A HIGHLY EFFICIENT EXTERNAL MEMORY INTERFACE ARCHITECTURE FOR AVS HD VIDEO ENCODER

Xiaofeng Huang, Chuang Zhu, Lei Zhang, Kaijin Wei, Huizhu Jia, Don Xie, Wen Gao

National Engineering Laboratory for Video Technology, Peking University, Beijing, China
xfhuang@jdl.ac.cn

ABSTRACT

This paper presents a highly efficient external memory interface architecture to improve memory bandwidth utilization for AVS HD video encoder. Both burst and bank interleaved SDRAM accesses are intelligently adopted in the memory interface design. Our proposed architecture is composed of an address mapping layer and an arbitration layer. In the address mapping layer, according to the data request pattern and quantity, the clients in the encoder are divided into four groups which are assigned to different banks of the SDRAM. In each group, efficient address mapping schemes are proposed to minimize inner client overhead. In the arbitration layer, a straightforward group-based interleaved arbitration scheme is proposed to minimize inter client overhead. Experimental results show that the data access overhead cycles of our proposed interface design are reduced significantly and the bandwidth utilization is improved by up to 10% compared to the tile-linear address mapping scheme.

Index Terms— address mapping, arbitration, memory bandwidth, AVS encoder

1. INTRODUCTION

AVS video coding standard, which is developed by China Audio Video Coding Standard (AVS) Working Group, has been accepted as an option by ITU-TFGIPTV for IPTV applications [1]. The AVS part 2 (AVS-P2) is high resolution friendly profile, which shows comparable performance with H.264/AVC for most HD sequences [7].

As the resolution of video-coding applications increases, explosive video data should be stored in off-chip memory such as DDR SDRAM [2]. In addition to the huge data transfers, the extra overhead cycles incurred by row-activation of SDRAM have a very negative effect on the video processing systems. How to reduce the overhead cycles and thus improve bandwidth efficiency has already become a hot research topic currently.

Numerous optimization strategies to improve bandwidth efficiency in video applications have been reported. H. Kim and I-C. Park proposed the array address-translation method to minimize the number of row-activation command and the power consumption [2]. Kaijin Wei et al adopted a novel Level C+ data reuse scheme to reduce the required external memory bandwidth for motion estimation [4][5]. A lossy reference frame compression technique has been proposed in [6] to reduce the SDRAM bandwidth, but this method will encounter quality degradation problem.

In this paper, we propose a highly efficient memory interface design including an address mapping layer and an arbitration layer. The remainder of this paper is organized as follows. Section 2 presents the problem which will be solved in this paper. The address mapping layer and the arbitration layer are described in section 3 and section 4, respectively. Section 5 presents the hardware implementation of the proposed design. Section 6 shows the experimental results and section 7 concludes this paper.

2. PROBLEM DEFINITION

In this section, the memory bandwidth related problem in AVS encoder is described. It is noted that the SDRAM can achieve its best performance when an application accesses the memory in the burst access mode. In video-processing applications, the block-based access pattern can incur many row-activation overhead cycles which decreases the bandwidth utilization greatly.

2.1. Features of SDRAM

Figure 1 shows a simplified architecture of a four-bank SDRAM [2]. All memory banks share the same data and address buses, whereas each bank has its own row decoder, column decoder and sense amplifier. The mode register stores several SDRAM operation modes such as burst length, burst type, CAS (column address strobe) latency, etc. A series of commands need to be issued when accessing the SDRAM. Firstly, a row-activation (ACTIVE) command is used to copy the row data of a designated bank into the sense amplifier. Secondly, a column address is selected and the corresponding data is transferred by the READ or

WRITE command. Finally, the activated SDRAM row is re-charged for next SDRAM access by PRECHARGE command.

When accessing the SDRAM, the overhead cycles for each command decrease bandwidth efficiency. The ACTIVE to column access delay called t_{RCD} and PRECHARGE command period called t_{RP} are the majority parts of overhead cycles [11]. The t_{RCD} and t_{RP} latency can be hidden by accessing different banks due to the independent processing capability of each bank [3].

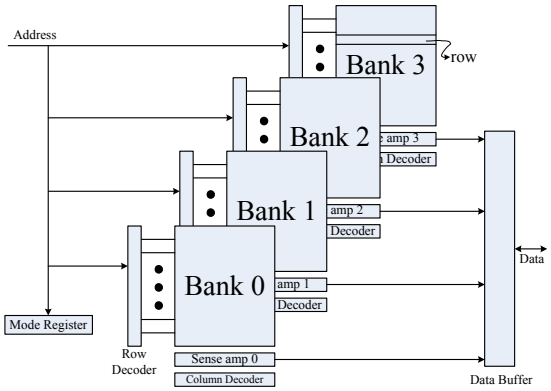

Fig. 1. Simplified architecture of a four-bank SDRAM.

2.2. Memory access clients in AVS encoder

In AVS encoder, motion estimation (ME) is the most bandwidth consuming part. Level C+ zigzag coding order is adopted in our encoder system to reduce the redundant data access of luminance reference pixels [5]. The bandwidth can be roughly reduced to 1/3 compared to the raster order [5]. Besides, the chrominance reference windows are also needed for fractional motion estimation (FME). In detail, the chrominance reference pixels are a 24×24 search window where the start pixel depends on the predicted motion vector (PMV) from integer motion estimation (IME).

In addition to the reference windows for ME, original pixels, motion vectors for predicted motion vectors and reconstructed pixels for reference store are needed in AVS encoder. For each MB of I frame, one MB original pixels are fetched from external memory and one MB reconstruction pixels are written back to SDRAM to be referenced for P and B frame. The P and B frame encoding is almost the same as I frame except the additional reference frame load. P frame uses two forward reference frames to predict the current frame; B frame uses one forward frame and one backward frame as references. The corresponding motion vector (MV) information of the MB is stored when encoding P frame which is for future B frame encoding.

The memory access clients in AVS encoder are shown in Table 1. F_ORG client fetches original pixels and F_REF_Y_{1-4} clients fetch luminance reference pixels for IME. F_REF_UV_{1-4} clients fetch chrominance reference windows for FME. There are four clients to load luminance and chrominance windows because AVS-P2 restricts the maximum number of reference pictures to 2, and the maximum number of reference to 4 for interlaced contents [7]. F_MV and S_MV clients fetch and store MV, respectively. S_REC_Y and S_REC_UV clients store luminance and chrominance reference pixels into external memory for P and B frame reference. Additional clients (S_BG and F_BG) are required in the system due to the Level C+ zigzag order bit-stream is not supported by decoder. A module named SPLICE will transform zigzag order bit-stream into standard order. S_BG client stores the zigzag bit-stream into external memory and F_BG client fetches bit-stream in standard order from external memory for SPLICE. The √ indicates the enabled state and - indicates the disabled state for each client in Table 1.

2.3. Data access and data processing pipeline

In the encoder system, data access and data processing are pipelined, as shown in Figure 2. The cycles spent in the data access stage impacts the encoder performance when it is longer than the data processing period. Besides the bandwidth requirement for the encoder, the bandwidth required for CPU also inspires us to improve bandwidth efficiency.

The problem to be solved in this paper is how to minimize the bandwidth in the encoder system. As talked above, overhead cycles are seriously degrading the bandwidth efficiency and in the next we will focus on how to reduce these overhead cycles.

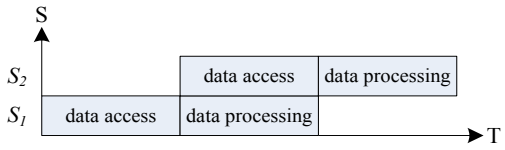

Fig. 2. Data access and data processing pipeline space time-diagram.

Table 1. Enabled and disabled memory access state of clients in AVS encoder for IPB frame

	F_ORG	F_REF_Y_{1-4}	F_REF_UV_{1-4}	F_MV	F_BG	S_REC_Y	S_REC_UV	S_MV	S_BG
I frame	√	-	-	-	√	√	√	-	√
P frame	√	√	√	-	√	√	√	√	√
B frame	√	√	√	√	√	-	-	-	√

3. ADDRESS MAPPING

Various address mapping schemes have been proposed for different applications [2][8][9]. A poor address mapping scheme generates unnecessary SDRAM latency such as t_{RCD} which largely degrades system performance. The type of SDRAM latency can be categorized into two groups: inner client overhead and inter client overhead. The inner client overhead denotes the latency during a client access. The inter client overhead denotes the incurred latency when switching from one client access to another.

3.1. Address mapping scheme to minimize inner client overhead

Before designing a memory address mapping scheme to minimize the inner client overhead, the data access patterns of clients in AVS encoder are thoroughly analyzed. There are three kinds of data access patterns in AVS encoder as shown in Figure 3. The first pattern is one MB unit data access. Clients such as F_ORG and F_MV fetch one MB data of original pixels and MV for the current MB to be encoded. The second pattern is MB column data access. The F_REF_$Y_{1\text{-}4}$ clients belong to this pattern due to adoption of the Level C+ zigzag encoding order [5]. The third pattern is 2-dimensional (2D) MB access. The F_REF_$UV_{1\text{-}4}$ clients belong to this pattern because of the PMV dependent 2D reference windows.

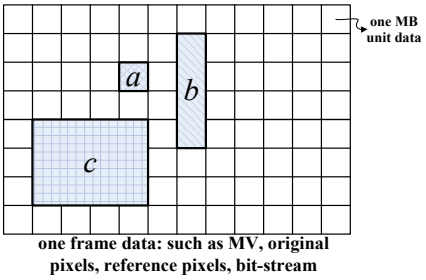

Fig. 3. Access patterns in AVS encoder. *a*: one MB unit data access, *b*: MB column data access, *c*: 2-dimensional (2D) MB data access.

A simple and effective scheme named MB-based column address mapping is proposed as shown in Figure 4(a) which will minimize latency for inner client access. The MB-based column address mapping method maps one frame data into external memory by sequentially storing MB-based column data. The *init_addr* is the start address of one frame data. The *one_col_addr_offest* means the address space occupied where stores one MB-based column data. The translation of mapped address into SDRAM row, bank and column addresses is shown as Figure 4(b). The lowest 10 bits is mapped to column address, 10-12 bits is mapped to bank address and the highest 14 bits is mapped to row address of SDRAM, respectively.

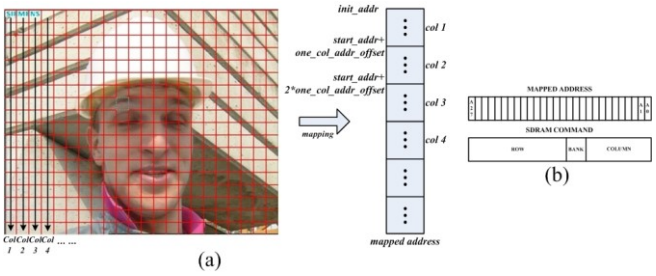

Fig. 4. (a) MB-based column address mapping scheme (b) mapped address to SDRAM command translation.

The MB unit data access would not generate SDRAM latency which can adopt burst access due to the continuous address space in external memory. The same conclusion can be derived for the MB column data access. The third access pattern can be seen as the combination of the second access pattern, so we only need to care about the access switching from one column to another. The switching skips about one MB-based column data (C_{mb_col}) of the frame. If the capacity of one row data of all the chosen banks (C_{row_banks}) is larger than C_{mb_col} as shown (1), the switching will not cross different rows in the same bank. In our system, the maximum data amount of C_{mb_col} is (256B+128B)×68=25.5KB for 1080P; the SDRAM bank number is 8, SDRAM data width is 64b and column address width of SDRAM is 10b [10]. We can get that C_{row_banks} is $64\times2^{10}\times8b$=64KB which is larger than C_{mb_col}. Then we can get the conclusion that the switching from one column to another in the third access pattern will not generate row-activation overhead and therefore inner client overhead gets the minimum value.

$$C_{row\_banks} > C_{mb\_col} \quad (1)$$

3.2. Address mapping schemes to minimize inter client overhead

The inner client overhead problem has been perfectly solved by MB-based column address mapping method. The inter client overhead may be incurred when switching from one client to another. For example, the inter client overhead will be incurred when switching from F_ORG to F_REF_Y_1 client if they locate in the different rows of same bank as shown in Figure 5. To avoid inter client overhead, a group-based address mapping method is proposed.

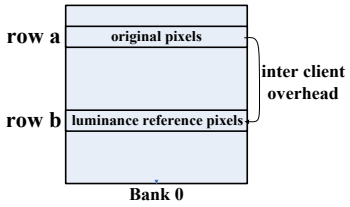

Fig. 5. Inter client overhead example.

3.2.1. Group division for encoder clients

The group division is based on the data access pattern and data amount of encoder clients. Based on the access patterns as shown in Figure 3, we divide all the encoder clients into three groups (one MB unit access group, MB column access group, 2D MB access group). In the one MB unit data access group, the data access patterns of F_BG and F_ORG clients are different: the access manner of F_ORG is in zigzag order, but the access manner of F_BG is in raster order. Based on the analysis above, we divide one MB unit data access group into two sub-groups: raster order group and zigzag order group. Finally, we summarize the grouped-clients into Table 2. Store clients are allocated to each group based on the consistence of corresponding fetch clients.

Table 2. Grouped-clients

group	client
zigzag order group	F_ORG, F_MV, S_MV
raster order group	F_BG, S_BG
MB-based column access group	F_REF_Y1-4, S_REF_Y
2D MB access group	F_REF_UV1-4, S_REF_UV

3.2.2. Bank allocation for each group

For raster and zigzag order group, we restrict the one MB data to locate in the same row of SDRAM based on the small data access amount fact. Thus the required bank is 1 for raster and zigzag order group. Different from raster and zigzag order groups, the needed bank for MB column access group is 2 to hide the overhead cycles when access happened in the row border of SDRAM. In order to satisfy (1), the least bank required for 2D MB access group is 4. The minimum bank required for each group is shown as Table 3.

Table 3. Minimum bank required for each group

group	bank required
zigzag order group	1
raster order group	1
MB-based column access group	2
2D MB access group	4

3.2.3. Minimal bank array address mapping scheme for 2D access

In each group, we adopt MB-based column address mapping scheme. The inner client overhead will be minimized if the bank for each group satisfies Table 3. The inter client overhead will be minimized by group-based interleaved arbitration method which will be described in Part 4.

For 2D MB access group, the needed bank is dependent on the C_{row_banks} and C_{mb_col}. In order to be more general for different applications, we propose a minimal bank array address mapping scheme for 2D MB access.

As shown in Figure 6, the red rectangle in the frame is decomposed into residing in different banks but of the same row in SDRAM. frame_width and frame_height represent the width and height of the frame. page_width and page_height represent the MB width and MB height contained in one row of the bank. There are three patterns in 2D MB data access. The (a) pattern access represents data access in the frontier among three banks. As we can see, there is no overhead cycle when we switch columns because of the bank interleaved access. The same conclusion can be derived for access pattern (b) and (c).

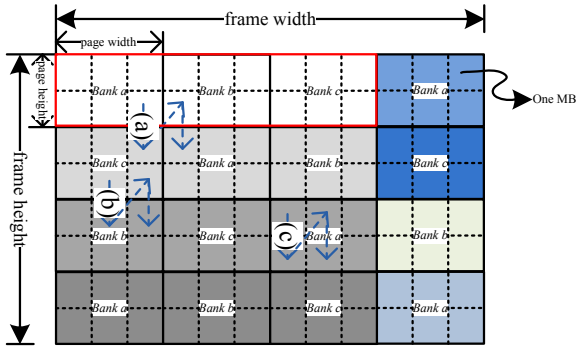

Fig. 6. Minimal bank array address mapping scheme for 2D MB access.

4. ARBITRATION

The arbitration strategy affects the inter client overhead because of the SDRAM access order for each client. In order to minimize inter client overhead, we propose a straightforward group-based interleaved arbitration scheme.

Our proposed arbitration method is shown as Figure 7. It is a fact that different groups are assigned to different banks; this is the reason the group-based interleaved arbitration is adopted in this work. Compared with write command, read command needs extra cycles to return the request data which affects the encoder performance. The subroutine of *get_wr(rd)_group* returns the maximum accessed group index for read or write request. The *ger_wr_(rd)_client* subroutine returns one of the requested clients in the selected group based on the fixed priority.

```
subroutine arbiter(rd_req,wr_req) {
  while(rd_req || wr_req) {
    if(rd_req) {
      group = get_rd_grp(rd_req,grp_last);
      client = get_rd_client(group);
    }
    else if(wr_req) {
      group  = get_wr_grp(wr_req,grp_last);
      client = get_wr_client(group);
    }
    access(client);
    grp_last = group;
  }
}
```

```
subroutine get_wr(rd)_grp(req,last_grp) {
  group=last_group;
  if(other groups have req) {
    group= group of maximum req;
  }
  return group;
}

subroutine get_wr(rd)_client(group) {
  client = select one request client in group;
  return client;
}
```

Fig. 7. A straightforward group-based interleaved arbitration scheme.

5. HARDWARE IMPLEMENTATION

5.1. Address mapping layer and its hardware implementation

Figure 8 shows block diagram of the address mapping schemes which takes MB and frame information as its input and generates proper mapped address. *mbx* and *mby* are the MB coordinate locations. MBH means the MB height of the frame. MB_SIZE represents the SDRAM address space occupied to store one MB data.

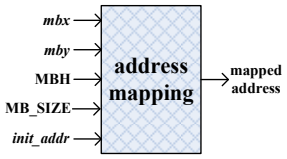

Fig. 8. Block diagram of the address mapping schemes.

```
mb_addr = (mbx*MBH+mby)*MB_SIZE;
row_addr = {corresponding SDRAM row address of mb_addr};
{bank_2,bank_1,bank_0} = {corresponding SDRAM bank address of mb_addr};
col_addr = {corresponding SDRAM column address of mb_addr};
mapped_addr = init_addr+{row_addr,bank_2,bank_1,2'b00,bank_0,col_addr};
```

(a) Zigzag order group address mapping realization

```
mb_addr = (mbx*MBH+mby)*MB_SIZE;
row_addr = {corresponding SDRAM row address of mb_addr};
{bank_2,bank_1,bank_0} = {corresponding SDRAM bank address of mb_addr};
col_addr = {corresponding SDRAM column address of mb_addr};
mapped_addr = init_addr+{row_addr,bank_2,bank_1,bank_0,3'b010,col_addr};
```

(b) Raster order group address mapping realization

```
mb_addr = (mbx*MBH+mby)*MB_SIZE;
row_addr = {corresponding SDRAM row address of mb_addr};
{bank_2,bank_1,bank_0} = {corresponding SDRAM bank address of mb_addr};
col_addr = {corresponding SDRAM column address of mb_addr};
if(bank_0==0) mapped_bank=3'b011;
else mapped_bank=3'b100;
mapped_addr = init_addr+{row_addr,bank_2,bank_1,mapped_bank,col_addr};
```

(c) MB column access group address mapping realization

```
row= {SDRAM row address of init_addr} +
     +(mby/page_height)+(mbx/(3*page_width))*[(MBH/page_height)];
tmp={(mbx%(3*page_width))/page_width),((mby%(3*page_height))/page_height)};
case(tmp) {
   4'b0000: bank=5; 4'b0001: bank=6; 4'b0010: bank=7;
   4'b0100: bank=7; 4'b0101: bank=5; 4'b0110: bank=6;
   4'b1000: bank=6; 4'b1001: bank=7; 4'b1010: bank=5;
}
col=((mbx%page_width)*page_height+mby%page_height)*MB_SIZE;
mapped_addr={row,bank,col};
```

(d) 2D MB access group address mapping realization

Fig. 9. Address mapping realizations for (a) zigzag order group (b) raster order group (c) MB column access group and (d) 2D MB access group.

Address mapping realizations for zigzag order group, raster order group, MB column access group and 2D MB access group are shown in Figure 9. The multiplication and division operation in Figure 9 can be replaced by shift operation if we set parameters such as MBH to 2^n.

The address mapping module is integrated into fetch and store clients as shown in Figure 10. *para_parse* module parses the parameters to *addr_mapping* module after the assertion of *mb_start* signal which is dispensed by *mb_ctrl* module. The *req_to_ddr* signal will be valid after *addr_mapping* module is finished. In fetch client, data from external memory is organized to return to the request user module. In store client, we buffer the data-to-be-stored in a ping-pong RAM in order to access external memory in burst access mode.

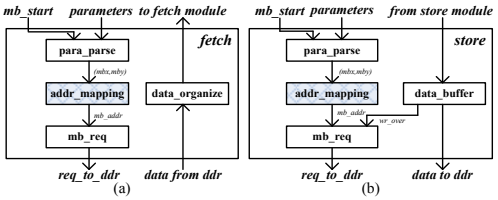

Fig.10. (a) fetch client (b) store client.

Our memory interface architecture is shown as Figure 11. An address mapping layer and an arbitration layer are included in the architecture. The address mapping layer includes the client interface which interacts with encoding modules such as IME, FME and so on. The arbitration layer reorders the request clients in a highly efficient manner by group-based interleaved arbitration scheme.

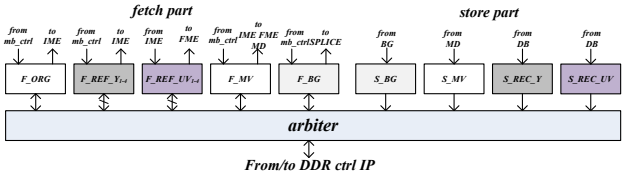

Fig.11. Memory interface architecture.

5.2. System architecture

System architecture is shown as Figure 12. The memory interface is a bridge for all external memory data request modules in encoder. Firmware is used for frame-level control, responsible for parameter configuration. Capture module captures original video data and stores it into external memory. Efficient five-stage MB-pipelining architecture is applied for our encoder core in the purple block and the five major components are IME, FME, mode decision (MD), deblocking filter (DB) and bit stream generating (BG).

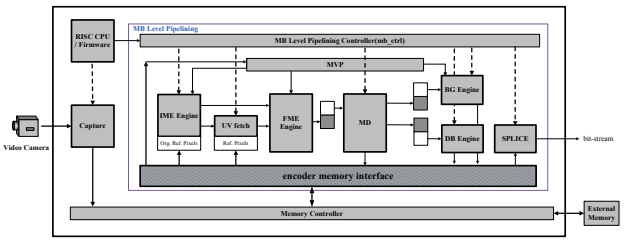

Fig. 12. System architecture.

Table 4. Experimental results

Encoding mode	sequence	overhead cycles for TLAM	overhead cycles for proposed	valid data transfer cycles	speedup	I	P	B
progressive	foreman	1192712	628806	8977760	5.16%	0.87%	4.03%	5.29%
	CREW	4676248	2511300	35914912	4.97%	0.92%	4.32%	5.31%
	flowergarden	4687210	2560638	36792720	4.78%	0.93%	4.21%	4.86%
	city	11170604	5650104	81763360	5.55%	0.97%	4.67%	5.60%
	blue_sky	25018708	12803334	186739336	5.38%	0.90%	4.94%	5.88%
interlace	foreman	2136670	778968	12508720	8.71%	3.40%	7.21%	9.52%
	CREW	7511100	2578608	43541168	9.20%	3.60%	8.50%	10.70%
	flowergarden	7882234	2654046	45351540	9.30%	3.70%	7.60%	10.60%
	blue_sky	39608802	13105944	230329024	9.30%	3.80%	7.70%	10.80%

6. RESULTS

In order to verify the performance of our proposed method, the tile-linear address mapping scheme (TLAM) [9] is used for comparison. The performance evaluation is performed on VCS simulation of AVS-P2 encoder.

Table 4 shows the results of both proposed and TLAM address mapping schemes. The results show that our proposed method reduces much more overhead cycles compare to the TLAM method. The overhead cycles are hidden in our proposed method except for the data access at the beginning of each pipeline. On average, the speedup factor of the proposed method over the TLAM method is 5% for frame and 10% for interlace.

7. CONCLUSION

This paper presents a novel memory-interface architecture which is composed of an address mapping layer and an arbitration layer to improve memory access efficiency. In address mapping layer, a group-based address mapping scheme is proposed to minimize inner client overhead. Group-based interleaved arbitration scheme is realized in the arbitration layer which guarantees the access to SDRAM in the highest efficient way. Experimental results show that the proposed method improves the SDRAM bandwidth by up to 10% compared to the TLAM method.

8. REFERENCE

[1] Chuang Zhu, Yuan Li, Hui-zhu Jia, Xiao-dong Xie, Hai-bing Yin, "A highly efficient pipeline architecture of RDO-based mode decision design for AVS HD video encoder," ICME, July 2011, 2011.

[2] H. Kim and I.-C. Park, "High-Performance and Low-Power Memory-Interface Architecture for Video Processing Applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, pp. 1160-1170, Nov. 2001.

[3] S. Rixner et al, "Memory access scheduling," in *Proc. ISCA*, Vancouver, BC, Canada, Jun. 2000, pp. 128–138.

[4] Kaijin Wei, Shanghang Zhang, Huizhu Jia, Don Xie, Wen Gao, "A flexible and high-performance hardware video encoder architecture," IEEE Picture Coding Symposium(PCS), Krakow, Poland, pp.373-376, May 2012.

[5] Kaijin Wei, Rongwei Zhou, Shanghang Zhang, Huizhu Jia, Don Xie, Wen Gao, "AN OPTIMIZED HARDWARE VIDEO ENCODER FOR AVS WITH LEVEL C+," ICME, July 2012, 2012.

[6] A. D. Gupte, B. Amrutur, M. M. Mehendale, A.V. Rao, M. Budagavi, "Memory Bandwidth and Power Reduction Using Lossy Reference Frame Compression in Video Encoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, pp. 225-230, Nov. 2011.

[7] Wen Gao et al, "AVS Vide Coding Standard," *Studies in Computational Intelligence*, 2010, Volume 280, Intelligent Multimedia Communication: techniques and Applications, Pages 125-166.

[8] K. Asheesh, P. R. Panda, N. D. Dutt, and A. Nicolau, "High-level synthesis with synchronous and RAMBUS DRAMs," in *Proc. SASIMI'98*, 1998, pp. 186–193.

[9] K. Iwata et al, "A 342mW Mobile Application Processor With Full-HD Multi-Standard Video Codec and Tile-Based Address-Translation Circuits," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 1, pp. 59–68, 2010.

[10] Micron Technology, Inc. Micron 512Mb: x4, x8, x16 DDR2 SDRAM Datasheet, 2006.

[11] Kun-Bin Lee, Tzu-Chieh Lin, Chein-Wei Jen, "An Efficient Quality-Aware Memory Controller for Multimedia Platform SoC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, pp. 620-633, May. 2005.