}}$	$W_{4,15}$	$W_{4,13}$	$W_{4,11}$	$\boldsymbol{W_{4,9}}$
11																										$W_{4,30}$	$W_{4,28}$	$\boldsymbol{W_{4,26}}$		$W_{4,14}$	$W_{4,12}$	$\boldsymbol{W_{4,10}}$		$W_{4,31}$	$W_{4,29}$	$\boldsymbol{W_{4,27}}$		$W_{4,15}$	$W_{4,13}$	$\boldsymbol{W_{4,11}}$
12																											$W_{4,30}$	$\boldsymbol{W_{4,28}}$			$W_{4,14}$	$\boldsymbol{W_{4,12}}$			$W_{4,31}$	$\boldsymbol{W_{4,29}}$			$W_{4,15}$	$\boldsymbol{W_{4,13}}$
13																												$\boldsymbol{W_{4,30}}$				$\boldsymbol{W_{4,14}}$				$\boldsymbol{W_{4,31}}$				$\boldsymbol{W_{4,15}}$

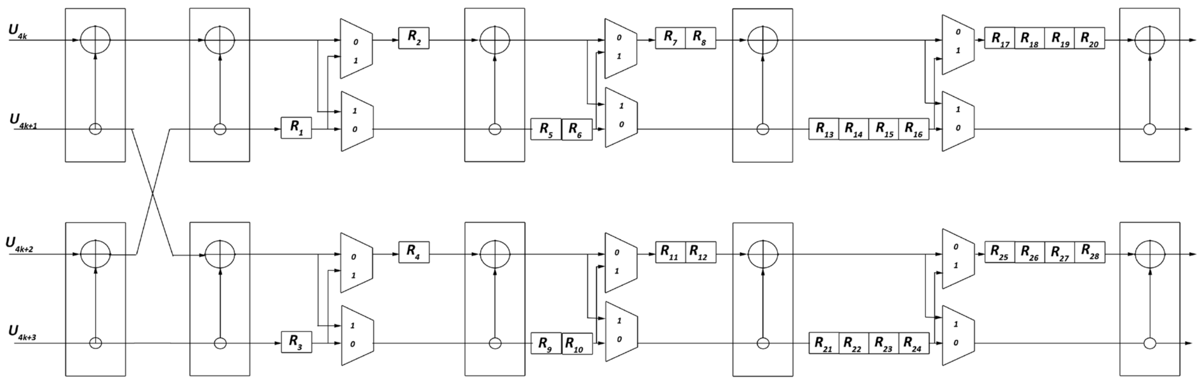

U_{4k}
U_{4k+1}
U_{4k+2}
U_{4k+3}
R_1
R_2
R_3
R_4
R_5 R_6
R_7 R_8
R_9 R_{10}
R_{11} R_{12}
R_{13} R_{14} R_{15} R_{16}
R_{17} R_{18} R_{19} R_{20}
R_{21} R_{22} R_{23} R_{24}
R_{25} R_{26} R_{27} R_{28}
0
1

Figure 4. Proposed 4-parallel folded architecture for encoding the (32, k) polar codes.

Stage 3: {R_0, R_4, R_8, R_{12}} {R_1, R_5, R_9, R_{13}} {R_2, R_6, R_{10}, R_{14}} {R_3, R_7, R_{11}, R_{15}}
Stage 4: {S_{12}, S_0, S_4, S_8} {S_{13}, S_1, S_5, S_9} {S_{14}, S_2, S_6, S_{10}} {S_{15}, S_3, S_7, S_{11}}
Stage 5: {T_4, T_8, T_{12}, T_0} {T_5, T_9, T_{13}, T_7} {T_6, T_{10}, T_{14}, T_2} {T_7, T_{11}, T_{15}, T_3}

The corresponding cut-set is shown in **Figure 2**, with blue connected lines. The delay requirement table D (W_{ij}) is filled up using Equation (2). This table contains negative delays; it can be set right by using the recalculated delay requirement for eight parallelisms as depicted in **Table 4**.

The linear lifetime chart is drawn for W_{3j} and W_{4j}, since there exists no cross over with other stage inputs on the W_{2j} stage. This chart minimizes the registers to a count of 24 as shown in **Figure 5**.

This register count has been used to perform register allocation as illustrated in **Table 5**.

In the folded architecture the stages 1 and 2 include zero delay and hence no registers are needed. The stage 3 requires eight registers as shown in the linear lifetime chart. The stage 4 requires sixteen registers to obtain the encoded output. **Figure 6** depicts the pipeline implementation of the proposed architecture.

9. Results and Discussion

The above designs of 32 bit polar encoder for fully parallel, eight and four folded architectures are simulated using Xilinx 14.6 ISE and implemented in Virtex 5 FPGA and the corresponding outputs are obtained.

The simulation of 32 bit fully parallel architecture for a polar encoder with the input of 32'h FFFFFFFF using Xilinx 14.6 ISIM results in an output of 32'h 80008000 as depicted in **Figure 7**.

The four parallel folded architecture is simulated with the same input stream as given for fully parallel and verifies the same results as shown above in **Figure 8**. The synthesis results for 32 bit polar encoder using four folded structure can be done by eight successive executions on the same architecture utilizes 224 registers for execution.

The 8-parallel folded architecture is simulated with the same input stream given in fully parallel results in the same output verifying the functionality of polar code as in **Figure 9**. The synthesis results for 32 bit polar encoder using eight folded structure can be done by four successive executions on the same architecture utilizes 96 registers.

The partially parallel implementation in [15] concludes that P parallelism of (N, K) polar encoder will arrive at ex-or gates formulated as $\left\lceil \frac{P}{2} \right\rceil \log_2 N$.

In addition, there exist N-P delay elements with the throughput of P bits/cycle. Thus the 32 bit polar encoder design for four and eight folding matches exactly with the same criteria. The comparison of resource utilization for the 32 bit polar encoder for fully parallel, four and eight folded architectures is depicted in **Table 6**.

10. Conclusion and Future Scope

This paper is focused to minimize the hardware resources for the 32 bit polar encoder. Many optimization techniques are implemented in steps to arrive at the proposed architecture for various folding levels. The simulation results show that the folded structure abides the polar encoder functionality. The implementation in Virtex 5

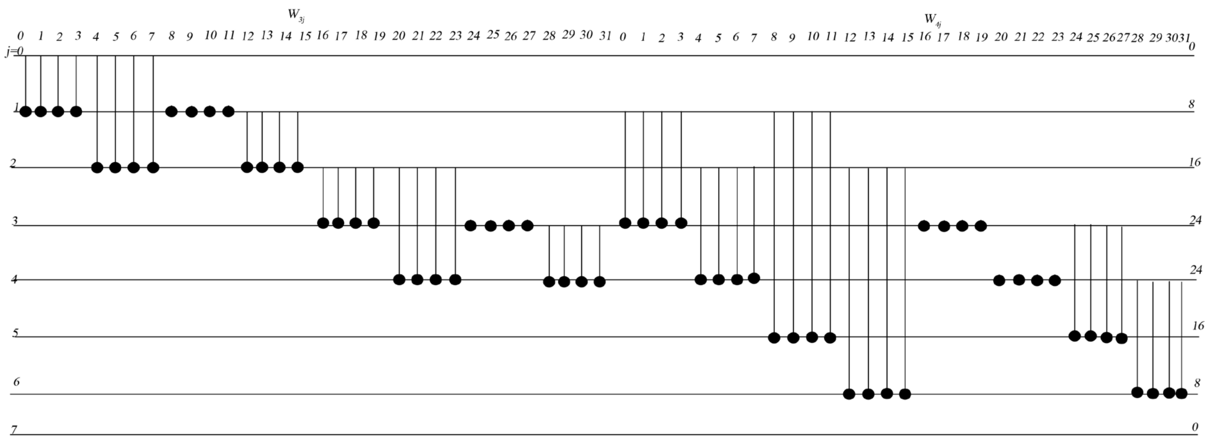


Figure 5. Linear lifetime chart for W_{3j} and W_{4j}.

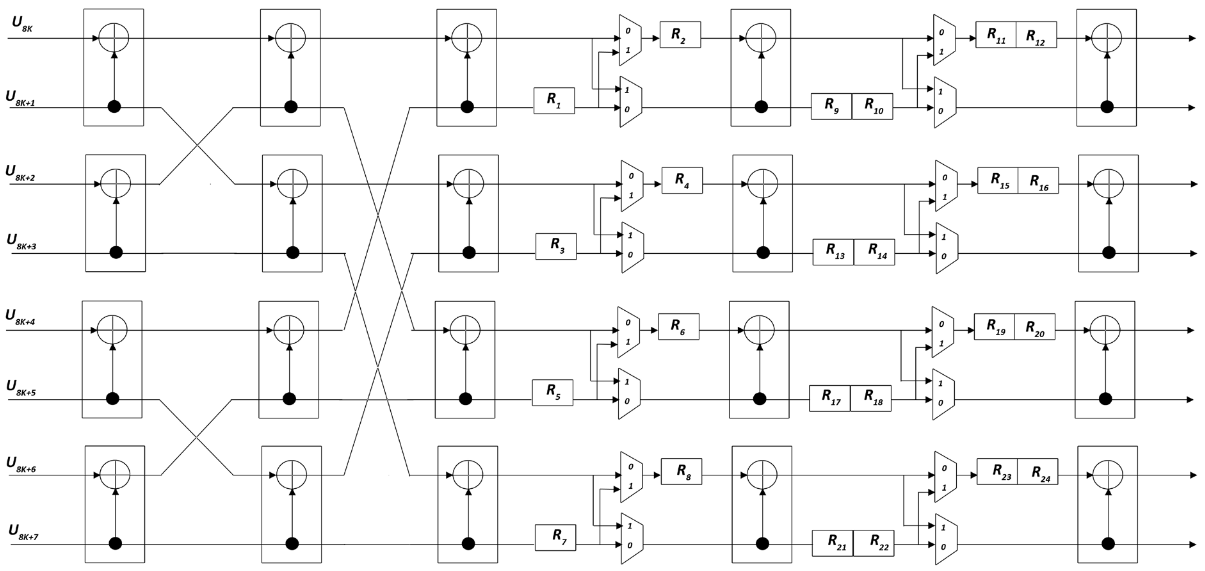
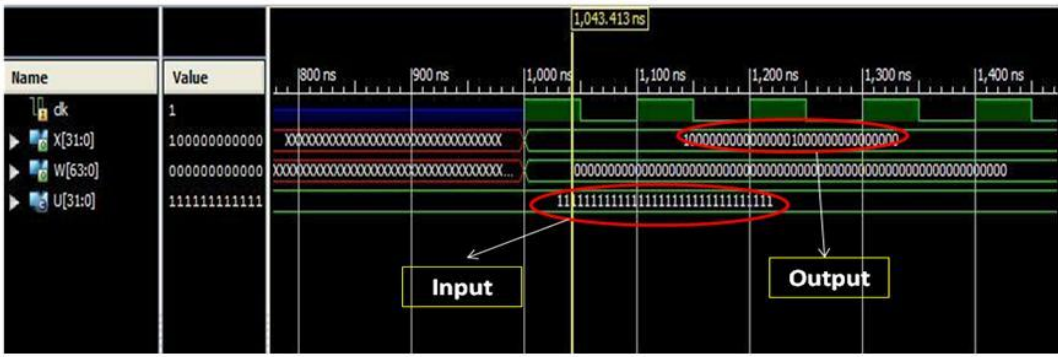


Figure 6. Proposed 8-parallel folded architecture for encoding the polar (32, *k*) codes.

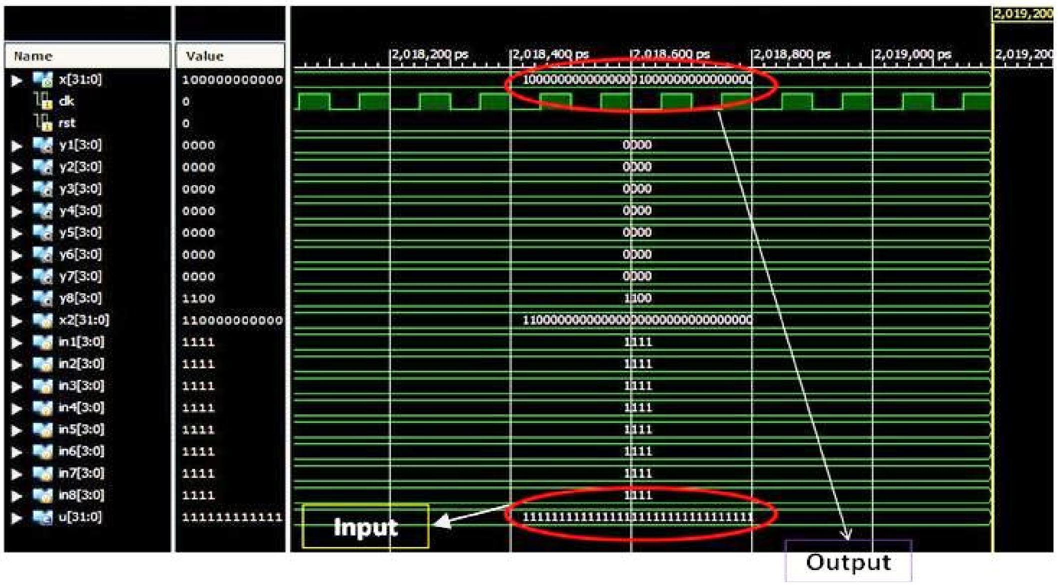
Table 4. Original $D(W_{ij})$ and recalculated $D'(W_{ij})$ delay requirement table for 8-parallel folded polar encoder.

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$D(W_{1j})$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$D(W_{2j})$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$D(W_{3j})$	1	1	1	1	2	2	2	2	0	0	0	0	1	1	1	1	1	1	1	1	−2	−2	−2	−2	0	0	0	0	−3	−3	−3	−3
$D(W_{4j})$	2	2	2	2	−2	−2	−2	−2	−0	−0	−0	−0	−0	−0	−0	−0	0	0	0	0	0	0	0	0	−2	−2	−2	−2	2	2	2	2
$D'(W_{1j})$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$D'(W_{2j})$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$D'(W_{3j})$	1	1	1	1	2	2	2	2	0	0	0	0	1	1	1	1	1	1	1	1	2	2	2	2	0	0	0	0	1	1	1	1
$D'(W_{4j})$	2	2	2	2	2	2	2	2	4	4	4	4	4	4	4	4	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2



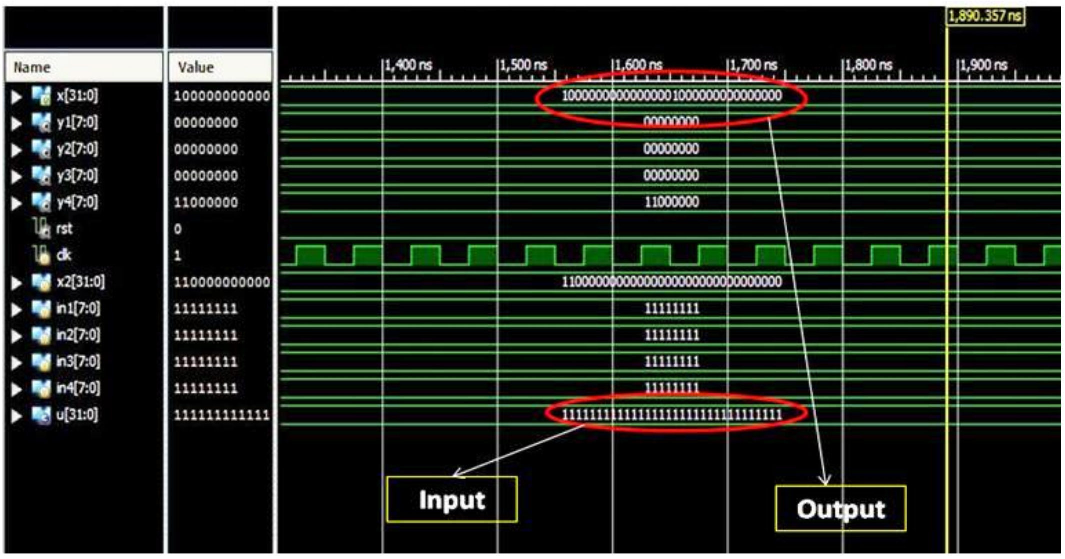
```
Name      Value
clk       1
X[31:0]   100000000000
W[63:0]   000000000000
U[31:0]   111111111111
```

Figure 7. Simulation result for fully parallel architecture.



```
Name       Value
x[31:0]    100000000000
clk        0
rst        0
y1[3:0]    0000
y2[3:0]    0000
y3[3:0]    0000
y4[3:0]    0000
y5[3:0]    0000
y6[3:0]    0000
y7[3:0]    0000
y8[3:0]    1100
x2[31:0]   110000000000
in1[3:0]   1111
in2[3:0]   1111
in3[3:0]   1111
in4[3:0]   1111
in5[3:0]   1111
in6[3:0]   1111
in7[3:0]   1111
in8[3:0]   1111
u[31:0]    111111111111
```

Figure 8. Simulation result for 4-parallel folded architecture.



```
Name       Value
x[31:0]    100000000000
y1[7:0]    00000000
y2[7:0]    00000000
y3[7:0]    00000000
y4[7:0]    11000000
rst        0
clk        1
x2[31:0]   110000000000
in1[7:0]   11111111
in2[7:0]   11111111
in3[7:0]   11111111
in4[7:0]   11111111
u[31:0]    111111111111
```

Figure 9. Simulation result for 8-parallel folded architecture.

Table 5. Register allocation table for W_{3j} and W_{4j}.

Cycle	Stage 3								R1	R2	R3	R4	R5	R6	R7	R8	Stage 4								R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24
0	$W_{3,0}$	$W_{3,2}$	$W_{3,4}$	$W_{3,6}$	$W_{3,1}$	$W_{3,3}$	$W_{3,5}$	$W_{3,7}$																																
1	$\boldsymbol{W_{3,8}}$	$\boldsymbol{W_{3,10}}$	$W_{3,12}$	$W_{3,14}$	$\boldsymbol{W_{3,9}}$	$\boldsymbol{W_{3,11}}$	$W_{3,13}$	$W_{3,15}$	$W_{3,4}$	$\boldsymbol{W_{3,0}}$	$W_{3,6}$	$\boldsymbol{W_{3,2}}$	$W_{3,5}$	$\boldsymbol{W_{3,1}}$	$W_{3,7}$	$\boldsymbol{W_{3,3}}$	$W_{4,0}$	$W_{4,2}$	$W_{4,8}$	$W_{4,10}$	$W_{4,1}$	$W_{4,3}$	$W_{4,9}$	$W_{4,11}$																
2	$W_{3,16}$	$W_{3,18}$	$W_{3,20}$	$W_{3,22}$	$W_{3,17}$	$W_{3,19}$	$W_{3,21}$	$W_{3,23}$	$\boldsymbol{W_{3,12}}$	$\boldsymbol{W_{3,4}}$	$\boldsymbol{W_{3,14}}$	$\boldsymbol{W_{3,6}}$	$\boldsymbol{W_{3,13}}$	$\boldsymbol{W_{3,5}}$	$\boldsymbol{W_{3,15}}$	$\boldsymbol{W_{3,7}}$	$W_{4,4}$	$W_{4,6}$	$W_{4,12}$	$W_{4,14}$	$W_{4,5}$	$W_{4,7}$	$W_{4,13}$	$W_{4,15}$	$W_{4,8}$		$W_{4,0}$		$W_{4,10}$		$W_{4,2}$		$W_{4,9}$		$W_{4,1}$		$W_{4,11}$		$W_{4,3}$	
3	$\boldsymbol{W_{3,24}}$	$\boldsymbol{W_{3,26}}$	$W_{3,28}$	$W_{3,30}$	$\boldsymbol{W_{3,25}}$	$\boldsymbol{W_{3,27}}$	$W_{3,29}$	$W_{3,31}$	$W_{3,20}$	$\boldsymbol{W_{3,16}}$	$W_{3,22}$	$\boldsymbol{W_{3,18}}$	$W_{3,21}$	$\boldsymbol{W_{3,17}}$	$W_{3,23}$	$\boldsymbol{W_{3,19}}$	$\boldsymbol{W_{4,16}}$	$\boldsymbol{W_{4,18}}$	$W_{4,24}$	$W_{4,26}$	$\boldsymbol{W_{4,17}}$	$\boldsymbol{W_{4,19}}$	$W_{4,25}$	$W_{4,27}$	$W_{4,12}$	$W_{4,8}$	$W_{4,4}$	$\boldsymbol{W_{4,0}}$	$W_{4,14}$	$W_{4,10}$	$W_{4,6}$	$\boldsymbol{W_{4,2}}$	$W_{4,13}$	$W_{4,9}$	$W_{4,5}$	$\boldsymbol{W_{4,1}}$	$W_{4,15}$	$W_{4,11}$	$W_{4,7}$	$\boldsymbol{W_{4,3}}$
4									$\boldsymbol{W_{3,28}}$	$\boldsymbol{W_{3,20}}$	$\boldsymbol{W_{3,30}}$	$\boldsymbol{W_{3,22}}$	$\boldsymbol{W_{3,29}}$	$\boldsymbol{W_{3,21}}$	$\boldsymbol{W_{3,31}}$	$\boldsymbol{W_{3,23}}$	$\boldsymbol{W_{4,20}}$	$\boldsymbol{W_{4,22}}$	$W_{4,28}$	$W_{4,30}$	$\boldsymbol{W_{4,21}}$	$\boldsymbol{W_{4,23}}$	$W_{4,29}$	$W_{4,31}$	$W_{4,24}$	$W_{4,12}$	$W_{4,8}$	$\boldsymbol{W_{4,4}}$	$W_{4,26}$	$W_{4,14}$	$W_{4,10}$	$\boldsymbol{W_{4,6}}$	$W_{4,25}$	$W_{4,13}$	$W_{4,9}$	$\boldsymbol{W_{4,5}}$	$W_{4,27}$	$W_{4,15}$	$W_{4,11}$	$\boldsymbol{W_{4,7}}$
5																									$W_{4,28}$	$\boldsymbol{W_{4,24}}$	$W_{4,12}$	$\boldsymbol{W_{4,8}}$	$W_{4,30}$	$\boldsymbol{W_{4,26}}$	$W_{4,14}$	$\boldsymbol{W_{4,10}}$	$W_{4,29}$	$\boldsymbol{W_{4,25}}$	$W_{4,13}$	$\boldsymbol{W_{4,9}}$	$W_{4,31}$	$\boldsymbol{W_{4,27}}$	$W_{4,15}$	$\boldsymbol{W_{4,11}}$
6																										$\boldsymbol{W_{4,28}}$		$\boldsymbol{W_{4,12}}$		$\boldsymbol{W_{4,30}}$		$\boldsymbol{W_{4,14}}$		$\boldsymbol{W_{4,29}}$		$\boldsymbol{W_{4,13}}$		$\boldsymbol{W_{4,31}}$		$\boldsymbol{W_{4,15}}$

Table 6. Comparison of various folding with fully parallel architecture.

Design/features	Fully parallel	Eight parallel folding	Four parallel folding
No. of ex or gates	80	20	10
No. of delay elements	0	24	28
No. of registers	0	96	224
Timing report (CPU to XST)	4.09 s	5.73 s	8.77s
Memory usage	164,328 KB	168,424 KB	169,448 KB

FPGA shows that the folding decreases the functional blocks (ex-or) operations, but needs trade off in the number of delay elements, registers and speed of execution.

References

[1] Arikan, E. (2009) Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels. *IEEE Transactions on Information Theory*, **55**, 3051-3073. http://dx.doi.org/10.1109/TIT.2009.2021379

[2] Hennessy, D.J. (1996) Computer Architecture: A Qualitative Approach. Morgan Kaufmann, USA.

[3] Hwang, F.K. (1984) Computer Architecture and Parallel Processing. McGraw-Hill, USA.

[4] Kogge, P. (1981) The Architecture of Pipelined Computers. McGraw-Hill, USA.

[5] Zhang, C., Yang, J., You, X. and Xu, S. (2015) Pipelined Implementations of Polar Encoder and Feed-Back Part for SC Polar Decoder. *IEEE International Conference Proceedings*, Lisbon, 24-27 May 2015, 3032-3035. http://dx.doi.org/10.1109/iscas.2015.7169326

[6] Parhi, K.K. (1999) VLSI Digital Signal Processing Systems: Design and Implementation. Wiley, USA.

[7] Oommen, M.S. and Ravishankar, S. (2015) FPGA Implementation of an Advanced Encoding and Decoding Architecture of Polar Codes. *International Conference on VLSI Systems*, *Architecture*, *Technology and Applications* (VLSI-SATA), 1-6. http://dx.doi.org/10.1109/VLSI-SATA.2015.7050456

[8] Mahdavifar, H. and Vardy, A. (2011) Achieving the Secrecy Capacity of Wiretap Channels Using Polar Codes. *IEEE Transactions on Information Theory*, **57**, 6428-6443. http://dx.doi.org/10.1109/TIT.2011.2162275

[9] Sasoglu, E., *et al*. (2010) Polar Codes for the Two-User Binary-Input Multiple-Access Channel. *IEEE Information Theory Workshop on Information Theory*, Cairo, 6-8 January 2010, 1-5. http://dx.doi.org/10.1109/itwksps.2010.5503184

[10] Mahdavifar, H., *et al*. (2014) Achieving the Uniform Rate Region of General Multiple Access Channels by Polar Coding. arXiv preprint arXiv1407.2990.

[11] Arikan, E. (2010) Source Polarization. *IEEE International Symposium on Information Theory Proceedings* (*ISIT*), 899-903. http://dx.doi.org/10.1109/isit.2010.5513567

[12] Goela, N., Abbe, E. and Gastpar, M. (2013) Polar Codes for Broadcast Channels. *IEEE International Symposium on Information Theory*, Istanbul, 7-12 July 2013, 1127-1131. http://dx.doi.org/10.1109/isit.2013.6620402

[13] Eslami, P.N. (2011) A Practical Approach to Polar Codes. *IEEE International Symposium on Information Theory Proceedings*, 16-20. http://dx.doi.org/10.1109/isit.2011.6033837

[14] Mizuochi, T., *et al*. (2009) Experimental Demonstration of Concatenated LDPC and RS Codes by FPGAs Emulation. *IEEE Photonics Technology Letters*, **21**, 1302-1304. http://dx.doi.org/10.1109/LPT.2009.2025867

[15] Yoo, H. and Park, I.C. (2015) Partially Parallel Encoder Architecture for Long Polar Codes. *IEEE Transactions on Circuits and Systems II*: *Express Briefs*, **62**, 306-310. http://dx.doi.org/10.1109/TCSII.2014.2369131

[16] Parhi, K.K. (1995) Calculation of Minimum Number of Registers in Arbitrary Life Time Chart. *IEEE Transactions on Circuits and Systems II*: *Analog and Digital Signal Processing*, **41**, 434-436. http://dx.doi.org/10.1109/82.300209

[17] Wang, C. and Parhi, K.K. (1995) High Level DSP Synthesis Using Concurrent Transformations, Scheduling, Allocation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **14**, 274-295. http://dx.doi.org/10.1109/43.365120