TRANSPORT AND TELECOMMUNICATION INSTITUTE

The 16th International Conference

RELIABILITY and STATISTICS in TRANSPORTATION and COMMUNICATION (RelStat'16)

19–22 October 2016. Riga, Latvia

Organised by

Transport and Telecommunication Institute (Latvia)
in co-operation with
Latvian Academy of Science (Latvia)

PROCEEDINGS

Edited by

Igor V. Kabashkin

Irina V. Yatskiv

RIGA - 2016

Proceedings of the 16th International Conference *RELIABILITY and STATISTICS in TRANSPORTATION and COMMUNICATION* (RelStat'16), 19–22 October 2016, Riga, Latvia.

Papers were reviewed by the Scientific Committee members;
the responsibility for context and grammar of the papers rests upon the authors.

Transport and Telecommunication Institute
Lomonosova iela 1, LV-1019, Riga, Latvia
http://RelStat.tsi.lv

ISBN 978-9984-818-83-2

Proceedings of the 16th International Conference "Reliability and Statistics in Transportation and Communication" (RelStat'16), 19–22 October 2016, Riga, Latvia, p. 283–293. ISBN 978-9984-818-83-2
Transport and Telecommunication Institute, Lomonosova 1, LV-1019, Riga, Latvia

ENCODER IMPROVEMENT FOR SIMPLE AMPLITUDE FULLY PARALLEL CLASSIFIERS BASED ON GREY CODES

Sergejs Šarkovskis[1], Aleksandrs Jeršovs[2], Deniss Kolosovs[3] and Elans Grabs[3]

[1]The Faculty of Computer Science and Telecommunication, Transport and Telecommunication Institute
Lomonosova str. 1, LV-1019, Riga, Latvia
E-mail: sarkovskis.s@tsi.com

[2]SAF Tehnika JSC
Ganibu dambis str. 24a, LV-1005, Riga, Latvia
E-mail: Aleksandrs.Jersovs@saftehnika.com

[3] Department of Fundamentals of Electronics, Riga Technical University,
Azenes str. 12, LV-1048, Riga, Latvia
E-mail: elans.grabs@rtu.lv

The present article describes functionality of real-time classifier usable for data flow statistical parameters calculations, different modulation types symbol detecting and in other applications, where the fastest association of input signal sample is required with one of the predefined categories. The effective implementation of encoder with high number of bits for fully parallel classifier is provided based on Gray codes (Gray, 1953). The work is concluded with comparative analysis of encoder standard implementation and its optimized version for FPGAs manufactured by Xilinx and Altera companies.

Keywords: FPGA, big data, simple amplitude classifier, Gray code

1. Introduction

The problem of high speed data processing gains its actuality with wide spreading of embedded control/monitoring systems for physical world processes (PWP), which leads to:

- increased complexity of implemented algorithms (i.e., it is possible to implement more complex processing algorithm with higher data rate and same time interval, since PWP wasn't altered);
- development of new application fields (i.e., increase of performance allows former algorithm to be used in such areas, where it couldn't be applied before, since implementation environment was lagging behind PWP).

Another factor leading to more complex algorithm is information volume increase, since improvement of algorithm structure makes it possible to process higher volumes of data.

That is why programmable logical devices (PLD) are getting increasingly popular for solutions in fast processing and/or operating with high volumes of data. The main advantage of PLD in specified field is a possibility to implement complex parallel algorithms with performance improvement (Bashkirov and Muratov, 2012) compared to serial algorithms implemented in standard processing devices. The following facts confirm the foregoing:

- Intel corporation added FPGA structures into latest server processors Xeon (Shah, 2016),
- FPGA devices are used in Microsoft Corporation datacenters to speed-up data processing, in particular, performance gain in case of Bing search engine is 95% (Putnam, 2014),
- the further development of this idea in Microsoft Corporation leads to research of FPGA applications in neural networks (Deep Convolutional Neural Networks) for performance improvement (Ovtcharov *et al.*, 2015),
- ATI company has been using Xilinx company manufactured FPGA devices to provide ATI Crossfire support for their graphic cards (ATI, 2005),
- FPGA devices are used for search of dark matter in CHIME telescope (Leibson, 2014) to split digitized data into 1024 frequency channels from 16 ADC with transfer rate of 15.5 Gbps,
- Mango Communications in cooperation with Rice University developed system for 802.11s standard wireless system dynamics real-time research, which uses 24 FPGA devices connected to 96 antennas (Murphy and Zhong, 2014).

One of the most important objectives of information processing algorithms is execution of specific procedures over objects according to their affiliation to some groups with specified properties. Such tasks can be found in multiple fields:

- In communications: analog-digital conversions, symbol detectors of various modulation types, enhancement of calculation consuming functions tabular implementation;
- big data & statistics: histogram construction, frequency analysis of text data arrays,
- computer vision (theory of objects recognition): calculation of similarities between objects, partitioning of objects set into separate groups;
- networking technologies: users distributions in priority groups, for example based on their activity results;
- automated trading systems: filtering the most profitable orders and hiding the unfavorable deals;
- neural networks: making decision at output of neuron, finding greatest weight in neural network;
- hardware implementation of cryptographical algorithms: privacy-preserving classification);
- and other.

Such type of applications will be further referred to as processing with pre-classification of objects. To improve performance of entire system it is desirable to merge these two operations (classification and processing).

If classification implies distribution of objects interpreted by numbers into groups based on their values (amplitudes), then such task can be solved by device further in text referred to as simple amplitude classifier (CAC).

2. Taxonomy of simple amplitude classifiers

CAC implementation variations are shown in Figure 1. Let's describe advantages and shortcomings of these methods with special accent on FPGA implementation due to effective capabilities for high speed data processing on this platform.

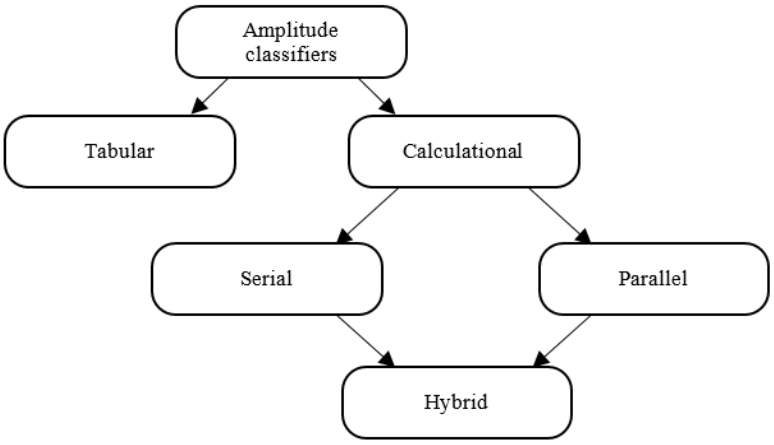


Figure 1. Simple amplitude classifier implementation variations

Tabular implementation – creation of special table, which consists of all possible classifier operation results sorted by input value amplitude. Functionality of such device is based on data reading from table cells with address specified by input data.

Tabular approach is appropriate when table is small-sized relative to implementation platform used, which depends both on address space specified by input value and on bitwidth of cell contents. Such implementation has number of benefits:

- high performance (result is obtained during 1 cycle),
- simple implementation and utilization logic.

However, there are several disadvantages as well:

- big-sized tables demand large amounts of resources,

- non-universal and complex reconfiguration in tasks requiring different classifiers implemented in same hardware,
- complex functionality for input data with floating point.

When shortcomings specified above do not allow implementation of classifier via tables, it is necessary to use more complex decision making algorithms to determine affiliation of data to specific group. These are computational classifiers (CC). Information processing with pre-classification by CC consists of 3 stages (see Fig. 2)

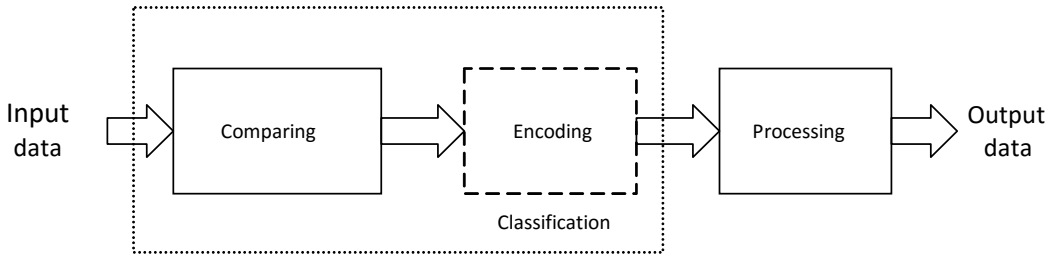


Figure 2. Stages of information processing with pre-classification

The first stage is performed by a set of comparators, each of them has simple comparison function shown in Figure 3.

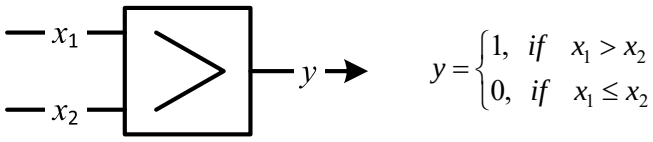


Figure 3. Comparator and comparison function

At this stage the input samples are compared with comparator's thresholds which yields resulting vector formed by outputs of comparators. This vector controls processing of different groups either directly or via an encoder. The architecture of comparators set defines belonging of CC to one of the groups shown in Figure 1.

2.1. Serial CCs

A serial comparing variant assumes that affiliation of input word to some specific group is performed in multiple cycles (see Fig. 4).

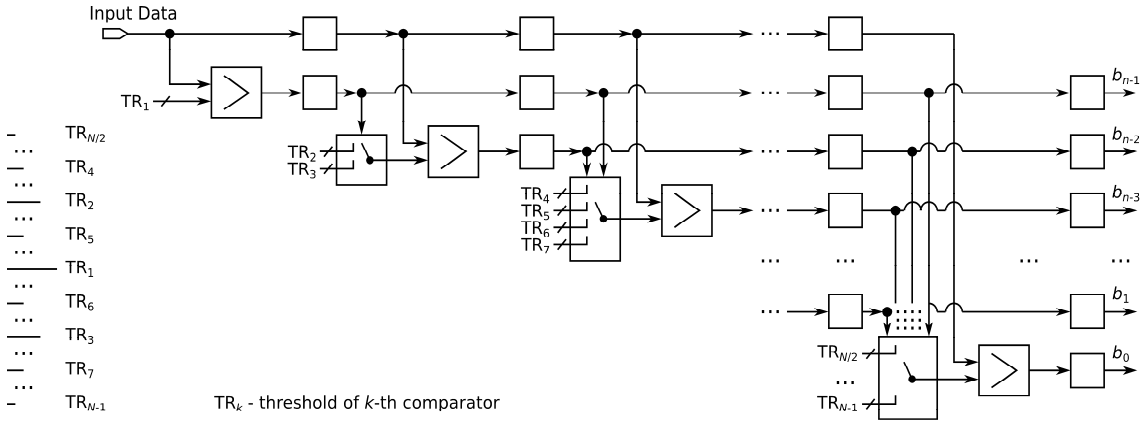


Figure 4. Serial computational real-time classifier

The main advantage of this approach is minimal amount of required comparators: 1 comparator for non real-time applications (with $\lfloor \log_2(N) \rfloor$ cycles for one number processing at most) and $\lfloor \log_2(N) \rfloor$

comparators for real-time pipeline implementation (a result is always delayed by $\lfloor \log_2(N) \rfloor$ cycles), where *N* is a number of groups. This approach, nonetheless, has multiple significant disadvantages:

- there is an unnecessary latency of result for real-time classifier, which may lead to impairment of regulation in control systems with feedback classifier;
- relative complexity of algorithm operation: comparing thresholds commutation, latency matching, additional registers for intermediary results storage and so on;
- difficult real-time change of thresholds during operation if classification system is modified.

Considering shortcomings specified above, the application of serial comparing in high speed data processing jobs is undesirable, especially when SAC is in system feedback loop.

2.2. Parallel CCs

Parallel type of comparing has been used for years in architecture of parallel ADCs (Kester, 2005). Its key concept is transfer of the code word to all $N-1$ comparators at once. The other inputs of comparators are connected to thresholds $\left(\mathrm{TR}_1 \div \mathrm{TR}_{N-1}\right)$, sorted in increasing order (see Fig. 5).

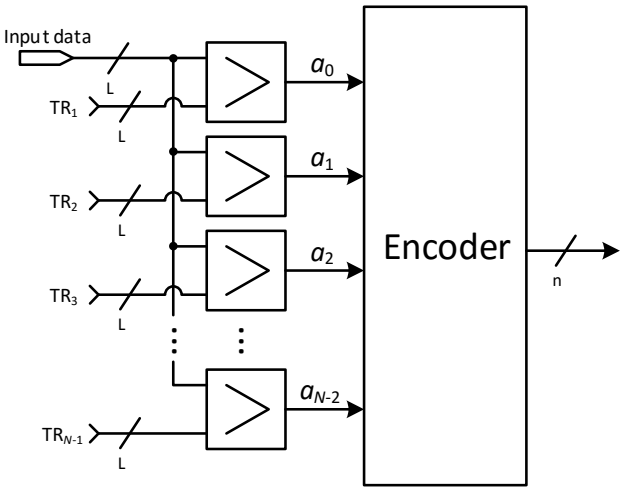


Figure 5. Parallel real-time computational classifier

Output vector of all comparators $A = \left[a_{N-2} \ldots a_2 a_1 a_0\right]$, with each of them functioning according to Figure 5, is thermometric code. It is obvious, that this code is redundant and thus often before processing stage thermometric code is converted by encoder into *n* bits binary code, where

$$n = \lfloor \log_2(N-1) \rfloor \tag{1}$$

Advantages:

- minimal latency (1-2 cycles regardless of number of classifier groups), and as of such, the highest achievable performance;
- simple reconfiguration: change of thresholds, reduction of classifier groups without changes in hardware, including encoder and subsequent circuits.

Shortcomings:

- large number of comparators used, which leads to allocation of great resources volume and as a such to increased power consumption, routing complexity and heatsink problems;

2.3. Hybrid CCs

If required volume of resources does not allow parallel implementation of comparing and minimal latency of the result is required, a hybrid scheme (serial-parallel) can be used. In such a case the part of comparators with respective encoder can be implemented in parallel with further operating with this sub-block multiple times (non real-time) or multiple copies of it in series (real-time) with proper change of thresholds.

Considering topicality of high rate data processing the parallel CC are described further in text.

3. Feasibility of encoder use in parallel CCs

Let us consider an example, where input data must be classified over $N = 32$ groups based on amplitude. For every group its own event is defined by comparators output vector $A = [a_{30}a_{29} \ \ldots \ a_1a_0]$ according to Table 1.

Table 1. Vector *A* description for different classification groups

A / No.	a_{30}	a_{29}	...	a_3	a_2	a_1	a_0
0	0	0	···	0	0	0	0
1	0	0	···	0	0	0	1
2	0	0	···	0	0	1	1
3	0	0	···	0	1	1	1
4	0	0	···	1	1	1	1
⋮	⋮			⋮			
30	0	1	···	1	1	1	1
31	1	1	···	1	1	1	1

The processing of particular group implemented in HDL language can be done by case (or if) construction, where conditional expression of branches is comparators output vector *A*. For instance, in VHDL:

```
if    A(30 downto 0) = "0000000000000000000000000000001" then <statement>;
elsif A(30 downto 0) = "0000000000000000000000000000011" then <statement>;
elsif A(30 downto 0) = "0000000000000000000000000000111" then <statement>;

. . .

elsif A(30 downto 0) = "1111111111111111111111111111111" then <statement>;
else                                                          <statement>;
end if;
```

Implementation of such straightforward solution leads to complex multilevel asynchronous logic shown, and as of such to disproportionate resources consumption and performance degradation.

It is possible to try to eliminate this problem by taking advantage of specific structure of thermometric code. Since input number encounter in particular group can be determined from change between sequences of ones and zeros, then conditional expression can be created from two bits with mentioned transition between sequences.

```
if    A( 1 downto  0) = "01" then <statement>;
elsif A( 2 downto  1) = "01" then <statement>;
elsif A( 3 downto  2) = "01" then <statement>;

. . .

elsif A(30 downto 30) =  "1" then <statement>;
else                             <statement>;
end if;
```

The solution with different sets of control signals in if-construction (not case-construction!) branches is acceptable (Sisterna, 2013), however, it will lead to creation of priority encoded logic (Flaxer, no date), that impairs performance and system resource intensity, which increases the risk of meta-stability. According to recommendations (Xilinx, no date) use of priority encoded logic is best to be avoided, especially considering that experimental results (see Table 5, Spartan 6. "Straightforward" reduced) show no improvements for this method compared to straightforward implementation.

It is obvious from the mentioned above, that for great number of groups and as of such – comparators, additional encoding operation is required, that will reduce vector *A* into vector *B* of smaller bitwidth.

The necessity of separate encoder in CC is not always justified and depends on further operation ("Processing" stage in Fig. 2).

- Encoder is not required if operation performed during processing of comparators outputs for each classification group is individual. For example, when for histogram construction separate

counter is used for every group, that keeps data of its own and neighbour's comparator output and does not take into account any other outputs. In such a case HDL implementation is a set of independent conditional constructions:

```
if    A( 1 downto  0) = "01" then
   Cnt1(7 downto 0) <= Cnt1(7 downto 0) + ”00000001”;
else
   Cnt1(7 downto 0) <= Cnt1(7 downto 0);
end if;

. . .

if    A(30 downto 29) = "01" then
   Cnt30(7 downto 0) <= Cnt30(7 downto 0) + ”00000001”;
else
   Cnt30(7 downto 0) <= Cnt30(7 downto 0);
end if;
```

which follow Xilinx recommendations mentioned above. Thus encoder is not required, since it only makes implementation more complex. However, it shouldn't be supposed that example described in text above excludes the use of encoder in general for histogram construction. It is rational to use classical approach when RAM array is used for histogram construction to store counted numbers of bins and single counter, that increments currently classified group number. In this case it is impossible to use a set of independent conditional constructions, since it will lead to multi-sourcing problem, and thus encoder is required.

- When operation performed for processing of comparators outputs is definition of a constant (different for each group), encoder can be combined/replaced with this assignment. In this case each output bit of such constants can be separately described with logical functions.

4. Encoder implementation in PLD

In order to improve performance of entire system, and consequently, combine encoding operation with further processing, it is necessary to implement encoder with asynchronous logic. Asynchronous logic, depending on its topology and complexity, can potentially have problems with timely response and signal propagation (different delays), and as of such it is necessary to pay special attention to logic mapping & routing. Thus, from implementation point of view, the following requirements must be met:

- minimal resources consumption;
- maximal performance, which allows encoder to operate in the same cycle when processing is performed.

In our case, encoder perform conversion of thermometric code *A* into binary code *B*. The simplest option – choose for vector $B = \left[b_{n-1} \ \ldots \ b_1 b_0 \right]$ a binary positional code, which shows a number of a bit, where sequence of ones transforms into sequence of zeros, where *n* is calculated according to (1). An example with $N = 32$ and $n = 5$ is shown in Table 2.

Table 2. Example of vectors *A* and *B* combinations

Bin No.	a_{30}	a_{29}	···	a_3	a_2	a_1	a_0	b_4	b_3	b_2	b_1	b_0
0	0	0	···	0	0	0	0	0	0	0	0	0
1	0	0	···	0	0	0	1	0	0	0	0	1
2	0	0	···	0	0	1	1	0	0	0	1	0
3	0	0	···	0	1	1	1	0	0	0	1	1
4	0	0	···	1	1	1	1	0	0	1	0	0
⋮				⋮						⋮		
30	0	1	···	1	1	1	1	1	1	1	1	0
31	1	1	···	1	1	1	1	1	1	1	1	1

Based on data from Table 2, each bit of vector C can be formed with logical function obtained from canonical disjunctive normal form (CDNF). However, this solution, in its essence, is one of the mentioned above possible variations with all its advantages and shortcomings, described by discrete logic instead of conditional constructions.

4.1. Encoder implementation simplification.

Let us examine encoder bit patterns (EBP) of binary positional code (Table 3). Gray color marks logical ones.

Table 3. EBP of binary positional code

b_0		■		■		■		■		■		■		■		■		■		■		■		■		■		■	
b_1			■	■			■	■			■	■			■	■			■	■			■	■			■	■		
b_2					■	■	■	■					■	■	■	■					■	■	■	■					■	■
b_3									■	■	■	■	■	■	■	■									■	■	■	■	■	■
b_4																	■	■	■	■	■	■	■	■	■	■	■	■	■	■

Output bits of encoder can be described with discrete logic as follows:

$$b_0 = a_0\overline{a}_1 + a_2\overline{a}_3 + a_4\overline{a}_5 + a_6\overline{a}_7 + a_8\overline{a}_9 + a_{10}\overline{a}_{11} + a_{12}\overline{a}_{13} + a_{14}\overline{a}_{15} + a_{16}\overline{a}_{17} + a_{18}\overline{a}_{19} + a_{20}\overline{a}_{21} + a_{22}\overline{a}_{23} + a_{24}\overline{a}_{25} + a_{26}\overline{a}_{27} + a_{28}\overline{a}_{29} + a_{30} \quad (2a)$$

$$b_1 = a_1\overline{a}_2 + a_2\overline{a}_3 + a_5\overline{a}_6 + a_6\overline{a}_7 + a_9\overline{a}_{10} + a_{10}\overline{a}_{11} + a_{13}\overline{a}_{14} + a_{14}\overline{a}_{15} + a_{17}\overline{a}_{18} + a_{18}\overline{a}_{19} + a_{21}\overline{a}_{22} + a_{22}\overline{a}_{23} + a_{25}\overline{a}_{26} + a_{26}\overline{a}_{27} + a_{29}\overline{a}_{30} + a_{30} \quad (2b)$$

$$b_2 = a_3\overline{a}_4 + a_4\overline{a}_5 + a_5\overline{a}_6 + a_6\overline{a}_7 + a_{11}\overline{a}_{12} + a_{12}\overline{a}_{13} + a_{13}\overline{a}_{14} + a_{14}\overline{a}_{15} + a_{19}\overline{a}_{20} + a_{20}\overline{a}_{21} + a_{21}\overline{a}_{22} + a_{22}\overline{a}_{23} + a_{27}\overline{a}_{28} + a_{28}\overline{a}_{29} + a_{29}\overline{a}_{30} + a_{30} \quad (2c)$$

$$b_3 = a_7\overline{a}_8 + a_8\overline{a}_9 + a_9\overline{a}_{10} + a_{10}\overline{a}_{11} + a_{11}\overline{a}_{12} + a_{12}\overline{a}_{13} + a_{13}\overline{a}_{14} + a_{14}\overline{a}_{15} + a_{23}\overline{a}_{24} + a_{24}\overline{a}_{25} + a_{25}\overline{a}_{26} + a_{26}\overline{a}_{27} + a_{27}\overline{a}_{28} + a_{28}\overline{a}_{29} + a_{29}\overline{a}_{30} + a_{30} \quad (2d)$$

$$b_4 = a_{15}\overline{a}_{16} + a_{16}\overline{a}_{17} + a_{17}\overline{a}_{18} + a_{18}\overline{a}_{19} + a_{19}\overline{a}_{20} + a_{20}\overline{a}_{21} + a_{21}\overline{a}_{22} + a_{22}\overline{a}_{23} + a_{23}\overline{a}_{24} + a_{24}\overline{a}_{25} + a_{25}\overline{a}_{26} + a_{26}\overline{a}_{27} + a_{27}\overline{a}_{28} + a_{28}\overline{a}_{29} + a_{29}\overline{a}_{30} + a_{30} \quad (2f)$$

Analysis of equations (2a-f) shows, that each encoder's bit is formed from multiple conjunctions with same arguments a_k, so implementation of each b_k with separate truth table may cause use of redundant resources. If cross-optimization is performed to reduce resource cost for different b_k, then additional logic layer shows up and routing becomes more complex, that degrades performance of asynchronous logic. In order to eliminate specified potential problems, the further improvement of logic is described further.

An important point of classifier operation is occurrence of input number in only one group, which excludes occurrence in other groups. Which means that only one conjunction can be equal to one in equations (2a-f) for each response of encoder. This important feature can be used to replace disjunctions in (2a-f) by XOR logical function (see Fig.6a), so in this specific case truth tables of OR and XOR functions are equivalent.

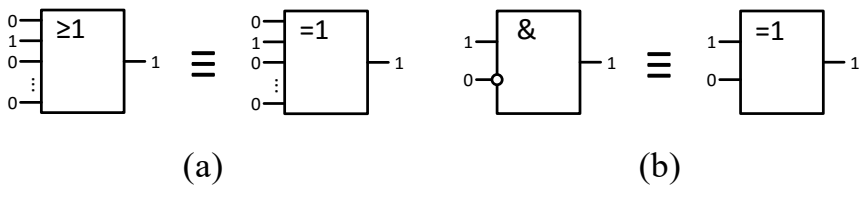


Figure 6. Equivalent transforms of logical functions

Since conjunctions of expressions (2a-f) are used for detecting transition of ones sequence into zeros sequence, they can be replaced by XOR functions as is shown I Figure 6b. Note, however, that XOR function can detect both transition of zero to one and transition of one to zero. However the second case in this specific thermometric code is impossible.

After conducted transforms the output of encoder is a set of separate comparators outputs combined only by XOR functions. For example, b_4:

$$b_4 = a_{15} \oplus a_{16} \oplus a_{16} \oplus a_{17} \oplus a_{17} \oplus a_{18} \oplus a_{18} \oplus a_{19} \oplus a_{19} \oplus a_{20} \oplus a_{20} \oplus a_{21} \oplus a_{21} \oplus a_{22} \oplus a_{22} \oplus a_{23} \oplus a_{23} \oplus a_{24} \oplus a_{24} \oplus a_{25} \oplus a_{25} \oplus a_{26} \oplus a_{26} \oplus a_{27} \oplus a_{27} \oplus a_{28} \oplus a_{28} \oplus a_{29} \oplus a_{29} \oplus a_{30} \oplus a_{30} \quad (3)$$

Then, after expanding brackets and using expression (4)

$$a_k \oplus a_k = 0 \quad (4)$$

it is possible to reduce same components of (2a-f) and decrease total complexity of encoder:

$$b_0 = a_0 \oplus a_1 \oplus a_2 \oplus a_3 \oplus a_4 \oplus a_5 \oplus a_6 \oplus a_7 \oplus a_8 \oplus a_9 \oplus a_{10} \oplus a_{11} \oplus a_{12} \oplus a_{13} \oplus a_{14} \oplus a_{15} \oplus a_{16} \oplus a_{17} \oplus a_{18} \oplus a_{19} \oplus a_{20} \oplus a_{21} \oplus a_{22} \oplus a_{23} \oplus a_{24} \oplus a_{25} \oplus a_{26} \oplus a_{27} \oplus a_{28} \oplus a_{29} \oplus a_{30} \quad (5a)$$

$$b_1 = a_1 \oplus a_3 \oplus a_5 \oplus a_7 \oplus a_9 \oplus a_{11} \oplus a_{13} \oplus a_{15} \oplus a_{17} \oplus a_{19} \oplus a_{21} \oplus a_{23} \oplus a_{25} \oplus a_{27} \oplus a_{29} \quad (5b)$$

$$b_2 = a_3 \oplus a_7 \oplus a_{11} \oplus a_{15} \oplus a_{19} \oplus a_{23} \oplus a_{27} \quad (5c)$$

$$b_3 = a_7 \oplus a_{15} \oplus a_{23} \quad (5d)$$

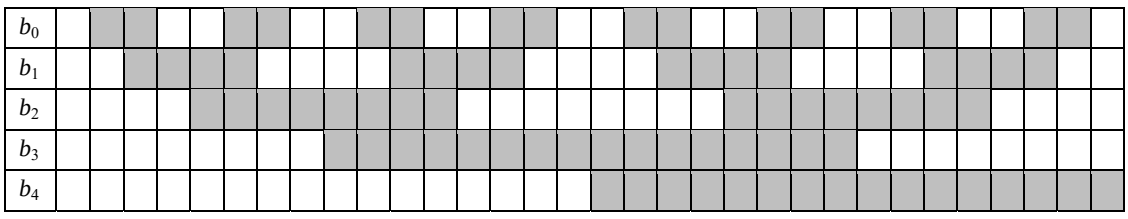
$$b_4 = a_{15} \quad (5f)$$

In this case of implementation part of outputs, such as b_0, has great number of components, which will increase proportionally to the number of classification groups, which will naturally have a negative impact on performance and resources consumption.

However, if specific description of vector B combinations is not specified explicitly, the specific choice of encoder states table will allow to substantially reduce its complexity. The table of encoder output vectors must be chosen so that columns contain as least “breaks” between long sequences of ones as possible.

One of good options is Gray code (Gray, 1953), since sufficiently long and monotonous sequences of ones and zeros can be expected in its columns due to variation of single bit in two consequent rows. However, such solution won’t be the best, since Gray codes require change of a single bit (which is not mandatory in considered case) and doesn’t specify minimum number of monotonous fragments in columns (which is important condition to minimize number of operands in (5)). A Gray code with EBP shown in Table 4 has been used for experiment.

Table 4. EBP of Gray code



The output of encoder is described with following set of equations:

$$b_0 = a_0 \oplus a_2 \oplus a_4 \oplus a_6 \oplus a_8 \oplus a_{10} \oplus a_{12} \oplus a_{14} \oplus a_{16} \oplus a_{18} \oplus a_{20} \oplus a_{22} \oplus a_{24} \oplus a_{26} \oplus a_{28} \oplus a_{30} \quad (6a)$$

$$b_1 = a_1 \oplus a_5 \oplus a_9 \oplus a_{13} \oplus a_{17} \oplus a_{21} \oplus a_{25} \oplus a_{29} \quad (6b)$$

$$b_2 = a_3 \oplus a_{11} \oplus a_{19} \oplus a_{27} \quad (6c)$$

$$b_3 = a_7 \oplus a_{23} \quad (6d)$$

$$b_4 = a_{15} \quad (6f)$$

The code portion in VHDL language, that implements encoder with Gray coding and post-processing is as follows

```
-- Processing inside process construction
if    B(4 downto 0)="00001" then <statement>;
elsif B(4 downto 0)="00011" then <statement>;
elsif B(4 downto 0)="00010" then <statement>;
. . .
elsif B(4 downto 0)="10000" then <statement>;
else                             <statement>;
end if;

-- Encoder on async. Logic
B(4 downto 4) <= A(15 downto 15);

B(3 downto 3) <= A(23 downto 23) xor A( 7 downto  7);

B(2 downto 2) <= A(27 downto 27) xor A(19 downto 19) xor
                 A(11 downto 11) xor A( 3 downto  3);

B(1 downto 1) <= A(29 downto 29) xor A(25 downto 25) xor A(21 downto 21) xor
                 A(17 downto 17) xor A(13 downto 13) xor A( 9 downto  9) xor
                 A( 5 downto  5) xor A( 1 downto  1);

B(0 downto 0) <= A(30 downto 30) xor A(28 downto 28) xor A(26 downto 26) xor
                 A(24 downto 24) xor A(22 downto 22) xor A(20 downto 20) xor
                 A(18 downto 18) xor A(16 downto 16) xor A(14 downto 14) xor
                 A(12 downto 12) xor A(10 downto 10) xor A( 8 downto  8) xor
                 A( 6 downto  6) xor A( 4 downto  4) xor A( 2 downto  2) xor
                 A( 0 downto  0);
```

The experimental test of this solution efficiency follows next.

5. Experimental results

In order to evaluate efficiency of encoders for different manufacturers FPGA architectures the corresponding projects have been created in VHDL language in following environments:

- Xilinx ISE Design Suite v. 14.7,
- Altera Quartus v. 13.0 sp1,

for FPGA chips

- Xilinx, Spartan 6, XP6SLX45T-3,
- Altera Cyclone IV E, EP4CE75F23C7,

and compilations have been performed to determine theoretical maximum frequency of design and resources consumption. At the beginning implementations of two encoders with different EBP were compared.

- with binary positional coding (Table 3, expressions 5a-f);
- with Gray coding (Table 4, expressions 6a-f).

Also note, that for input data of encoders (vector *A*) a generator of pseudo-random 31 bit numbers based on LFSR has been used, which takes 9 Slices and must be considered during evaluation of resources consumption of solutions.

The data on maximum frequency (Synthesis Report; TimeQuest Timing Analysis, slow @ 85C) and resources consumed (Map Report; Fitter (Place & Rout)) were obtained after compilation of projects in Table 5. As it can be observed from table, an encoder with Gray coding outperformed encoder with binary positional code both in maximum frequency (381.7 MHz vs 353.2 MHz for Spartan 6 and 519.5 MHz vs 457.5 MHz for Cyclon IV E), and in resources consumed (15 vs 22 for Spartan 6 and 36 vs 42 for Cyclon IV E).

Since encoder with Gray coding shows better results, it has been used for the further comparison with straightforward solution, where constant assignment was chosen to be processing operation.

The results of experiments show that if-constructions implementation in Altera and Xilinx platforms differs greatly, so for Altera projects case-construction has been chosen instead.

In addition to estimates obtained during compilations, the comparative performance specifications of classifier solutions were evaluated in a process of empirical search of maximum tact frequency of real hardware (PCB). There were two testbenches created for these purposes with following PCBs:

- Xilinx SP605,
- Altera board based on Cyclone IV E EP4CE75F23C7.

A testbench includes generator of data pair: input classification value and true number of group. Generator guarantees exhaustive search of all possible input data variations and provides pseudo-random order of these variations. The architecture of this design part is such that fault probability when frequency increases is minimal compared to classifier being researched. Furthermore, generated input value was sent to encoder input and a priori known answer was delayed by a number of cycles required for classifier to process input data. The output of tested device was compared with known number of group. In case of a single mismatch during detection the testing environment set flag indicating impossibility of fault-free operation of such architecture at this specific cycle frequency.

Table 5. Results of conducted experiments

Implementation	Resources consumption.	F_{max}, compilation.	F_{max}, experimental
Spartan 6. Encoder with binary coding.	Number of Slice LUTs: 22	353.170 MHz	—
Spartan 6. Encoder with Gray coding.	Number of Slice LUTs: 15	381.665 MHz	—
Spartan 6. Straightforward solution with post-processing.	Number of Slice LUTs: 129	131.686 MHz	229 MHz
Spartan 6. Encoder with Gray coding and post-processing.	Number of Slice LUTs: 16	349.638 MHz	505 MHz
Spartan 6. Straightforward, reduced.	Number of Slice LUTs: 62	209.087 MHz	—
Cyclon IV E. Encoder with binary coding.	Total logic elements: 42	457.46 MHz	—
Cyclon IV E. Encoder with Gray coding.	Total logic elements: 36	519.48 MHz	—
Cyclon IV E. Straightforward solution with post-processing.	Total logic elements: 97	158.96 MHz	210 MHz
Cyclon IV E. Encoder with Gray coding and post-processing.	Total logic elements: 40	373.69 MHz	400 MHz

Natural experiments of maximum frequency search are rather labor-consuming, so they were performed only for two implementations with post-processing: straightforward and encoder with Gray coding for both platforms (Xilinx and Altera). Since there is obvious correlation between theoretical and practical results, it is expected that in other cases the same match of results was present.

6. Conclusion

In applications with further data processing on FPGA devices, which required performance, reconfiguration of bins system and so on, it is recommended to use computational parallel classifiers or hybrid variation in presence of resources consumption limits. When encoding operation is necessary, aa specific encoding table must be chosen, such as Gray coding table or search for alternative solutions for specific application, considering partial match of Gray codes to optimization requirements.

Analysis of obtained results shows unambiguous efficiency of optimized encoder use for parallel comparators output vector data processing. The obtained solution with maximum operational frequency definitely outperforms straightforward HDL implementation variations on both evaluated platforms, which makes it especially suitable for high rate processing systems. There is also significant improvement observed for resources consumption of PLD implemented encoder. Even though the resources consumed by this part is small relative to those of parallel comparators, for applications with multiple uses of such classifiers the area of chip gained is also considerable advantage of solution proposed.

References

1. Bashkirov, A and Muratov, A. (2012) An advantage of parallel digital signal processing algorithms over a sequential algorithm when implemented on FPGAs. *Vestnik VGTU*, 8(1), 89-92.
2. Flaxer, E. (no date) *VHDL. Chapter 7. Behavioral Modeling.* Aviable at: http://www.tau.ac.il/~flaxer/edu/course/vhdl/slides/VHDL01.pdf (Accessed: 6 July 2016).
3. Gray, F. (1953) *Pulse code communication.* US2632058. (Patent).

4. Kester, W. (2005) *The Data Conversion Handbook.* Oxford: Elsevier. 976 p.
5. Leibson, S. (2014) FPGAs Aid Search for Dark Energy with CHIME Telescope. *XceLL Journal*, (89), 32–37.
6. Murphy, P. and Zhong, L. (2014) FPGAs Help Characterize Massive-MIMO Channels. *XceLL Journal*, (89), 18–25.
7. Ovtcharov, K. *et al*. (2015) *Accelerating deep convolutional neural networks using specialized hardware*. Microsoft Research Whitepaper.
8. Putnam, A. *et al*. (2014) A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. In: *41st Annual International Symposium on Computer Architecture*, June 2014, Minneapolis, MN, USA: IEEE Press Piscataway, pp. 13-24.
9. Shah, A. (2016) *Intel starts baking speedy FPGAs into chips.* Aviable at: http://www.pcworld.com/article/3055526/intel-starts-baking-speedy-fpgas-into-chips.html (Accessed: 10 June 2016).
10. Sisterna, C. (2013) Introduction to VHDL for Implementing Digital Designs into FPGAs (Presentation). *Presented at the International Training Workshop on FPGA Design for Scientific Instrumentation and Computing.* Rome. Available at: http://indico.ictp.it/event/a12223/session/2/contribution/2/material/0 (Accessed: 6 July 2016).
11. Wasson, S. (2015) *ATI's CrossFire dual-graphics solution. Dually Radeons at last.* Aviable at: http://techreport.com/review/8826/ati-crossfire-dual-graphics-solution (Accessed: 5 July 2016).
12. Xilinx (no date) *Coding Style Guidelines.* Available at: https://wiki.electroniciens.cnrs.fr/images/Xilinx_HDL_Coding_style.pdf (Accessed: 6 July 2016).