Efficient Architecture for Motion Vector Management of H.264/AVC Decoder for 4K Resolution

Tian Song¹, Naoyuki Ishikura¹, Kenji Watanabe² and Takashi Shimamoto¹

¹Department of Electrical and Electronic Engineering, Graduate School of Engineering, Tokushima University
Minami-Josanjima 2-1, Tokushima City 770-8506, Japan
²Synthesis Corporation, Awajimachi TC Bldg. 4th Floor, 2-6-9 Awajimachi, Chuo-ku, Osaka 541-0047, Japan
E-mail: tiansong@ee.tokushima-u.ac.jp

Abstract H.264/AVC has been successful in the past 10 years owing to its wide use in many fields from high-definition digital TV (HDTV) to low-resolution one-segment mobile TV. In recent years, with the high demand for high-resolution applications, 4K resolution (UHDTV) which has four times the resolution of 1080p HDTV, has become highly required for video streaming and broadcasting. However, the computational complexity for 4K-resolution applications is still the bottleneck preventing the realization of real-time decoders. In this work, a motion vector management architecture that is designed for total motion vector management and calculation is proposed. The proposed architecture can calculate predicted motion vectors, select reference motion vectors for the direct mode, and calculate the motion vectors for interface decoding. The proposed architecture can perform 176 cycles/MB decoding in the worst case. The simulation result shows that when working at 200 MHz, the proposed architecture can be implemented by about 99k gates.

Keywords: 4K resolution, motion vector, H.264/AVC

1. Introduction

H.264/AVC [1] is widely used for broadcasting from high-definition digital TV (HDTV) to low-resolution one-segment mobile TV. In recent years, with the development of higher resolution applications, 4K resolution with four times the resolution of 1080p HDTV, has become highly required for video streaming and broadcasting. However, the computational complexity of H.264/AVC, especially for 4K-resolution applications, is still the bottleneck preventing the realization of real-time decoders.

Many previous works have proposed efficient VLSI architectures for the decoder of H.264/AVC to realize fast processing. Some of these works realize real-time decoding for up to 1080p high-definition applications. As discussed in many previous works, the design of the real-time architecture of motion compensation (MC), including the external and internal data memory control is the main bottleneck [2][3]. In addition to the MC architecture, fast intra coding and real-time CABAC architecture are also critical modules. In the case of real-time decoding for 4K-resolution applications, it becomes more difficult for these function modules to realize real-time decoding. However, a fast MC architecture can be realized by using more processing elements, and fast intra decoding can also be realized by parallel processing with multiple calculation units. Other function modules such as DCT, quantization, deblocking filter, and CABAC, can also be realized by enhancing hardware cores [4]-[6].

However, in the design of a hardware decoder for H.264/AVC, it should be noted that the management of motion vectors (MVs) becomes another critical issue because fewer cycles are assigned for the management of MVs. With the traditional design methodology, MV management is distributed in each sub-module, which cannot realize efficient MV control. It should also be noted that the calculation of the predicted motion vector (PMV), and the generation of the MV for the direct mode and for interface coding perform similar calculation functions. Therefore, a centrally controlled MV management module can improve the efficiency of the total decoder. In this work, we design an efficient centrally controlled module and its VLSI architecture for all the modules con-
cerning MV generation. The design of the proposed architecture is dedicated to an application-specific integrated circuit (ASIC) with current 40 nm technology to achieve the smallest hardware area with a typical clock frequency of 200MHz.

2. Motion Vector in H.264/AVC

![Motion estimation](image)

**Fig. 1 Motion estimation**

As shown in Fig. 1, motion estimation (ME) is a process to search for the MV. First, the PMV is calculated from the MV of adjacent macroblocks (MBs). Next, the search center is determined on the basis of the PMV and the ME is performed within a dedicated search range (SR). The position with the smallest SAD (sum of absolute difference) is selected as the best point and the MV is thus obtained.

H.264/AVC achieves high coding efficiency by introducing multiple-block-size ME. It uses four types of large block of 16x16, 16x8, 8x16, and 8x8 and three other types of smaller block of 8x4, 4x8, and 4x4. Each MV in H.264/AVC is strongly correlated with the MVs of surrounding blocks. Therefore, the PMV from this relationship is predicted from neighboring blocks, and the bitrate is reduced by encoding the vector difference. Basically, as shown in Fig. 2, the PMV of the MVs of the horizontal component (x direction) and vertical component (y direction) are calculated by taking the median of the value on the left of the target coding block (A), upper (B), upper right (C), and upper left (D). As an exception, in some cases when the target coding blocks are coded by 16x8 or 8x16 blocks and the target coding block and neighboring blocks are coded with blocks of different size, the calculation of the median value is different.

In the decoding process the surrounding MVs of the current decoding block are used. Figure 3 shows three different patterns. In the normal case of 4x4 blocks as shown in Fig. 3 (a), the median of the left, upper, and upper-right MVs is calculated in the x and y directions. In the case of 16x16 blocks, the median of the surrounding MVs shown in Fig. 3 (b) is calculated. In the case of a frame boundary, the upper-right MV is not available. mvD shown in Fig. 3 (c) is then used.

![Median value calculation](image)

**Fig. 2 Median value calculation**

To achieve higher coding efficiency, a direct mode is introduced in the predictive B slices, according to which the MV can be derived from the previous coding information without additional encoding information for an MB or block. In the decoding process, when an MB is encoded by a direct mode, the MV can be calculated directly from the relevant MVs in the reference frames. Therefore, direct mode decoding also concerns MV management. H.264/AVC allows two fields to be coded either jointly or separately. The frame/field coding concept can be extended to the MB level which is called macroblock-adaptive frame/field (MBAFF) coding. When MBAFF coding is used, MV management of the frame and field decoding have to be performed.

3. Proposed architecture

In the proposed motion vector management (MVM) architecture, the calculation of the PMV, the MV calculation for the direct mode, and the MV calculation for MBAFF coding are implemented. Figure 4 shows the proposed architecture.

The proposed MVM module receives the control command and processing parameters from the system bus. The reference MVs for calculating the PMV are transferred from the external DRAM and the calculated MVs are output to the data bus. There are two main calculation cores in the proposed MVM module, which are designed for direct mode MV prediction and normal MV prediction. The REGISTERS module is used for the temporary storage of some other input decoding parameters and provides signals at the correct timing for the other connecting modules. An internal SRAM is prepared to store the MVs surrounding current MB including the MVs on the left, upper-left, upper, and upper-right of the MB. Upon the calculating the median of the three MVs, the required MVs are read out from the SRAM. The PMV calculation process is performed in the PMV module with a two-stage pipeline, which can reduce the number of processing cycles by about half. Moreover, in order to shorten the critical path, it is divided into six steps. When the MV for direct-mode-encoded blocks is being processed, the direct mode process module is run parallel to the normal mode process module to avoid output delay.
3.1 Flowchart of MVM

The MV calculation is processed in terms of the decoding parameters. In the normal case, only the median of the surrounding MVs needs to be calculated. When the direct mode is used, either the spatial or temporal direct mode is selected and the MVs have to be calculated. When MBAFF coding is used, the MVs for the bottom and top fields have to be calculated sequentially. Figure 5 shows the flowchart of the MVM.

When starting the decoding process, several necessary parameters are input from an external main processor by the system bus. An external DRAM is used as the frame memory with the DRAM bus connected with the proposed module. The MVs of the surrounding MB are read out from the SRAM to prepare for the calculation of the median MV. Next, if the MB type is intra mode coded, the calculation is skipped and is written back to the SRAM directly. When the MB is encoded by the direct mode, either the temporal or spatial direct mode is used. When the MB is encoded by the direct mode, the reference MVs of the reference frames have to be read from the external DRAM. In the case of the MBAFF coding mode, the MVs for the top and bottom fields are calculated one by one.

3.2 Decoding schedule

In the proposed architecture a six-step pipeline schedule is introduced to realize low-cost implemen-
tation and a reduction in the number of processing cycles as shown in Fig. 6. In the first step, the surrounding MVs including the MVs on the left, upper-left, upper, and upper-right MB are loaded from the SRAM. In the second step, on the basis of the loaded MVs and parameters, the adjacent blocks for decoding are selected. In the third step, the median MV is calculated on the basis of the loaded surrounding MVs, and the PMV is calculated. In the fourth step, the MV is calculated on the basis of the PMV and MVD. In the fifth step, it is checked whether the decoding MB is the skip mode. In the sixth step, the MV is output. After the processing for the L0 direction, the same processing for the L1 direction is performed after one clock cycle. In the case of the direct mode when starting the PMV processing, the direct mode decoding starts at the same time. Parallel processing of the PMV and direct mode reduces the total number of processing cycles. For the normal decoding of a 16x16 MB, seven cycles are necessary including the L0 and L1 directions. In the worst case, when one MB is encoded by 16 4x4 blocks, 112 cycles are necessary including the L0 and L1 directions.

The register arrays are designed to save the MVs of surrounding MBs. Figure 7 shows the PMV architecture and Fig. 8 shows the function of the register arrays in the six-step pipeline.

In the first step, the surrounding MVs, which are loaded from the SRAM, are input in the registers including the L0 and L1 directions. In the next step, blk.a, blk.b, blk.c, and blk.d are selected to calculate the median value, which are saved in other registers. In the third step, the median value is calculated using the selected MVs.

On the other hand, after the reconstruction of the MVs by accumulating the MVD to the PMV, the MVs are saved in the register array in the fourth step. These register arrays store the MVs of 16 4x4 blocks and wait for the outputs. These outputs are also saved in the SRAM at the same time for the processing of the next MB. In the case of the skip mode, the skip processing is performed in the fifth step. The pipeline processing for 16 4x4 blocks is also a reasonable solution for the proposed architecture. However, the proposed pipeline has fast enough processing for real-time decoding. Although the pipeline for 16 4x4 blocks is not considered in the proposed architecture, it may become a solution when using some previous technology.

3.3 SRAM

The width of the data bus is 67 bits. Two words, namely 134 bits, are used to save the reference MVs.
and relevant information for the MV calculation. Figure 9 shows the bit field description of the SRAM.

![Bit field description of SRAM](image)

Table 2 - Implementation results

<table>
<thead>
<tr>
<th>module</th>
<th>gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTERS</td>
<td>9594</td>
</tr>
<tr>
<td>CALC</td>
<td>17802</td>
</tr>
<tr>
<td>DIRECT</td>
<td>27412</td>
</tr>
<tr>
<td>PMV</td>
<td>27660</td>
</tr>
<tr>
<td>DRAM</td>
<td>4149</td>
</tr>
<tr>
<td>SD REFGEN</td>
<td>850</td>
</tr>
<tr>
<td>CTRL</td>
<td>262</td>
</tr>
<tr>
<td>SRAM</td>
<td>6344</td>
</tr>
<tr>
<td>total</td>
<td>99079</td>
</tr>
</tbody>
</table>

The processing time is evaluated by the number of processing cycles. The number of processing cycles is summarized in Table 3.

Table 3 - Processing cycles

<table>
<thead>
<tr>
<th>MVs</th>
<th>calculate MV</th>
<th>read</th>
<th>write</th>
<th>other</th>
<th>total cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>28</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>47</td>
</tr>
<tr>
<td>10</td>
<td>35</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>54</td>
</tr>
<tr>
<td>14</td>
<td>49</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>68</td>
</tr>
<tr>
<td>16</td>
<td>66</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>75</td>
</tr>
<tr>
<td>20</td>
<td>70</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>89</td>
</tr>
<tr>
<td>22</td>
<td>77</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>96</td>
</tr>
<tr>
<td>26</td>
<td>91</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>110</td>
</tr>
<tr>
<td>28</td>
<td>98</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>117</td>
</tr>
<tr>
<td>32</td>
<td>112</td>
<td>11</td>
<td>6</td>
<td>2</td>
<td>131</td>
</tr>
</tbody>
</table>

In most case, 16x16 blocks are used and a total of 26 cycles are necessary. When the coding block size becomes smaller more MVs need to be processed. The worst case is when one MB is encoded by 16 4x4 blocks. In this case, a total of 131 cycles are needed. When the temporal direct mode and MBAFF mode are used, many more processing cycles are necessary. Figure 10 shows the number of cycles in every case.

As shown in Fig. 10, in the worst case 176 cycles are necessary. However, even in the worst case, the proposed architecture is also fast enough for the real-time decoding of 4K applications.

5. Conclusions

In this paper, we propose a centrally controlled MV management architecture for a 4K-resolution H.264/AVC decoder. The proposed architecture can calculate the predicted motion vectors, select the reference motion vectors for the direct mode, and calcu-
late the motion vectors for interlace decoding. The proposed architecture can perform 176 cycles/MB real-time decoding even in the worst case. The simulation result shows that, when working at 200MHz, the proposed architecture can be implemented by about 99k gates.

Acknowledgements

The authors would like to express their sincere thanks to Mr. Kentaro Takei and Mr. Takafumi Bando, who graduated from our university. This work was supported by a Grant-in-Aid for Young Scientists (B) and with the generous support of Synthesis Corporation.

References


Tian Song received his B.E. degree from Dalian University of Technology, China, in 1995, and his M.E. and Dr.E. degrees from Osaka University in 2001 and 2004, respectively. He joined Tokushima University in 2004 as an Assistant Professor. Presently, he is an Associate Professor of the Department of Electrical and Electronic Engineering, Graduate School of Advanced Technology and Science, Tokushima University. He is a member of IEICE and IEEE. His current research interests include video coding algorithms, VLSI architectures, and system design methodology.

Naoyuki Ishikura received his B.E. degree from Tokushima University in 2011. He is working for his masters degree at the Graduate School of Advanced Technology and Science, Tokushima University, Japan. His current research interests include video coding algorithms and video extraction algorithms.

Kenji Watanabe received his B.E., M.E., and Ph.D. degrees from Osaka University in 2004, 2006, and 2009, respectively. He joined Synthesis Corporation in 2009. He is a member of IEICE, IPSJ, and IEEE. His current research interests include video coding algorithms, VLSI architectures, and system design methodology.

Takashi Shimamoto received his B.E. and M.E. degrees in electrical engineering from Tokushima University, and his Dr.E. degree from Osaka University in 1982, 1984, and 1992, respectively. He joined Tokushima University in 1984 as an Assistant Professor. Presently, he is a Professor of the Department of Electrical and Electronic Engineering, Graduate School of Advanced Technology and Science, Tokushima University. His current research interests include heuristic algorithms for VLSI CAD.

(Received May 14, 2012; revised September 14, 2012)