ISSN: 2278 – 909X
International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)
Volume 5, Issue 10, October 2016

Implementation of Parameter based Partially Parallel Encoder Architecture for Long Polar Codes

SRINATH.P, SAMUEL JOHN.J

Abstract **— Polar coding is an encoding/decoding scheme that demonstrably attains the capacity of the class of symmetric binary memory-less channels. Due to the channel achieving attribute, the polar code turns out to be one of the most optimal error-correcting codes. But, polar code must be long enough to have a decent error- correcting capability. Even if the earlier fully parallel encoder is inherent and easy to realize, it is inappropriate for long polar codes due to vast hardware complexity required. In this work, partially parallel folded encoder architecture is designed. The fully parallel architecture is transformed by analyzing its data flow graph, delay requirement calculation, lifetime analysis and register allocation, which gives the resultant folded architecture with mitigated hardware use. The encoder enhanced in the Very Large Scale Integration orientation which allows a tradeoff between the throughput and hardware complexity. It can be methodically used to the design of any polar code and to any degree of parallelism. Finally the power, area and delay reports of folded encoder architecture with comparison of existing work are presented.**

Key Terms— **Error correcting codes, FPGA, Polar codes, folding transformation.**

I. INTRODUCTION

The aim of communication systems is to transmit data reliably over a noisy channel. The conventional block diagram of a communication system is shown in *Figure1*. The source encoder eliminates the unnecessary information from the source's data. The channel encoder adds redundancy to the data such that authentic communication can be accomplished over a noisy channel. The function of the channel decoder is to replicate the data sent over the channel from the channel output. In the end, the source decoder reproduces the source's data from the output of channel decoder. Our prime concern in this thesis will be the blocks channel encoder, channel.

Figure 1: Communication System

A. Channel coding

The channel coding objective is to increase reliability of data transmission at the cost of reduction in information rate. This goal is accomplished by appending redundancy to the information symbol vector resulting in a longer coded vector of symbols that are detectable at the output of the channel.

B. Channel Parameters

Two important parameters of symmetric B-DMC's: The mutual information I(W) and the Bhattacharyya parameter Z(W).

The capacity of a symmetric B-DMC equals the mutual information between the input and output of the channel with uniform distribution on the inputs. I(W) is a measure of rate in a channel. It is familiar that reliable communication is possible over a symmetric B-DMC at any rates up to I(W). Bhattacharyya parameter is a measure of the reliability of a channel

In the past decades, the core objective of coding theory is to invent the new codes that approach Shannon limit closer than the prior codes. Inspired by this aim, information theoreticians have proposed generations of channel codes (*Figure* 2).

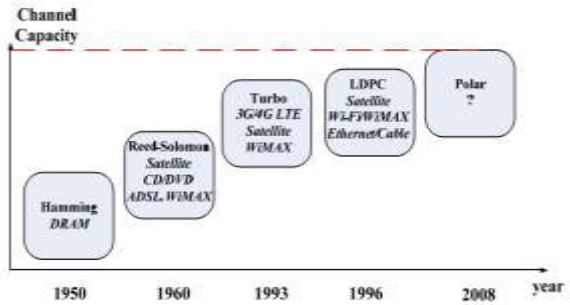

Figure2: Comparison of Channel coding techniques

Since the date being brought in, polar codes have received compelling attention from coding theory association. Compared to the prior best channel codes, polar codes can potentially exceed LDPC codes in terms of error-correcting performance with the similar code rate and code-length . Nevertheless, the approval of new

channel codes in IEEE standards does not only depend on the error-correcting performance, but also needs to consider the efficient VLSI design of codec. Inefficient hardware performance of polar codes decoder has become the critical challenge that restrain the practical use of polar codes.

Even if the polar code has been regarded as being associated with low complexity of O (N log N) for a polar code of length N and takes n stages when $N=2^n$, such a long polar code suffers from huge hardware complexity and long latency.

To be close to the channel capacity, the code length should be at least 2^{20} bits, and many literature works introduced polar codes ranging from 2^{10} to 2^{15} to attain decent error-correcting performances in practice.

The performance of improved folded encoder architecture can be analyzed by increasing the size of the input data and reduce the area coverage and delay. The code will be designed based on parameter method, it represent the value 'n'. The n value can be changed here based upon the memory size of FPGA used.

II. POLAR ENCODER

The name "polar" code is derived from the phenomenon of channel polarization [1]. As proved in, with efficient construction approach, the reliability of decoded bits will be polarized based on their different positions at the source data.

Therefore, an effective polar-based transmitter can be constructed based on the following principles: 1) sending required information bits at "good" positions, which can strongly guarantee the reliability of transmission; and 2) sending fixed "0" at "bad" positions, since after the transmission any decoded bits at these "bad" positions are highly unreliable. In those "0" bits are called "frozen" bits since these are fixed and their positions are known at both the encoder and the decoder. Similarly, the non-frozen information bits are referred as "free" bits. Accordingly, an (*n*, *k*) polar code contains *k* information ("free") bits and (*n*-*k*) "frozen" bits.

III. FULLY PARALLEL ENCODING

The fully parallel architecture (*Figure 3)* will contain 4 stages of operation, it will based on FFT Butterfly architecture, here the 16 input-bits are given (U0, U1, U2, U3, …..... ,U15) and get 16-bits output (X0, X1, X2, X3,..... X15).The inside operation uses for fixed XOR gate, and the stage1 output we can get W1,0 W1,1 W1,2 W1,3.. W1,15 and the Stage2 output we can get W2,0 W2,1 W2,3 W2,4...W2,15 and Stage3 output we can get W3,0 W3,1 W3,2 W3,3...W3,15 and finally we can get the 16-bit Polar code output[2].

This logic will be same for 32-bits, 64-bits, 128-bits..., based upon this architecture, as the bit size increases, the area size also increases. So we need the partially parallel encoder architecture for long polar codes

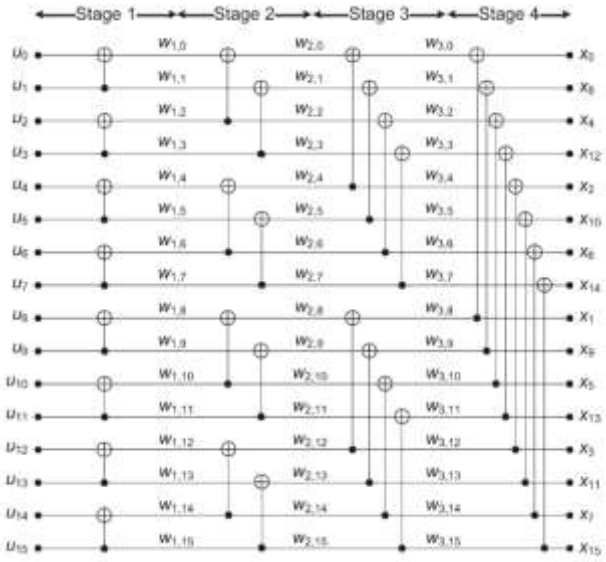

Figure 3: Parallel architecture for encoding a 16-bit polar code

Disadvantages: It has huge hardware complexity with $N/2(Log2^{n)}$ number of XOR gates. It requires more Power dissipation. It is not feasible for long polar codes to achieve channel capacity.

The folding transformation [3], [4] 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 *Figure 4*, where a node represents the kernel matrix operation *F*, and W*ij* denotes the *j*th edge at the *i*th 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 uses the kernel matrix instead of the butterfly operation.

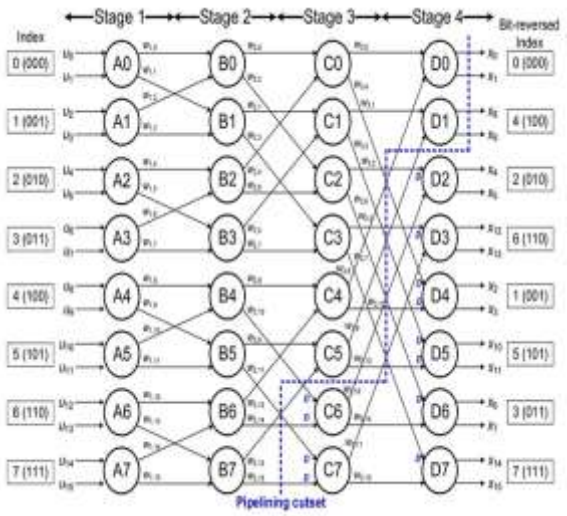

Figure 4: DFG of 16 bit polar encoding

IV. ENHANCED SYSTEM

The folding transformation is widely used to save hardware resources by time-multiplexing several operations on a functional unit.

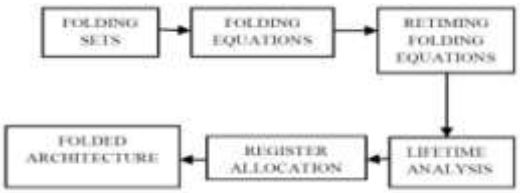

Figure 5: Folding transformation Techniques

A. Folding sets

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. 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 3 at the next cycle, and so forth.

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, 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 $A0$ and $B0$ 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 $\log 2\ P$, 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 $\log P$ has the folding sets obtained by cyclic shifting the previous folding sets of stage $s-1$ right by $s-\log P$. The logarithm has its base as 2.

B. Folding equations

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.

When an edge Wij from functional unit U to functional unit V has a delay of d, the delay requirements for Wij in the F-folded architecture can be calculated as

$$D(Wij) = Fd+v-u \quad (1)$$

Where u and v denote the position in the folding set corresponding to U and V, 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 summarized in *Figure 6.* For instance, $w2,0$ from $B0$ to $C0$ demands one delay since $d = 0$, $v = 1$, and $u = 0$. Note that some edges indicated by circles have negative delays.

Every edge with a negative delay should be compensated by inserting at least one delay element to make the value (1) of 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 elements are inserted to make the paths have the same delay elements. The delay elements can be observed in the Figure7.

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 6: Original delay requirements D(wij) and recalculated delay requirements D_(wij)

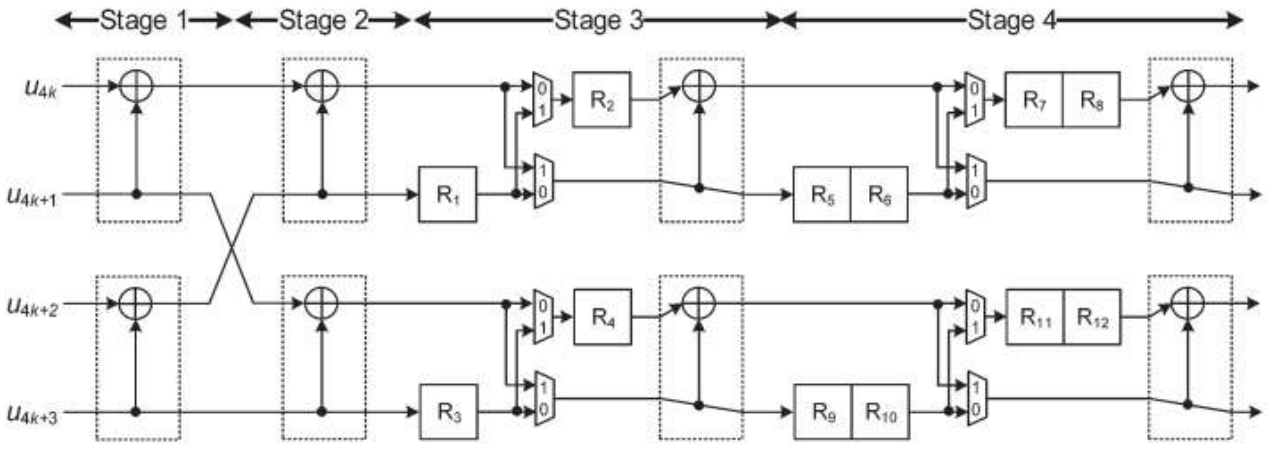

Figure 7: Proposed 4-parallel folded architecture for encoding the polar (n, K) codes.

C. Lifetime Analysis and Register Allocation

For the folded architecture of 16-bit polar encoder, there is room for minimizing the number of delay elements. The lifetime analysis 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 *Figure8*. Since all the edges starting from stage 1 demand no delay elements, only *w*2*j* and *w*3*j* are presented in *Figure8*.

For instance, *w*3, 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 architecture can be implemented with 12 delay elements instead of 48.

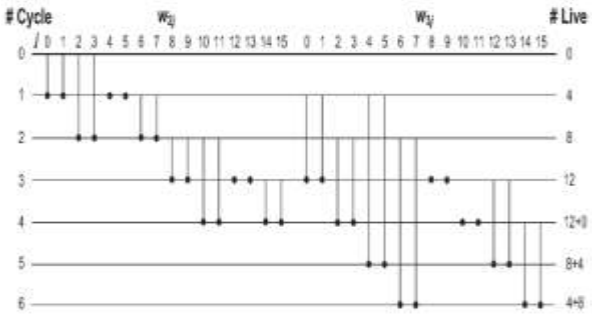

Figure 8: Linear lifetime chart for w2j and w3j.

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 Figure 9 [3].

D. Register Minimization Technique

The technique for minimizing register is lifetime analysis which analyzes the time for when a data is produced (T input) and when a data finally is consumed (T output).

$$T_{input} = u + PU \quad (2)$$

$$T_{output} = u + PU + \max_{v} \{DF(U \rightarrow V)\} \quad (3)$$

Where u is the folding order of U and PU is the number of pipelining stages in the functional unit that executes u.

Register allocation can be performed using an allocation table. The allocation scheme dictates how the variables are assigned to registers in the allocation table.

1) Determine the minimum number of registers using lifetime analysis.
2) Input each variable at the time step corresponding to the beginning of its lifetime.
3) Each variable is allocated in a forward manner until it is dead or it reaches the last register.
4) Since the allocation is periodic the allocation of the current iteration also repeats itself in subsequent iterations.
5) For variables that reach the last register and are not yet dead the remaining life period is calculated and these variables are allocated to a register in a backward manner on a first come first served basis.
6) Repeat steps 4 & 5 as required until the allocation is complete.

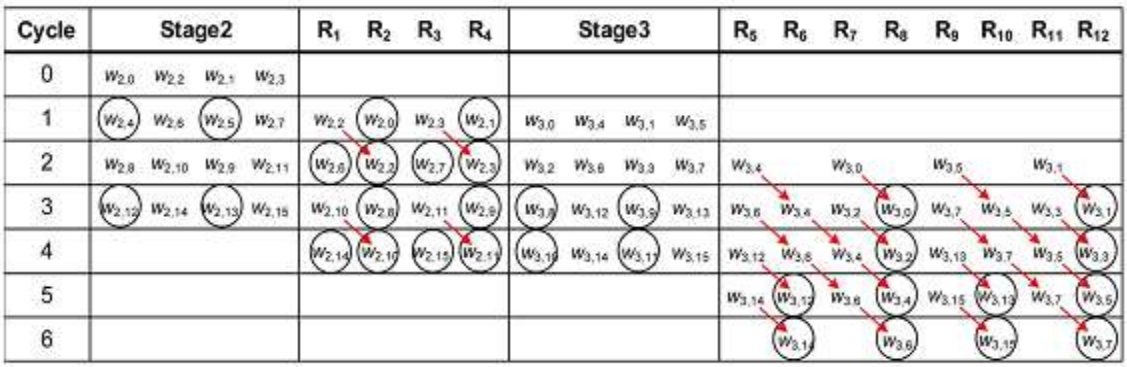

Figure 9: Register allocation table for w2j and w3j.

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 the *Figure 9*,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 [4]

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 folded architecture continuously processes four samples per cycle according to the folding sets and the register allocation table. Note that the folded encoder takes a pair of inputs in a natural order and generates a pair of outputs in a bit-reversed order. As the functional unit in the folded architecture processes a pair of 2 bits at a time it maintains the consecutive order at the input side and the bit reversed order at the output side if a pair of consecutive bits is regarded as a single entity [5],[6].

V. RESULTS AND COMPARISON

When the enter bit size increases although the delay remains constant, gate count increases, raises and there may be mild variants in power.

Table I: SYNTHESIS RESULTS OF POLAR (65536, K) ENCODERS

Bit size	**32**	**128**	**2048**	**8192**	**32768**	**65536**
Delay(ns)	2.698	2.698	2.698	2.698	2.698	2.698
Gate count	261 (165+96)	328(203+125)	452 (279+173)	514 (317+197)	576(355+221)	607(374+233)
Power(W)	0.121	0.123	0.132	0.135	0.137	0.138

Table II: GATE COUNTS OF POLAR (N, K) ENCODERS

Code length	**1024**		**2048**		**8192**		**32768**	**65536**
Design	Existing	proposed	Existing	proposed	Existing	proposed	proposed	proposed
Logic	1486	260	1651	279	2127	317	355	374
Register	7283	161	16661	173	66337	197	221	233
Total	8769	421	18312	452	68464	514	576	607

The black box defines what are the inputs and outputs of our design

Figure 10: Black box

The Register Transfer Level converts the code into block diagram

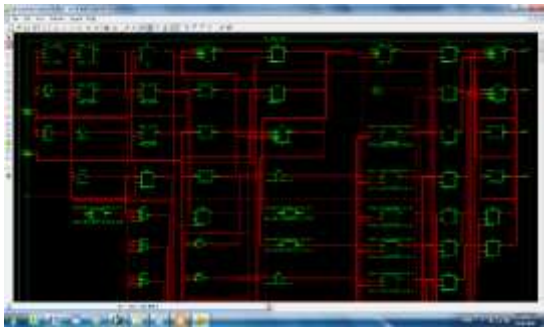

Figure 11: RTL schematic

The Technology schematic defines the usage of the LUT's, number of slices, number of IOB's.

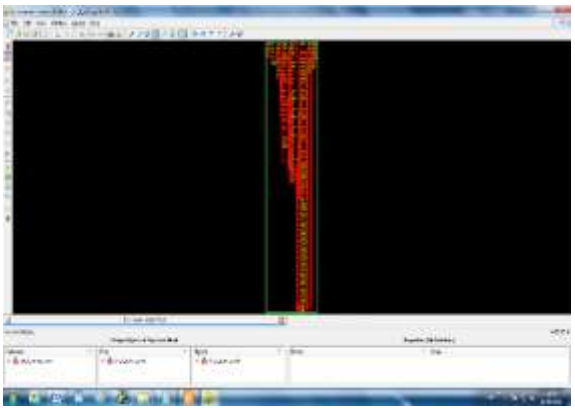

Figure 12: Technology schematic

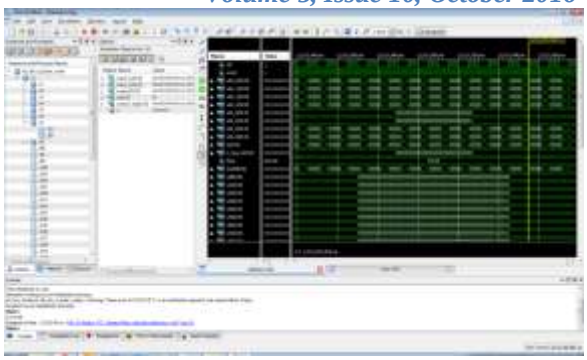

Figure 13: Simulation results

VI. CONCLUSION

The partially parallel encoder architecture which supports long polar codes is developed. For the this procedure, we are increasing the size of the inputs but field optimization is achieved. The experimental results exhibit the hardware self-discipline to curb within the proposed technique. The polar encoder can be implemented with various parallelism depend on application. It has applications in wireless communication protocols .Polar codes can be concatenated with other powerful error correcting codes to achieve more optimal solutions for data storage applications.

VII.REFERENCE

1. Arikan, E. (2009) Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memory less Channels. IEEE Transactions on Information Theory, 55, 3051-3073.

2. HoyoungYoo, and In-Cheol Park, „Partial parallel encoder architecture for long polar codes", IEEE Transactions on Circuits and Systems II, Oct. 21, 2014.
3. K.K. Parhi, VLSI Digital Signal Processing Systems: Design and Implementation. Hoboken, NJ, USA: Wiley, 1999
4. 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
5. Hwang, F.K. (1984) Computer Architecture and Parallel Processing. McGraw-Hill, USA.
6. Hennessy, D.J. (1996) Computer Architecture: A Qualitative Approach. Morgan Kaufmann, USA.