A HIGH-SPEED PATTERN DECODER IN MPEG-4 PADDING BLOCK HARDWARE ACCELERATOR

Hyeon-Cheol Mo, Jong-Sun Kim, and Lee-Sup Kim
Department of EECS, KAIST,
373-1 Kusong-dong, Yusong-gu, Taejon, 305-701, KOREA

ABSTRACT

A new pattern decoder in MPEG-4 padding block is proposed. Necessities have been growing for the padding block to be implemented as a dedicated hardware because padding block belongs to the critical path of MPEG-4 encoding procedure. In padding block, the pattern decoder is the most complicated part. Therefore, an efficient hardware design is needed for the pattern decoder. To implement it as a hardware, the proposed decoder uses the modified repetitive padding algorithm. In the modified repetitive padding algorithm, the conventional three cases of Binary Alpha Block (BAB) pattern were simplified to just one case. Its regularity makes it easy for the pattern decoder to be hardware-implemented. The simulation results show that the proposed pattern decoder can be effectively used for implementing high-speed MPEG-4 padding block.

1. INTRODUCTION

MPEG-4 [1] standard provides a set of technologies to satisfy the needs of authors, service providers, and end users such as greater reusability, flexibility and interactivity of media objects, which include natural audio, video and synthetic content. The major difference between MPEG-4 and MPEG-1/2 is that the former specifies algorithms for the object-oriented coding. In other words, the data structure in MPEG-4 represents an image frame by a composition of video objects (VOs). Each video object is divided into a number of object layers. Under each object layer, there is an ordered sequence of snapshots in time, called video object plane (VOP). VOP is the basic unit where MPEG-4 encoding algorithm is applied. It is a rectangular area that completely covers VO with the minimum number of macro blocks (MBs). Macro block (MB) is a square rectangular area with 16 by 16 pixel size.

A special technique is required for coding the MBs that contain the shape edge of an object in ME/MC process because a VOP has an arbitrary shape [2]. Their non-object pixels are filled by the padding technique. The basic operation of padding is explained in section 2. In section 3, various cases encountered in padding process and its simplification are described. In section 4, a hardwired pattern decoder based on the simplified padding algorithm is introduced. In section 5, simulation results that place emphasis on the speed are presented. This paper is finished with the conclusion.

2. MPEG-4 PADDING ALGORITHM

The padding technique was introduced owing to the macroblock-based algorithm inside MPEG-4. In a single VOP, which is a minimum sized rectangular bounding box, there are MBs that are completely outside the object (Fig. 1-a), completely inside the object (Fig. 1-c), and that straddle the boundaries of the object (Fig. 1-b). The MBs that lie completely inside the object remain untouched. In the mean time, for the MBs that straddle the boundary of the object, the repetitive padding process is carried out, and the MBs that are completely outside the object are filled using extended padding. The extended padding is done by copying the first row or column of the next MB that is chosen depending on the position of the current MB. The simple copying operation does not need any hardware acceleration [3].

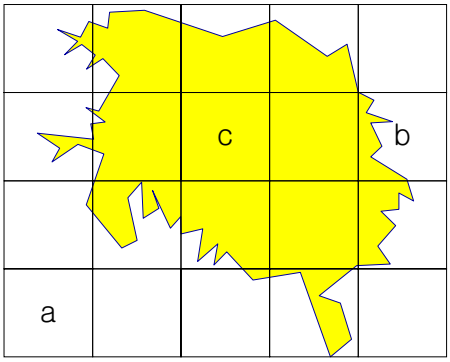


Figure 1. MBs of an arbitrary shaped VO (VOP)

The repetitive padding is even more complicated because it depends on the shape of the object. In the padding process, two data structures are needed; binary shape data, which is referred to as binary alpha block (BAB), and luminance data including chrominance data. By examining the BAB, non-object area is found and filled with the valid luminance data. The repetitive padding is carried out through two sequential steps; the horizontal padding and the vertical padding.

Fig. 2-(a) shows the horizontal padding process that is done by sequential row operation. All non-object pixels are filled with the next border value. If non-object pixels are located between two valid object pixels, the non-object pixels are filled with the average value of the two valid pixel data. An empty row or an initially filled row remains untouched.

Fig. 2-(b) shows the vertical padding process following the horizontal padding process. Likewise, all non-object pixels are filled vertically with the next border value of the object or the next pixel value that is filled in the horizontal padding process. If there are empty pixels between the valid ones, the average value of them is taken. After all columns of the MB are processed, the padding is finished.

0-7803-6685-9/01/$10.00©2001 IEEE

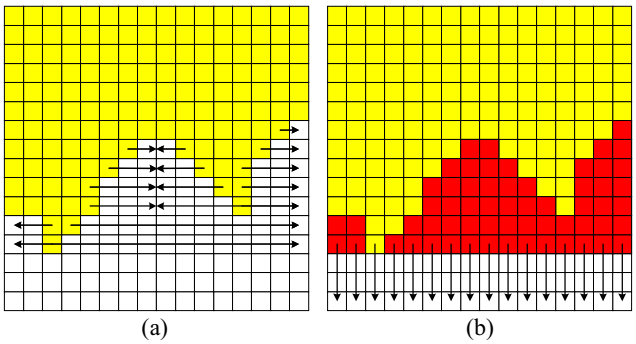


Figure 2. Repetitive padding algorithm (horizontal and vertical)

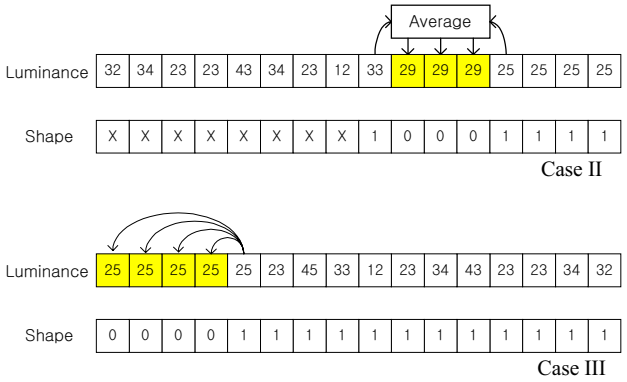


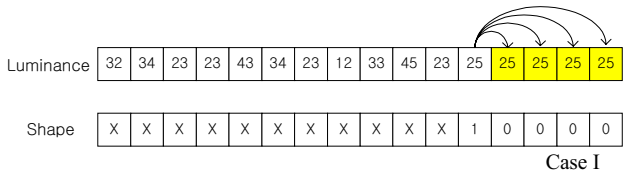
Figure 3. Three patterns of the shape-data

3. DECODING ALGORITHM

3.1 Conventional Cases

Basically, padding is processed with a row or a column as the fundamental unit. Therefore, the pattern of a single row or a column should be fully analyzed before the filling process of pixel value is started. The patterns of a row or a column in BAB have a wide variety because it is a slice of VOP which contains an arbitrary shape. However, they can be categorized into three cases (Fig. 3). To make the analysis easier, the row is traversed from the right edge to the left one. In case I, it starts with '0' and encounters '1'. In case II, it starts with '1' and has the 'hole'. In case III, it starts with '1' and ends with '0'. Owing to the arbitrary shape characteristics, above three cases can be mixed in a single row, which necessitates another padding process to the same row. A padding process to the same row comes to an end when all the pixels in the row become '1'. In the case of bus width limitation, 16 bits of a row can be divided into appropriate sub modules. For example, provided the row is divided into two 8 bit of rows, it brings forth increased 5 cases [3]. In this paper, three-case analysis is adopted.

All the information that is needed from the shape data for a row or a column is 'next' signal, source address where pixel luminance value is copied from, second source address (if needed), destination addresses, which is non-object pixel. In Fig. 3, for example, source address is the fifth bit from the right end in case I, fourth bit in case II, twelfth bit in case III. Second source address is the eighth bit in case II and it is needed only in the case II. It is necessary to simplify the cases, since too many cases increase decoder overhead. In the next section, modification of the algorithm for the simplicity is addressed.



3.2 Case Simplification

In three cases above, case II is more general than others because it needs two source addresses, while others need only one source address. Hence, it would be better to aim at case II in simplifying the cases, and how to integrate the three cases in the form of case II should be devised. One trial is as follows. In any cases of shape data pattern, if '1' bit is added on both sides of the shape data, all three cases become to have the same form as the original case II, which starts with '1' and has hole(s) (Fig. 4). Therefore, if there is a regular decoding solution for the case II, the decoder can be easily hardware-implemented.

As mentioned above, two source addresses should be found first, in case II ('simplified case' hereafter). Then, the luminance data for the addresses are averaged and the result is written in the destination. However, one of the source addresses is additionally added '1' bit, special care is needed because the luminance data for the additionally added '1' bit is zero. In this case, the sum of luminance, which is not averaged, is the right data to be written. It will be addressed in the next section for the hardware implementation again.

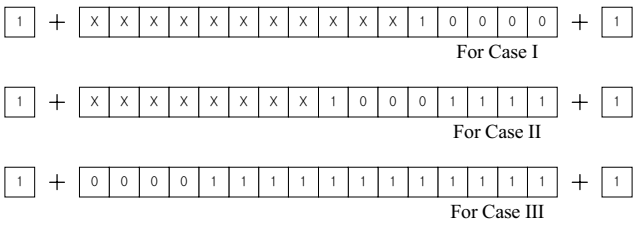


Figure 4. Case simplification

3.3 Edge Detection Algorithm for Case II

The primary object of the speculation and modification of the various cases is to implement the pattern decoder as an efficient dedicated hardware. Therefore, the decoding solution for the simplified case should be devised from the hardware point of view. As stated above, shape data is analyzed by traversing it

from right end to the left. Hence, to detect an edge, a special circuitry, named first-zero detector, is very useful (Fig.5). The operation is as follows. GND is carried through the NMOS chain from the right end until it encounters zero because zero-input bit turns on the PMOS to change the output bit data from GND to VDD. Once it is changed, VDD is carried through the remained NMOS chain, which leads to the output as a thermometer code.

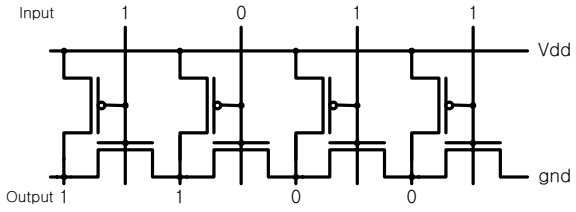


Figure 5. First-zero detector

With the first-zero detector, the shape data can be analyzed in hardware aspect as explained in Fig. 6. (A) is the input shape data. The information needed from the input shape data is the first and the second source address, the destination address, and 'next' signal. According to the input shape data (A), the first source address is the third bit and the second source address is the eighth bit. When the first zero detector is used with (A) as the input, (B) is obtained as the output. If (B) is right-shifted by one bit, the first source address is obtained as a thermometer code. (C) is the result of 'NAND' operation between (A) and (B), which is an intermediate process for (D). (D) is the first-zero-detected result like (B), which is the exact second source address, which is a thermometer code as well. (E) is obtained by the 'XOR' operation between (B) and (D). (E) is the write-enable signals for the averaged luminance data to be written.

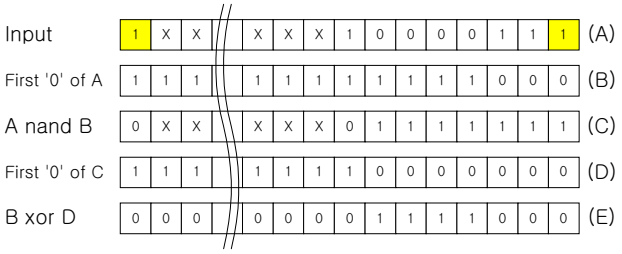


Figure 6. Decoding algorithm for the simplified case

4. HARDWARE IMPLEMENTATION

4.1 Decoder Implementation

Based on the decoding solution above, the pattern decoder can be implemented as a hardware (Fig. 7). 'a[0-17]' is the input shape data and 'b[0-17]' is the first source address that is left-shifted by one bit. 'd[0-17]is the exact second address, and 'e[0-17]' is the destination address. The 'next' signal is made such that when all the bits of a single row in shape data are all '1' or all '0', the padding process to the row is finished, and another row is fed to the padding procedure. In the illustrated decoder, the first and the last bit stage are for the additionally added '1' bit. If 'a0' is always '1', 'b0', 'd0' and 'e0' are always '0'. Hence, strictly speaking, four transistors and gates in the first stage can be omitted.

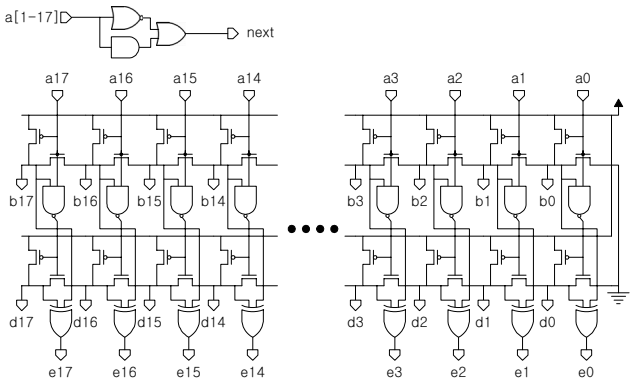


Figure 7. Implemented decoder

Fig. 8 shows the example schedule of decoder operation. Once an input shape data is fed into the FSM, the next input waits for the end of the precedent pad operation. When all the input bits become '1', the next shape data is inserted. For the given shape input data, 4 clock cycles are spent.

4.2 Overall Architecture

As mentioned above, it is necessary to handle the extra control procedure due to the case simplification, in the padding process,. The luminance data for the two source addresses are averaged and the result is written in the appropriate destination. However, if one of both source addresses contains the additionally added '1' bit, the luminance data for the other source address should be written without any change. Hence, there should be a correct selection between the sum of luminance data and the averaged luminance data, which is the shifted sum, according to whether the any source address contains the additionally added '1' bit.

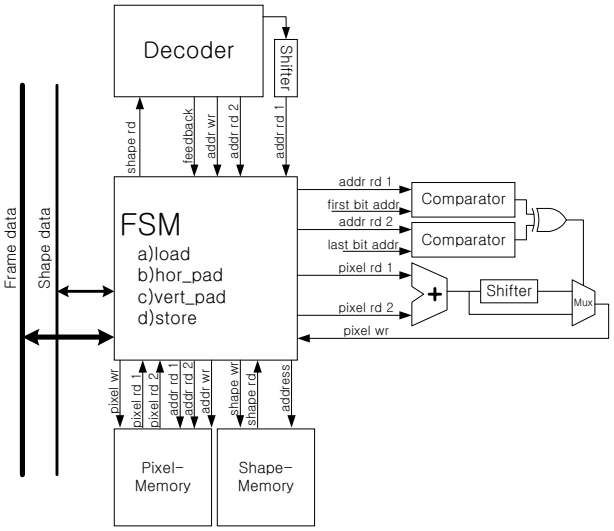


Figure 9. Overall architecture

Fig. 9 shows the overall architecture of padding block hardware. The FSM has the four states as load, horizontal pad, vertical pad and store. In the 'load' state, the pixel and shape data is fed into the inner memory of padding block through the input stream. After the horizontal and the vertical padding are finished, the processed output data is streamed out in the 'store' state.

On the right of the FSM, there is a data-path for pixel-value selection. If one of the source addresses contains the additionally added '1' bit, the sum that is not shifted is selected for the data to be written.

5. SIMULATION RESULT

The pattern decoder was simulated in 0.25 um CMOS technology. Fig. 10 shows that the maximum delay time is 0.5 ns, which is from 100.1 ns to 100.6 ns. The sample delay path which was chosen is from the 'a0' to the 'b17', 'd17' and 'e17' in Fig. 7. As a result, theoretical maximum speed is 2 GHz. On the assumption that the speed of FSM is high enough and delay time of load/store time is short enough, the speed of the padding block can reach GHz thanks to the reduction of the overhead of the pattern decoder by the proposed simplification algorithm.

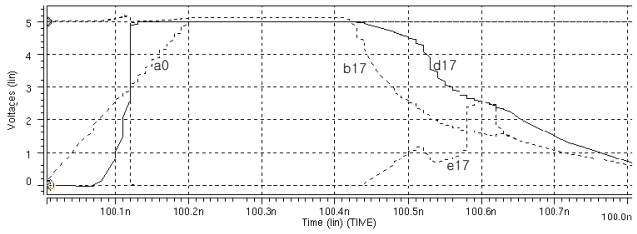


Figure 10. Maximum delay

The proposed pattern decoder can be effectively used in implementing much faster padding block compared to the conventional padding block hardware accelerator whose maximum speed is only 100 MHz [3].

6. CONCLUSIONS

This paper proposes a hardware-implemented pattern decoder in MPEG-4 padding block. Lack of coherence in various cases of the conventional pattern detection algorithm prevented the decoder from being implemented as an efficient hardware. Therefore, simplification of 3 cases is indispensable. A new decoding algorithm for simplified single case is schemed out using the first-zero detector. Finally, its possible hardware implementation and overall padding block architecture are introduced. Simulation results show that the possible maximum speed of the padding block could reach GHz with the reduction of the delay of load/store and the FSM unit.

7. ACKNOWLEDGMENT

This work was supported by KOSEF through the MICROS at KAIST, Korea.

8. REFERENCES

[1] Overview of the MPEG-4 Standard, ISO/IEC JTC1/SC29/WG11 N3536 Beijing – July 2000

[2] E. A. Edirisinghe, J. Jiang, C. Grecos, "Object boundary padding technique for improving MPEG-4 video compression efficiency". *Electron. Lett.,* 1999, pages 1453-1455.

[3] C. Heer, K. Migge. "VLSI Hardware accelerator for the MPEG-4 padding algorithm". *IS&T/SPIE Conference on Media Processors,* 1999

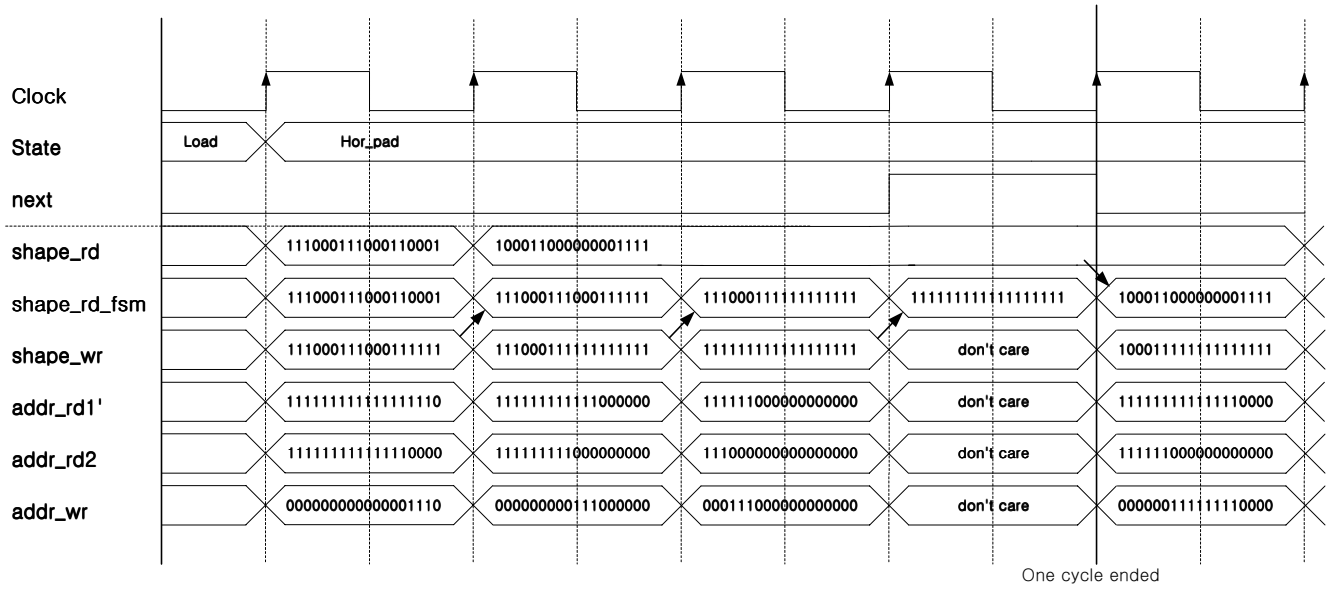


Figure 8. Example schedule of decoding operation