Implementation of MADM algorithms on FPGA based platforms

Ivan Kanev

Abstract: *MADM algorithms are a version of the ADM method for compression of speech signals. In this paper the implementation of these algorithms on FPGA based platforms is presented. The outputs of the research carried out may be used in the design of IVR or VM systems.*
Key words: *MADM, ADM, FPGA, IVR, VM*

1. INTRODUCTION

As a result of the sampling of a speech signal S(t) by a continuous parameter within a discrete time interval T, a discrete sequence S(nT) (hereafter used as Sn only) is obtained; n, n = 1..N is an integer variable and N is the number of discrete samples needed to represent the speech signal. The relation Fs 1/T defines the sampling frequency. The S(t) => Sn transformation is known as the PCM (Pulse Code Modulation). For the S(t) approximation with regular sampling (Fs = const) L levels are needed. As an immediate output the sampling process a block code of a length of B = $B_{m+1} = \log_2 L$ is obtained; m+1 is the number of the binary digits used for presentation of Sn. Additional operations can be performed on the discrete sequence Sn (compression of Sn) for the purpose of reducing the information redundancy. In this respect a good solution is provided by the Modified Adaptive Delta Modulation algorithms [1] compression of speech signals. MADM algorithms present a version of the method for compression of speech signals ADM [2]-[5]. Defining of MADM (r) algorithms as Moore finite-state automatons, their presentation in terms of tables and on this basis their realisation on FPGA based platforms is the objective of this research work.

2. BASIC DEFINITIONS

When the MADM (r) algorithms are applied on the discrete sequence S_n a new discrete sequence S'_n is obtained. The basis of *MADM(r),* $r = A, B, C$ is provided by the recurrent dependency.

$$S'_n = S'_{n-1} + \alpha_r \delta_{n-1} \tag{1}$$

where:

- δ_n is a relative change of S'_n

$$\delta_n = S'_n - S'_{n-1} \tag{2}$$

- α_r are functions that determine the adaptive character of algorithms

The functions α_r depend on the following conditions:

$$\alpha_A = \begin{Bmatrix} \alpha_+ \text{, if } \sigma_n = Ss_{n-1} \text{ and } |\delta_{n-1}| < \delta_{\max} \\ 1 \text{ , if } \sigma_n = Ss_{n-1} \text{ and } |\delta_{n-1}| = \delta_{\max} \\ \alpha_- \text{, if } \sigma_n \neq Ss_{n-1} \text{ and } |\delta_{n-1}| \neq 1 \\ \text{-1 , if } \sigma_n \neq Ss_{n-1} \text{ and } |\delta_{n-1}| = 1 \end{Bmatrix}, \tag{3}$$

$$\alpha_B = \begin{Bmatrix} \alpha_+ \text{, if } \sigma_n = Ss_{n-1} \text{ and } |\delta_{n-1}| < \delta_{\max} \\ 1 \text{ , if } \sigma_n = Ss_{n-1} \text{ and } |\delta_{n-1}| = \delta_{\max} \\ -\dfrac{1}{|\delta_{n-1}|}\text{, if } \sigma_n \neq Ss_{n-1} \end{Bmatrix}, \tag{4}$$

$$\alpha_C = \begin{cases} \alpha_A \text{ , if } |\delta_{n-1}| \leq J^2 \\ \alpha_B \text{ , if } |\delta_{n-1}| > J^2 \end{cases}, \quad (5)$$

where:

- Ss_n is a variable determine a sign of δ_n:

$Ss_n = sig\delta_n$, $Ss_n = 1,-1$ (6)

- α_+, α_- are coefficients that determine the increase of S'_n:

$\alpha_+ = 2$, $\alpha_- = -\alpha_+^{-1}$; (7)

- $\delta_{\max}$ is coefficient limiting the increase of δ_n:

$\delta_{\max} = 2^k, 1 \leq k$; (8)

- J is coefficient switching A and B methods

$1 < J < k$. (9)

The condition σ_n which serves to define the way of obtaining of adaptive function α, provides two alternatives for operations with the algorithm (1).

2.1 Compression

The operation C performs the algorithm (1) on the discrete sequence S_n, whereupon a new discrete sequence is obtained.

$$S_n \underset{n \in N}{\overset{C}{\Longrightarrow}} Ss_n , \quad (10)$$

having for σ_n a valid dependence

$\sigma_n = sige_{n-1}$, (11)

where:

- e_{n-1} is the error of the transformation $S_n \Rightarrow S'_n$:

$e_{n-1} = S_{n-1} - S'_{n-1}$; (12)

$sige_{n-1}$ is the sign of the error e_{n-1}:

If $e_{n-1} \geq 0$, $\sigma_n = 1$; otherwise $\sigma_n = -1$; (13)

is referred to as ADM compression algorithm.

The operation C transforms the block code S_n, of length B_{m+1} into block code Ss_n of length 1. Therefore the compression with ratio $B_{m+1} : 1$ is obteined. In this case S'_n has a property of intermediate variable.

2.2 Decompressing

The operation D performs the algorithm (1) on the discrete sequence Ss_n, whereupon a new discrete sequence is obtained S'_n.

$$Ss_n \underset{n \in N}{\overset{D}{\Longrightarrow}} S'_n \quad (14)$$

under the condition that for σ_n the following dependency is valid:

$\sigma_n = Ss_n$ (15)

is referred to as decompression.

The operation D transforms the block code Ss_n, of length 1 into block code S'_n of length B_{m+1}. Therefore the decompression with ratio $1 : B_{m+1}$ is obteined and S'_n is restored. The function $\alpha_r \delta_{n-1}$ generate the set:

$$\alpha_r \delta_{n-1} = \{\pm 1, \pm 2, .., \pm 2^{k-1}, \pm \delta_{\max}\}, \quad (16)$$

with the number variable quantities of the set:

$$|A_r| = 2(k+1) \quad (17)$$

3. DEFINING OF *MADM(r)* ALGORITHMS AS A MOORE MACHINE

The Moore machine is defined with six parameters:

$$M = \langle Q, V, W, d, \lambda, q_0 \rangle \quad (18)$$

when

Q - present state alphabet;

V - input alphabet

W - output alphabet

d - function of transitions with domain of definitions $D(d): D(d) \subseteq Q \times V$ and domain of variable quantities. $R(d): R(d) \subseteq Q$.

λ - output function with domain of definitions Q and domain of variable quantities W

$q_0 \in Q$ - initial state

To define Moore finite-state automaton realizing *MADM(r)* algorithms it is necessary to provide definitions for:

3.1 Input alphabet V

If we replace the values of Ss_n (6) and σ_n (13) when they are equal to –1 with 0 then

Compressing

$$V = \begin{Bmatrix} 0, \text{ IF } \sigma_n = Ss_{n-1} \\ 1, \text{ IF } \sigma_n \neq Ss_{n-1} \end{Bmatrix}, \textit{ or} \quad (19)$$

$$V = \sigma_n \oplus Ss_{n-1} \quad (20)$$

Decompressing

$$V = \begin{Bmatrix} 0, \text{ IF } Ss_n = Ss_{n-1} \\ 1, \text{ IF } Ss_n \neq Ss_{n-1} \end{Bmatrix}, \textit{ or} \quad (21)$$

$$V = Ss_n \oplus Ss_{n-1} \quad (22)$$

3.2 Output alphabet W

The output alphabet is define by the elements of set that is generated from function (16) $\alpha_r \delta_{n-1}$:

$$W = \{\pm(2^i)\}, \quad i = 0,1,..,k. \quad (23)$$

3.3 Present state alphabet Q

If $q(+)$ denotes the states that satisfy the condition $\delta_{n-1} > 0$ and with $q(-)$ denotes the states that satisfy the condition $\delta_{n-1} < 0$. Then for alphabet of input state we get:

$$Q = \{q_i(\pm)\}, \quad i = 0,1,..,k. \quad (24)$$

3.4 Output function λ

After definition of V,W and Q *we can get the output function* λ:

$$\lambda(q_i(\pm)) = \{\pm 2^i\}, \; i = 0,1,..,k. \quad (25)$$

3.5 Initial state q_0

Compression

The initial state q_0 for compression depend on the sign of speech signal error σ_n, *n* = 1:

$$q_0 = \begin{Bmatrix} q_0(+), \text{ IF } \sigma_{(1)} = 1 \\ q_0(-), \text{ IF } \sigma_{(1)} = 0 \end{Bmatrix}$$

Decompression

The initial state q_0 for decompression depend on the sign of relative change (6) of speech signal Ss_n, *n* = 0:

$$q_0 = \begin{Bmatrix} q_0(+), \text{ IF } Ss_{(0)} = 1 \\ q_0(-), \text{ IF } Ss_{(0)} = 0 \end{Bmatrix}.$$

3.6 Function of transitions d

The function of transitions can be proved by the following theorems:

Theorem 1.

If the adaptive function α_r is used to implement the operation:

$\alpha_r \lambda(q_i) = \lambda(q_j)$,

only the following transition will be performed

$d(q_i, V) \to q_j$.

Proof.

Suppose that $\alpha_r \neq 0$ and there are such $p, p \neq j$, where:

$\alpha_r \lambda(q_i) = \lambda(q_j) = \lambda(q_p)$.

Therefore two distinct nodes with the same output functions exist in automaton, which opposes to the condition (25) for output function λ.

Then if $\alpha_r \lambda(q_i) = \lambda(q_j)$ is true only the transformation will be done

$d(q_i, V) \to q_j$.

Theorem 2.

The condition for adaptive function

$\alpha_A = \alpha_+, \text{if } \sigma_n = Ss_{n-1} \text{ and } |\delta_{n-1}| < \delta_{\max}$,

defines $2k$ transitions for which the following is valid:

$d(q_{i-1}(\pm), V = 0) \to q_i(\pm), \; 1 \le i \le k$.

Proof

Suppose that the adaptive function α_A(3) consist of i sequential iteration, $i = 1,2,..,k$ and for all i: $V = 0$.

We assume that:

$\delta_{n-1} = \lambda(q_0(\pm)) = \pm 1$.

Then for i:

$i = 1 : \delta_n = \alpha_+ \lambda(q_0(\pm)) = \lambda(q_1(\pm))$,

the following transitions will be performed:

$d(q_0(\pm), V = 0) \to q_1(\pm)$;

$i = 2 : \delta_{n+1} = \alpha_+ \lambda(q_1(\pm)) = \lambda(q_2(\pm))$

the following transitions will be performed:

$d(q_1(\pm), V = 0) \to q_2(\pm)$.

By induction for all i,

$1 \le i < k : \delta_{n+i-1} = \alpha_+ \lambda(q_{i-1}(\pm)) = \lambda(q_i(\pm)$, will be performed $2k$ transitions

$d(q_{i-1}(\pm), V = 0) \to q_i(\pm)$.

Theorem 3

The condition for adaptive function

$\alpha_A = 1$, if $\sigma_n = Ss_{n-1}$ and $|\delta_{n-1}| = \delta_{\max}$,

defines 2 transitions for which the following is valid:

$d(q_k(\pm), V = 0) \to q_k(\pm)$.

Proof

Suppose that the adaptive function α_A(3) consist of i sequential iterations $i = 1,2,..$ and for all i: $V = 0$.

We assume that:

$i = k-1 : \delta_{n+k} = \alpha_+ \lambda(q_{k-1}(\pm)) = \lambda(q_k(\pm))$.

Therefore:

$\lambda(q_k(\pm)) = \delta_{\max}$ и $\alpha_A = 1$

Then for all i,

$i : \delta_{n+i} = \lambda(q_k(\pm))$ two transitions will be done

$d(q_k(\pm), V = 0) \to q_k(\pm)$.

Theorem 4.

The condition for adaptive function

$\alpha_A = \alpha_-$, if $\sigma_n \neq Ss_{n-1}$ and $|\delta_{n-1}| \neq 1$,

defines $2k$ *transitions for which the following is valid:*

$d(q_i(\pm), V = 1) \to q_{i-1}(\mp)$, $1 \le i \le k$

Proof

Suppose that the adaptive function α_A(3) consist of i sequential iterations $i = 1,2,..,k$ and for all i: V = 1.

We assume that:

$$\delta_{n-1} = \lambda(q_k(\pm)) = \pm 2^k$$

Then for i:

$i = 1 : \delta_n = \alpha_- \lambda(q_k(\pm)) = \lambda(q_{k-1}(\mp))$,

the following transitions will be done:

$d(q_k(\pm), V = 0) \to q_{k-1}(\mp)$;

$i = 2 : \delta_{n+1} = \alpha_- \lambda(q_{k-1}(\pm)) = \lambda(q_{k-2}(\mp))$,

the following transitions will be done:

$d(q_{k-1}(\pm), V = 0) \to q_{k-2}(\mp)$.

By induction for all i:

$1 \le i \le k : \delta_{n+i} = \alpha_- \lambda(q_i(\pm)) = \lambda(q_{i-1}(\mp)$ two transitions will be done

$d(q_i(\pm), V = 0) \to q_{i-1}(\mp)$.

Theorem 5.

The condition for adaptive function

$\alpha_A = -1$, if $\sigma_n \neq Ss_n$ and $|\delta_{n-1}| = 1$,

defines 2 transitions for which the following is valid:

$d(q_0(\pm), V = 1) \to q_0(\mp)$.

Proof

Suppose that the adaptive function α_A(3) consist of i sequential iterations $i = 0,1,..$ and for all i: V = 1.

We assume that:

$\delta_{n-1} = \lambda(q_0(\pm) = \pm 1$

Then for all i:

$\delta_{n+i} = -1\lambda(q_0(\pm)) = \lambda(q_0(\mp)$ two transitions will be done

$d(q_0(\pm), V = 1) \rightarrow q_0(\mp)$.

Theorem 6

The condition for adaptive function

$\alpha_B = \frac{-1}{|\delta_{n-1}|}$, if $\sigma_n \neq Ss_{n-1}$,

defines $2k+2$ *transitions for which the following is valid:*

$d(q_i(\pm), V = 1) \rightarrow q_0(\mp)$. $0 \leq i \leq k$.

Proof

Suppose that the adaptive function α_A(4) consist of i sequential iterations $i = 0,1,..,k$ and for all i: V = 1.

We assume that:

$\delta_{n-1} = \pm 2^i = \lambda(q_i(\pm))$.

Then for all i:

$\delta_{n+i} = -\left(\frac{\lambda(q_i(\pm))}{|\lambda(q_i(\pm))|}\right) = \mp 1 = \lambda(q_0(\mp))$ $2k+2$ transitions will be done

$d(q_i(\pm), V = 1) \rightarrow q_0(\mp)$.

4. TABLE REPRESENTATION FOR *MADM(r) ALGORITHMS*

The definition of *MADM(r)* algorithms as a Moore machines can be used for their table representation. The goal of table representation is to describe the automaton as arrays (one dimensional or multi dimensional) Moreover we can use it as basis for creation of algorithmic or hardware solutions independent of automaton configuration. The different configurationsof *MADM(r)* finite automatons can be represented with table using the following rules:

1. *When we have a given k and method r we create a MADM Moore machine(fig. 1)*
2. *We can re-enumerate the present states of automaton* $q_i(\pm)$, $i = 0,1,..k$ *on its internal contour. The choice of initial state and direction for re-enumeration are not important. Assume that for the initial state the first node is chosen* $q_0(+)$ *and direction is clockwise.*

2.1 *Two dimensional array. Lets denote the present state of graph with* $State_j$, $j = 0,1,..,k+1$ *and for the initial state we assign number 0.*

2.2 *One dimensional array. Lets denote the new present state of graph with* $State_j$, $j = 0,2,..,4k+2$ *and for the initial state we assign number 0.*

3. *Lets put together pair of numbers* $\lambda' = \{\lambda_i, State_j\}$ *to each edge of graph, where the first number is formed by output function* λ_i *from node i where the edge come from. And the second number is formed by the state* $State_j$, *where the edge goes in.*
4. *Let us construct two tables* TBL_V0, TBL_V1 where:
   - *TBL_V0 includes all pairs fot which for a certain State, V=0.*
   - *TBL_V1 includes all all pairs fot which for a certain State, V=1.*

The tables TBL_V0 and TBL_V1 represent *MADM(r)* algorithm as arrays of automatons states which is a basis for their realization without the utilization of operations subtraction, addition, multiplication, shift and rotation.

The table representation of *MADM (A)* finite automaton is given in Table 1.

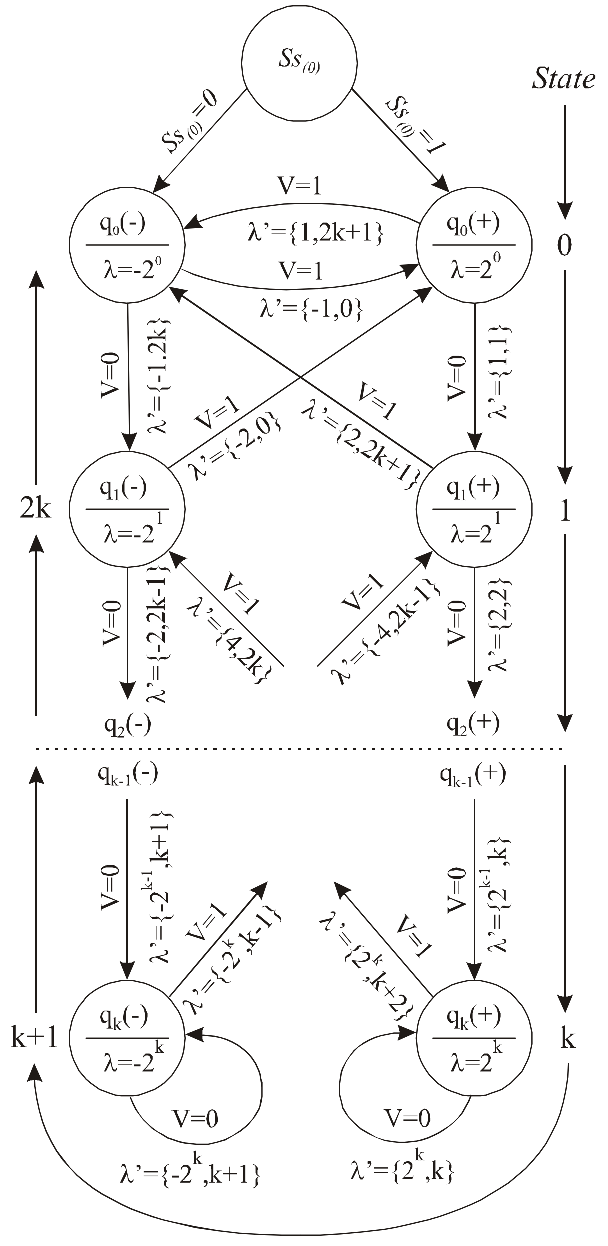

Fig. 1. MADM finite automaton – method "A"

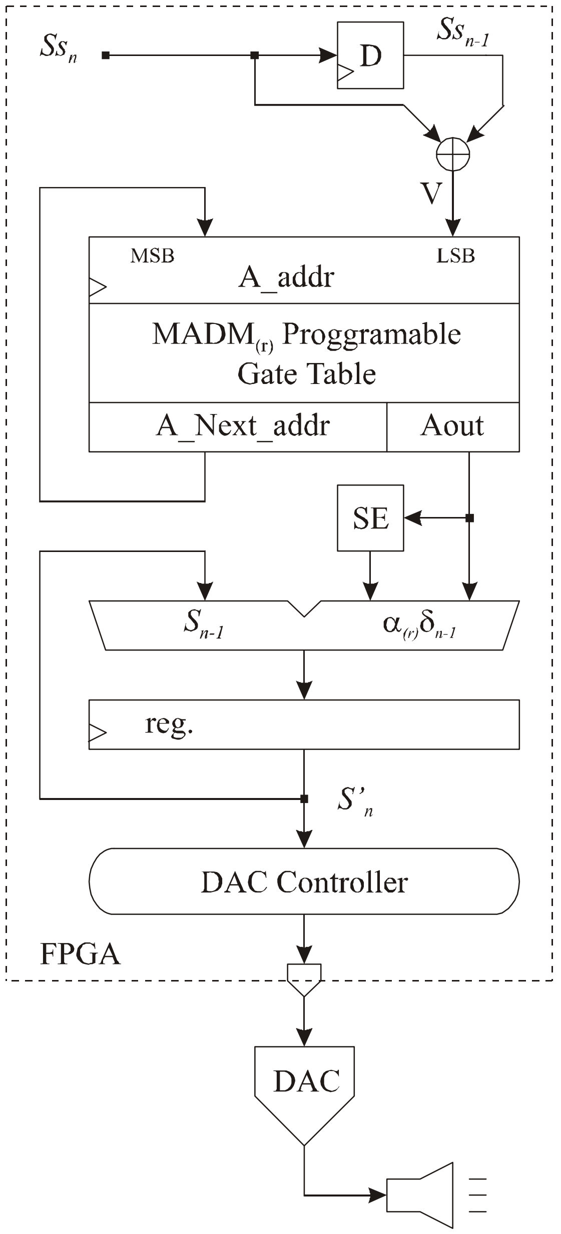

Fig. 2. MADM(r) decoder

q_i	$q_0(+)$	$q_1(+)$	….	$q_k(+)$	$q_k(-)$	….	$q_1(-)$	$q_0(-)$
State	0	1	….	k	k+1	….	2k	2k+1
TBL_V0	1,1	2,2	….	2^k,k	-2^k,k+1	….	-2,2k-1	-1,2k
TBL_V1	1,2k+1	2,2k+1	….	2^k,k+2	-2^k,k-1	….	-2,0	-1,0

Table1. Table representation of *MADM* finite automaton – method *A* .

5. IMPLEMENTATION OF MADM ALGORITHMS ON FPGA BASED PLATFORMS

We can use the table representation of *MADM(r)* finite automatons for their implementation on FPGA based platforms. There are several methods to transform table representation to practice realization. In this paper we present an approach where *MADM(r)* finite-state automatons are realized as a logical circuit with programmable gates array. This approach is appropriate for MADM algorithms with coefficient (8) k that limit the increase of δ_n not higher than 5.

To create a table representation of *MADM(r)* finite automaton as a logical circuit (Table 2) is necessary to define:

A_addr				
State	V	Aout	A_next_addr	Source: Table 1.
000	0	0001	001	q_0(+): *TBL_V0 = 1,1*
000	1	0001	101	q_0(+): *TBL_V1 = 1,2k+1*
001	0	0010	010	q_1(+): *TBL_V0 = 2,2*
001	1	0010	101	q_1(+): *TBL_V1 = 2,2к+1*
010	0	0100	010	q_k(+): *TBL_V0 =* 2^k*,k*
010	1	0100	100	q_k(+): *TBL_V1 =* 2^k*,k+2*
011	0	1100	011	q_k(-): *TBL_V0 =* -2^k*,k+1*
011	1	1100	001	q_k(-): *TBL_V1 =* -2^k*,k-1*
100	0	1110	011	q_1(-): *TBL_V0 = -2,2k-1*
100	1	1110	000	q_1(-): *TBL_V1 = -2,0*
101	0	1111	100	q_0(-): *TBL_V0 = -1,2k*
101	1	1111	000	q_0(-): *TBL_V1 = -1,0*

Table 2. Table representation of MADM finite automaton as logical circuits Method "A"; $k = 2$.

- A_addr - input function. This function is determined as concatenation of $State$ and input alphabet V.

- Aout – output function. Aout is determined from the elements of set, which is created by function (16) $\alpha_r \delta_{n-1}$. For given $State$ and V, this functions is obtained by the first elements of the tables TBL_V0 and TBL_V1. In order to express sign numbers the output function Aout is in complement notation. When adding Aout to S'_n, Aout is sign extended in order to fit the code word of S'_n.

- A_next_addr is determined by the state $State_j$, where edge of automaton comes from. When $State$ and V, are given, this function is determined from the second elements of tables TBL_V0 and TBL_V1.

Fig 2 shows block diagram of $MADM_{(r)}$ decoder, based on logical circuit with programmable gates array.

The following program shows an example of $MADM_{(A)}$ finite automation in VHDL realization.

```
--    VHDL MADM(A) finite automation - method "A"; k=2
library ieee;
use ieee.std_logic_1164.all;
entity Aa_k2_tbl is
 port (
      A_addr      : in  std_logic_vector(3 downto 0);
      Aout        : out std_logic_vector(3 downto 0);
      A_Next_addr : out std_logic_vector(2 downto 0);
      );
end   Aa_k2_tbl ;
architecture rtl of Aa_k2_tbl is
 begin
  process(A_addr) begin
      case A_addr is
            when "0000" => Aout <= "0001"; A_Next_addr <= "001";
            when "0001" => Aout <= "0001"; A_Next_addr <= "101";
            when "0010" => Aout <= "0010"; A_Next_addr <= "010";
            when "0011" => Aout <= "0010"; A_Next_addr <= "101";
            when "0100" => Aout <= "0100"; A_Next_addr <= "010";
            when "0101" => Aout <= "0100"; A_Next_addr <= "100";
            when "0110" => Aout <= "1100"; A_Next_addr <= "011";
            when "0111" => Aout <= "1100"; A_Next_addr <= "001";
            when "1000" => Aout <= "1110"; A_Next_addr <= "011";
            when "1001" => Aout <= "1110"; A_Next_addr <= "000";
            when "1010" => Aout <= "1111"; A_Next_addr <= "100";
            when "1011" => Aout <= "1111"; A_Next_addr <= "000";
            when others => Aout <= "0000"; A_Next_addr <= "000";
      end case;
  end process;
end rtl;
```

CONCLUSIONS

The development of the proposed method can be realized in two directions:

- through the development of methodology for synthesis of finite-state automatons for coefficients $k > 5$
- through the development of methodology for synthesis of a set of finite-state automatons which can change their state dynamically in the process of decompression of different segments of the speech signal.

All the theoretical and practical statements included in this research work are proved through modeling and testing on FRGA Apex 1k-50 of Altera Company. The outputs os this research work can be used in the design of IVR or VM systems.

REFFERENCES

[1] Kanev I. Calculated Aspect of ADM algorithms, J.T.U., Fundamental Sciences and Applications, Vol.10, 2003

[2] Rabiner L., Shafer R., Digital Processing of Speech Signals, Prentice Hall, NJ, 1978.

[3] Rabiner L., Applications of Voice Processing to telecommunications, Proceedings of the IEEE, vol. 82, No 6, June 1994.

[4] Boite R., Kunt M., Treitement de la Parole, PPR, 1986.

[6] Aldajani M. A., Sayed A. H., A Stable Adaptive Structure for Delta Modulation with Improved Performance, Proc. ICASSP, Salt Lake City Utah, May 2001.

[6] Truss J.K., Discrete Mathematics for Computer Scientists, Addison Wesley, 1992.

ABOUT THE AUTOR

Ivan Kanev, Department of Computer Systems, Technical University Sofia – Branch Plovdiv Phone +359 32 659 704, E-mail: ikanev@tu-plovdiv.bg.