UNIWERSYTET TECHNOLOGICZNO-PRZYRODNICZY
IM. JANA I JĘDRZEJA ŚNIADECKICH W BYDGOSZCZY
ZESZYTY NAUKOWE NR 253
TELEKOMUNIKACJA I ELEKTRONIKA 12 (2009), 57-71

PARALLEL 4X4 TRANSFORM ON BIT – SERIAL SHARED MEMORY ARCHITECTURE FOR H.264/AVC

Grzegorz Rubin

Department of Computer Science
Bialystok University of Technology
Bialystok, Poland
gregor@wi.pb.edu.pl

Summary. The aim of this paper is to present an implementation and simulation of parallel 4x4 transform on bit-serial shared memory architecture for H.264/AVC. Compared with the existing parallel implementations, the proposed architecture reduces interconnection resources of physical elements of FPGA device. The results of simulation show that the transform can be realized in real–time on bit–serial arithmetic. The paper concludes with a summary.

Keywords: FPGA, shared memory, video coding

1. INTRODUCTION

H.264/AVC is the latest standard of video coding for applications used in mobile devices. There are few known ways of its implementations [1, 2, 3]. Reduction of power consumption or increasing an image quality and real-time processing are the main goals for portable multimedia devices. Therefore, H.264/AVC can be used for video coding in wireless video applications if such requirements will be fulfilled.

Long-range bit-parallel data links provide high data rates at the cost of large chip area, routing difficulty, noise and power [4]. Additionally, such links are often utilized only a small portion of the time, but dissipate leakage power at all times. Parallel link performance is bounded by available clock rate and by clock skew, delay uncertainty due to process variations, cross-talk noise, and layout geometries. There is an alternative to bit-parallel interconnects, mitigating the issues of area and leakage power, when bit-serial few wires are used. However, to provide the same throughput as an N-bit parallel interconnect, the serial link must operate N times faster. For bit-serial implementation, H.264/AVC has relatively higher complexity than other video standards, which might result in increase of power consumption and difficulty of real-time processing.

In this paper, a new parallel 4x4 transform architecture based on bit-serial shared memory architecture is proposed to improve processing rate for H.264/AVC and reduce power consumption. This paper is organized as follows. H.264 transform algorithms are described in section 2. The new architecture and scheduling method by special toolbox

are presented in section 3. Simulation and implementation results are shown in section 4. Finally, conclusion is given in section 5.

2. 2-D INTEGER TRANSFORM IN H.264

The H.264/AVC standard is known as: ISOMPEG4 Part 10, ITU-T H.264, and the Advanced Video Coding (AVC)[5]. This paper discusses the Residual Transform (RT) block, although the standard specifies a complete decoder. Both, the encoder and the decoder use this block, and perform a transform on a macro block level. The transform block in H.264/AVC is one of the key components and there are several aspects of its design that are considered[1]. First, the transform is orthogonal, separable, low gain, and has a strong decorrelating performance. Second, the transforms are performed in integer arithmetic, because floating point arithmetic is harder to implement in hardware. Third, the transform block is designed to reduce the memory access and computation overhead. There are three different transforms used in H.264/AVC, one for 4x4 DC luma coefficients, another for 2x2 chroma DC coefficients and a third for all other 4x4 residual data [3]. It has been shown through complexity analysis [4] that the 4x4 residual data transform takes up majority of computation, therefore it will be used as an example in this paper. However, the proposed architecture can be used to implement other transforms. The ability of adapting proposed designs to incorporate other calculations will be discussed. The authors of H.264/AVC start with a well known two dimensional Discrete Cosine Transform (DCT). This transform can be represented by:

$$Y = AXA^T = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} [X] \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix} \tag{1}$$

where:

$$a = \frac{1}{2}, b = \sqrt{\frac{1}{2}} \cos\left(\frac{\pi}{8}\right), c = \sqrt{\frac{1}{2}} \cos\left(\frac{3\pi}{8}\right)$$

Then matrix equation (1) can be factorized into equivalent form:

$$\boldsymbol{Y} = \left(\boldsymbol{CXC}^T\right) \otimes \boldsymbol{E} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \boldsymbol{d} & -\boldsymbol{d} & -1 \\ 1 & -1 & -1 & 1 \\ \boldsymbol{d} & -1 & 1 & -\boldsymbol{d} \end{bmatrix} [X] \begin{bmatrix} 1 & 1 & 1 & \boldsymbol{d} \\ 1 & \boldsymbol{d} & -1 & -1 \\ 1 & -\boldsymbol{d} & -1 & 1 \\ 1 & -1 & 1 & -\boldsymbol{d} \end{bmatrix} \right) \otimes \begin{bmatrix} \boldsymbol{a}^2 & \boldsymbol{ab} & \boldsymbol{a}^2 & \boldsymbol{ab} \\ \boldsymbol{ab} & \boldsymbol{b}^2 & \boldsymbol{ab} & \boldsymbol{b}^2 \\ \boldsymbol{a}^2 & \boldsymbol{ab} & \boldsymbol{a}^2 & \boldsymbol{ab} \\ \boldsymbol{ab} & \boldsymbol{b}^2 & \boldsymbol{ab} & \boldsymbol{b}^2 \end{bmatrix} \tag{2}$$

Where E is matrix scaling factors and the symbol $\otimes$ indicates that each element of CXC^T is multiplied by the scaling factor in the same position in matrix E. Values a and b are the same as listed in (1) and d=c/b.

More simply implementation of the transform can be done, when we modify constants like this:

$$a = \frac{1}{2}, b = \sqrt{\frac{2}{5}}, d = \frac{1}{2}$$

Therefore final forward transform becomes:

$$\boldsymbol{Y} = \left(\boldsymbol{CXC}^{T}\right) \otimes \boldsymbol{E} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & 2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} [X] \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} \boldsymbol{a}^2 & \frac{\boldsymbol{ab}}{2} & \boldsymbol{a}^2 & \frac{\boldsymbol{ab}}{2} \\ \frac{\boldsymbol{ab}}{2} & \frac{\boldsymbol{b}^2}{4} & \frac{\boldsymbol{ab}}{2} & \frac{\boldsymbol{b}^2}{4} \\ \boldsymbol{a}^2 & \frac{\boldsymbol{ab}}{2} & \boldsymbol{a}^2 & \frac{\boldsymbol{ab}}{2} \\ \frac{\boldsymbol{ab}}{2} & \frac{\boldsymbol{b}^2}{4} & \frac{\boldsymbol{ab}}{2} & \frac{\boldsymbol{b}^2}{4} \end{bmatrix} \tag{3}$$

CXC^T part is the core of transform and can be carried out with integer arithmetic using only additions, subtractions and shifts. The last operation $\otimes E$ requires multiplication, which can be done into the quantization process.

Several hardware design methods for the implementation of the 2-D integer transform have been developed in recent years. In this paper an architecture which performs one dimensional transform on a column of input data by a matrix multiplication architecture was used [2] as shown in Fig. 2. Figure 1 and formula (4) show 1-D transform and that operation needs to be performed four times along the vertical dimension and four times along the horizontal dimension on X. Each of these eight 1-D column/row transforms requires four adders and four subtractors. In the paper [3] with this novel architecture it is possible to perform the whole DCT in one cycle, where the cycle duration is close to a carry propagation through a 64–bit adder. The large amount of hardware used is the drawback of this design, which totals 32 16–bit adders and 32 16–bit subtractors. This amount of hardware can be reduced by introducing various pipeline splits. An addition of a register stage in the middle of the combinational. This papers based on given in figure 2 transform what is detailed in[3], but hardware implementation uses shared memory architecture approach. Moreover, proposed approach uses bit-serial arithmetic and synchronous calculations.

$$\begin{aligned} x_0^{'} &= (x_0 + x_3) + (x_1 + x_2), \\ x_1^{'} &= (x_0 - x_3) \cdot 2 + (x_1 - x_2), \\ x_2^{'} &= (x_0 + x_3) - (x_1 + x_2), \\ x_3^{'} &= (x_0 - x_3) + (x_1 - x_2) \cdot 2, \end{aligned} \tag{4}$$

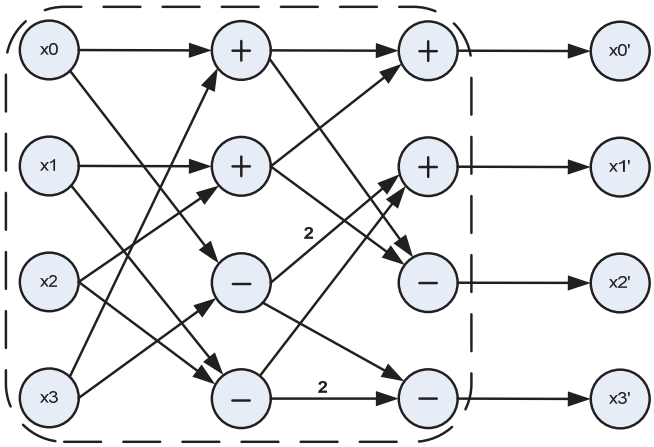

Fig. 1. 1-D transform
Rys. 1. Przekształcenie 1-wymiarowe

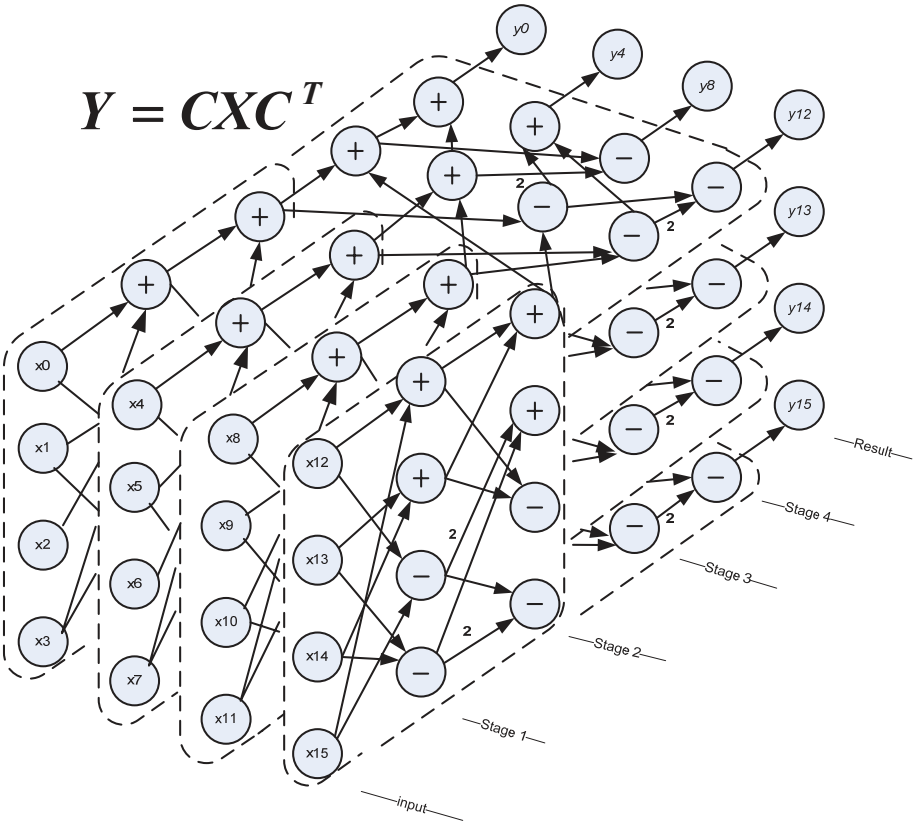

Fig. 2. Parallel 2-D transform
Rys. 2. Równoległe przekształcenie 2-wymiarowe

3. PROPOSED BIT-SERIAL ARCHITECTURE

In the following subsections two usages of hardware architectures of the transform block are presented. The first set of designs is concentrated on unit of bit-serial shared memory architecture, and the second set of designs concentrate on parallel usage of bit–serial blocks. In addition, the description of the two approaches could be useful for designing other area and timing optimized designs.

3.1. SHARED MEMORY ARCHITECTURE

Shared-memory architecture approach (SMA) is detailed in [6]. Figure 3 shows proposition of that architecture. The idea is very simple. In order to simultaneously provide the PEs (Processing Elements) with input data, the shared memory is partitioned into blocks. PEs usually perform simple memoryless mapping of the input values to a single output values. Using a rotating access scheme, each processor gets access to the memories once per N (N – number of PE's) cycles. During this time, processor either writes or reads data from memory. All processors have the same duration time slot to access to the memories and access conflict is completely avoided. Examples of usage and algorithm implementations can be found in [7]. Presented paper presents some modifications of architecture. Separate input and output registers were replaced by registers with input/output interface. There is possible to save result of serial calculation into free cells of input registers during shifting data into processing elements. The second modification is 16–bit width of data extension.

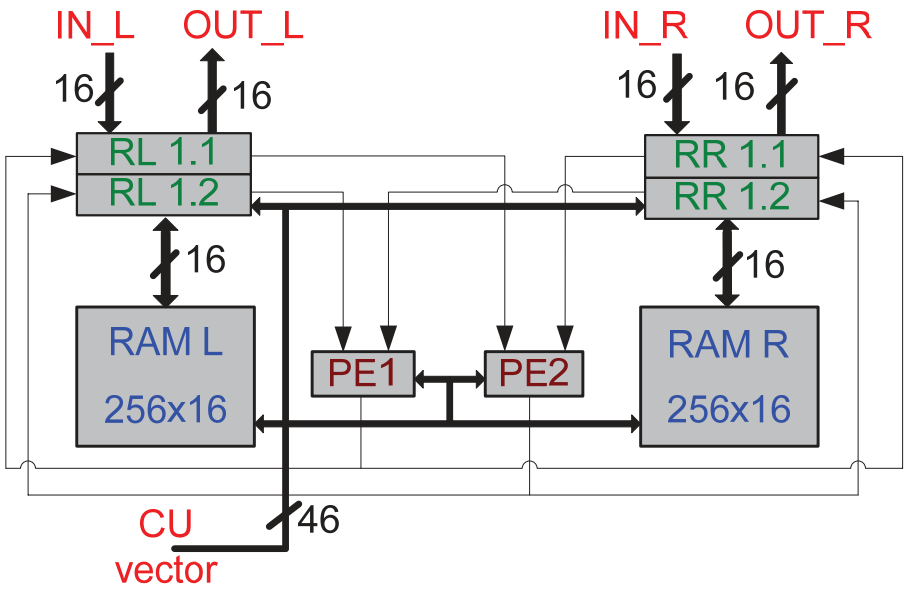

Fig. 3. Shared Memory Architecture unit
Rys. 3. Jednostka Architektury o współdzielonej pamięci

3.2. PARALLEL FORM OF SMA

One SMA unit can calculate the algorithm witch depends on "CU vector" bus state. Implementation of integer residual transform on one SMA requires additions and multiplications. Because of multiplier value on bit-serial arithmetic shifting can be replaced against multiplication. For each block of the 2-D 4x4 integer transform the same calculations has to be done. Therefore, we can repeat the same operations eight times on the same SMA unit or take parallel connection form of few identical SMA blocs with the same dynamic instructions of scheduling (Fig.4). Note that, to save input pins in physical device we must share input lines for data and control vectors. The input data are coming in 16-bit parallel form step by step, but calculations in 16-bit serial form. H.264/AVC supports 8-bit residual pixel data, but it is likely that this will be extended in the future [2]. The time period between two next data depends on bit-serial calculations. Proposed approach of calculations takes 879 steps for one block. During that we can use input pins for data of next 4x4 block, but we can't use input of control vector. Otherwise the control vector is the same for each block, but if we want to use it for parallel calculations, delay by few steps should be applied according to proper input data. One of possibilities based on additional instruction register for every SMA block. Then delayed input of control vector can be applied and the additional set of data calculated.

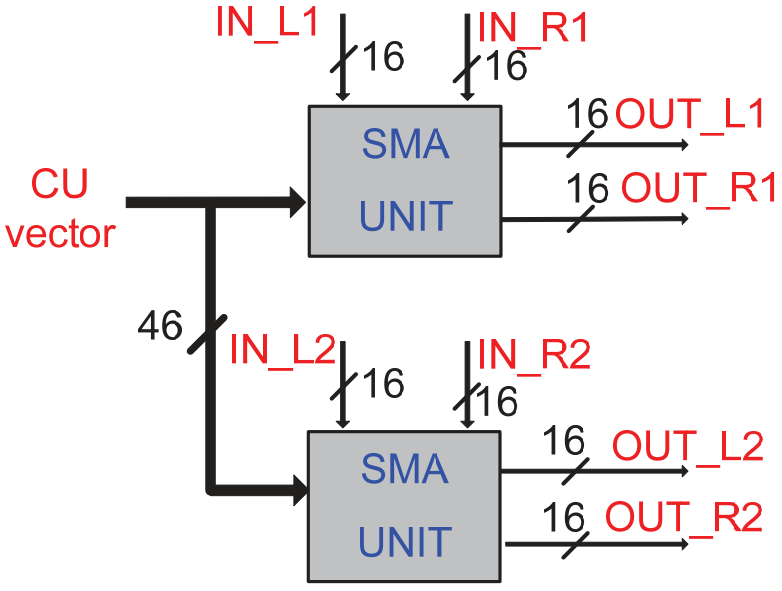

Fig. 4. Two parallel SMA units in the same operating mode
Rys. 4. Dwie równoległe jednostki SMA w jednakowym trybie pracy

We can use set of SMA blocks, but note that, the number of blocks can't exceed resources of physical device and steps of calculation of the first SMA unit. Figure 5 shows proposition of three SMA connected architecture.

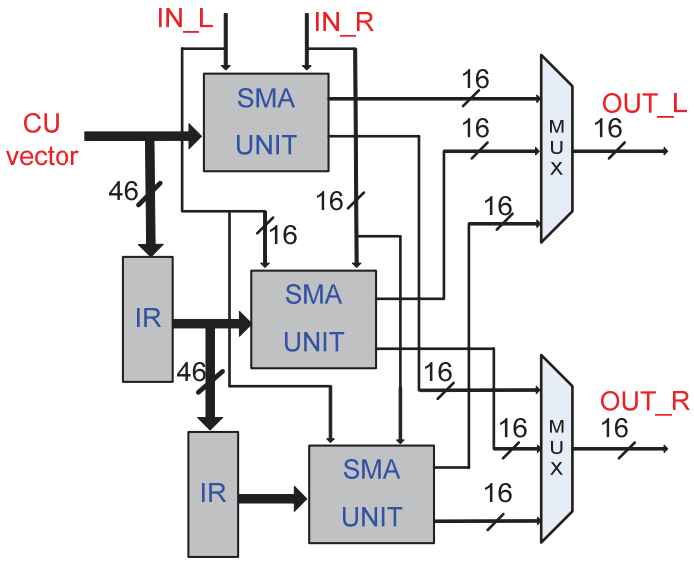

Fig. 5. Three parallel SMA units in the same operating mode with sharing input/ output pins

Rys. 5. Trzy równoległe jednostki SMA w jednakowym trybie pracy ze współdzielonymi pinami wejścia/wyjścia

3.3. SMA TOOLBOX FOR CONTROL UNIT

For fast and efficient scheduling of SMA elements an application was designed. Using that tool the designer can very quickly generate control unit vectors for any algorithm realized on proposed SMA architecture. That's very helpful during simulation process, because of width of 46-bit vector. It's a big probability of setting wrong values into the control vectors witch enables and runs elements of SMA architecture, especially in bit serial arithmetic. Proposed architecture was designed that it's possible to run every part inside the SMA in any time period, therefore the user decide of proper scheduling of algorithm realization. Such architectures are dynamic reconfigurable, so modification of control vector can change scheduling of SMA unit and another algorithm can be realized without any physical changes. The interface of SMA toolbox is presented on figure 6. Upper right corner consist scheme of used architecture for scheduling. Chosing „operation" IN or OUT the left side boxes are enabled or disabled, then DATA IN or DATA OUT options are available to set. There is also possibility to load or save the results of scheduling.

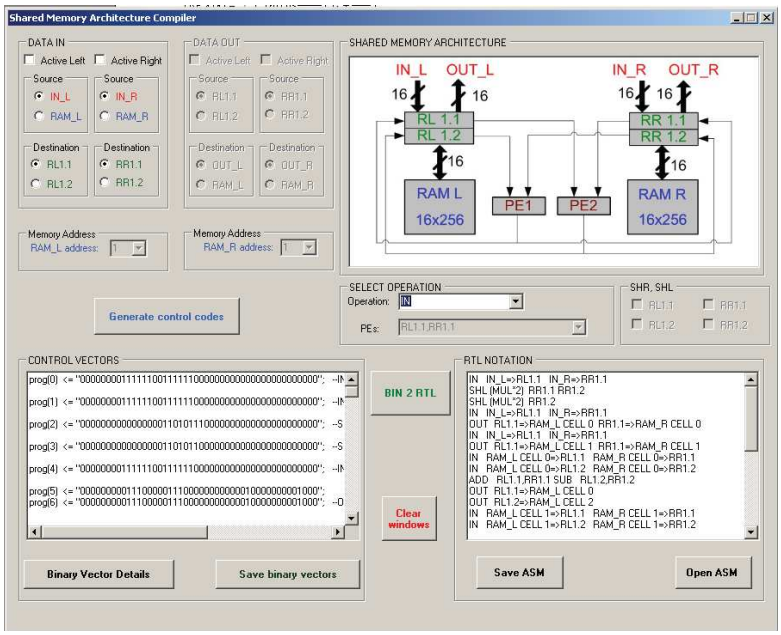

Fig. 6. SMA compiler interface
Rys. 6. Interfejs kompilatora SMA

Proposed SMA compiler is very simple at this time. There are two ways of programming required control vectors – by setting each step using buttons and checkboxes or writing by text in RTL notation. Then just clicking "Generate control vector" in the control vectors window we can see expected set of bits. For example input two set of data, add them together and put it out, RTL notation is as follow:

```
IN    IN_L=>RL1.1    IN_R=>RR1.1
ADD   RL1.1,RR1.1
OUT   RL1.1=>OUT_L   RR1.1=>OUT_R
```

and control vector listed bellow:

```
prog(0)  <= "00000000111111001111110000000000000000000000000";
--IN   IN_L=>RL1.1   IN_R=>RR1.1
prog(1)  <= "10000000000000000000000000000000000000000000000";
--ADD   RL1.1, RR1.1
prog(2)  <= "00100000010000010100000100000000000000000000000";
prog(3)  <= "00100000010000010100000100000000000000000000000";
prog(4)  <= "00100000010000010100000100000000000000000000000";
prog(5)  <= "00100000010000010100000100000000000000000000000";
prog(6)  <= "00100000010000010100000100000000000000000000000";
prog(7)  <= "00100000010000010100000100000000000000000000000";
prog(8)  <= "00100000010000010100000100000000000000000000000";
prog(9)  <= "00100000010000010100000100000000000000000000000";
```

```
prog(10) <= "00100000010000010100000100000000000000000000";
prog(11) <= "00100000010000010100000100000000000000000000";
prog(12) <= "00100000010000010100000100000000000000000000";
prog(13) <= "00100000010000010100000100000000000000000000";
prog(14) <= "00100000010000010100000100000000000000000000";
prog(15) <= "00100000010000010100000100000000000000000000";
prog(16) <= "00100000010000010100000100000000000000000000";
prog(17) <= "00100000010000010100000100000000000000000000";
prog(18) <= "00100000010000010100000100000000000000000000";
prog(19) <= "00000000111111011111110100000000000000000000";
--OUT  RL1.1=>OUT_L  RR1.1=>OUT_R
```

Given example shows that we have strict rules of Processing Elements usage. They can work together at the same time or one do nothing. That limitations are only because of application and SMA architecture can do independent operations, but scheduling must be done manually.

3.4. CONTROL VECTORS FOR 4-POINT 2-D TRANSFORM

For proper work of any algorithm, set of control vectors are required. Using SMA compiler it's possible to generate set of bit vectors for 4-point 2-D transform presented in Figure 2. The SMA unit is able to calculate arithmetic operations such as adding, subtracting, multiply and negation. Additionally it can store some temporary results of any calculation using two separate memory blocks. Therefore SMA unit can by applied for any algorithm using proper control vectors. Because of paper space only the RTL notation for input data and first part of calculations for residual transform is given bellow:

- read input data
```
 IN IN_L=>RL1.1 IN_R=>RR1.1
```
- save input data into RAM L and R - address 0
```
 OUT RL1.1=>RAM_L CELL 0 RR1.1=>RAM_R CELL 0
```
- read input data
```
 IN IN_L=>RL1.1 IN_R=>RR1.1
```
- save input data into RAM L and R - address 1
```
 OUT RL1.1=>RAM_L CELL 1 RR1.1=>RAM_R CELL 1
```
- read input data
```
 IN IN_L=>RL1.1 IN_R=>RR1.1
```
- save input data into RAM L and R - address 2
```
 OUT RL1.1=>RAM_L CELL 2 RR1.1=>RAM_R CELL 2
```
- read input data
```
 IN IN_L=>RL1.1 IN_R=>RR1.1
```
- save input data into RAM L and R - address 3
```
 OUT RL1.1=>RAM_L CELL 3 RR1.1=>RAM_R CELL 3
```
- read input data
```
 IN IN_L=>RL1.1 IN_R=>RR1.1
```
- save input data into RAM L and R - address 4
```
 OUT RL1.1=>RAM_L CELL 4 RR1.1=>RAM_R CELL 4
```
- read input data
```
 IN IN_L=>RL1.1 IN_R=>RR1.1
```
- save input data into RAM L and R - address 5
```
 OUT RL1.1=>RAM_L CELL 5 RR1.1=>RAM_R CELL 5
```

- read input data
```
IN IN_L=>RL1.1 IN_R=>RR1.1
```
- save input data into RAM L and R - address 6
```
OUT RL1.1=>RAM_L CELL 6 RR1.1=>RAM_R CELL 6
```
- read input data
```
IN IN_L=>RL1.1 IN_R=>RR1.1
```
- save input data into RAM L and R - address 7
```
OUT RL1.1=>RAM_L CELL 7 RR1.1=>RAM_R CELL 7
```

- the first 1-D transform
```
IN RAM_L CELL 0=>RL1.1 RAM_R CELL 0=>RR1.1
IN RAM_L CELL 0=>RL1.2 RAM_R CELL 0=>RR1.2

ADD RL1.1,RR1.1 SUB RL1.2,RR1.2
OUT RL1.1=>RAM_L CELL 8
OUT RL1.2=>RAM_L CELL 9
IN RAM_L CELL 1=>RL1.1 RAM_R CELL 1=>RR1.1
IN RAM_L CELL 1=>RL1.2 RAM_R CELL 1=>RR1.2
ADD RL1.1,RR1.1 SUB RL1.2,RR1.2
OUT RR1.1=>RAM_R CELL 8
OUT RR1.2=>RAM_R CELL 9
IN RAM_L CELL 8=>RL1.1 RAM_R CELL 8=>RR1.1
IN RAM_L CELL 8=>RL1.2 RAM_R CELL 8=>RR1.2
ADD RL1.1,RR1.1 SUB RL1.2,RR1.2
OUT RL1.1=>RAM_L CELL 10
OUT RL1.2=>RAM_L CELL 11
IN RAM_L CELL 9=>RL1.1 RAM_R CELL 9=>RR1.1
SHL (MUL*2) RL1.1
IN RAM_L CELL 9=>RL1.2 RAM_R CELL 9=>RR1.2
SHL (MUL*2) RL1.2
ADD RL1.1,RR1.1SUB RL1.2,RR1.2
OUT RL1.1=>RAM_L CELL 12
OUT RL1.2=>RAM_L CELL 13
```

Presented transformation uses 8 identical programmed operations (1-D residual transform), only memory addresses for temporary data differs. One operations takes 113 synchronous steps for 16-bit serial arithmetic. In this paper proposed approach uses one SMA unit for one 4x4 set of data and calculates coefficient during 879 synchronous steps. As was written previously for the test, two parallel connected SMA units were used in Xilinx ISE simulator. Both have the same control vector and separate input/output pins, so the units work together for two sets of data simultaneous. Proposed SMA units can be connected in cascade or parallel form and number of units is limited only by physical programmable device. Such approach allows for better scheduling of algorithm, for example simulated transformation can be calculated by eight SMA units for one 4x4 block. According to Figure 2, firstly compute 4 of 1-D transformation in parallel form, secondly next 4 computation. Then the result will be after 226 steps. Flexible scheduling allows for many combinations of calculations.

3.5. SCHEDULING USING PETRI NETS

Scheduling of parallel processes can be done based on Petri Nets theory. There are few know examples of that approach [8, 9, 10]. For proposed approach based on shared memory architecture an application was desing. Presented environment allows on graphical designing and simulation of algorithms step by step. Formal analysis, possibility of hierarchical designing and simulation are allowed too. That is useful for error correction by simulation of each part of designed algorithms. As the result of simulation of presented graph on figure 7 corresponding to SMA unit for bit-serial calculations we have control vectors for FPGA device. It works similar to previous toolbox described in section 3.3.

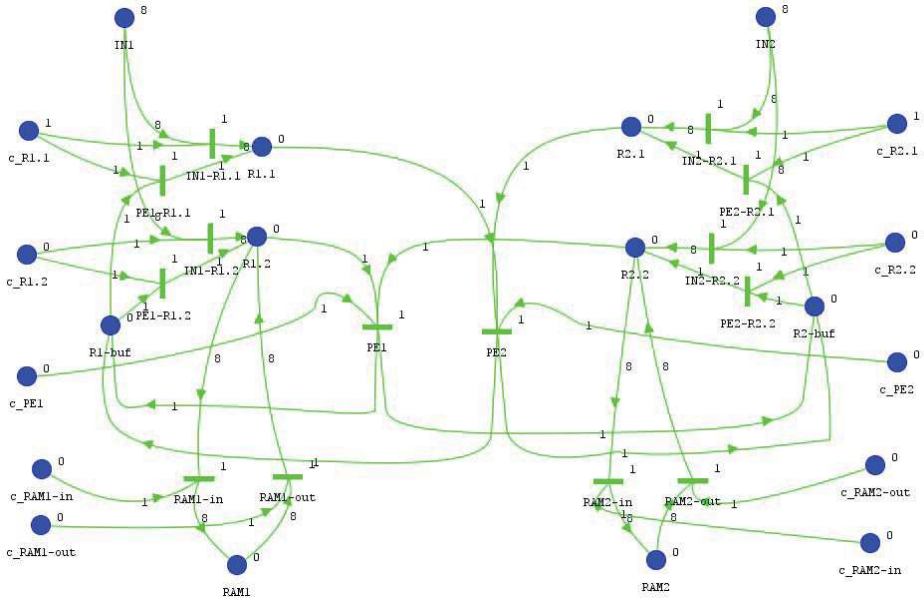

Fig. 7. Petri Net for SMA unit
Rys. 7. Sieć Petriego przedstawiająca jednostkę SMA

Proposed approach allows for deadlock prevention in parallel processes. Every transition corresponds to enable signal in physical device. Figure 8 presents piece of simulation result. Two processing elements works together but PE1 is delayed by two steps. Moreover there is possibility to design hierarchical structure of the net. Every transition and place can be design as the subnet. Simulation of hierarchical structure is also possible. The control vectors of simulation result can be written as a text file, then control vectors can be loaded to Xilinx simulator for FPGA device simulation. Such approach allows for better scheduling because of running every elements of architecture as fast as it's possible, when proper data are ready for processing.

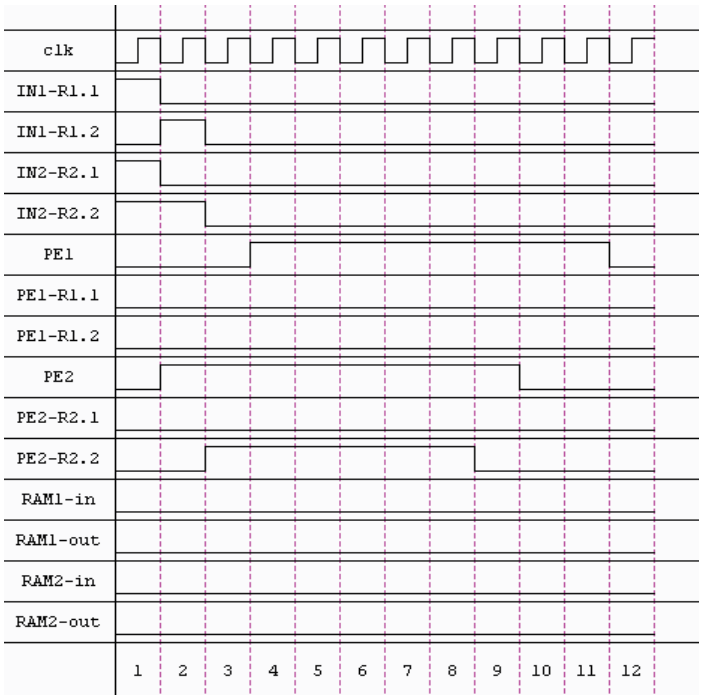

Fig. 8. Piece of simulation of Petri Net for SMA unit

4. IMPLEMENTATION AND SIMULATIONS RESULTS

The transformations were synthesized with Xilinx Project Navigator 10.1 for Virtex II xc2v3000. Top design is schematic (Fig. 9) and realized according with proposition on Fig. 4. Two blocks of data:

$$X_1 = \begin{bmatrix} 1 & 6 & 11 & 16 \\ 2 & 7 & 12 & 17 \\ 5 & 10 & 15 & 20 \\ 3 & 8 & 13 & 18 \end{bmatrix} \qquad X_2 = \begin{bmatrix} 255 & 226 & 225 & 226 \\ 224 & 225 & 225 & 226 \\ 224 & 225 & 226 & 226 \\ 224 & 225 & 225 & 228 \end{bmatrix}$$

were set as input in Xilinx ISE Simulator and the result was purchased:

$$Y_1 = \begin{bmatrix} 164 & -140 & 0 & -20 \\ -28 & 0 & 0 & 0 \\ -12 & 0 & 0 & 0 \\ 16 & 0 & 0 & 0 \end{bmatrix} \qquad Y_1 = \begin{bmatrix} 3605 & -18 & 1 & -9 \\ -1 & 15 & -3 & 0 \\ 3 & 0 & 3 & -5 \\ 2 & 5 & -4 & 5 \end{bmatrix}$$

The "controlUnit" block gets control vectors from the text file which was generated by SMA compiler, reads line by line and puts the 46-bit vector into "C_Signal" of "arch" units. There is only one main clock signal for SMA architecture.

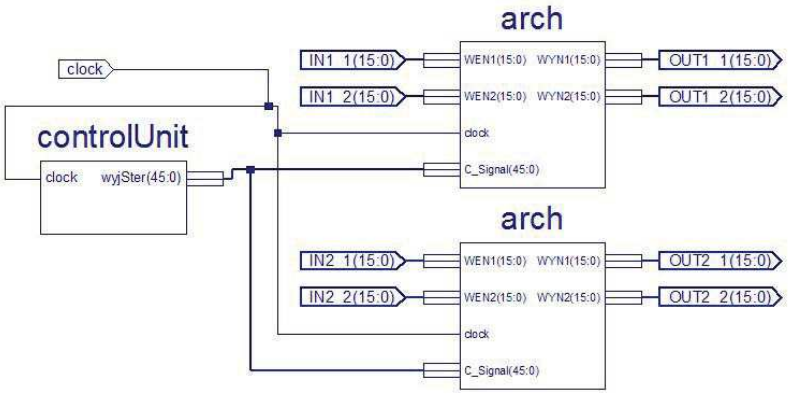

Fig. 9. Parallel form of SMA units
Rys. 9. Połączenie równoległe jednostek SMA

Simulation results shows that SMA units works as were programmed. Figures 10,11 shows start and the end of simulation process.

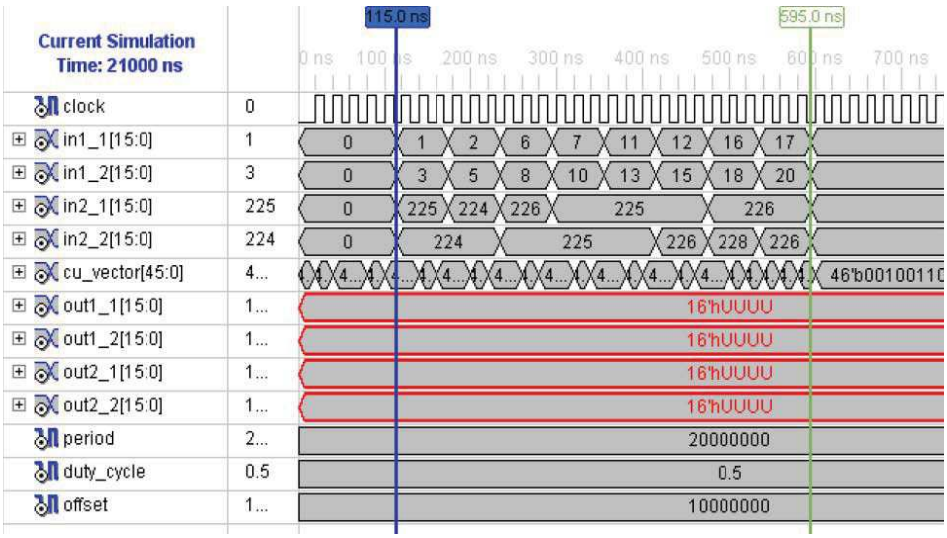

Fig. 10. Begin of simulation process
Rys. 10. Początek procesu symulacji

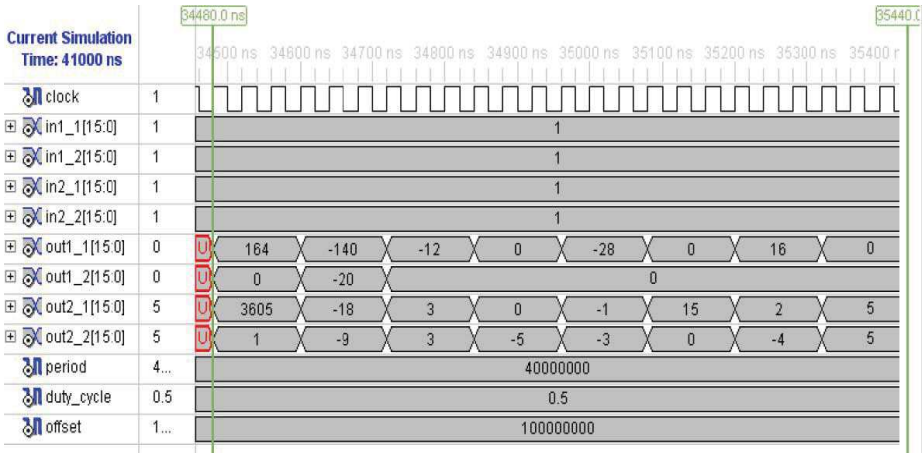

Fig. 11. End part of simulation process
Rys. 11. Końcowy fragment procesu symulacji

Table 1. Device utilization summary
Tabela 1. Podsumowanie wykorzystania zasobów układu

Number of Slices:	1243 out of 14336	8%
Number of Slice Flip Flops:	578 out of 28672	2%
Number of 4 input LUTs:	2192 out of 28672	7%
Number of bonded IOBs:	173 out of 684	25%
Number of 4 BUFGMUXs:	1 out of 16	6%

The same results of simulations using control vectors from SMA toolbox and Petri Nets approach are the same, but the second case allows for clock steps reduction. For better comparison of effectiveness for parallel processing, more tests will be done in the future work.

5. SUMMARY

In this paper, a new parallel 4x4 transform architecture based on bit-serial shared memory architecture was shown to improve processing rate for H.264/AVC and reduce power consumption. Reducing the power consumption is achieved by bit-serial communications as an alternative to bit-parallel interconnects. Simulations results of proposed SMA approach shows that calculations for one 4x4 set of data can be done in 879 steps of one SMA unit. Real time processing of FPGA device working with 100 MHz frequency, with implemented SMA unit can calculate 113705 of 4x4 blocks with 16-bit data. For video frame size 176x144 (3G standard) we have 1584 of 4x4 blocks. One SMA unit can calculate transformation with speed about 71 fps, so for 30 fps we can lower the clock rate to 50MHz. Applying proposed SMA approach for higher resolutions we need more SMA units connected in parallel form. For example 640x480 frames for 30 fps residual transform requires device of 100 MHz clock rate and 5 SMA units connected in parallel form.

BIBLIOGRAPHY

[1] H.S. Malvar, A. Hallapuro, M. Karaczewicz, L. Kerofsky, 2003: Low-Complexity Transform and Quantization in H.264/AVC. IEEE Trans. Circuits Syst Video Technol., vol. 13, no. 7.
[2] E. Hong, E. Jung, H. Fraz, D. Har, 2005: Parallel 4×4 transform architecture based on bit extended arithmetic for H.264/AVC. Proc. Int. Sym. On Circuits and Systems, vol. 1, pp. 95- 98.
[3] R. Kordasiewicz, S. Shirani, 2007: On Hardware Implementations Of DCT and Quantization Blocks for H.264/AVC. Journal of VLSI Signal Processing 47, pp. 189-199.
[4] R. Dobkin, A. Morgenshtein, A. Kolodny, R. Ginosar, 2008: Parallel vs. serial on-chip communication. Proc. of the 2008 International Workshop on System-Level Interconnection Prediction, NewCastle, pp. 43-50.
[5] ITU-T Rec. H.264/ISO/IEC 11496-10, 2002: Advanced video coding. Final Committee Draft, Document JVT-G050, December.
[6] L. Wanhammar, 1999: DSP integrated circuits. Academic Press, USA.
[7] G. Rubin, M. Omieljanowicz, A. Petrovsky, 2007: Reconfigurable FPGA-based hardware accelerator for embedded DSP," MIXDES'2007, Ciechocinek, pp.147-151.
[8] M. Adamski, M. Węgrzyn, 1994: Hierarchically Structured Colored Petri Net Specification and Validation of Concurent Controllers. Proc. in 39th International Scientific Colloquium, IWK'94, Ilmenau, Germany, Band 1, pp. 517-522.
[9] A. Węgrzyn, 2003: The symbolic analysis of binary control units using given methods of Petri nets. Rozprawa doktorska, Politechnika Warszawska (in Polish).
[10] А.А. Петровский, 1988: Techniques and microprocessors tools of fast and wideband processing of real-time processors, Наука и Техника, Минск (in Russian).

RÓWNOLEGŁE PRZEKSZTAŁCENIE 4X4 NA BITOWO-SZEREGOWEJ ARCHITEKTURZE O WSPÓŁDZIELONEJ PAMIĘCI DO ZASTOSOWAŃ H.264/AVC

Streszczenie

Praca zawiera opis implementacji oraz symulacji równoległego przekształcenia 4x4 stosowanego w H.264/AVC, bazując na bitowo-szeregowej architekturze o współdzielonej pamięci. W porównaniu z istniejącymi rozwiązaniami implementacji równoległej, proponowana architektura obliczeniowa redukuje liczbę linii połączeń wewnętrznych fizycznego układu FPGA. Zawiera ona również wyniki symulacji, pokazujące możliwość wykonywania przekształcenia w czasie rzeczywistym, przy zastosowaniu arytmetyki szeregowej.

Słowa kluczowe: FPGA, pamięć współdzielona, kodowanie wideo