國立交通大學

電子工程學系 電子研究所碩士班

碩 士 論 文

適用於 H.264 解碼器的可調式雙層外部記憶體管理器

A Flexible Two-Layer External Memory Management for H.264/AVC Decoder

研 究 生：張長軒

指導教授：黃 威 教授

中 華 民 國 九 十 五 年 七 月

適用於H.264解碼器的可調式雙層外部記憶體管理器

A Flexible Two-Layer External Memory Management for H.264/AVC Decoder

研 究 生：張長軒　　Student：Chang-Hsuan Chang

指導教授：黃　威 教授　　Advisor：Prof. Wei Hwang

國 立 交 通 大 學
電 子 工 程 學 系 電 子 研 究 所
碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical Engineering and Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Electronics Engineering

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年七月

適用於 H.264 解碼器的可調式雙層記憶體管理器

學生：張長軒　　　　　　　　　　指導教授：黃　威　教授

國立交通大學電子工程學系電子研究所

摘　　要

在 H.264 解碼器中有大量的資料需要去存取記憶體,外部記憶體資料存取所需的時間和功率消耗對於整個 H.264 系統的效能影響很大,因為外部記憶體受到頻寬的限制所以外部記憶體的效能對於系統仍然是個瓶頸.在 H.264 解碼器中為了達到即時播放的效果,如何設計一個良好的記憶體子系統變的十分重要.

在這篇論文中,我門針對 H.264 解碼器提出了一個計憶體子系統,這個子系統包含了一個可調變的記憶體控制器,靜態隨機存取記憶體,雙倍數同步動態隨機存取記憶體,和一條匯流排.

子系統裡面的記憶體控制器除了要確保資料傳輸的正確性以外還要能改善外部記憶體的效能,我們所提出來的記憶體控制器共分成兩層,第一層是用來做位址轉換,這一層提供了一個存取記憶體的位址產生方法,這個方法可以就由減少記憶體中誤失的數目來達到節省功率的效果.

第二層是外部記憶體介面,除了產生適當的命令給記憶體外還可以降低因為記憶體誤失所耗費的時間,這個記憶體介面可以用來控制單倍數和雙倍數同步動態隨機存取記憶體,因為這個外部記憶體介面的可調變性,使用者可以快速的把這個介面隨著不同型態的外部記憶體整合到系統裡.

這個 H.264 解碼器中的記憶體控制器可以減少資料存取延遲時間到大約 30%,而且頻寬的使用率可以從 42%提升到 51%左右.

A Two-Layer Flexible External Memory Management for H.264/AVC Decoder

Student : Chang-Hsuan Chang   Advisors : Prof. Wei Hwang

Department of Electronics Engineering & Institute of Electronics

National Chiao-Tung University

ABSTRACT

In the H.264/AVC decoder there are large amount of data need to be fetched to/from the off-chip memory. The latency of accessing data and power consumption in the off-chip memory greatly affect the performance of the whole system. The performance of the off-chip memory is still the bottleneck of the video process due to the limited bandwidth. The real-time requirement in H.264/AVC decoder at level 4 result in the request of a well designed memory sub-system.

In this thesis, we proposed a memory sub-system for the H.264/AVC decoder. This memory sub-system contains a flexible memory controller, SRAM, AHB-bus, and DDR SDRAM.

This memory controller inside the memory sub-system not only keeps the correctness of the data transfer between the module and DRAM but is also responsible to improve the performance of the external memory. The proposed memory controller consists of two layers. The first layer is address translation which provide an efficient memory map method. This layer is designed to decrease the number of row-miss status and bank-miss status. By this way the power consumption can be saved.

The second layer is external memory interface. Besides generate appropriate commands to the off-chip memory, this layer is used to increase bandwidth utilization and decrease the latency induced by the row-miss status and bank-miss status. This external memory interface is design to control the SDR SDRM and DDR SDRAM. Due to the flexibility of this EMI, the users can rapidly integrate this EMI into their system design with various kind of external memory.

The experimental results of a H.264 decoder show that the proposed controller can further reduce access latency by approximately 30% and the memory utilization is accelerated from 42 % to 51%.

Acknowledgements

我要感謝我的老師黃威教授這兩年對我的指導和鼓勵,在研究過程中提供了很多方向和指引,才讓我的研究可以順利完成,特別感謝老師能讓我同時結合記憶體和多媒體這個研究領域,讓我這兩年的研究雖然辛苦但是充滿了樂趣.

另外要特別的感謝就是跟我同一個團隊的老師,學長和同學,要謝謝蔣迪豪教授讓我加入他們研究的 H.264 (TLM)團隊.使我有機會可以接觸多媒體這一塊領域.在團隊工作期間,成功大學的李國君教授給了我門團隊很多的指導,我們的團隊的主導人彭文孝學長,日前已經到資工系當教授了,更是提供了很多不同的方向讓我們來思考,從學長的身上學到很多東西.

另外要特別由衷感謝給我最多幫助的同一個團隊的博班學長李志鴻和陳治傑同學,在研究過程給了我相當多的支援與協助,才使得我的論文可以順利完成

接下來要感謝同一個實驗室的黃柏蒼學長,華重憲學長,和張銘宏學長,在我的研究過程幫助了我很多也教導了我很多,從學長身上得到很多寶貴的建議.

最後要感謝我的家人和朋友在研究過程給我的打氣與鼓勵以及關心,讓我的研究過程雖然辛苦但還是充滿歡樂.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background

Multimedia processing technologies have been widely applied in many systems. These technologies have not only provided existing applications like desktop video/audio but also spawned brand new industries and services like digital video recording, video-on-demand services, high-definition TV, etc. To support complex multimedia applications, architectures of multimedia systems must provide high computing power and high data bandwidth. Furthermore, a multimedia operation system should support real-time scheduling.

Previous researches have shown that data transfer and storage of multidimensional array signals dominate the performance and power consumption in system-on-a-chip (SoC) designs. In particular, the data transfer to off-chip memory is especially important due to the scarce resource of off-chip bandwidth. In fact, although the tremendous progress in VLSI technology provides an ever-increasing number of transistors and routing resource on a single chip, and hence allows integrating heterogeneous control and computing functions to realize SoCs, the improvement on off-chip communication is limited due to the number of available input/output (I/O) pins and the physical design issues of these pins. As many recent studies have shown, the off-chip memory system is one of the primary performance bottlenecks in current systems.

1.2 Motivation

As the resolution of video-processing applications becomes high, video signal processors should deal with a large amount of data within a tightly bounded time. Due to the large quantities, video data are stored in off-chip memories that are usually slow, and thus the system performance strongly depends on the memory bandwidth between processors and external memories. For example, in decoding MP@HL video streams, an MPEG-2 video decoder must feature 10-MB memory as frame buffers and should access about 370 million data/s.

H.264/AVC is the newest international video coding standard. Relative to prior video coding methods such as MPEG-2 video, H.264/AVC has the higher coding efficiency. With an increasing number of services and growing popularity of high definition TV are creating greater request for higher coding efficiency. Moreover, other transmission media such as Cable Modem, XDSL, or UMTS offer much lower data rates than broadcast channels, and enhanced coding efficiency can enable the transmission of more video channels or higher quality video representations within existing digital transmission capacities.

In the H.264 decoder, it process 245760 MB/s in maximum at level 4. In order to support the complex mode in the H.264 decoder, there are even large data transfer between H.264 decoder and external memory.

The issue of latency, bandwidth utilization, and power consumption are very important in the video process. These issues in the video process are greatly influenced by the memory subsystem. Hence a memory controller that can efficiently communicate with external memory is the most significant part over the entire video decoding system. Within the video decoding system, motion compensation always dominates the total amount of data transmission especially when SDRAM is adopted as external frame memories. How to manage the data transfer and utilize the limited bandwidth is the most important issue of the memory controller.

1.3 Organization

The organization of this thesis is as follows. An overview of external memory organization is introduced in Chapter 2. Besides the DRAM architecture and basic operation the past DRAM controller will be described. The DRAM development and DRAM trend are also discussed here.

My memory controller is applied in the H.264 decoder. This H.264 decoder is designed in transaction level model (TLM). The concept of TLM and the TLM model of our H.264 decoder will be thoroughly explained in Chapter 3. The role of my controller in this H.264 is mentioned here. There are two layers in my memory controller. Layer 0 is used to control the memory. Layer 1 is used to improve the performance in video process.

Chapter 4 presents layer 0 of my memory controller. This layer is called external memory interface (EMI). The system can communicate with external memory by using this EMI. The flexibility if this EMI can help users to integrate this EMI to their design easily. The different setting of EMI will influence the performance. The improvement on performance including latency and power will be compared here. The experimental results of different setting will also be listed in this chapter.

The whole memory subsystem and the layer 1 of my memory controller which is called address translation machine are proposed in Chapter 5. The H.264 decoder needs large amount of data transfer. The efficiency of the memory controller is very important. To deal with the bandwidth loss which degrades the performance, the layer 1 is combined into my memory controller. We have proposed a memory map method by using the characteristic of the H.264 process, and Layer 1 is used to do the address translation. This address translation machine not only reduces the power consumption on the bus line but also increases the bandwidth utilization. Finally, the conclusions and future work will be discussed in Chapter 6.

Chapter 2
Overview of External Memory Organization

In this chapter, it introduces the overview of external memory organization for video process. Firstly, DRAM characteristic is described in section 2.1. Then, section 2.2 discussed the techniques which were proposed in the past used in memory controller. In addition, the design trends of modern DRAM is presented in section 2.3.

2.1 DRAM characteristic

2.1.1 DRAM architecture

DRAM architecture is usually composed of the data memories, address decoders, row buffer, mode register, data buffer. Fig. 2.1 shows a simplified block diagram

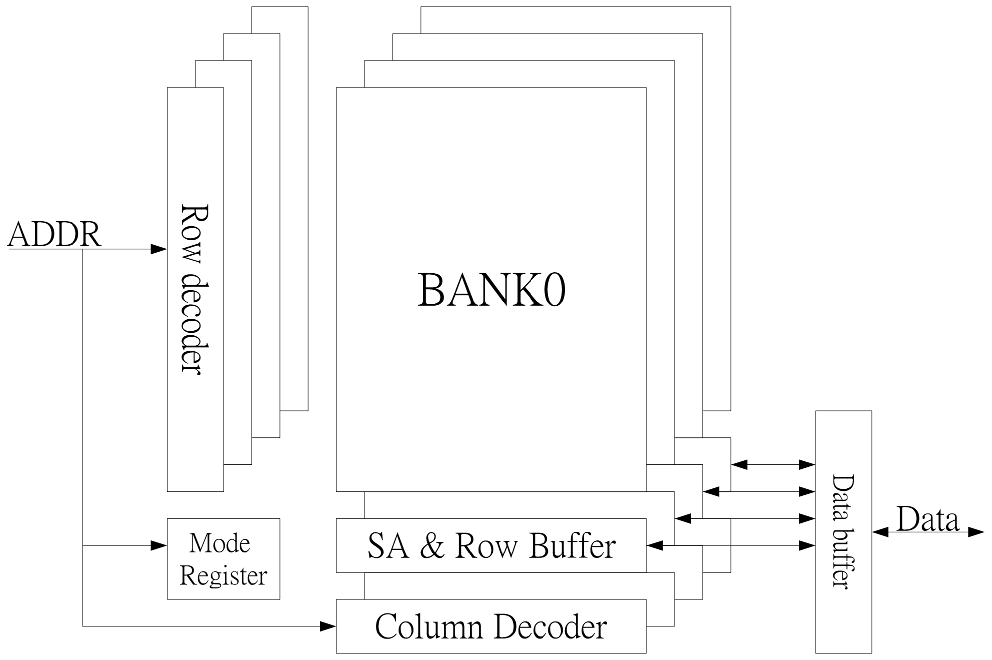


Fig. 2.1 Simplified architecture of a DRAM.

Four bank share the address bus and command bus. Each bank has its own row decoder, column decoder, and sense amplifier. The mode register stores the DRAM operation mode, including burst length (BL), column address strobe latency (CL), and burst type, etc. Users can set the value of the mode register through address bus with proper command.

2.1.2 DRAM command and operation

The normal commands and its operation used in DRAM will be introduced as follow.

NO OPERATION (NOP):

The NOP command can prevent unwanted commands from being registered during idle or wait states. Operations already in progress are not affected.

ACTIVE:

This command is used to open a row in a particular bank. The row remains open for accesses until a PRECHARGE command is issued to that bank.

READ/WRITE:

The read/write command is used to initiate a read/write access to an active row, if auto precharge is selected, the row being accessed will be closed at the end of read.

PRECHARGE:

The precharge command is used to deactivate the open row in a particular bamk. The bank will be available for a subsequent row access a specified time (tRP)

REFRESH:

The refresh command can be used to retain data in the DRAM.

A memory access operation, which simplified state diagram is depicted in Fig. 2.2, contains three operation including row activation (ACTIVE), column access (read/ write), and precharge.

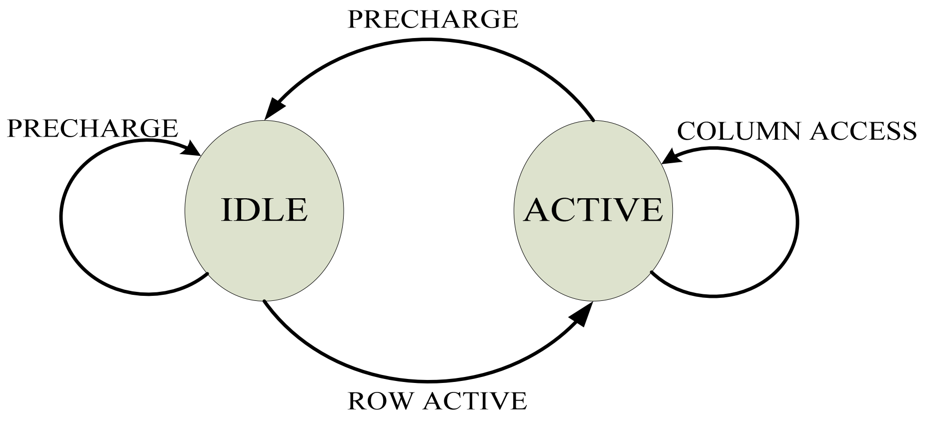


Fig. 2.2 Bank state diagram.

The active command opens a particular row in one of the bank, and copies the row data into the row buffer. The active command needs a latency period called tRCD to accomplish this operation. Then, after tRCD delay a column access command (read / write) can be issued to sequential access data or single data according to the burst length and burst type set in the mode register. During the tRCD time, no other commands can be issued to the bank. However, commands to other banks are permissible due to the parallel processing capability of each bank. For read operation the valid data-out from the starting column address will be available following the CAS latency after the read command, as shown in Fig. 2.3. For write command in SDRAM, the first data-in is coincident with the write command, as shown in Fig. 2.4. For write command in DDR DRAM, the first data-in is sent to DQ [1] strobe after a DQS [1] delay, as shown in Fig. 2.5. Finally a precharge command must be issued before opening a different row in the same bank.

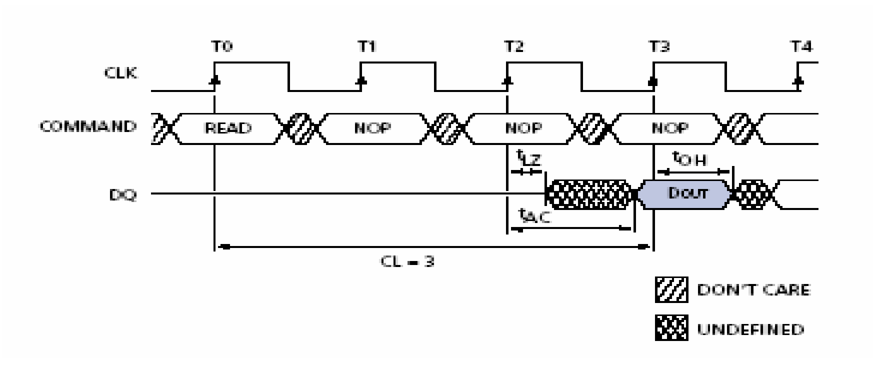


Fig. 2.3 Read command CAS latency.

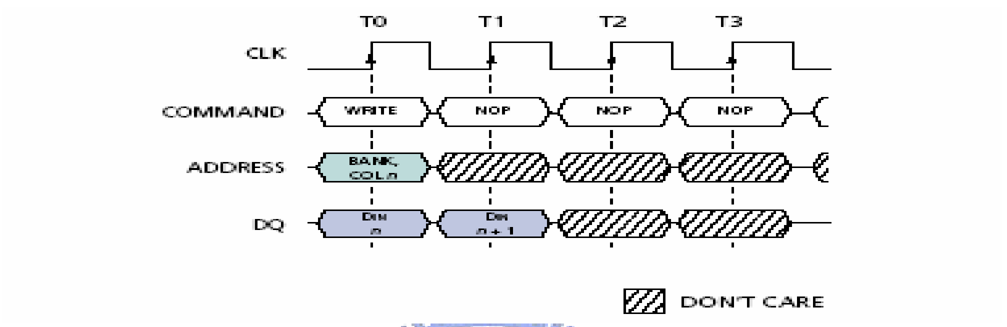


Fig. 2.4 Write command (SDRAM).

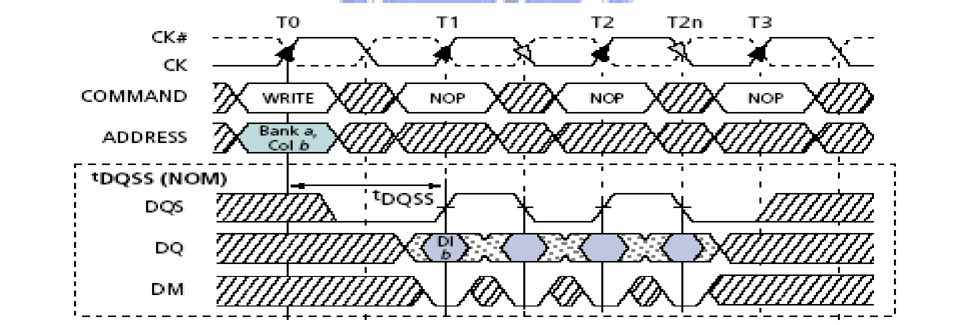


Fig. 2.5 Write command (DDR).

2.2 Past DRAM controller techniques

In the design of system-on-a-chip such as portable wireless device and multimedia system, several factor such as increased system complexity, time-to-market pressure, cost effectiveness, and various functionality requirement have made the trend of system-on-a-chip design indispensable. In general, SoC device are connect to the off-chip memory that store data to be transferred between functional blocks. As the SoC integrate more functional block and need high performance, high data bandwidth is required to meet a given system specification.

Similarly Because of the tremendous data transfer and storage in video process, software or hardware must provide high data bandwidth to achieve the real-time request in multimedia system. External memory has the largest data capacity so it is often used as the frame memory in the video process. Nevertheless, the data transfer to off-chip memory is bound to the limited bandwidth.

Many external memory controllers have been proposed to improve the memory bandwidth utilization and achieve efficient memory access. In this section, some important techniques used in memory controller will be introduced.

2.2.1 Techniques and Improvement

According to the environment, the controllers can be categorized into two classes: single channel and multiple channel environments. For single channel environment, Rixner memory access scheduler [2] reorder the access addresses from each bank controller and sends command to DRAM through arbiter. It can reduce the latency after reorder the address. However, the output command may be out-of-order, many command FIFOs and extra circuits are required to reorder commands and addresses.

Miura dynamic-SDRAM-mode-control scheme [3] eliminates the above disadvantage and it can both reduce operating current and the latency of an SDRAM. Nevertheless, it only supports scheduling of single-channel. For multi-channel environment, Lee advances a quality-aware memory controller [4]. It supports different scheduling policies according to the current channel situation. These memory controllers mainly focus on general-purposed environment.

Concerning the particular-purpose memory controllers for the video codec application, several papers have been proposed on improvement of performance. They focus on the power consumption, latency (speed), and bandwidth utilization.

Kim memory interface architecture [5] reorganizes data arrangement in SDRAM The interface reduces energy consumption and increase memory bandwidth by placing the data in the same block in the same row. Thus, when accessing the block, the row-hit rate increases. The time and power waste on pre-charging the rows are decreased. Fig. 2.6 (a) is the original placement, and Fig. 2.6 (b) is the new placement after data arrangement.

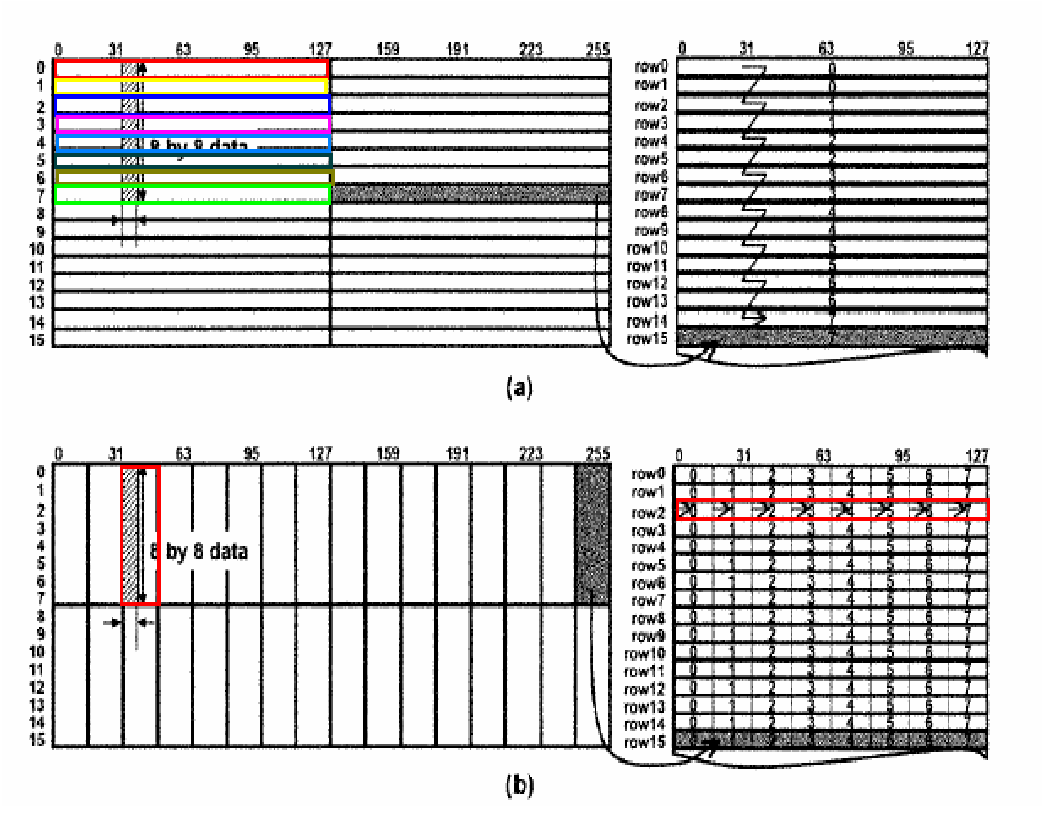


Fig. 2.6 Data placement

Park history-based memory mode control [6] reduce row-miss rate to achieve 23.3 % reduced energy consumption and 18.8 % reduced memory latency. It uses history-based prediction to predict the next command is row-hit or row-miss. The prediction is implemented by a finite sate machine, as shown is Fig. 2.7. It will pre-charge the current bank if it predict the next command is row-miss, and it will let the current row stay in the active state if it predict the next command is row-hit. The proposed memory controller is shown in Fig. 2.8.

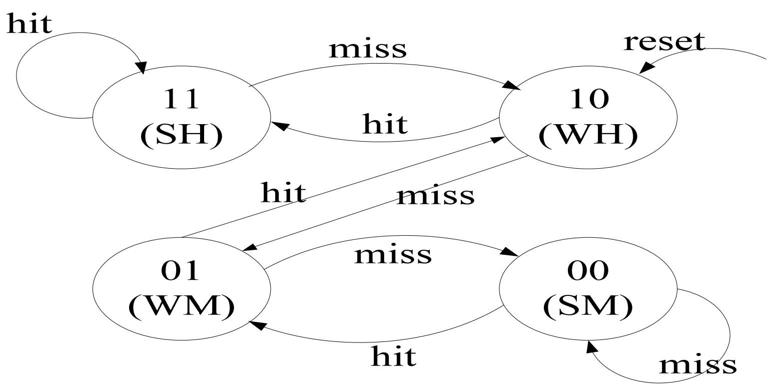


Fig. 2.7 Mode control prediction

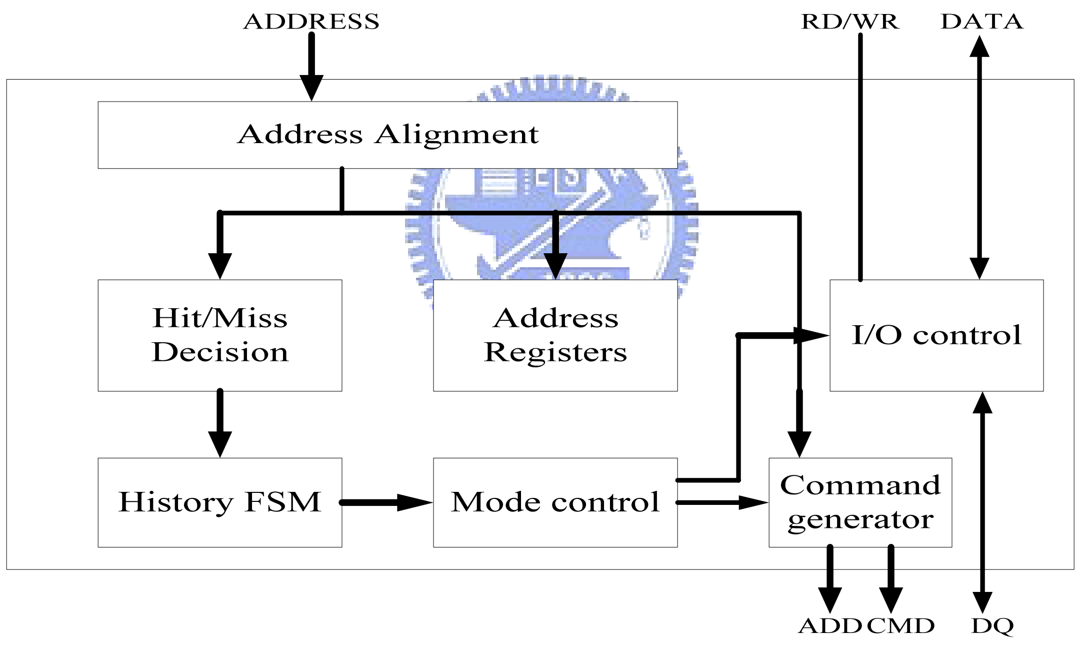


Fig. 2.8 Mode control memory controller

Zhu SDRAM controller in H.264 HDTV Decoder [7] focus on memory mapping and data arrangement in SDRAM to reduce the row active cycle, it also improves throughput and provide less power consumption. However, it adopts auto precharge rather than manual pre-charge leads to some loss of bus bandwidth and increase the access latency.

Heithecker proposed a mixed QOS SDRAM controller [8]. The controller uses two types of scheduler to do the memory access. It can improve the memory bandwidth and latency for multiple access streams with different access sequence types running in parallel. Fig. 2.9 shows the overall controller structure with two scheduler variants.

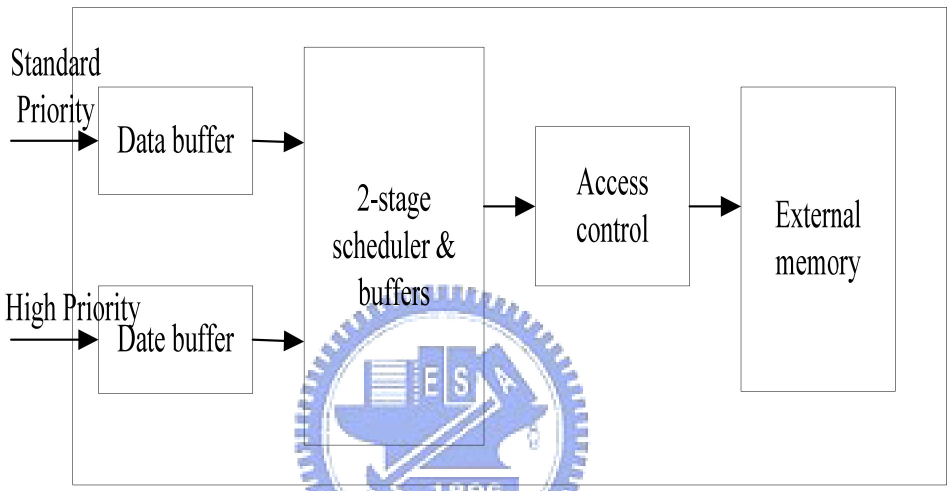


Fig. 2.9 2-stage scheduler memory controller

Kang proposed a memory controller in the MPEG-4 AVC/H.264 decoder [9]. It uses a dual memory controller and dual bus to improve the memory bandwidth

The above memory control technique concentrate on three part, including memory scheduler, mode control, and data arrangement in SDRAM. The above discussion of related controllers and its techniques is summarized in Table 2.1.

Besides the techniques mentioned above, there are still other techniques can apply on the video process such as frame recompression and frame memory reorganization. Frame recompression is recompressing data before storing to memory, and decompression is required when reading data from memory, as shown in Fig. 2.10.

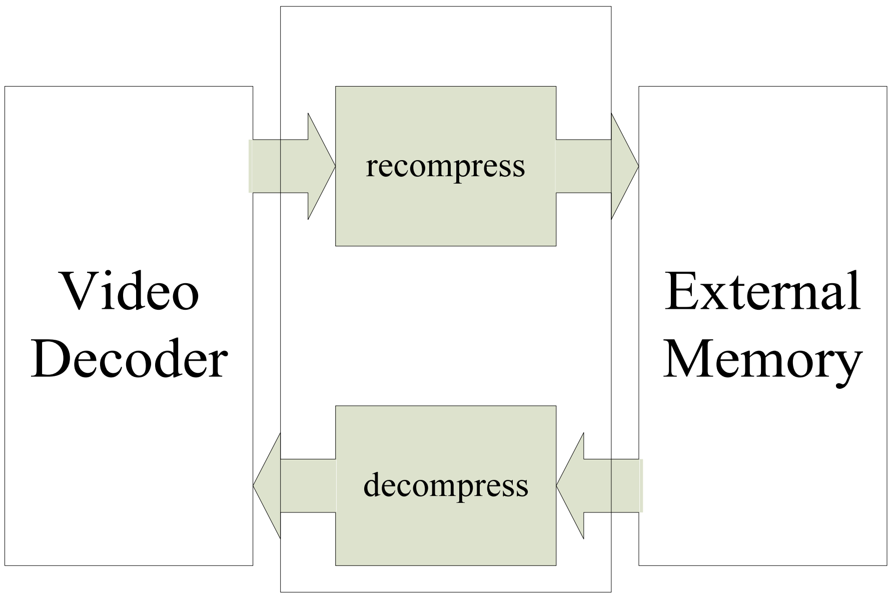


Fig. 2.10 recompression method

In this respect, many algorithms, such as Tajime [10] 2-D adaptive DPCM in pixel domain, and Lee [11] modified Hadamard transform and Golomb-Rice (GR) coding, etc have been proposed. However, frame recompression method need extra circuit and require additional execution cycles to compress data such that the throughput of video decoder degrades. For the memory reorganization can be found in De Greef [12]. Beside, Interuniversity MicroElectronic Center (IMEC) widely exploited this idea to H.264 decoder system [13], MPEG-4 motion estimation [14] and video decode [15]. The concept of memory hierarchy [16] combined with merging structured frame memory can achieve data reuse and reduce the redundancy if data access. However, they only focus on C level simulation. If we want implement on ASIC design, many issues still have to be overcome. For advanced development, Chang combined frame memory architecture [17] can reduce frame memory size up to 57 % and reduce up to 83 % average latency.

Table 2.1 Related works of SDRAM memory controller

Related	Application	Improve	Technique
Rixner	General-purpose single-channel	Bandwidth Latency	Memory schedule
Miura	32 bit RISC CPU single channel STB	Latency Power	Memory mode control
Lee	Multimedia SoC multiple-channel STB	Bandwidth Latency	Memory schedule
Kim	Video single channel	Power Bandwidth	Data arrangement
Park	HDTV decoder Multiple-channel	Latency power	Memory mode control
Zhu	H.264 decoder Multiple-channel	Bandwidth utilization Decoding throughput	Data arrangement
Heither	Image process Multiple-channel	Bandwidth utilization Latency	schedule
Kang	H.264 decoder Multiple-channel	Bandwidth utilization	Dual controller & dual bus

2.3 Modern DRAM Development

From the day DRAM has been invented, the requests of performance accelerate very fast. The most important issues are bandwidth, latency, and power. This section will introduce the development of DRAM that improve the performance and the future trend of DRAM.

2.3.1 Bandwidth

The improvement of DRAM bandwidth has never satisfied the increasingly complicated application such as multimedia and 3D processing. To fulfill the demand for high bandwidth, various new DRAM specifications have been announced by DRAM manufacturers. The SDRAM standards supported by JEDEC [18] have become the mainstream of DRAM market. Several techniques have been applied on the latest standards announced by JEDED to provide users higher bandwidth.

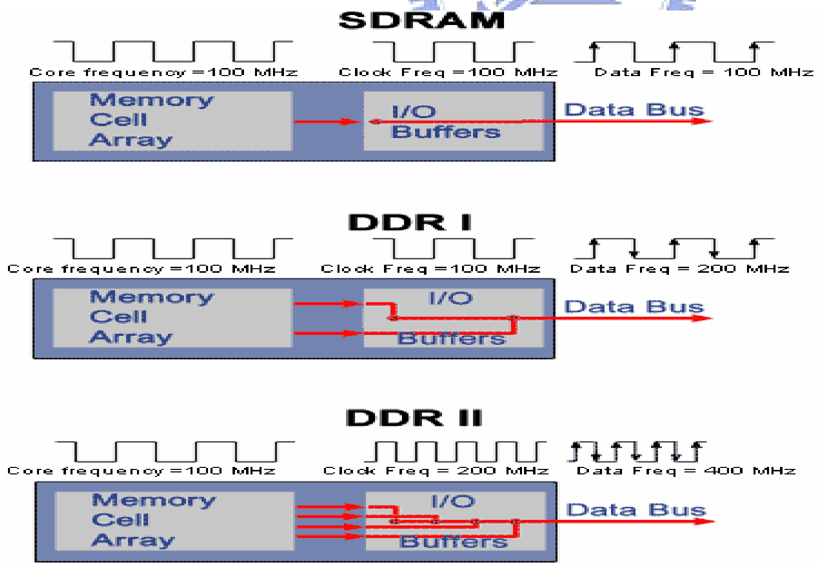


Fig. 2.11 Operating frequency of SDR, DDR-1, DDR-2

Fig. 2.11 shows the operating frequency of each component of SDR, DDR-1, and DDR-2. As we can see, in SDR, the core and the I/O are running at the same frequency, and the data is transferred at the positive edge of clock. In DDR-1, a 2-bit data PREFETCH technique is applied. Data is transferred at positive and negative edge of clock. The data rate of DDR-1 is twice as SDR. In DDR-2, the PREFETCH technique is further promoted up to 4-bit. The I/O frequency is twice as DDR-1, so DDR-2 can provide bandwidth twice as DDR-1 can. The PREFETCH technique makes DRAM be able to provide quadruple bandwidth than SDR with core frequency remains unchanged.

In SDR and DDR-1, row activation takes a period of time called tRCD before column access command can be issued. This may induce overhead due to the bus confliction. For example, Fig. 2.12 shows the difference between DDR and DDR-2. The top part is the timing diagram of DDR. Because of the bus confliction The third active command can not be issued at 4th cycle. It must wait a cycle therefore make a bubble on the data bus. The bottom part of Fig. 2.12 is the timing diagram of DDR-2. DDR-2 allows users to issue subsequent read command right after an active command. The read command will be buffered inside DDR-2 and will not be processed until the tRCD latency is reached. Thus the bus confliction is prevented, the bubble is eliminated and the bandwidth utilization becomes better.

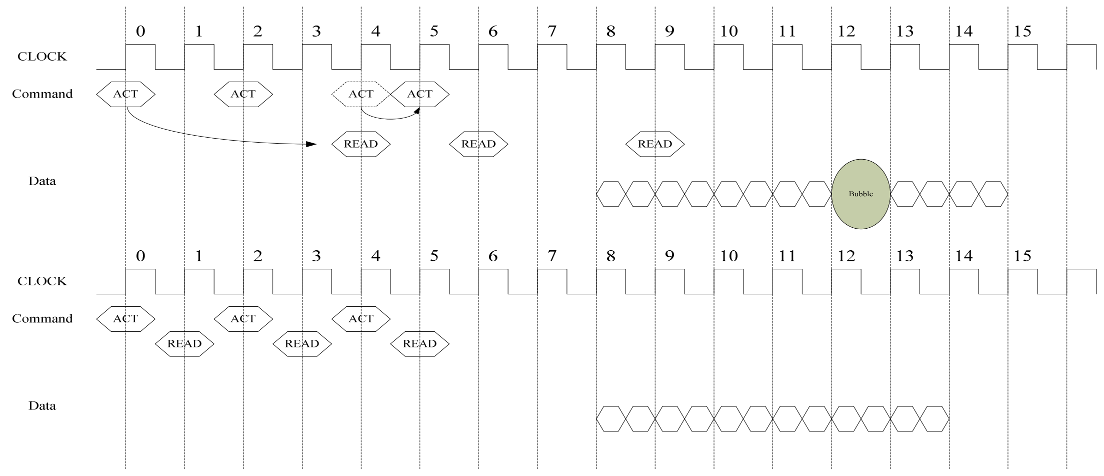


Fig. 2.12 Timing diagram of DDR & DDR-2

2.3.2 Latency

The DRAM response latency can directly influence the speed of the whole system. The speed of the system for the multimedia process is very essential to achieve the real-time request. So if the DRAM latency is shorter, the whole system can boost its performance. However, the situation is not as we expected. Fig. 2.13 compares the performance trend of CPU and DRAM. While CPU clock speed increases 7.65 times, DRAM latency also has a 4.6 times increase. The improvement of CPU is much faster than the improvement of DRAM. Long response latency waste its processing power on waiting and the performance is limited.

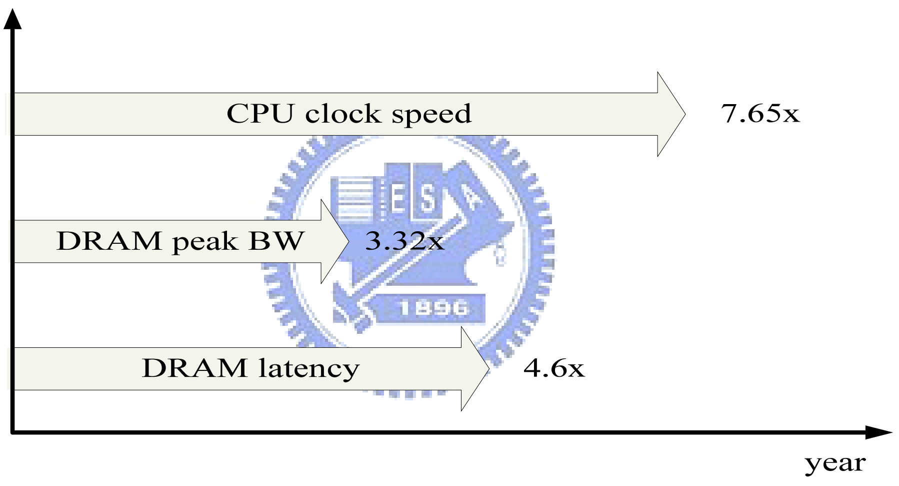


Fig. 2.13 CPU VS DRAM performance

Since Year 2000, DRAM manufacturers have begun to pay attention to the impact of DRAM latency and proposed several solution including RLDRAM (reduce latency DRAM) [19], FCRAM (Fast cycle RAM) [20], and Network DRAM [21] etc. In the following we will introduce RLDRAM for example.

RLDRAM-1 which mainly aimed at application requiring short latency and high bandwidth was announced by micron in 2001. RLDRAM-2 was announced in 2003 and can be taken as an enhanced version of RLDRAM-1. Some features if RLDRAM-2 are listed as followed.

The traditional DRAM is divided into 4 bank while RLDRAM can divide into 8 banks. The successive accesses to the same bank cost more latency than the successive accesses to the different banks, shows in Fig. 2.14 and Fig. 2.15. So if the number of banks increases, the rate of accessing different banks increases. So the latency of RLDRAM is shorter than the traditional DRAM. On the other hand, Row cycle time (tRC) defined the shortest time interval needed for two successive ‘active’ commands addressed to the same bank. The row cycle time is 20ns in RLDRAM and 72ns in RDRAM respectively. When the row miss happens, RLDRAM can apparently save much access latency.

RLDRAM has another advantage of saving latency. It separates data bus for read and write data, RLDRAM-2 can effectively reduce the latency caused by the turnaround cycles.

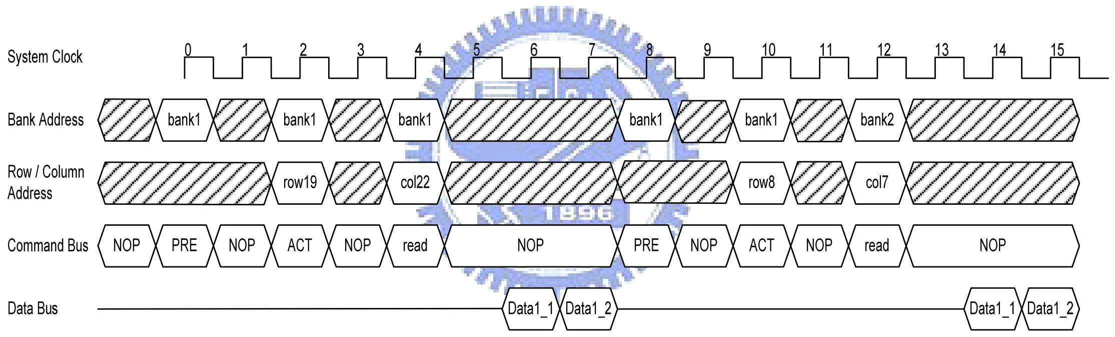


Fig. 2.14 Accesses addressed to same bank

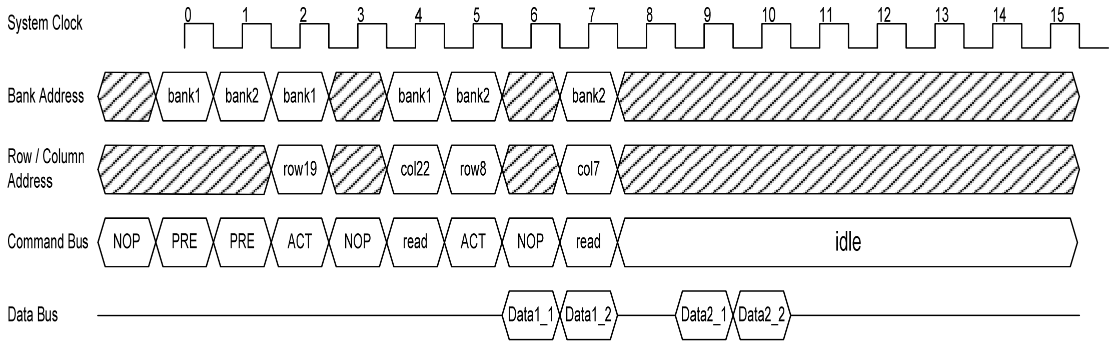


Fig. 2.15 Accesses addressed to different bank

2.3.3 Power

In many application of portable wireless devices such as mobile and PDA, power consumption is the significant issue because of battery life is limited. With the application of multimedia becomes popular, the request of memory size is larger. Accordingly, the designers often select DRAM to be the body memory component.

In order to reduce the power of DRAM, many products have been invented for low power such as BAT-RAM from micron [22] and Mobile-RAM from Infineon [23]. The low-power DRAM has some special features inside.

Low Operating voltage

Compare with SDR SDRAM, the operating voltage of low-power DRAM is lowered from 3.3v to 1.8v. Thus, the power consumption can significantly decreases.

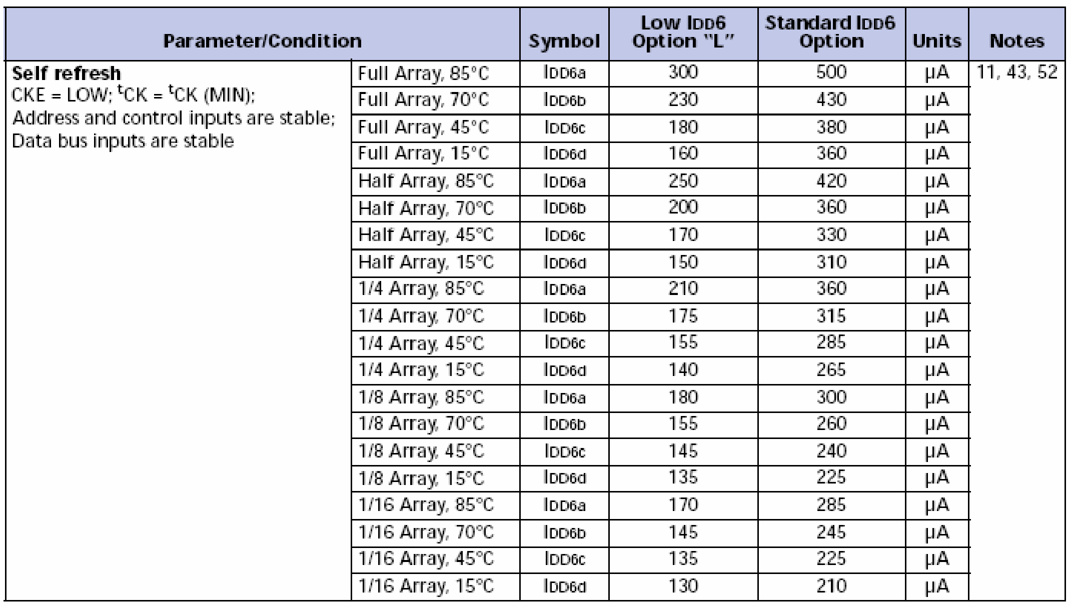
Output Driver Strength

Because the low-power DRAM is designed for use in smaller systems that are typically point-to-point connection, an option to control the drive strength of the output buffers is provided. Drive strength should be selected based on expected loading of the memory bus. There are four allowable setting for the output drivers, including full strength driver, half strength driver, quarter strength driver, and one-eighth strength driver.

Temperature Compensated Self Refresh (TCSR)

Most of the time mobile devices stay in standby mode and DRAM can enter sef refresh mode to save unnecessary power consumption. In the self-refresh mode, DRAM will refresh the data stored in the DRAM cell. The refresh period is inversely proportional to temperature, traditional DRAM can only support single refresh period which is the worst condition. In the low-power DRAM, a temperature sensor is implemented for auto control of the self refresh oscillator on the device. Therefore, the refresh current is decreasing while the temperature is low, as shown in table 2.2.

Table 2.2 Current of different condition

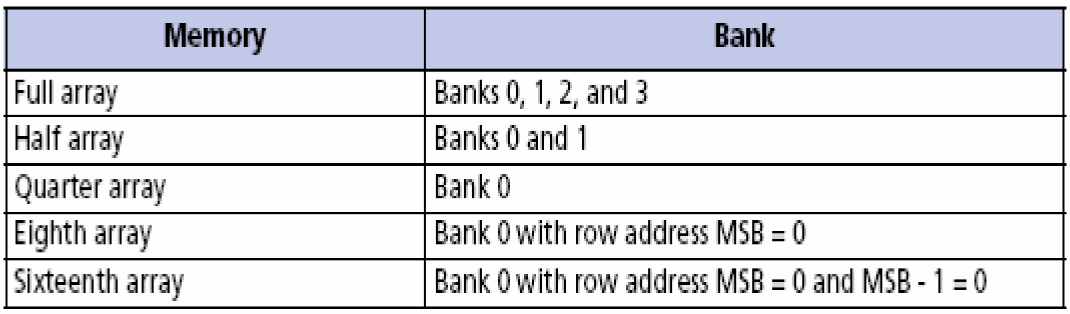


Parameter/Condition		Symbol	Low I_{DD6} Option "L"	Standard I_{DD6} Option	Units	Notes
Self refresh CKE = LOW; $^tCK = {}^tCK$ (MIN); Address and control inputs are stable; Data bus inputs are stable	Full Array, 85°C	I_{DD6a}	300	500	μA	11, 43, 52
	Full Array, 70°C	I_{DD6b}	230	430	μA	
	Full Array, 45°C	I_{DD6c}	180	380	μA	
	Full Array, 15°C	I_{DD6d}	160	360	μA	
	Half Array, 85°C	I_{DD6a}	250	420	μA	
	Half Array, 70°C	I_{DD6b}	200	360	μA	
	Half Array, 45°C	I_{DD6c}	170	330	μA	
	Half Array, 15°C	I_{DD6d}	150	310	μA	
	1/4 Array, 85°C	I_{DD6a}	210	360	μA	
	1/4 Array, 70°C	I_{DD6b}	175	315	μA	
	1/4 Array, 45°C	I_{DD6c}	155	285	μA	
	1/4 Array, 15°C	I_{DD6d}	140	265	μA	
	1/8 Array, 85°C	I_{DD6a}	180	300	μA	
	1/8 Array, 70°C	I_{DD6b}	155	260	μA	
	1/8 Array, 45°C	I_{DD6c}	145	240	μA	
	1/8 Array, 15°C	I_{DD6d}	135	225	μA	
	1/16 Array, 85°C	I_{DD6a}	170	285	μA	
	1/16 Array, 70°C	I_{DD6b}	145	245	μA	
	1/16 Array, 45°C	I_{DD6c}	135	225	μA	
	1/16 Array, 15°C	I_{DD6d}	130	210	μA	

Partial Array Self Refresh

For further power savings during SELF REFRESH, the PASR feature enables the control to select the amount of memory that will be refreshed during SELF REFRESH. The option is shown in Table 2.3. The current can be reduced as shown in Table 2.1.

Table 2.3 Amount of memory will be refreshed



Memory	Bank
Full array	Banks 0, 1, 2, and 3
Half array	Banks 0 and 1
Quarter array	Bank 0
Eighth array	Bank 0 with row address MSB = 0
Sixteenth array	Bank 0 with row address MSB = 0 and MSB - 1 = 0

Stopping the external clock

One method of controlling the power efficiency in applications is to throttle the clock that controls the SDRAM. There are two basic ways to control the clock:

1. Change the clock frequency, when the data transfers require a different rate of speed.
2. Stopping the clock altogether.

Both of these are specific to the application and its requirements and both allow power savings due to possible fewer transitions on the clock path.

The clock can also be stopped altogether if there are no data accesses in progress, either WRITE or READ, that would be affected by this change; i.e., if a WRITE or a READ is in progress, the entire data burst must be through the pipeline prior to stopping the clock.

For the full duration of the clock stop mode. One clock cycle and at least one NOP is required after the clock is restarted before a valid command can be issued. Fig. 2.16 illustrates the clock stop mode.

It is recommended that the DRAM be in a pre-charged state if any changes to the clock frequency are expected. This will eliminate timing violations that may otherwise occur during normal operations.

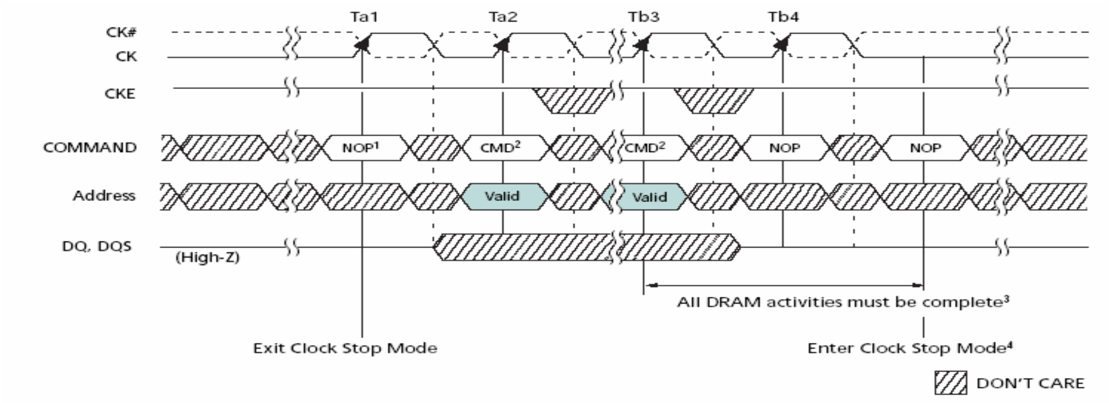


Fig. 2.16 Clock stop mode

Power-Down

Power down can occurs when all banks idle, this mode is referred to as precharge power-down. If power down occurs when there is a row active in the bank, this mode is referred as active power-down. Entering power-down mode deactivates all input and output buffers, therefore the power is saved.

Deep Power-Down

Deep power down is an operating mode used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained when the device enters power-down mode. Since DRAM is often used as temporary data buffers, enter DPD mode while the device is in standby mode won’t cause any loss.

2.4 Future trend of DRAM

DDR-3

Beside DDR-2, the draft of DDR-3 industry standard has been announced by JEDEC in Oct 2002. DDR-3 SDRAM which is due to enter volume production in 2006, operates from a 1.5 volts supply and can transfer data at up to 1,066-Mbits per second. That's twice as fast as DDR2, which is just coming into the market in volume, and four times the speed of DDR. Samsung said it plans to make the chips using an 80 nanometer manufacturing process and, citing data from research house IDC, believes DDR3 will account for 65 percent of the market in 2009.

DDR3 has some features that can improve the performance and more cost-effective [24]. In the 8b-prefetch data-path with 3-stage pipelining, a newly devised hybrid-type latency control scheme and a 2-step multiplexing can proficiently handle maximum 128b parallel data in the case of x16 configuration. An efficient protocol for the temperature read-out is proposed, supporting CPU while it controls the heat in high-speed operations. Per-bank-refresh is another experimental feature of this prototype, virtually removing the loss of the memory bandwidth due to the unavoidable requirement of refresh operation common in all DRAM.

Embedded DRAM

Embedded DRAM (EDRAM) technology has significant advantage in terms of performance, area, bandwidth, and power consumption by combining a high bandwidth DRAM macro with logic/analog circuits on the same chip.

Embedded DRAM (EDRAM) macros have been proposed as a way to achieve the low power and wide bandwidth required by graphic controllers, network systems, and mobile systems. To give an example; Advanced 3D graphics (3DG) technology will be used in console game machines [25], and it is desired to develop a rendering controller chip which can handle real time 3D animation with true colors. Embedded DRAM (EDRAM) [25] technology attracts attention of the 3DG systems, because only EDRAM can satisfy the required data rate. Fig 2.17 shows the trend of 3DG controller performance.

In the portable device such as mobile and PDA, the power consumption is a very important issue. In off-chip design, the power consumption of off-chip interconnection is larger one or two orders of magnitude than that of standalone DRAM itself. In SOC design, the situation is changed. The power consumption of on-chip interconnection is reduced to the same order of that of EDRAM because on-chip I/O capacitance is tremendously reduced. Therefore the power consumption of connection can be saved.

Although embedded DRAM technology has significant advantage in SOC design because of its performance such as low-power consumption and high bandwidth. However, process gap between commodity DRAM and logic make it difficulty to make highly reliable EDRAM product with low cost, and high yield. There are still other problems need to overcome in the design of EDRAM. But the implementation of EDRAM technology would be a trend in the design of SOC.

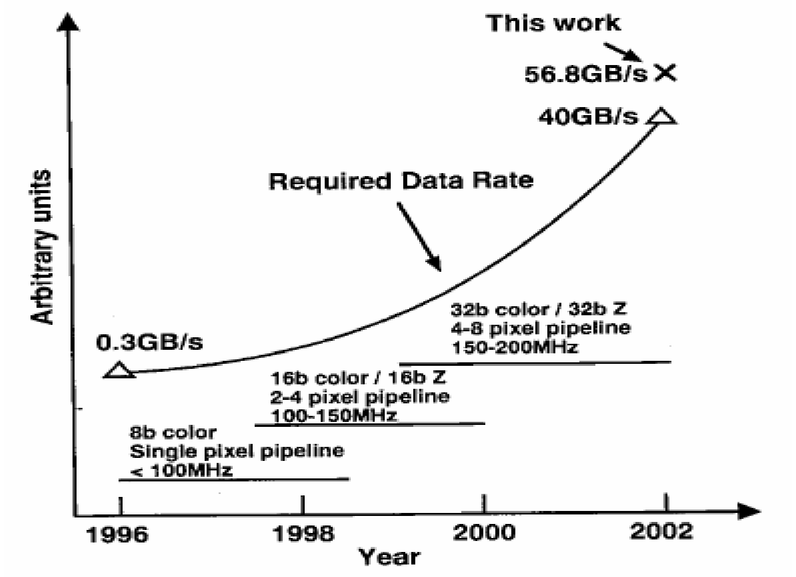


Fig. 2.17 The required data rate in 3DG engine

Chapter 3
Transaction Level Modeling of H.264/AVC Decoder

The proposed controller is designed for the video process which need large amount of data storage. We have a project to design a hardware architecture for the H.264 decoder that confirms to high profile at Level 4 (HP@L4). As a result of the complexity of the H.264 decoder, it needs a memory controller to communicate with external memory efficiently. We have a cooperation to integrate my memory controller into the H.264 decoder system design. The characteristic of H.264 and the system design of this H.264 decoder will be introduced in this chapter. The role of my memory control in this H.264 decoder system will also be described here.

3.1 Overview of H.264

3.1.1 Introduction

H.264/AVC is the newest international video coding standard. Relative to prior video coding methods such as MPEG-2 video, H.264/AVC has the higher coding efficiency. With an increasing number of services and growing popularity of high definition TV are creating greater needs for higher coding efficiency. Moreover, other transmission media such as Cable Modem, XDSL, or UMTS offer much lower data rates than broadcast channels, and enhanced coding efficiency can enable the transmission of more video channels or higher quality video representations within existing digital transmission capacities.

The scope of the standardization is illustrated in Fig. 3.1, which shows the typical video coding/decoding chain (excluding the transport or storage of the video signal). Only the central decoder is standardized. By imposing restrictions on the bit-stream and syntax, and defining the decoding process of the syntax elements such that every decoder conforming to the standard will produce similar output when given an encoded bit-stream that conforms to the constraints of the standard.

The new standard is designed for many application areas such as the following example.

• Broadcast over cable, satellite, cable modem, DSL, terrestrial, etc.

• Interactive or serial storage on optical and magnetic devices, DVD, etc.

•Conversational services over ISDN, Ethernet, LAN, DSL, wireless and mobile networks, modems, etc.

• Video-on-demand or multimedia streaming services over ISDN, cable modem, DSL, LAN, wireless networks, etc.

• Multimedia messaging services (MMS) over ISDN, DSL, LAN, wireless and mobile networks, etc.

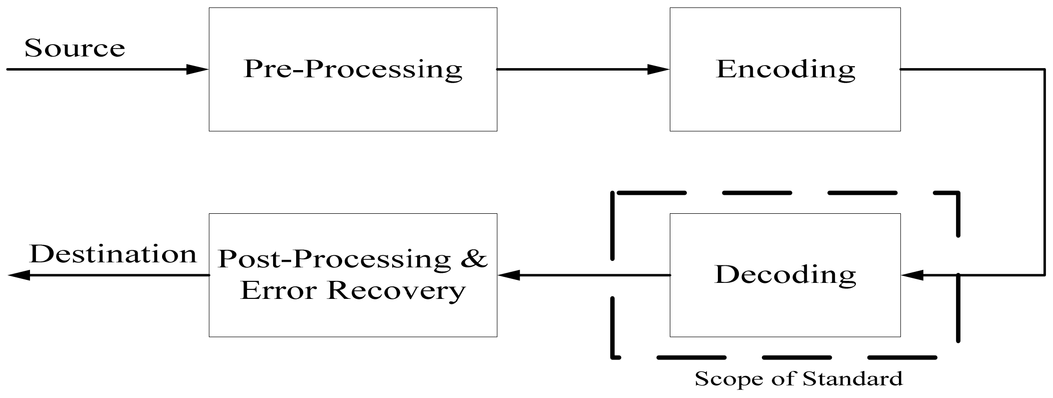


Fig. 3.1 Scope of video coding standardization

3.1.2 Characteristic of H.264

The coding gain of H.264 is achieved by efficiently exploiting spatial and temporal redundancy. For better temporal prediction, new coding tools such as long-term prediction, multiple reference frames, motion compensation with variable block size, in-loop filter, and 1/4-pel motion compensation are developed. In addition, for exploiting spatial redundancy, an intra prediction technique is adopted. Further, to reduce bit rate, a context-adaptive entropy coder is deployed. The following briefly summarizes the features of each coding tool.

• **Long-Term Prediction:** The prediction of a picture can refer to a prior coded picture that is not right before the current one. For sequences with periodic content, long-term prediction offers coding gain by having more flexibility on the selection of reference picture.

• **Motion Compensation with Variable Block Size and Multiple Reference Frames:** Motion compensation can be done by partitioning a macroblock into a few number of sub-blocks and each sub-block can refer to a larger number of pictures that have been coded and stored. The features of variable block size and multiple reference frames offer better trade-off between texture and motion information as well as better adaptation for macroblocks with varying characteristics.

• **1/4-pel and 1/8-pel Motion Compensation:** The prediction can come from 1/4-pel samples (or 1/8-pel for chroma) that are generated by using the interpolation with full-pel samples as input. The sub-pel motion compensation with higher accuracy improves the prediction efficiency by reducing the aliasing from sampling.

• **Intra Prediction:** An intra-coded block can be predicted from the edges of the adjacent and previously-coded blocks. Particularly, the prediction can come from different directions.

• **Transform with Variable Block Size:** The 4x4 integer transform and 8x8 DCT transform can be adaptively selected for a macroblock. The 4x4 integer transform can remove ringing artifact while the 8x8 DCT provides higher coding efficiency for smooth area. In addition, a double transform could be applied for the DC coefficients belonging to the 16 4x4 blocks within a macroblock.

• **Context-Adaptive Entropy Coding:** The entropy coding is done in a context-adaptive manner. The value of prior coded syntax elements (or bins) could be used to select the probability model or table for the coding of following syntax elements (or bins). Higher coding efficiency is achieved by using conditional probability models.

• **In-loop De-Blocking Filter:** A de-blocking filter is placed in the prediction loop to remove the blocking artifact for the reference picture so as to improve the quality of the reference picture and prediction efficiency.

While more correlations are used for coding, it suggests that stronger data dependency exhibit between successive computations and more buffers are required. Moreover, the very different types of predictors imply that intensive computations are inevitable. Also, the heterogeneous building blocks and operations bring new challenges to a system design such as synchronization, data flow control, error handling, buffering, software/hardware concurrency, and so on. With these design challenges, the SoC implementation for H.264 codec becomes much more difficult than prior coding standards. Due to the complexity of H.264, a proper top-level architecture is the key to shorten design cycle and increase chances of first-time silicon success. With a system having heterogeneous building blocks, the design regression would be time-consuming and the loss of cost is significant if the system architecture has any errors. The following introduces a new SoC design philosophy, transaction level modeling, which allows us to explore the design spaces at system level by providing trade-off between implementation details and simulation accuracy.

3.2 Transaction Level Modeling

3.2.1 Introduction of TLM

As described in the previous section, the traditional design methodology can not satisfy the need for the design of complex system. The reason is that many unnecessary implementation details are captured for the system-level modeling. Thus, the simulation speed could be so slow that the verification at system level may not be done thoroughly. Recently, a modeling technique called transaction level modeling (TLM) is proposed to achieve the system-level modeling. The idea is to make another level of abstraction between the system specification and its RTL implementation so that unnecessary implementation details can be hid from the system-level modeling. As far as the system is concerned, the implementation details for each component is not the most important in the early development phase. Instead, the system parameters, such as the partition of the tasks, the functionality of each component, the topology that connects different components, the communication protocol between components, the memory hierarchy, and so on, are of more interest.

There are four types of TLM including (1) the PE-assembly model, (2) the bus-arbitration model, (3) the cycle-accurate computation model, and (4) the timing-accurate communication model.

Fig. 3.2 (a) shows the system models at different levels of abstraction. The top level is the specification model and the bottom level is the implementation model. The marked level between specification model and implementation model are transaction levels. According to the modeling accuracy in computation and communication, each model represents an operating point in Fig. 3.2 (b), where the bottom-left corner stands for the specification of the system while the top-right corner denotes the detailed implementation at register-transfer level. Particularly, only the four modules, PE-assembly model, bus-arbitration model, time-accurate communication model, and cycle-accurate computation model, are considered as the TLM.

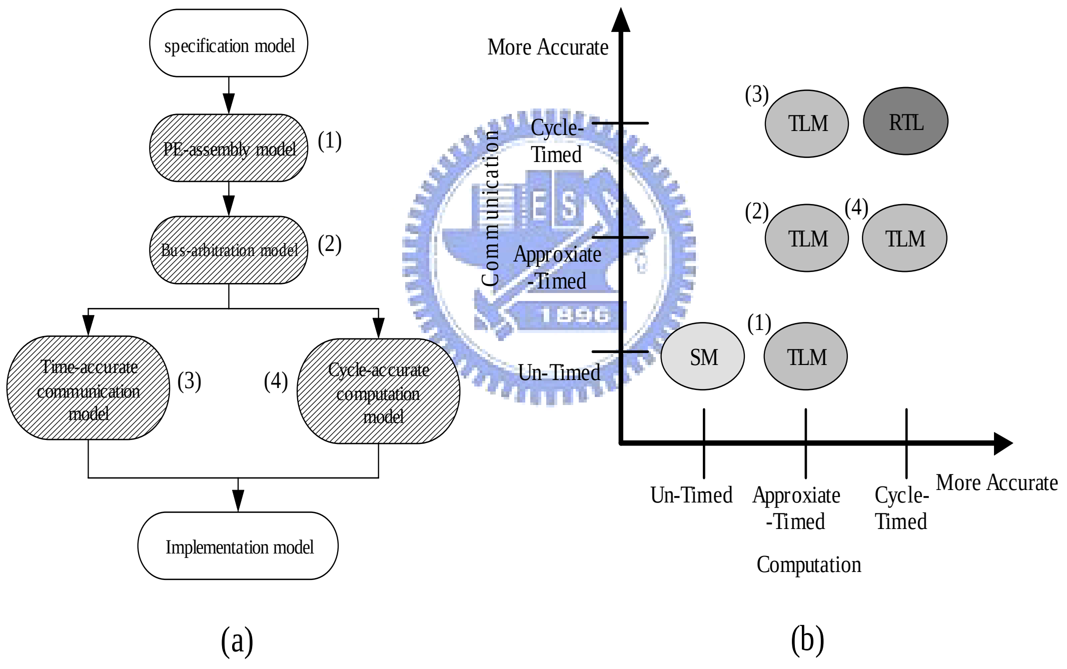


Fig. 3.2 System model at different levels of abstraction

Table 3.1 indicates the characteristics of different system models. As shown, different models capture different degrees of accuracy in computation and communication. The specification model and the implementation model represent the two extreme cases, where the system model specifies the functionality of the system while the implementation model defines its implementation at register-transfer level.

The models in between are the four types of TLM, which offers the flexibility on selecting the simulation accuracy and speed.

Table 3.1 Characteristics of different models

Models	communication time	computation time	communicatoin schene	PE interface	added Implementation detail
Specificaton model	no	no	variable	(no PE)	-
PE-assembly model	no	approximate	message-passing channel	abstract	PE allocation, process PE mapping
Bus-transaction model	approximate	approximate	abstract bus channel	abstract	bus topology, bus arbitration
Time-accurate communication model	time/cycle accurate	approximate	detailed bus channel	abstract	detailed bus protocol
Cycle-accurate computation model	approximate	cycle accurate	abstract bus channel	pin accurate	RTL/ISS PEs
Implementation model	cycle accurate	cycle accurate	wire	pin accurate	detailed bus protocol or RTL/ISS PEs

3.2.2 Design flow with TLM

Traditional design flow can not ensure the quality of the design when the system complexity increases dramatically. This section presents a new SoC design flow with TLM as the platform for concurrent software and hardware development. The new design flow mainly comprises two parts, which are (1) the new system-to-RTL extension and (2) the traditional RTL-to-layout flow. The first part is different from that used in the past while the second part is remained the same.

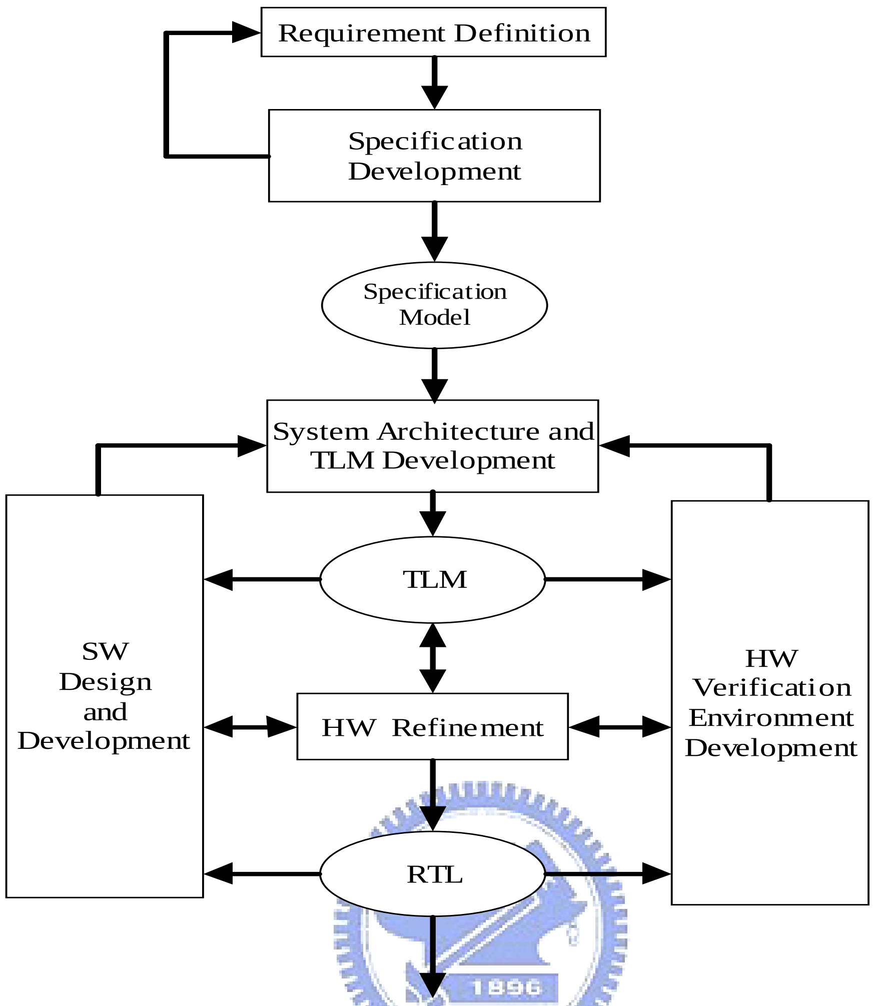

Requirement Definition
Specification Development
Specification Model
System Architecture and TLM Development
TLM
SW Design and Development
HW Refinement
HW Verification Environment Development
RTL

Fig. 3.3 Design floe with transaction level modeling

Fig. 3.3 indicates the new system-to-RTL extension. As shown, after the specification is defined, the system architecture is developed and verified by using TLM. Upon the completeness of the TLM model, it is used as a unique reference to both software and hardware teams. For the software team, the embedded software is developed and verified based on the TLM model. For the hardware team, the TLM serves as the golden model for the detailed implementation. Along with the development of software and hardware, the TLM model can be annotated with more accurate timing information. Consequentially, not only the functionality but also the timing can be verified together. Differs from the traditional design flow, the new design flow performs system integration and verification in the very beginning, which is the key for ensuring the quality of the design. The following summarizes the functionality of TLM in the SoC design flow:

1. Verification model for design space exploration.
2. Platform for early software development.
3. Specification and golden model for hardware development.

Nowadays, EDA tools are still not capable of automatically converting TLM to detailed hardware implementation. The hardware refinement is still done through a traditional paper specification and RTL coding. TLM appears to be an extra workload and unnecessary task. However, it still brings many benefits that significantly reduce the time to market:

1. System integration at the early stages so that the potential problems can be found and solved earlier.
2. Faster simulation speed while maintaining the accuracy of simulation.
3. Concurrent software and hardware development.
4. Platform for software/hardware co-design and co-verification.
5. Incremental hardware refinement and implementation details by means of hybrid abstraction level modeling.

For the implementation of TLM, we present the System C library. In the System C, there are 3 major components, which are process, channel, and interface function. The processes define the operations of a component and can be triggered by a set of predefined events. In addition, the channel specifies the connections among different components and the interface function provides the means for a component to communicate with the others. Within a component, the processes can call the interface functions for data transaction without knowing the detailed implementation of the interface functions. Consequentially, by well defining the interface functions, the communication part and the computation part can be developed and refined independently.

Due to the benefits of the TLM, it will become a critical step in the SoC design flow. In the next section, the system architecture of our H.264 video decoder that is designed by using will be introduced.

3.3 Transaction Level Modeling of H.264 Decoder

In most video coding standard, different profile level will support different coding tools such as transform8x8, supporting frame size, and MBAFF. As a result, the profile level definition is very important to the complexity and cost of the overall system architecture. This section will describe the profile level of our H.264 decoder first. The system architecture designed in TLM model will be introduced next.

3.3.1 Specification

In H.264 standard, there are many different profile levels which contain different coding tools to improve the coding efficiency. Thus, different decoder design supporting for differnet profile will be different in performance and cost. This section presents a design specification of decoder conforming to high profile at level 4. Any bit-streams conforming to main/high profile with a level lower than or equal to 4 shall be decoded. Specifically, the decoder supports the decoding throughput up to 1920x1080i@60Hz. In the following, some properties of high profile and level limits are listed.

1. Only I, P, and B slice types may be present.

2. No data partition.

3. Arbitrary slice order is not allowed.

4. No slice group and no redundant picture.

5. chroma_format_idc in the range of 0 to 1.

6. bit_depth_luma_minus8/bit_depth_chroma_minus8 equal to 0 only.

7. qpprime_y_zero_transform_bypass_flag equal to 0 only.

8. Up to 16 reference frames. (32 reference fields).

9. Vertical motion vector range does not exceed MaxVmvR as in Table 3.1.

10. Horizontal motion vector range does not exceed the range of -2048 to 2047.75

11. Up to 32MVs perMB.

12. Number of bits per macroblock is not greater than 3200.

Moreover, Table 3.2 shows more constraints of different profiles. With summarizing these profile limits, we can start to design our micro-architecture of H.264 decoder to satisfy all functionalities while minimizing the cost.

Table 3.2 Level limits

Level number	Max macroblock processing rate MaxMBPS (MB/s)	Max frame size MaxFS (MB/s)	Max decoded picture buffer size MaxDPB (1024 bytes for 4:2:0)	Max video bit rate MaxBR (1000 bits/s, 1200 bits/s, cpbBrVclFactor bits/s, or cpbBrNalFactor bits/s)	Max CPB size MaxCPB (1000 bits/s, 1200 bits/s, cpbBrVclFactor bits/s, or cpbBrNalFactor bits/s)	Vertical MV component range MaxVmvR (luma frame samples)	Min compression ratio MinCR	Max number of motion vectors per two consecutive MBs MaxMvsPer2Mb
1	1 485	99	148.5	64	175	[64,+63.75]	2	-
1b	1 485	99	148.5	128	350	[64,+63.75]	2	-
1.1	3 000	396	337.5	192	500	[128,+127.75]	2	-
1.2	6 000	396	891.0	384	1000	[128,+127.75]	2	-
1.3	11 880	396	891.0	768	2000	[128,+127.75]	2	-
2	11 880	396	891.0	2000	2000	[128,+127.75]	2	-
2.1	19 800	792	1 782.0	4000	4000	[256,+255.75]	2	-
2.2	20 250	1 620	3 037.5	4000	4000	[256,+255.75]	2	-
3	40 500	1 620	3 037.5	10000	10000	[256,+255.75]	2	32
3.1	108 000	3 600	6 750.0	14000	14000	[512,+511.75]	4	16
3.2	216 000	5 120	7 680.0	20000	20000	[512,+511.75]	4	16
4	245 760	8 192	12 288.0	20000	25000	[512,+511.75]	4	16

3.3.2 System Architecture of H.264 decoder

Fig. 3.4 shows the overall architecture of this system, which is developed based on the ARM platform. For the chip I/O, the compressed bit-stream is input via a hardware interface, which communicates with the host by a bridge, and the decoded frames are output to the monitor via HDMI interface. The reference pictures, decoded pictures, and motion vectors for each reference picture are stored in the external memory. All the data access to external memory will go through the memory controller.

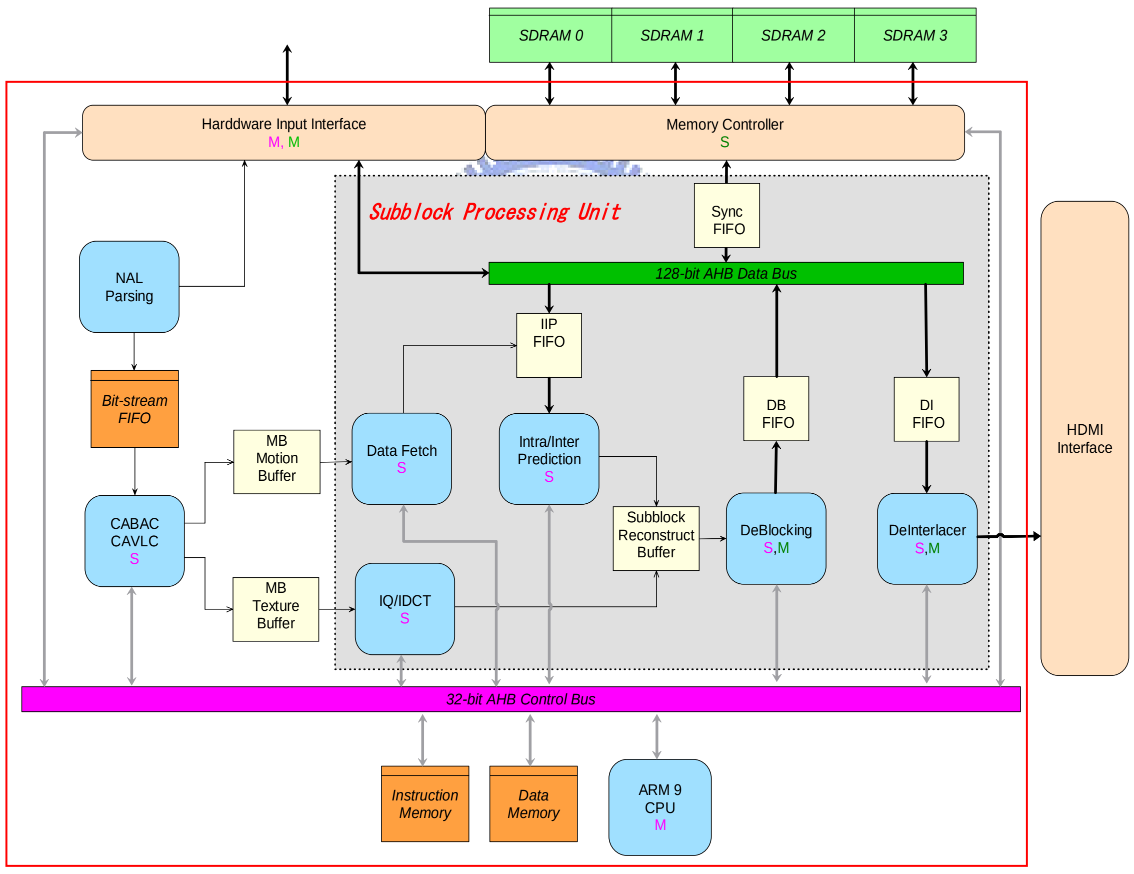


Fig. 3.4 System architecture diagram

Inside the chip, there is an embedded CPU and two AHB buses, which are control bus and data bus. The control bus is used by CPU for data flow control and the data bus is used by Data Fetch, De-blocking and De-interlacer for data transfer between these modules and external memory. In addition to the AHB buses, there are backdoor-to-backdoor connections between modules. The modules connected by backdoor channel make up a video pipe, where its input comes from bit-stream FIFO and its output is drive to the HDMI interface. Particularly, the data between modules are exchanged on block by block basis with block size being 8x8 except for CABAC.

In this system, the headers of sequence, picture, and slice are parsed by CPU. The data below slice level are processed in the video pipe. According to the information in sequence, picture, and slice headers, CPU can configure the modules in the video pipe through the control bus for various decoding modes. Each module can also be independently tested by CPU. The following briefly describes our video pipe and system schedule.

3.3.2.1 Video Pipe

In our H.264 decoder, our video pipe contains seven modules which are CABAC, IQ/IDCT, Data Fetch (DF), Intra-Inter prediction (IIP), De-Blocking, and De-Interlacer. CABAC is the first module in video pipe and the functionality of CABAC is decoding the bit-stream syntax below slice header. Because our decoder processes luma and chroma components in parallel and CABAC can not decode chroma components until all luma coefficients have decoded in one macroblock, it is more efficient to make CABAC operate in macroblock level. For saving the buffer size, all other modules operate in 8x8 block.

After CABAC the pipeline process to IQ/IDCT module and DF module. The IQ/IDCT module does the inverse quantization and inverse discrete cosine transform of the residuals while the DF module is responsible for motion prediction and fetching reference block for Inter block and intra prediction type decoding for Intra block. Following after the DF is IIP which produces the value of prediction block for Intra and inter prediction and adds the results with the residuals from IQ/IDCT. Because we let IQ/IDCT start earlier one eight by eight block than IIP, it can make sure that IIP always has the corresponding residuals.

After that, de-blocking is performed for reducing the blocking effect. For keeping the data order correct, there are three eight by eight block buffers between IQ/IDCT, IIP, and De-Block. For example, when IQ/IDCT is writing the third buffer and IIP is writing the second buffer, De-Block is reading the first block buffer. In addition, because de-blocking has specific process order in macroblock level but our process unit is eight by eight block, we must change the process order shown in Figure 3.2 but still keep the rule in specification. In Figure 3.2, the number means the process order in one block. Fig. 3.5 present the eight by eight block order in zig-zag scan of one macroblock.

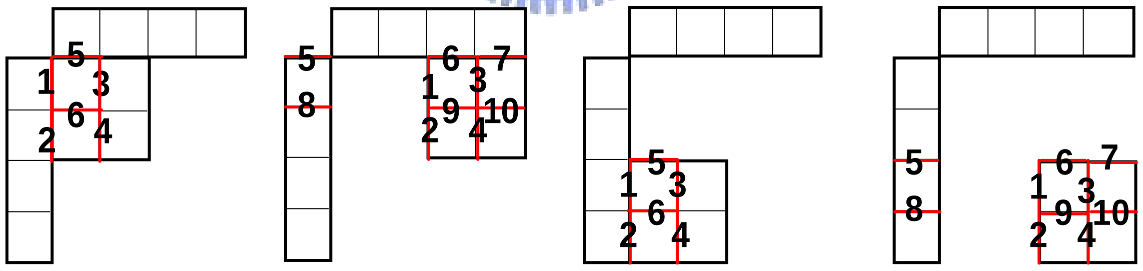


Fig. 3.5 De-blocking process order

The last module is De-Interlacer which will work only when the source sequence is field. The functionality of De-Interlacer is to translate a field picture to a frame picture. Because the algorithm of De-Interlacer will use the previous, current, and next one field in display order, it will also cost bus bandwidth. Besides, the fields for De-Interlacer and for reference may be different. As a result, it will also increase the size of external memory to store the data.

3.3.2.2 System Schedule

Without a system schedule to control data flow in video pipe, it is possible that the functionality of whole system is wrong even if every module is well verified. This section describes briefly our system schedule shown in Table 3.3. In the beginning of decoding, we need an initial period to decoding bit-stream above slice header and to set the control registers of every hardware modules by ARM CPU. After that, all hardware modules start to decode one by one. When CABAC starts to decode the second macroblock in current slice, the first macroblock is fed into the following modules in eight by eight block unit. As a result, IQ/IDCT and DF, IIP, and DeBlock process the different eight plus eight block.

During the slice changes, the CPU will detect the NAL unit first and start to decode the slice header. When CPU is decoding the slice, the hardware may stall depending on the time the NAL unit is detected, slice header decoding speed of CPU, and block decoding speed of hardware modules. Table 3.3 shows the condition that hardware modules will stall. In addition, when the picture changes, there is same condition as slice.

Looking into the hardware pipe, every module has different process time. As a result, when a module finish current block, it does not mean that the module can process next block right now because the input data may be not ready and it may over write the data which next stage is using. In our decoder design, if one module is finished, it will not start again until all hardware modules are finished. Thus, for supporting real-time decoding, all modules must make sure that they can finish their job in time.

Table 3.3 System schdule

			Field(t1)			Field(t2)
8x8	Hardware Parser	CABAC	IQ/IDCT	IIP	DeBlock	DeInterlacer
Cycles			and DF			
	Detect NAL		Initial of	sequence		
	Dec SPS					
	Set SPS_CR					
	Detect NAL		Initial of	fist picture		
	Dec PPS_0					
	Set PPS_CR[0]					
	Detect NAL		Initial of	first slice		
	Dec SH_0					
	Set SH_CR[0]					
0		P0S0B0				B0
1		P0S0B1	P0S0B0			B1
2		P0S0B2	P0S0B1	P0S0B0		B2
3		P0S0B3	P0S0B2	P0S0B1	P0S0B0	B3
.		.	P0S0B3	P0S0B2	P0S0B1	.
.		.	.	P0S0B3	P0S0B2	.
.		.	.	.	P0S0B3	.
479		P0S0B479	.	.	.	B479
480	Detect NAL	Bubble	P0S0B479	.	.	Bubble
	Dec SH_1					
	Check					
	Continuity()					
	Set SH_CR[1]					
481		P0S1B0	Bubble	P0S0B479	.	B480
482		P0S1B1	P0S1B0	Bubble	P0S0B479	B481
483		P0S1B2	P0S1B1	P0S1B0	Bubble	B482
484		P0S1B3	P0S1B2	P0S1B1	P0S1B0	B483
.		.	P0S1B3	P0S1B2	P0S1B1	.
.		.	.	P0S1B3	P0S1B2	.
.		.	.	.	P0S1B3	.
959		P0S1B479	.	.	.	B959
.		.	.	.	.	.
.		.	.	.	.	.
.		.	.	.	.	.
.		.	.	.	.	.
.	Detect NAL	Bubble	P0SNB479			Bubble
.	Dec PPS_1					
.	Set PPS_CR[1]					
.	Detect NAL	Bubble	Bubble	P0SNB479		Bubble
.	Dec SH_1					
.	Set SH_CR[1]					
0		P1S0B0	Bubble	Bubble	P0SNB479	B0
1		P1S0B1	P1S0B0	Bubble	Bubble	B1
2		P1S0B2	P1S0B1	P1S0B0	Bubble	B2
.		.	P1S0B2	P1S0B1	P1S0B0	.
.		.	.	P1S0B2	P1S0B1	.
.		.	.	.	P1S0B2	.

3.3.2.3 System Modeling

For transaction design at transaction level, Fig. 3.6 shows the concept about how we implement all modules with sc_thread. First, we define the input and output ports connecting with other modules. All data transmission is through channel by calling the interface functions of ports. In the program part, the module will read input data through input channel in the beginning. Then, the finite state machine decides which path th program will go through. In the main program, the functionality can be composed of one or more functions and we can decide to use wait function after the end of every function or only use once after all functions are finished. The more wait function we use, the more program switches happen. That will influence the simulation speed. After all work is done, the module will send a finish signal to synchronization channel and stall until all modules finish their work.

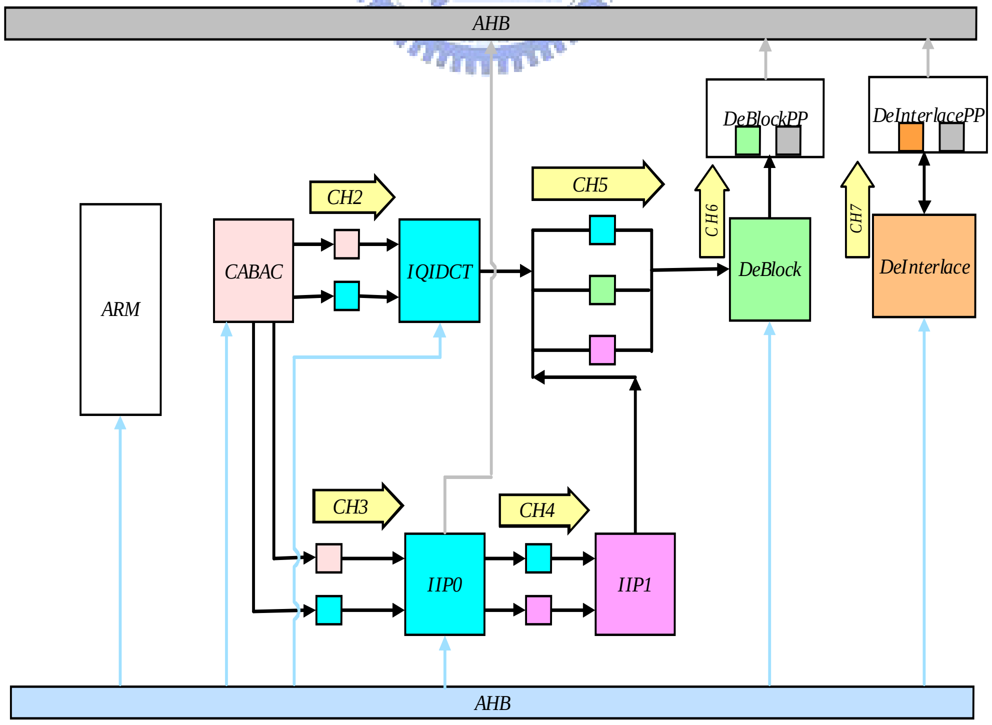


Fig. 3.6 Transaction level modeling at system level

3.4 Memory Controller in H.264 Decoder

There are three main modules in the H.264 decoder will access external memory to receive or give data. These three modules are inter prediction (motion compensation) module, de-blocking module, and de-interlace module. The relationship between these modules and external memory is shown in Fig. 3.7.

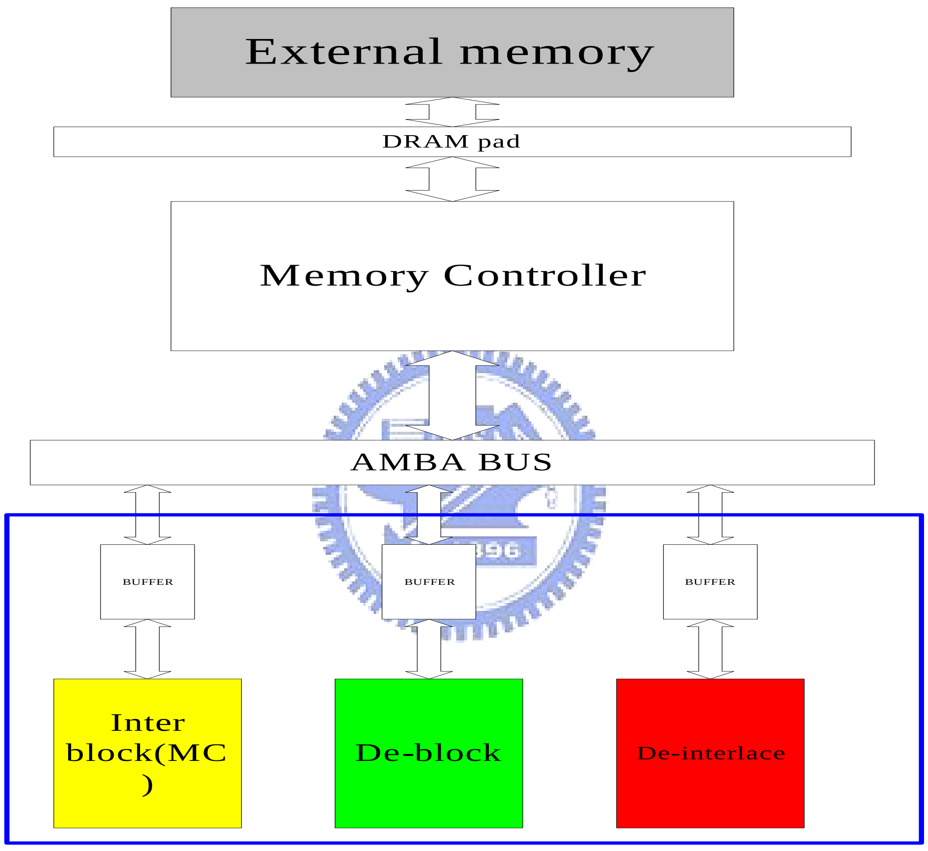


Fig. 3.7 Relationship between H.264 decider and external memory

The functionality of de-block module is writing the re-construct block into the external memory, all the decoded block will be stored in the external memory through de-blocking module.

The inter prediction module is used to do the motion compensation for the inter-block. The data fetch module obtains the motion vectors from CABAC block first, and it fetches the block in the reference frame which is placed in the external memory already. The data that is read from external memory will be placed in the first-in-first-out (FIFO) buffer before entering the inter prediction module.

In order to display the picture in field mode, a de-interlace module is need to read the decoded field stored in the external memory and de-interlace the fields.

The H.264 decoder needs a memory controller to communicate with external memory. Due the complexity of H.264/AVC decoder at level 4 we need a memory controller that not only keep the correctness of data but also can help the whole system to achieve better performance.

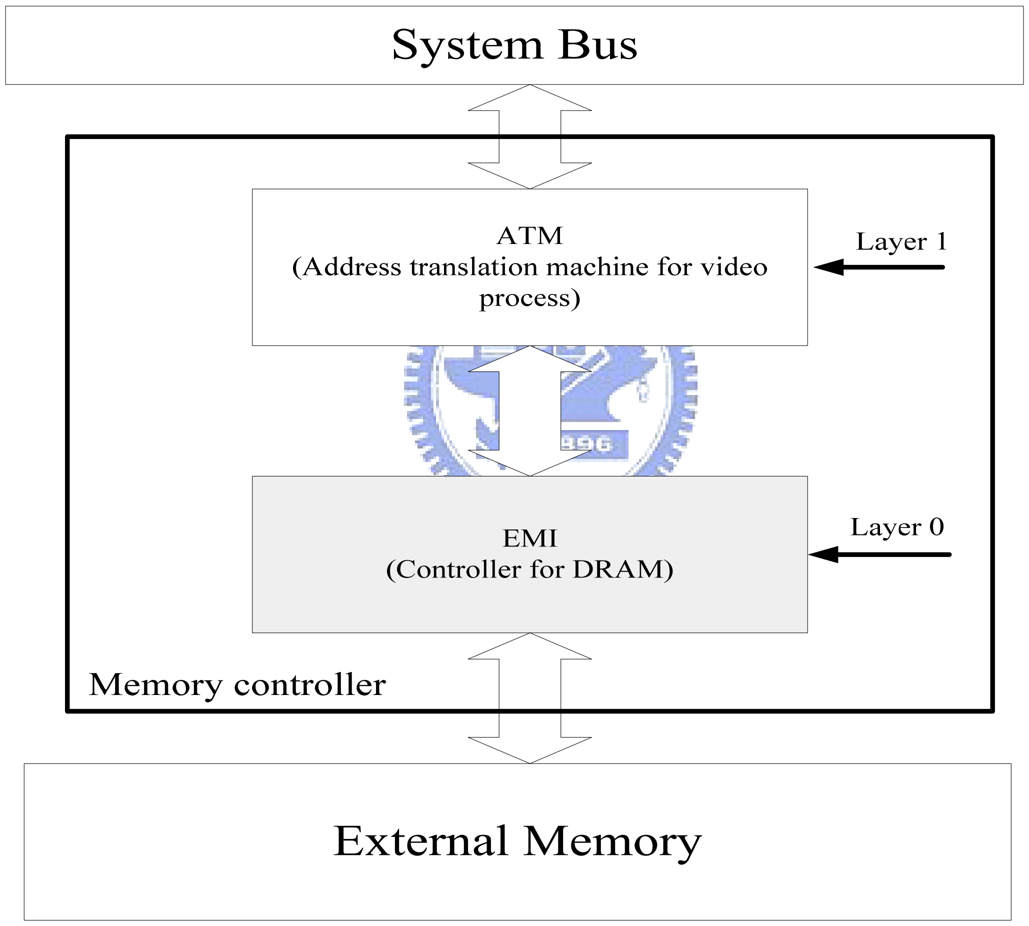


Fig. 3.8 Architecture of the memory controller in H.264 decoder

Fig. 3.8 shows the architecture of my memory controller inside the H.264 decoder. This memory controller combines two layers together. The Layer 0 is the external memory interface (EMI) which is used to control the external memory. The Layer 1 is the address translation machine (ATM) which is designed for the H.264

decoder. The Layer 0 part (EMI) will receive the addresses translated from Layer 1 (ATM) part and generate the appropriate commands to external memory.

The purpose of layer 0 (EMI) is to communicate with external memory. The EMI is designed for various kind of system. As a result, the EMI is independent of the system but dependent of the external memory. In order to make this EMI can fit variable type of external memory. The configurability of this EMI is very essential. The detail of EMI will be introduced in the Chapter 4.

Layer 1 is an address translation machine (ATM). It exploits the characteristics of the video process to make the accesses become more regular. The Layer 1 is designed for the memory subsystem. The detail of Layer 1 and the relationship between memory subsystem and H.264 decoder will be described in Chapter 5.

Chapter 4
External Memory Interface

In the design of SOC system, it usually needs an off-chip memory to storage the large amount of data. The external memory interface is used to communicate with the external memory for the system as shown in Fig. 4.1. To deal with tremendous data transfer and storage in H.264 decoder, the external memory must provide high data bandwidth to achieve the real time request. The bandwidth of the external memory is limited due to the pin number of I/O is finite. Accordingly the external memory interface must provide high data bandwidth utilization by using some techniques. The proposed external memory interface will be introduced in this chapter.

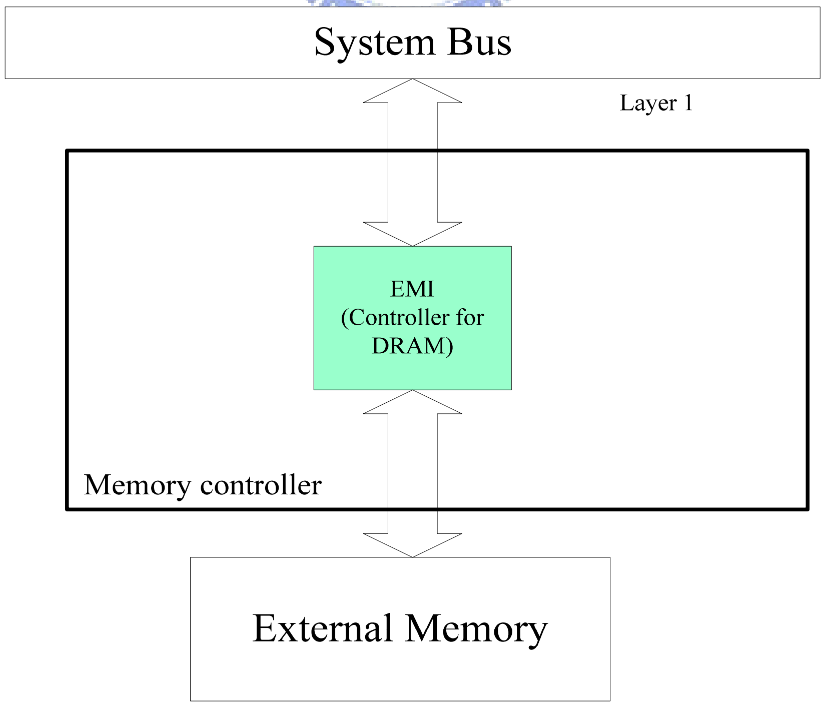


Fig. 4.1 Architecture of my memory controller

4.1 Concept of EMI

The external memory interface (EMI) is one of the Layers in this memory controller, that is, layer 0. EMI will receive the physical addresses from the address translation machine or the functional block, and use the addresses to access the DRAM. This external memory interface is designed to control the external memory, and it will directly connect to the DRAM as show in Fig. 4.2.

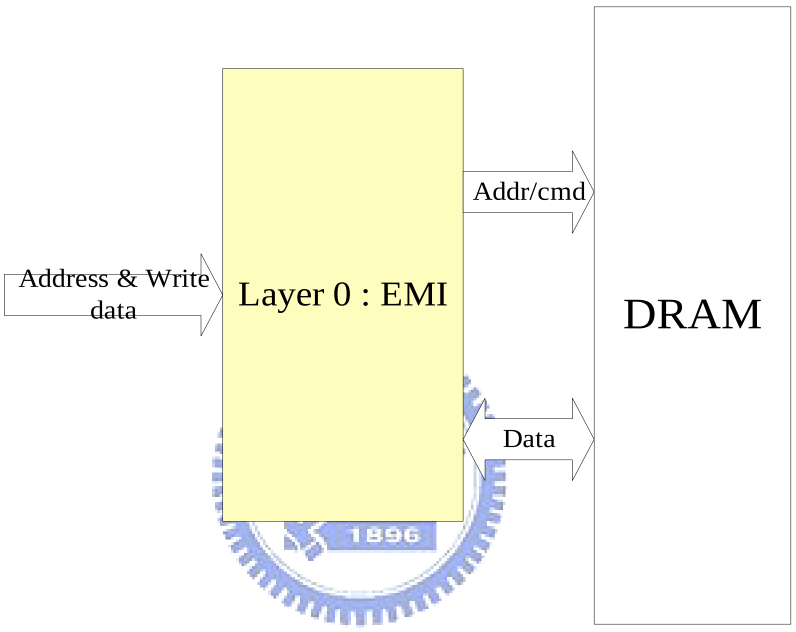


Fig. 4.2 Connection of EMI

The external memory interface will generate the appropriate commands to external memory (DRAM). The external memory can only accept the commands that it defined in the specification, and the commands have been introduced in chapter2.

The design of my EMI is configurable for different kinds for external memory. Users only have to set the specific parameter of the external memory that they use, and this EMI will translate the accesses into the commands. Thus the users can conveniently use this EMI to access the external memory (DRAM).

This EMI has another advantage of improving the performance of external memory. As mentioned in chapter2 the performance of DRAM usually places a restriction on the design of SOC. In order to increase the performance of the whole system this EMI is designed to achieve higher performance. The whole architecture will be introduced in next section.

4.2 Architecture of EMI

The detail architecture is shown in Fig. 4.3. It contains five fundamental parts inside the EMI. The input data and address will stored in the FIFO and generate the appropriate command through the mode control block and the FSM block. Then the commands will be send to DRAM. The detail of these parts will be described in the following.

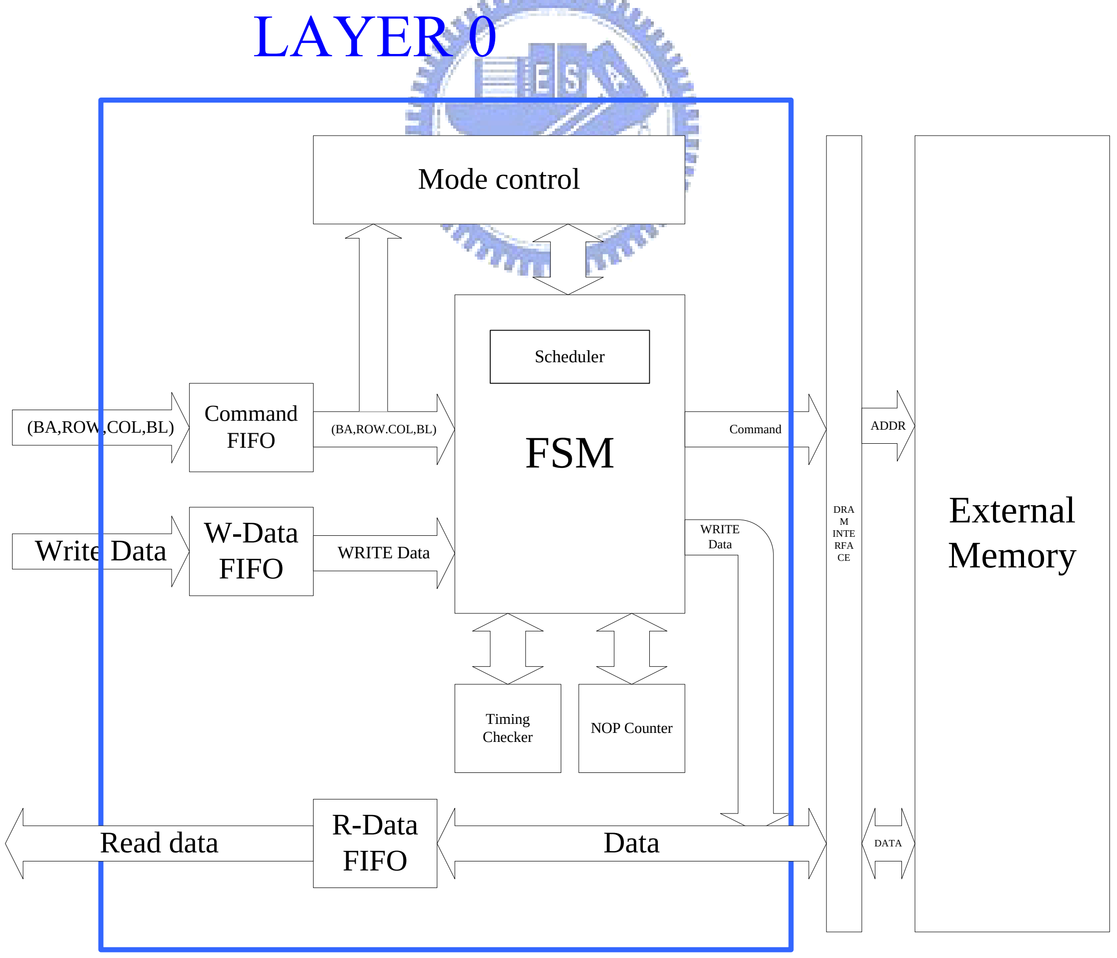


Fig. 4.3 Architecture of EMI

4.2.1 FIFO

The FIFO (first-in & first-out register) in the EMI is used to store the data and addresses temporarily. It can divide into data FIFO and command FIFO. The data FIFO is used to put the data that will write to the external memory for the write operation. The command FIFO is used to put the addresses that the system wants to access.

The command FIFO is designed in a particular way that can reduce the latency to get the data. In the system of H.264 decoder, there are large amount of consecutive read operations to external memory. How to reduce the latency of receiving data is very important.

In the external memory, the consecutive read (or write) operations to access the same row is called as a row-hit status. The next read operation can be issued right after the previous read operation as shown in Fig. 4.4. The consecutive read (or write) operations to different row in the same bank is called as a row-miss status. If the row-miss status happens, the next read operation can't be issued directly. The bank has to be pre-charged in order to turn off its current row, and issue an active command to turn on the row that the next read operation will access, as shown in Fig. 4.5.

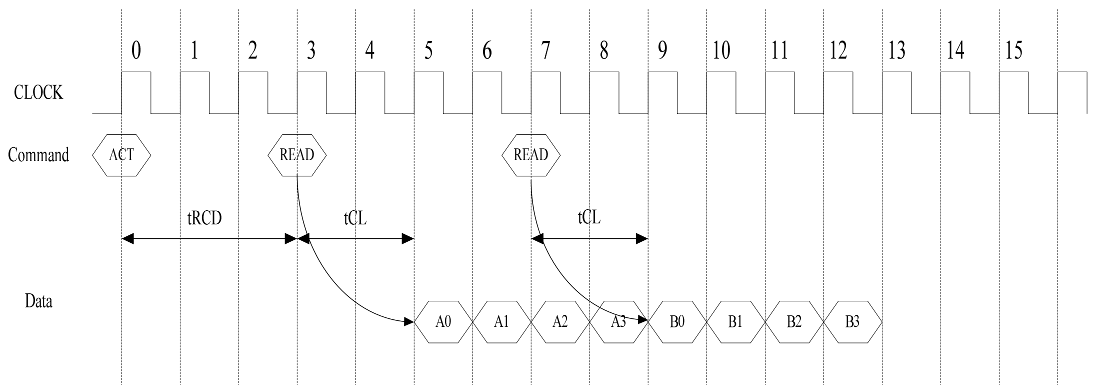


Fig. 4.4 Consecutive accesses to same row (burst length=4)

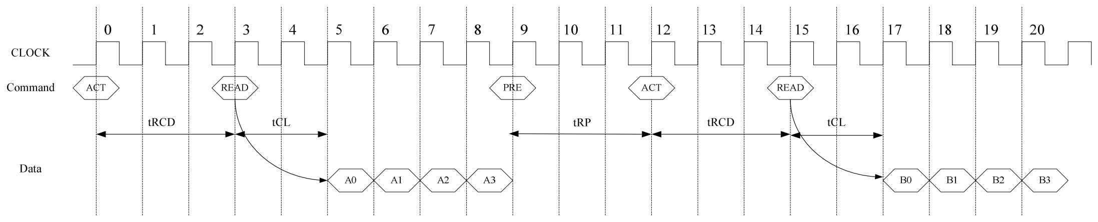


Fig. 4.5 Consecutive accesses to different row (burst length=4)

Compare Fig. 4.4 to Fig. 4.5, we can find the row-miss status consumes more latency on receiving the data. The row-miss status takes time to pre-charge the bank and open the next row. In order to decrease the latency when meeting the row-miss situation, the external memory supports auto-pre-charge function. The read (or write) command can be issued with auto-pre-charge capability. Thus the pre-charge time (tRP) can be hid in the time of data transfer, as shown in Fig. 4.6.

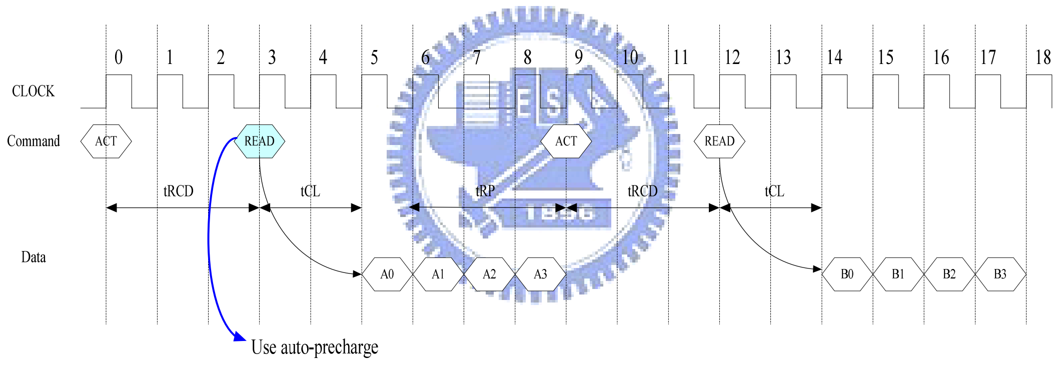


Fig. 4.6 Consecutive accesses to different row with auto-pre-charge

Apparently if we use auto-pre-charge capability appropriately, we can decrease the latency of reading data. On the contrary, it costs the latency to re-active the same row if we issue a read command with auto-pre-charge capability when the next command is a row-hit status.

The design of my command FIFO provides a dynamic way to use auto-pre-charge capability. It can accelerate the accuracy of prediction. The addresses will be stored in both of the command FIFO and previous-address register. The incoming address will compare the bank address and column address with the previous address register. If bank address and column address are same with the previous register, the hit bit of the previous command will change to 1. Otherwise the hit bit is still 0. Fig. 4.7 shows the architecture of the command FIFO.

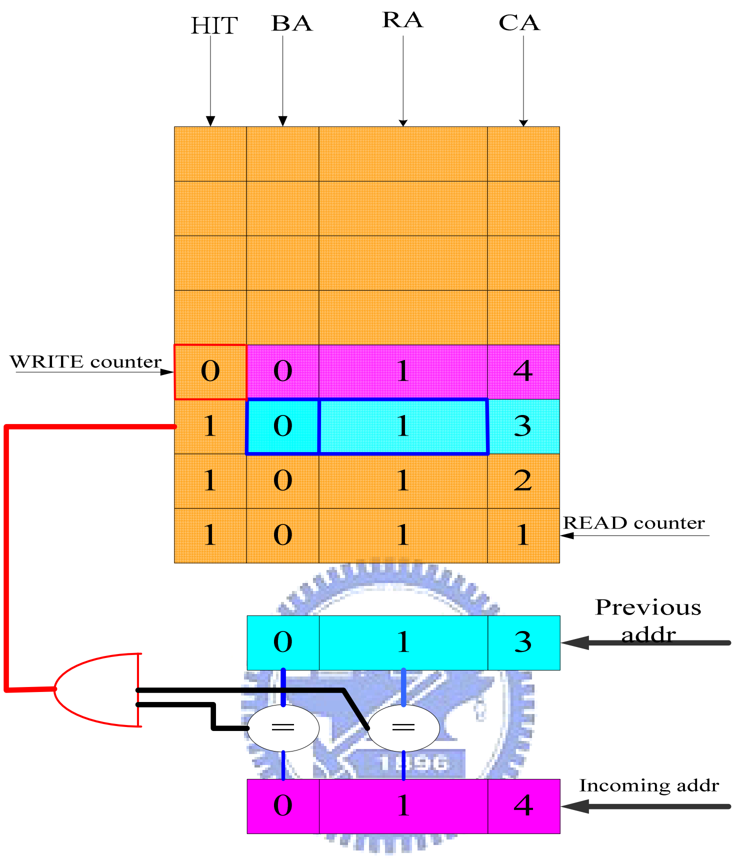


Fig. 4.7 Command FIFO

If the hit bit of the command is 1, it means the row address of next command is same as current command. The current command will read/write the row buffer without auto-pre-charge capability. Similarly it will do the access operation with auto-pre-charge capability when the hit bit is zero. This dynamic method of choosing the auto-pre-charge capability is useful. Even though sometimes the current command don’t know the address of next command, it is effective to pre-charge the bank. That is because of no information of the next command means the master is going to be changed, and the addresses of other masters have high probability of being located at other space in the external memory.

This dynamic method has two options for users to choose. One of the options is the bank will start the auto-pre-charge capability if it detect the next command is bank-miss. This method is suitable when there is high probability of accessing the original bank is row-miss. The other option is the bank will stop the auto-pre-charge capability if it detect the next command is bank-miss. This method is useful when there is high probability of accessing the original bank is row-hit.

Generally we often use auto-pre-charge capability when it is a random-access process and disuse auto-pre-charge capability when the system is regular access. The random-access process has more row-miss situation than regular-access process, so it is benefit to use the auto-pre-charge in the read operation. In the video process, the access is usually regular access. I used different method to read a HD-picture (1920x1080) from external memory in H.264 decoder. The simulation results of different methods are list in Table 4-1.

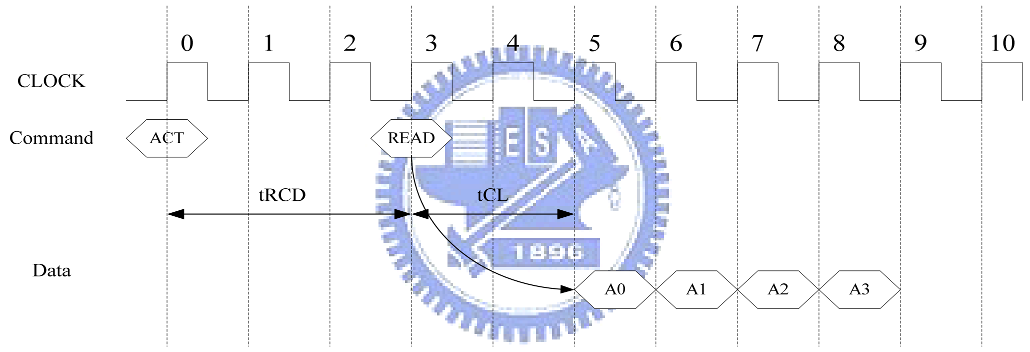
Table 4.1 Different method for receiving a picture

Auto-pre-charge method	The read timing per picture (1920x1080)	Cycle count
Row-close method	2025283 ns	337548 cycle
Row-open method	820431 ns	136739 cycles
Dynamic method 1 (bank-miss no pre-charge)	612377 ns	113442 cycles
Dynamic method 2 (bank-miss pre-charge)	680652 ns	102063 cycles

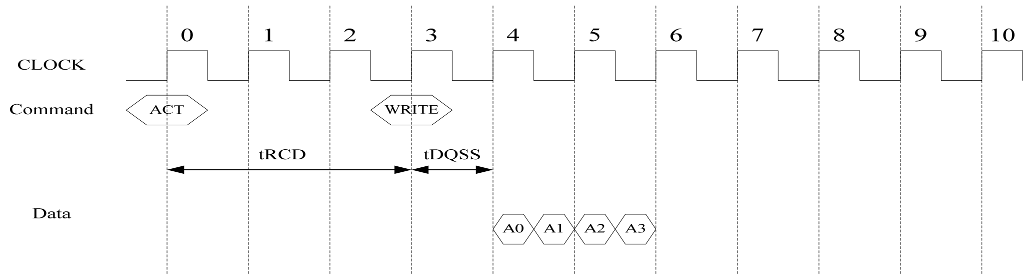
4.2.2 Mode Control

The mode control block is used to control the finite state machine block. It will record the state of each bank, and record the active row in each bank. There are two states of the bank. One is idle state, and it means there is no row open in this bank. The other state is active state, and it means there is a particular row open in this bank. The row register of the bank will record the row address of this particular row.

If the bank is in idle state, it is in bank-miss status. It induces (ACTIVE + CAS) latency for read access and induces (ACTIVE + DQSS) latency for write access in DDR SDRAM. The access state diagram is shown in Fig. 4.8 (a) and Fig. 4.8 (b).



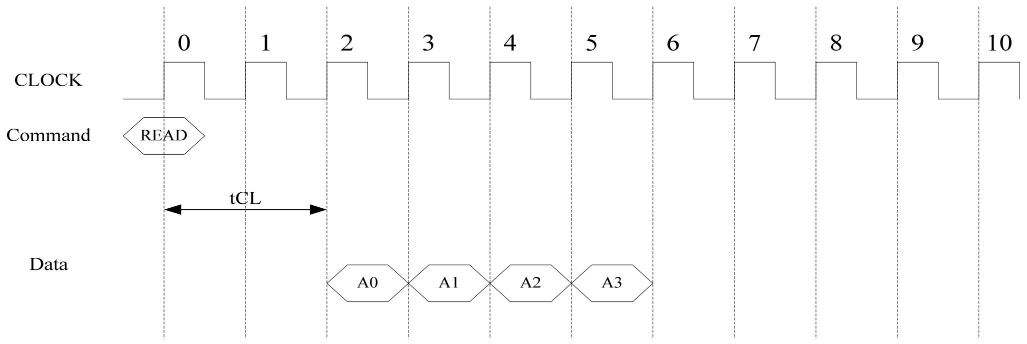
(a) READ OPERATION of bank-miss status



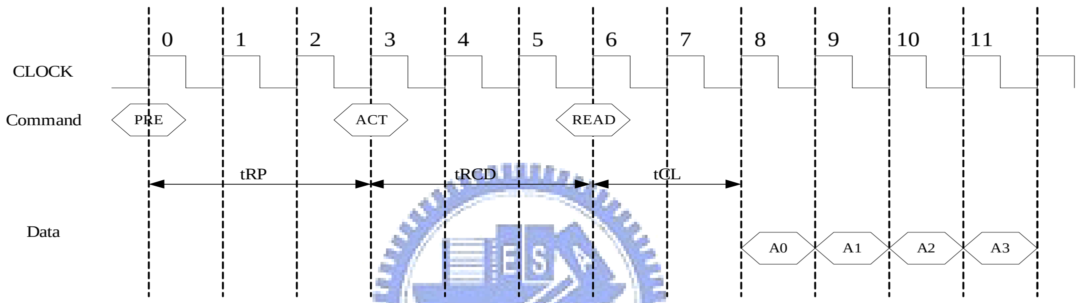
(b) WRITE OPERATION of bank-miss status in DDR

Fig. 4.8 State diagram of bank-miss status

Else if the bank is in active state and the row address in the row register is same as current command, it means a row-hit status. The access state diagram of row-hit status is shown in Fig. 4.9(a). Finally when the row in the row register is different to the current command, it is a row-miss status. The access state diagram of row-hit status is shown in Fig. 4.9(b)



(a) READ OPERATION of row-hit status



(b) READ OPERATION of row-miss status

Fig. 4.9 State diagram of row-hit and row-miss status

The mode control block receives the input signal from finite state machine to change the state of every bank. When the current command is updated, the mode control block will compare the address with the row register. The architecture is shown in Fig. 4.10.

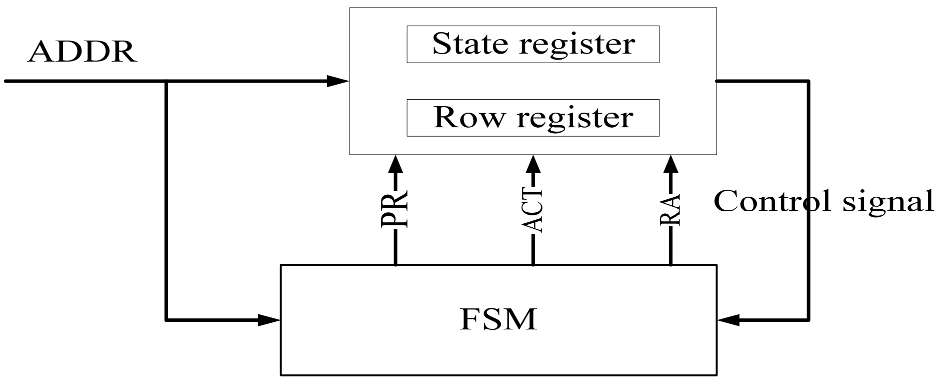


Fig. 4.10 Mode control block

4.2.3 Finite States Machine and schedule block

The functionality of this finite states machine (FSM) is to produce the appropriate commands that can access the external memory. These commands should not violate the timing restrictions which are specified in the memory. The most important of all is the commands should keep the correctness of data.

4.2.3.1 FSM

The basic state diagram of FSM is shown in Fig. 4.11. The NOP state will send out a NOP command to the external. The NOP command means no operation in the memory, the number of NOP commands is issued for different kind of timing restriction. The state flow is controlled by the mode control block. The next state in the FSM is decided by the result of mode control block.

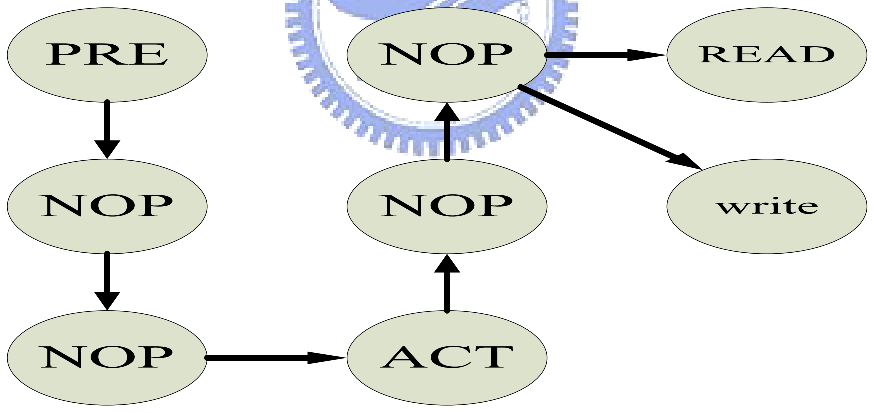
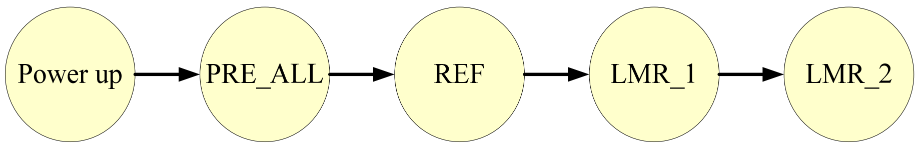
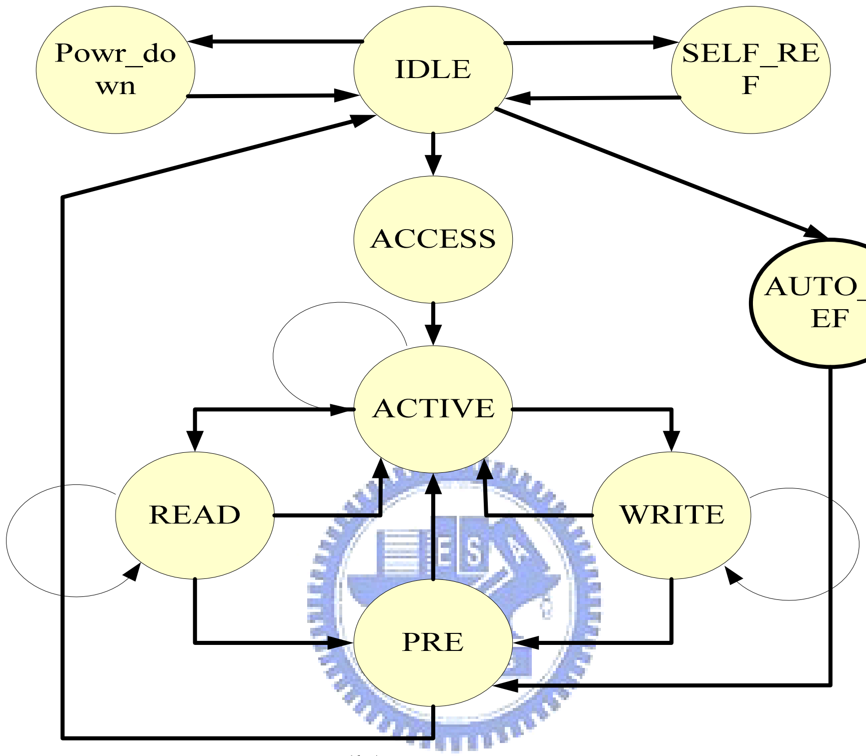


Fig. 4.11 Basic state diagram of FSM

We only use one finite states machine to control four banks inside the memory for the sake of reducing the area and circuit. Because the external memory can only accept one command at each clock cycle, it is wasteful to use many controllers to generate commands for each bank. The overall state diagram is shown in Fig. 4.12. Some states is designed for some particular external memory, the detail operation of states will be described as follows.



(a) Initialization



(b) Access

Fig. 4.12 Finite State Machine

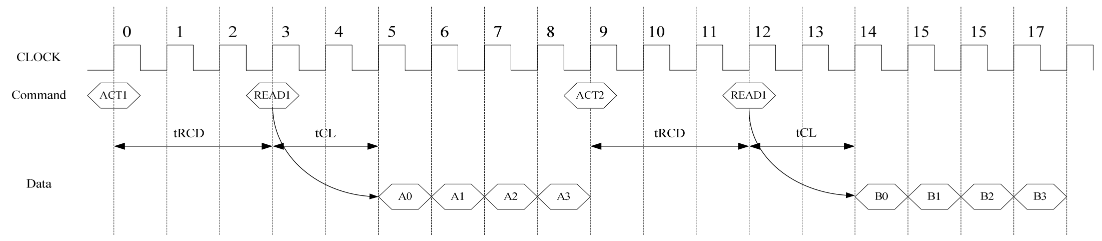
Fig. 4.12(a) shows the operation of initialization. The external memory needs a long latency to power up. After power up state is finished, one pre-charge command and two auto-refresh command must be issued to complete the initialization. After initialization, FSM will enter load-mode-register state. The mode registers are used to define the specific mode of operation of the external memory. LMR_1 state is used to load standard mode register. The standard mode register enables the selection of burst length, burst type, CAS latency, and operating mode. LMR_2 state is used to load extended mode register for some specific external memory such as mobile DRAM.

Fig. 4.12(b) shows the operation of access. The operation of access is to generate the proper commands that can access the DRAM. When the memory stays in idle state for a long time it will enter power down state to save the power. After the memory is being accessed again, it will come back to idle state. The state flow is controlled by mode control block and schedule block.

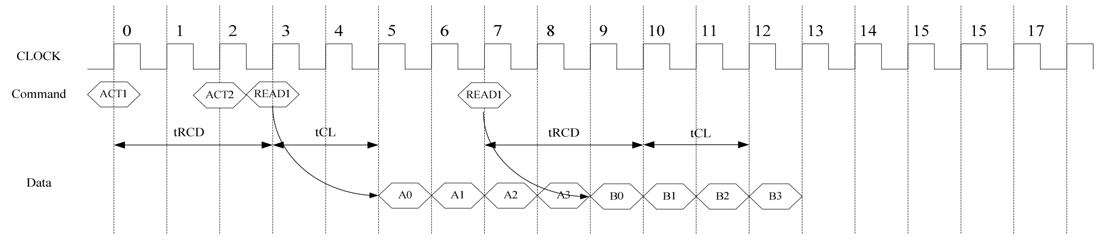
4.2.3.2 Schedule block

The performance of external memory sometimes is still not enough to satisfy applications which have demand for short access latency and high bandwidth. The performance of normal memory controller is constrained because it can not support parallel processing which are addressed to different banks.

The schedule block in the FSM is used to schedule the command. To fully utilize the DRAM bandwidth, it is necessary to parallel process accesses addressed to different banks. This work is done by the schedule block. The schedule block can insert commands that belong to other banks as shown in Fig. 4.13 and Fig. 4.14. The schedule block can utilize the command bus by arranging the command that access to different bank. It not only can improve the bandwidth but also reduce the latency.

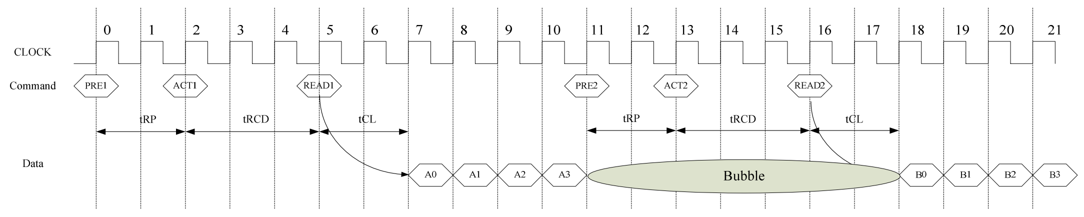


(a) Commands to different bank without schedule

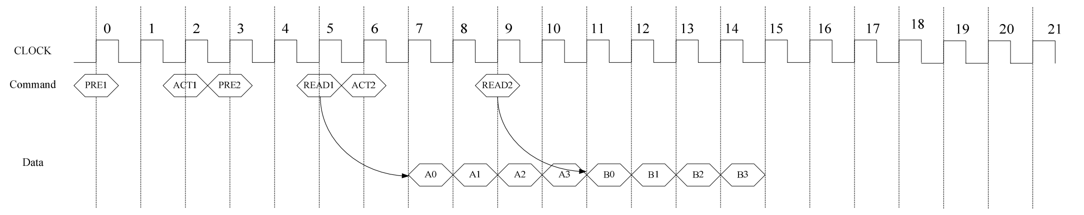


(a) Commands to different bank with schedule

Fig. 4.13 Command with schedule and without schedule (two bank-miss)



(a) Commands to different bank without schedule



(a) Commands to different bank with schedule

Fig. 4.14 Command with schedule and without schedule (two row-miss)

4.2.4 Counter and Timing Checker

The counter is used to calculate the different latency for different commands. After a command is issued, the counter will start to count until the latency is satisfied. The memory can accept next command after the counter is completed. During the time of counting, EMI will issue NOP commands to external memory. It is used to ensure no other command is issued during this latency.

The timing checker stored the timing parameters and some other settings that users can configure it. Some of the important settings inside are listed as follows.

Settings

• Data rate:

Users can choose single data rate or double data rate for the DRAM they use.

• Memory data bus bandwidth:

There are three selections including x8, x16, and x32.

• Memory size:

The number of rows per bank and the number of columns per row can be changed by users.

• Mode registers setting

The CAS latency, burst length, burst type, PASR, TCSR, drive length can be set by users

• Buffer size

The number of buffer inside the external memory can be decided by users.

• Auto-Pre-charge method

There are four method can be selected. The row-open method and row-close method are the traditional methods. Additionally the dynamic method 1 and dynamic method 2 are provided by this EMI. Users can choose different modes for the process they use.

• Timing parameter

The timing parameters are not the same for different types of external memory. The users can set the timing parameters in this timing parameter file. Thus the EMI can easily integrate this EMI to the external memory they use. Some important timing parameters are illustrated in Table 4-2.

Table 4.2 Timing parameters of Micron Mobile DDR DRAM

Parameter	Symbol	Micron DDR (-6)		Unit
		Min	Max	
Active to Read or write delay	tRCD	18		ns
Refresh period	tREF		64	ns
Pre-charge period	tRP	18		ns
Write recovery time	tWR	12		ns
Write to Read delay	tWTR	1		clk
Write to DQS transition	tDQSS	0.75	1.25	clk
Load Mode Register time	tMRD	2		clk
Active to pre-charge	tRAS	42		ns
Active a to Active b	tRRD	12		ns
Active to Active	tRC	60		ns

4.3 The external memory in H.264 decoder

In our design of H.264 decoder, it need to process a 8x8 Y block per 165 cycle. Consequently the latency of writing data and reading data should be less enough to achieve the real time request.

We choose Micron Mobile-DDR SDRAM to be our external memory model. The Mobile-DDR SDRAM is a high speed CMOS, dynamic random access memory. It is internally configured as a quad-bank DRAM. The Mobile-DDR SDRAM uses a double data rate architecture to achieve high-speed operation. The double data rate is essentially a 2n-prefetch architecture with an interface designed to transfer two data words per clock cycle at the I/O pins. A single read or write access for the Mobile-DDR SDRAM effectively consists of a single 2n0bit wide, one-clock-cycle data transfer at the internal DRAM core and two corresponding n-bit wide, one-half-clock cycle data transfer at the I/O balls.

Read and write accesses to the Mobile-DDR SDRAM are burst oriented. Accesses start at a selected location and continued for a programmed number. The address bits registered coincident with the ACTIVE command are used to select the bank and row to be accessed. The address bits registered coincident with the READ or WRITE command are used to select the bank and the starting column location for the burst access.

As with standard SDR SDRAM, the pipelined, multi-bank architecture of Mobile-DDR SDRAM enables concurrent operation, thereby providing high effective bandwidth by hiding row pre-charge and activation time.

The other advantage of Mobile-DDR SDRAM is that can support deep power-down mode. Deep power-down mode can achieve maximum power reduction by eliminating the power draw of the memory array. Data will not be retained when the device enters DPD mode. Some other methods that can save power such as PASR and TCSR are also supported.

The latency of writing and reading a 8x4 block in H.264 decoder can be decreased when compare with the SDR DRAM. The timing diagram of read operation and write operation in the SDR DRAM and DDR DRAM are shown in Fig. 4.15 and Fig. 4.16. As we can see, DDR takes fewer cycles than SDR. The advantage of improving the latency is the critical factor we choose DDR DRAM.

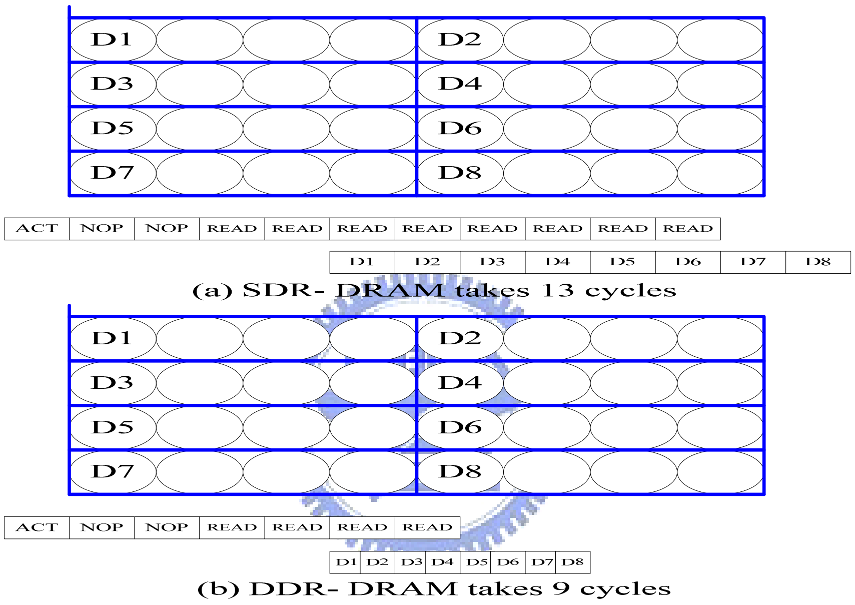


Fig. 4.15 Read operation of SDR and DDR

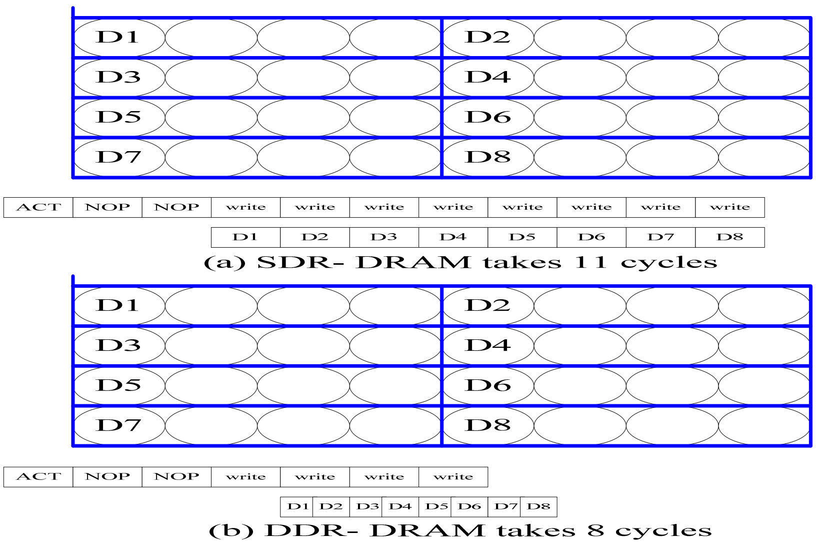


Fig. 4.16 Write operation of SDR and DDR

4.4 Analysis

In this section, I present an experimental work used to evaluate the performance of this external memory interface (EMI). I choose Micron Mobile DDR (256Mb) to be my external memory model and run the patterns from random process and video process (regular process).

4.4.1 Access Latency

I take two processes to test my external memory interface (EMI). One is random process, the other one is video process. The random commands pattern is generated from the random number. The video process commands pattern is generated from the H.264 decoder. As listed in Table 4.3, after simulation 20,000 commands for each process. We first look at random process; we can find the proportion of row-miss occupies more than 24% and the bank-miss is nearly 75%. In the video process, the row-hit status is occupies more than 98%.Apparently, the random process consumes more latency than the video process. The random process needs 1,010,867ns to run the 20,000 commands while the video process only needs 46,308ns to finish them.

Table 4.3 Status of different process

	Row-hit	Row-miss	Bank-miss	Latency
Random process	62 (0.31%)	4916 (24.58%)	15022 (75.11%)	1010867ns
Video process	19721 (98.6%)	8 (0.04%)	271 (1.35%)	46308ns

4.4.2 Power Estimation

Fetching data to/from SDRAM involves three memory operations. An activation operation decodes the row address, selects the appropriate bank and moves a row to the row buffer of the corresponding bank. After a row is opened, a read/write operation moves data to/from the output pins of the SDRAM. Only one bank can use the output pins at the time. When the next read/write accesses hit in the same page, the memory controller does not need to activate the row again (row hit). However, when another row is needed (a row miss), pre-charging the bank is needed first. Only thereafter the new page can be activated and the data can be read. The energy consumption of the SDRAM is computed with the following formula:

$$E = \sum_{i=1}^{Nbanks} (E_{static}^{i} + E_{dynaic}^{i})$$

$$E_{static}^{i} = P_{cs}.t_{cs}^{i} + P_{standby}.t_{standby}^{i} + P_{powerdown}.t_{powerdown}^{i}$$

$$E_{dynamic}^{i} = N_{active}.E_{active} + N_{precharge}.E_{precharge} + N_{rw}.E_{rw}$$

The total energy is dynamic energy add static energy. The static energy is sum of clock suspend mode, standby mode, and power down mode. The dynamic energy is sum of the active energy, pre-charge energy, and read/write energy. The power and energy of each state are listed in Table 4.4. This table is provided by Micron.

Table 4.4 Power of each state

E pre-charge/active	14000PJ/access
E read/write	2000PJ/access
P standby	50mW
P cs with self refresh	10.8mW
P power down	0mW

As we can see the active power consumption and pre-charge power consumption is much larger than the read/write power. Thus reducing the number of active and pre-charge can reduce the dynamic power. In the Table 4.5 we can see the video process takes less number of activation and pre-charge. It is apparently the regular process consumes less power than random access. This total number of commands in this simulation is 20,000.

Table 4.5 State transition of different process

	Active number	Pre-charge number	R/W number
Random process	19988	21220	20000
Video process	1891	7276	20000

4.4.3 Auto-pre-charge Method Effect

In the previous section, I have introduced a FIFO that can dynamically choose auto-pre-charge method when generate a read/write command. The latency of each method is show in Fig. 4.17 and Fig. 4.18.

I take two process patterns including random process and video process which are come from Table 4.3. The latency of video process is shown in Fig. 4.17. The row-close method takes almost double latency when compare with row-open method. This is because the accesses in video process are regular. It has a great opportunity to access the same row. Using row-close method takes additional tome to re-active the same row, and therefore it is better to use row-open method in the regular process. We can see the dynamic method 1 (bank-miss no pre-charge) achieve a better performance than dynamic method 2 (bank-miss pre-charge). This is because the probability of retuning the same row of the same bank is strong in the video process. We can see the dynamic method1 have no improvement on latency, this is because the dynamic method 1 is designed to reduce the latency of row-miss status. This video process pattern almost has no row-miss status so the improvement of dynamic method

1 is not distinct.

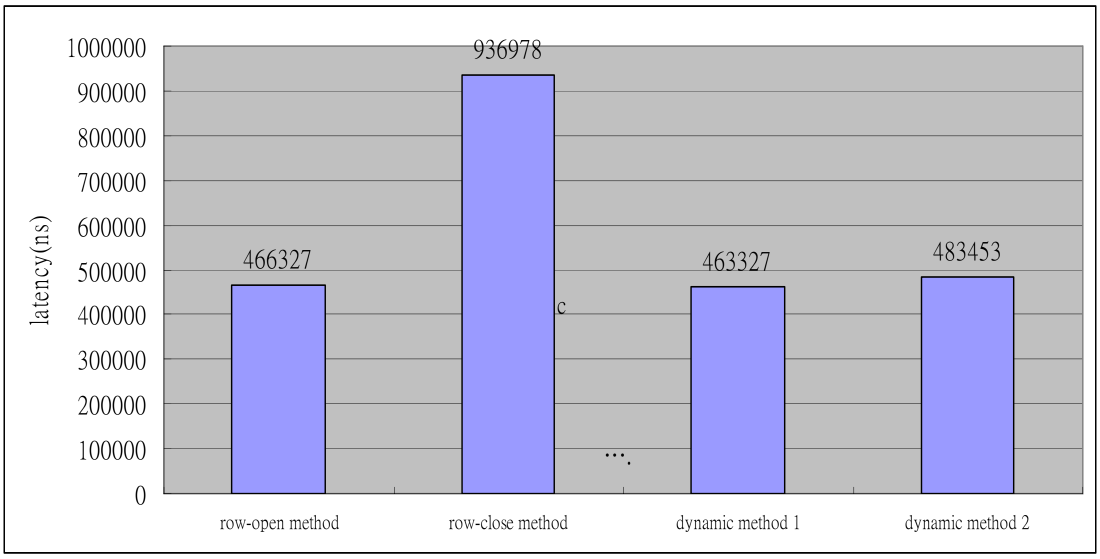


Fig. 4.17 Latency of different method for video process

The latency of random process is shown in Fig. 4.18. The row-open method takes more latency when compare with row-close method. This is because the accesses in random process are irregular. It has a small opportunity to access the same row. Thus it is useful to pre-charge the row before accessing the bank, the latency waste on pre-charge latency can be saved. We can see the dynamic method 2 (bank-miss no pre-charge) achieve a better performance than dynamic method 1 (bank-miss pre-charge). This is because the probability of retuning the same row is weak in the random access process. Pre-charging the bank before accessing it can decrease the latency. The random process will suffer a latency loss when meeting a row-hit status. The dynamic method 2 is useful to solve this problem. In this random process, it almost has no row-hit status so the improvement of this dynamic method 2 is not great.

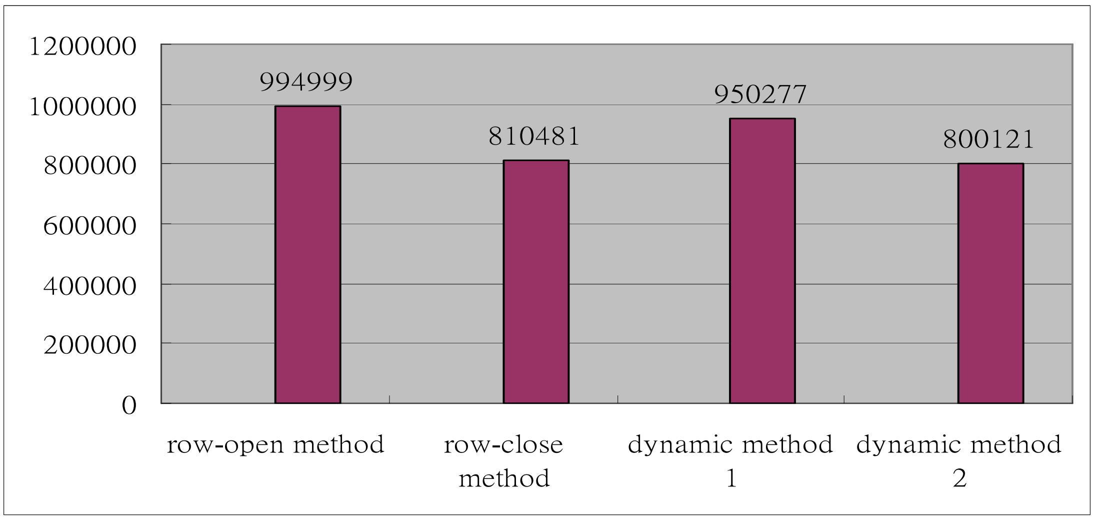


Fig. 4.18 Latency of different method for random process

I simulate another video process pattern which has more row-miss number. The simulation result is shown in Fig. 4.19. We can the dynamic methods have the better performance than row-open method and row-close method.

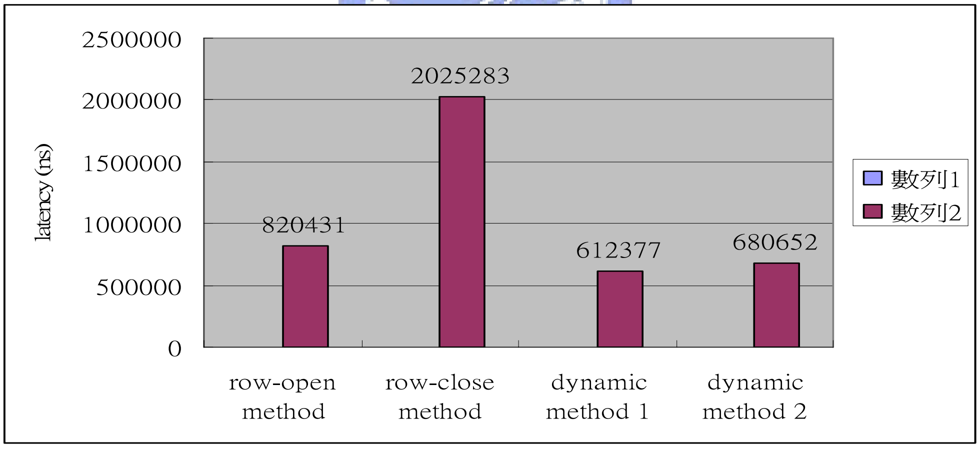


Fig. 4.19 Latency of different method for video process_1

To summarize the result, if the pattern is regular the dynamic method 1 will have more improvement than dynamic method 2. When the pattern is more random the dynamic method 2 does a better work than dynamic method 2.

The latency in the row-open method will be similar with the latency in the dynamic method 1 when the process doesn’t have row-miss status. But when the number of row-miss status increases in the video process we can see the improvement in Fig. 4.20.

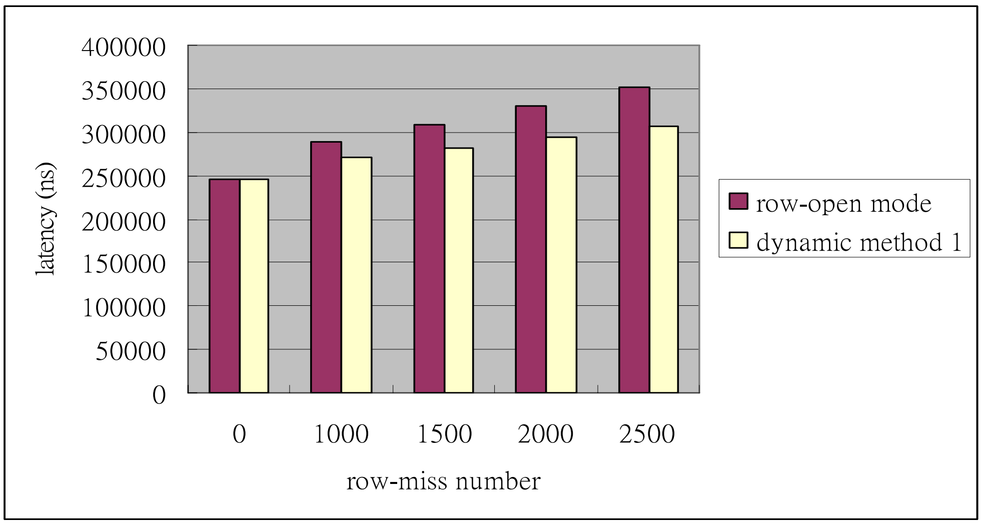


Fig. 4.20 Dynamic method 1 versus row-open method in the video process

By the simulation result we get a final conclusion, we prefer use dynamic method 1 in video process and use dynamic method 2 in random process. When the process is extremely regular or random using the conventional method is recommended.

4.4.4 FIFO Size Effect

The FIFO size in the EMI will affect the latency of accessing data. Because the external memory is double data rate, and the system bus is single edge triggered. The output data rate and input data rate are different in the FIFO. The output data rate of read-data FIFO is slower than input data rate. If the read-data FIFO is going be full, the external memory will stall and stop fetch commands from command-FIFO. Inversely when the write-data FIFO is going to be empty in write operation the external memory need to stall. The architecture is shown in Fig. 4.21.

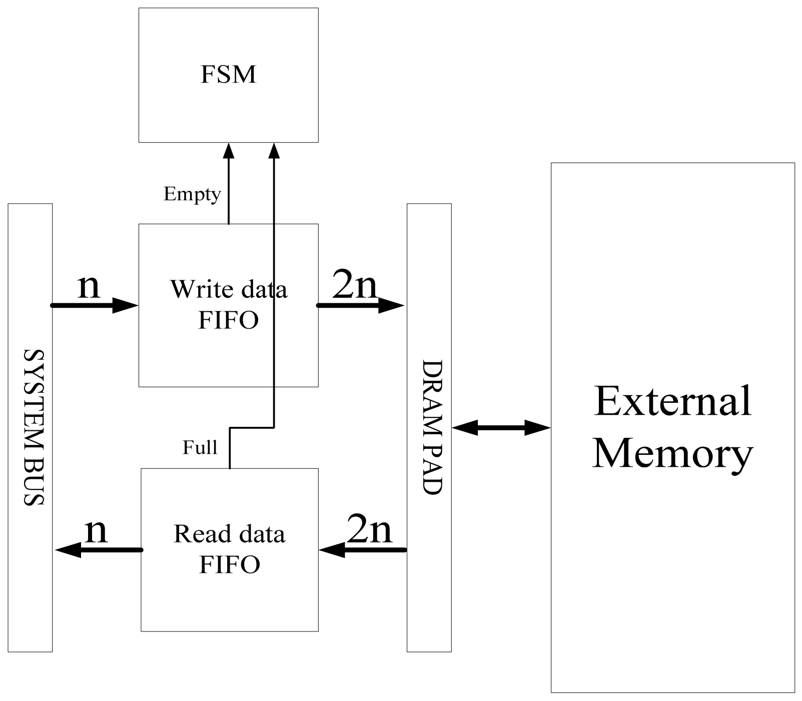


Fig. 4.21 Architecture of stall machine

If the external memory stalls too many times, the respond latency will become longer. This EMI has three option of FIFO size can be chosen including 8-word FIFO, 16-word FIFO, and 32-word FIFO. The respond latency of this three different read-FIFO size is shown in Fig. 4.22.

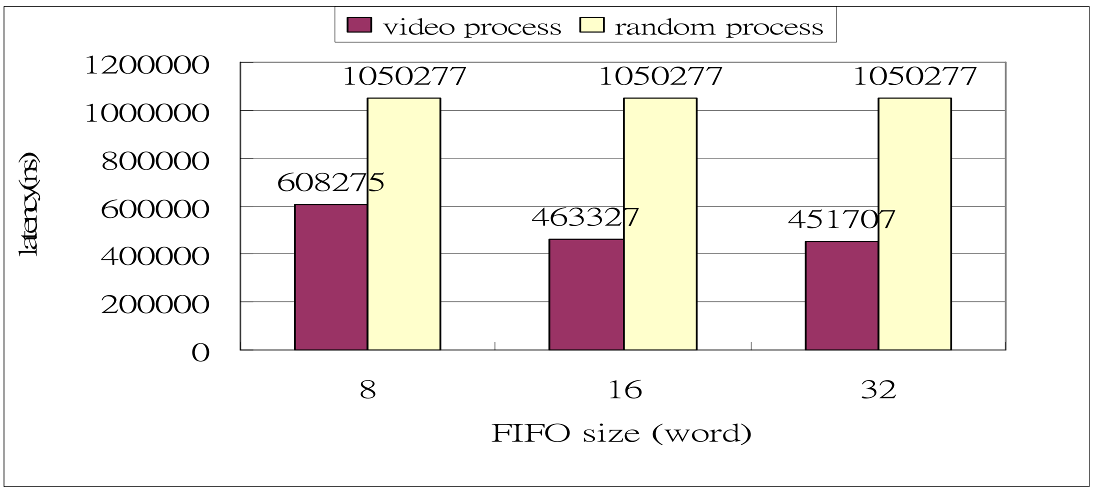


Fig. 4.22 Respond latency v.s read-data-FIFO Size

As we can see when the size of read-data FIFO is bigger the probability of being full degrades. The external memory don't need stall too many times so the respond latency is shorter when the FIFO size is bigger in video process. The respond latency of random process is not affected by FIFO size. Most of the time the external memory doesn't send data to the read-data FIFO, thus the data bus utilization is not high enough to make the read-data FIFO become full. The read-data FIFO size will not affect random process, but has a big impact on video process.

The latency of different write-FIFO size is shown in Fig. 4.23. When the size of write-data-FIFO is bigger, the system can write more data into the FIFO. Thus the external memory doesn't need to stall too many times for waiting data.

The data bus utilization in the external memory is very important to FIFO size. The bus utilization in video process is usually very high and so the size of FIFO should be large. The bus utilization in random process is usually very low and so the size of FIFO will not affect the performance. Therefore I provide three different size of FIFO. The users can dynamically choose the FIFO size based on their process.

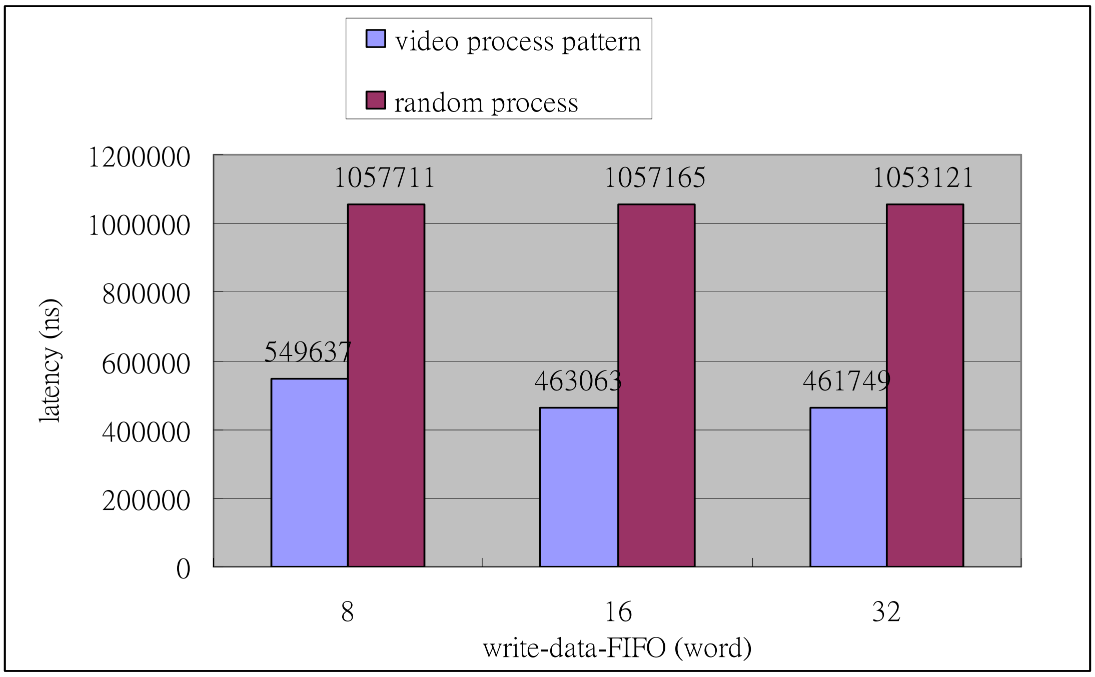


Fig. 4.23 Respond latency v.s write-data-FIFO Size

4.5 Summary

For multi-core SoC designs, the performance of memory subsystem is even more important, due to the share of memory bus with different access requirements of these heterogeneous cores. The data transfer to off-chip memory is especially important due to the scarce resource of off-chip bandwidth. Although the tremendous progress in VLSI technology provides an ever-increasing number of transistors and routing resource on a single chip, and hence allows integrating heterogeneous control and computing functions to realize SoCs, the improvement on off-chip communication is limited due to the number of available input/output (I/O) pins and. As many recent studies have shown, the off-chip memory system is one of the primary performance bottlenecks in current systems.

In this chapter, we have presented an external memory interface (EMI). Users can conveniently integrate this EMI to their system. The users just need to set the parameter for the type of external memory they use. Some other mode control methods and the buffer size are suitable for different kinds of process. Users can dynamically choose them which are beneficial to their system. The configurability of this EMI can alleviate the burden for system designer. Besides, this EMI is able to improve the performance including latency, bandwidth, and power.

To support complex multimedia applications, architecture of multimedia systems must provide high data bandwidth, high speed, and low power. Furthermore, a multimedia operation system should support real-time scheduling. In order to test the performance and the configurability, I choose mobile-DDR DRAM as my external memory model and select the H.264 decoder as our video process. In this chapter, this EMI only simulate with the mobile-DDR DRAM. The pattern I simulate is not complete and lacks the characteristic of the video process. In the next chapter, we will integrate this EMI and the mobile-DDR DRAM into the H.264 decoder system. By simulating with real video process patterns we can see the performance of our EMI.

Chapter 5
Address Translation Machine & Memory Subsystem for H.264 Decoder

The EMI (layer 0) of the memory controller has been proposed in previous chapter. This EMI can reduce the latency of row-miss status and bank-miss status, but the latency still longer than row-hit status and bank-miss status. For the reason gives above, we add an address translation machine which can increase the probability of row-hit and bank-hit status in my memory controller. The performance of this memory controller in video process can be improved after combined address translation machine with EMI. The detail of this address translation machine and the whole memory subsystem will be introduced in this chapter. The experimental result of the memory subsystem with different granularity in the H.264 decoder will also be shown in this chapter.

5.1 Introduction

To improve memory bandwidth and power consumption in video applications, a new address translation machine is proposed. This address translation machine is used for H.264 decoder. The advantage of the address translation machine is the accessing to external memory can become more regular. Since the translation can minimize the number of overhead cycles needed for row-activations in synchronous DRAM (SDRAM), we can improve memory bandwidth and energy consumption significantly. The features of SDRAM and memory-access patterns of video-processing applications are considered to find a suitable address translation which can improve the performance of whole system.

As the resolution of video-processing applications becomes high and H.264 supports the high compressing efficiency, video signal processors should deal with a large amount of data within a tightly bounded time. Due to the large amount of data transfer, video data are stored in the external memory that are usually slow, and thus the system performance strongly depends on the memory bandwidth between processors and external memories.

The data transfer in the video decoder is especially huge in order to support different level and complex mode. To meet the requirement, we must exploit the characteristics of video-processing algorithms. From the deterministic characteristic, most memory access patterns can be known at compile time. The regularity of memory-access patterns can be effectively used to reduce the number of clock cycles required in array accesses.

Besides the high memory bandwidth, low-power consumption becomes an important factor to be considered in system design. As the power related to memory accesses dominates the whole system power in data-dominated systems, it is essential to reduce the memory power consumption. In the external memory, row-activation and pre-charge operation are dominant in dynamic power consumption. When the accesses to memory become more regular the number of active and pre-charge operation can be decreased. Therefore the dynamic power consumption is greatly decreased.

Using the address translation machine has another advantage. Since the switching of the address bus between processors and external memory is also a major source of power consumption, we can save power spend on bus transition by using the address translation machine. For another issue, if we share the bus to both address line and data line. When we don't have to send address to memory every access, the bus will not be occupied by sending address. We can have more time on transferring data without waiting for the bus.

In this chapter we will combine an address translation machine (ATM) with the external memory interface (EMI) I introduced in the Chapter 4 to be the memory controller in the H.264 decoder. The architecture of this memory controller is shown in Fig. 5.1. The EMI use the characteristic of external memory to improve the performance and the ATM use the characteristic of video processing to improve the performance.

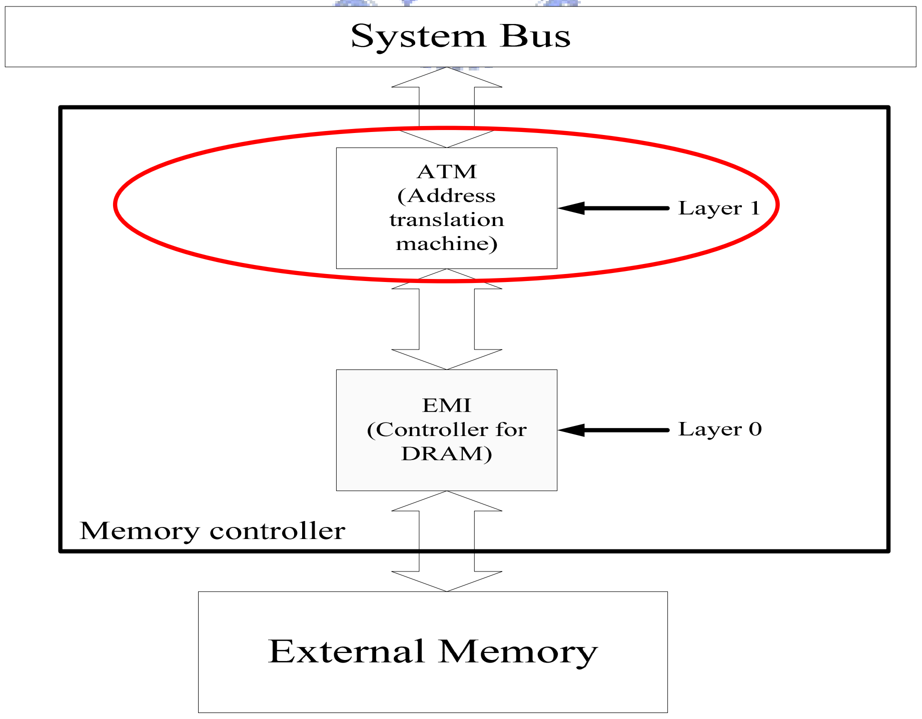


Fig. 5.1 The memory controller in H.264 decoder

5.2 Memory Subsystem Architecture

The memory subsystem we proposed is shown in Fig. 5.2. There are three modules that need to access the external memory via AHB-bus and memory controller. Specifically, the IIP module fetches the motion-compensated data from the external memory for prediction. The DB module writes back the reconstructed block and the DEI module reads the decoded fields for de-interlacing. The physical DRAM addresses are generated by the address translation unit with the motion vector as input. The external memory interface, on the other hand, generates associated DRAM commands. In particular, we use the DDR DRAM for higher data bandwidth.

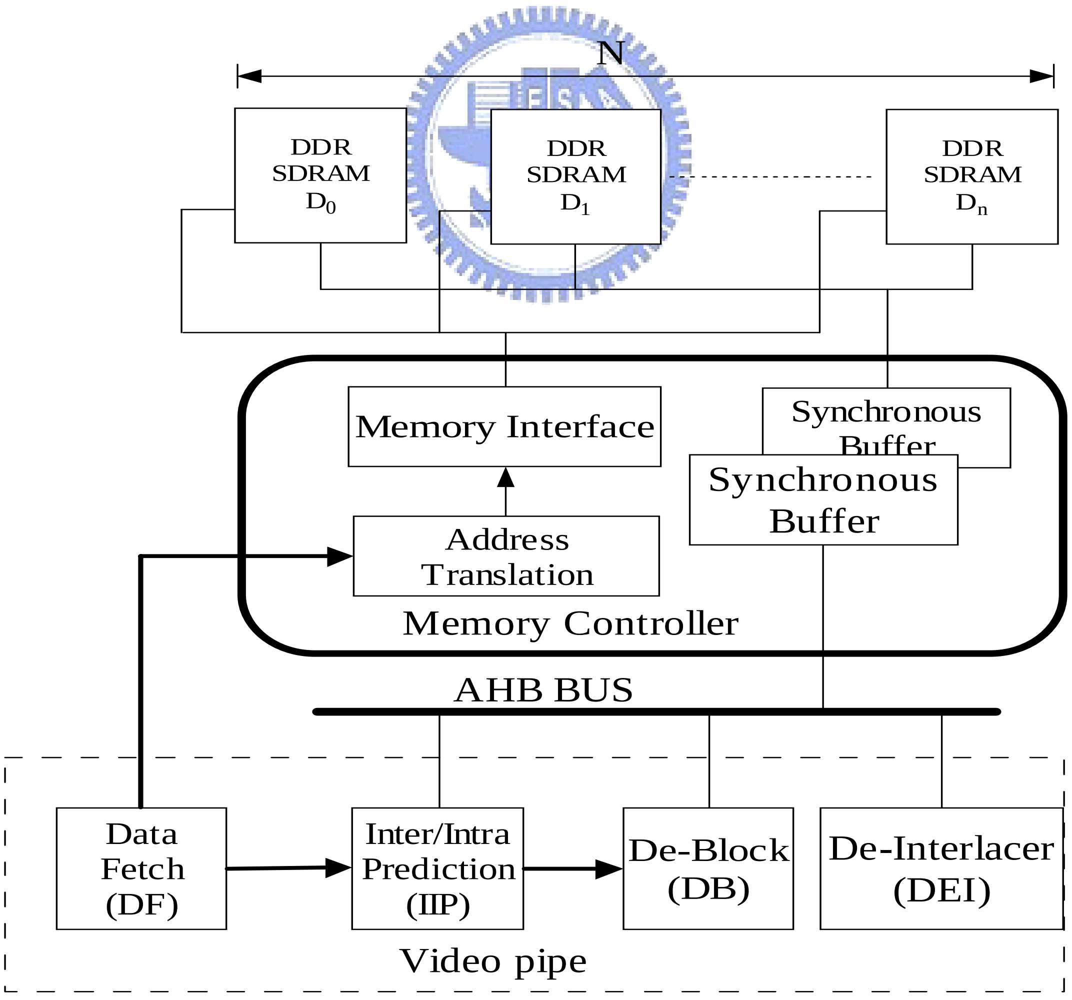


Fig. 5.2 Memory subsystem Architecture

In our design, all the data required for IIP, DB, and DEI modules will be firstly stored in the synchronization buffer so as to enable concurrent DRAM access and video decoding. Particularly, the synchronization buffer is implemented by two SRAMs with size being determined by the block granularity.

Table 1 shows the cycle counts per macroblock (MB) when the synchronization buffer is designed at the levels of 4x4, 8x8, and 16x16. Note that the number is based on the worst case assumption in which the sub-pel interpolation of a 4x4 block requires the most input data, no redundant data are fetched for different blocks, and the input data are distributed in different banks.

Table 5.1 Dram requirement for real time requirement in design of worst case

Granularity	**4 x 4**	**8 x 8**	**16x16**
Cycles/MB with 1 DRAM	1488	992	579
Equivalent configuration for real-time requirement			
# of DRAMs	8	4	1
Cycle/MB with N DRAMs	720	604	579
Bandwidth Utilization (DRAM)	25%	42.4%	79.4%
Width of AHB bus (bit)	128	108.5	50.84

As shown, smaller block size causes poor DRAM bandwidth utilization because of more frequent DRAM active and pre-charge operations. Equivalently, for the same real-time requirement and clocking rate, the configuration with smaller block size requires more number of DRAMs and wider AHB bus. In Table 1, when one DRAM is used, only the block size of 16x16 can fulfill the real-time requirement. The block size of 8x8 need four DRAM to achieve real-time requirement. The 4x4 block size can’t reach real-time even if we use eight DRAM, i. e., 660cycles/MB with clock rate being 162MHz.

Because of the 16x16 granularity costs large amount of synchronous buffer and the 4x4 granularity requires too many DRAMs, we finally choose 8x8 block size to be our granularity.

5.3 Data Arrangement

We use Mobile-DDR SDRAM (32bit width) to be our external memory. In order to increase the bandwidth of our H.264 decoder, we use 1~4 Mobile-DDR SDRAMs and use a AMBA bus. The relationship of our H.264 and memory controller and external memory is shown in Fig. 5.3

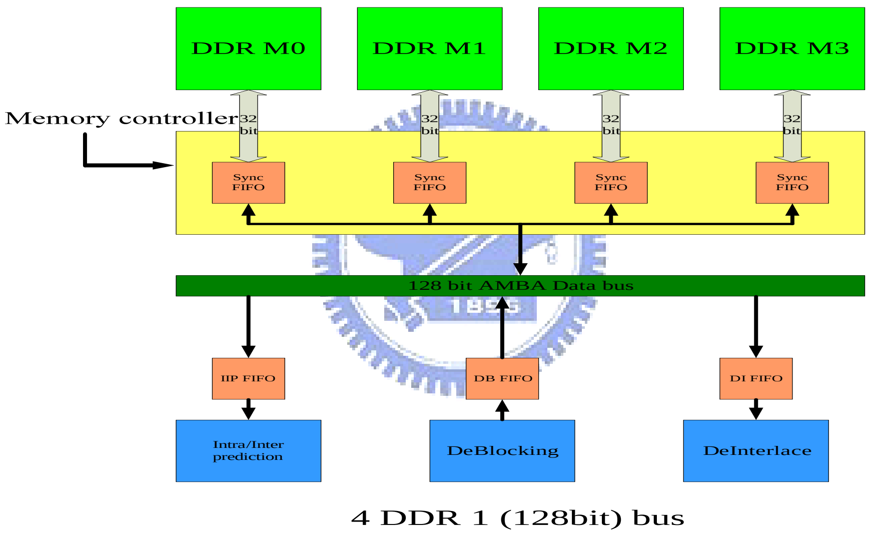


Fig. 5.3 Architecture of memory organization

5.3.1 Memory Mapping Method

There are one to four DRAMs in our subsystem and there are two kinds of memory method. If the number of external memory is four we will access luma block and chroma block simultaneously. If the number of memory is less than four we will interlace the luma block and the chroma block. The latency of different method is show in Table 5.2. We can see the interlaced method is more suitable for 2-memory configuration and simultaneous method is more useful for 4-memory configuration.

Table 5.2 Different memory mapping method for different memory configuration

	Luma (8x8) cycle	Chroma(8x4) cycle	Total cycle
1-memory(interlaced)	60	30	60+30=90
2-memory(interlaced)	60/2=30	30/2=15	30+15=45
2-memory(simultaneous)	60	30	60
4-memory(interlaced)	60/3=20	30/2.5=12	32
4-memory(simultaneous)	60/2=30	30/2=15	30

Memory mapping simultaneous method: (four DRAMs)

We use two memories to store the luma block and use another two memories to store the chroma block so that we can access luma block and chroma block at the same time. Take luma block for example, Fig. 5.4 and Fig. 5.5 illustrate how the luma block is stored in the memory.

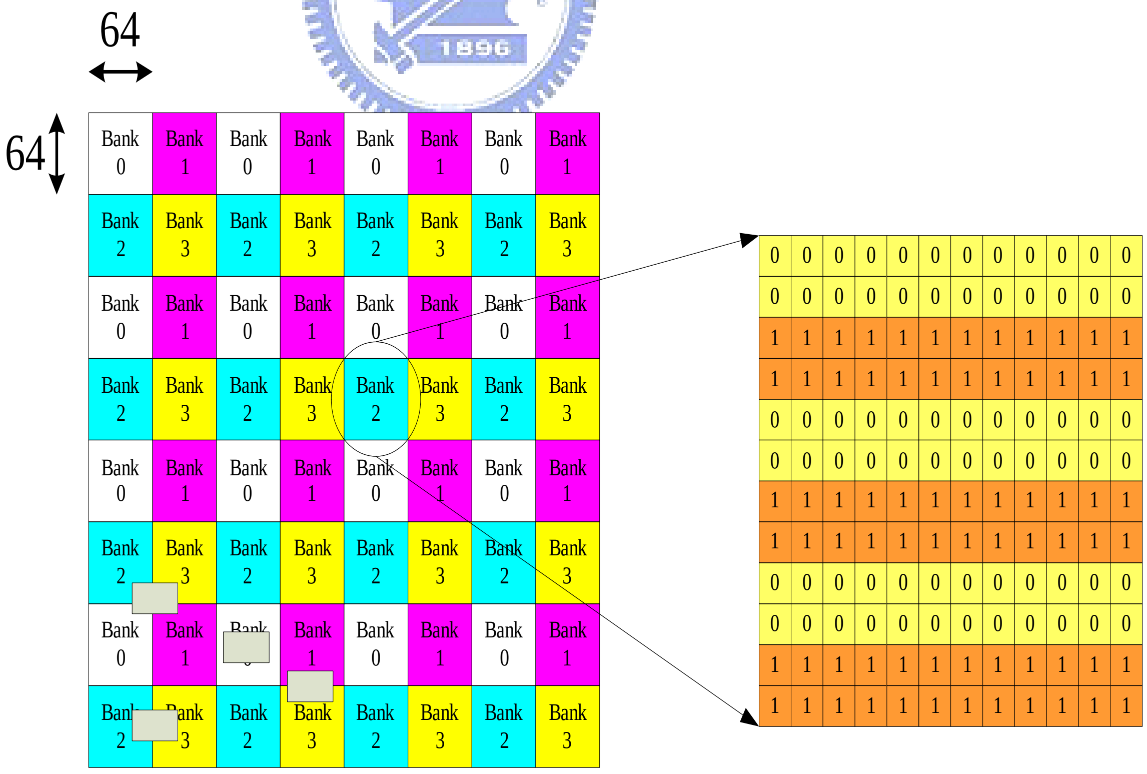


Fig. 5.4 Frame map to memory

As shown in Fig. 5.4 , the frame is divided into four parts. Each part is stored in the different banks. This frame is stored in memory 0 and memory 1. We can see the enlargement of a single bank; we change the memory per two pixels. The yellow part represents memory 0 and the orange part represents memory 1. The advantage of using two memories is it can reduce almost half the latency to access data.

Each check represents one particular row in that bank. As we can see no consecutive rows in the same bank is put together. As a result, when we want to reference a block in the frame the row-miss status will not appear. Only the row-hit status and bank-miss status occurs. As we have mentioned in the previous chapter the row-miss status causes most bandwidth utilization loss and longest latency. In this way, when we decrease the number of row-miss status we can utilize the finite bandwidth and shorten the latency.

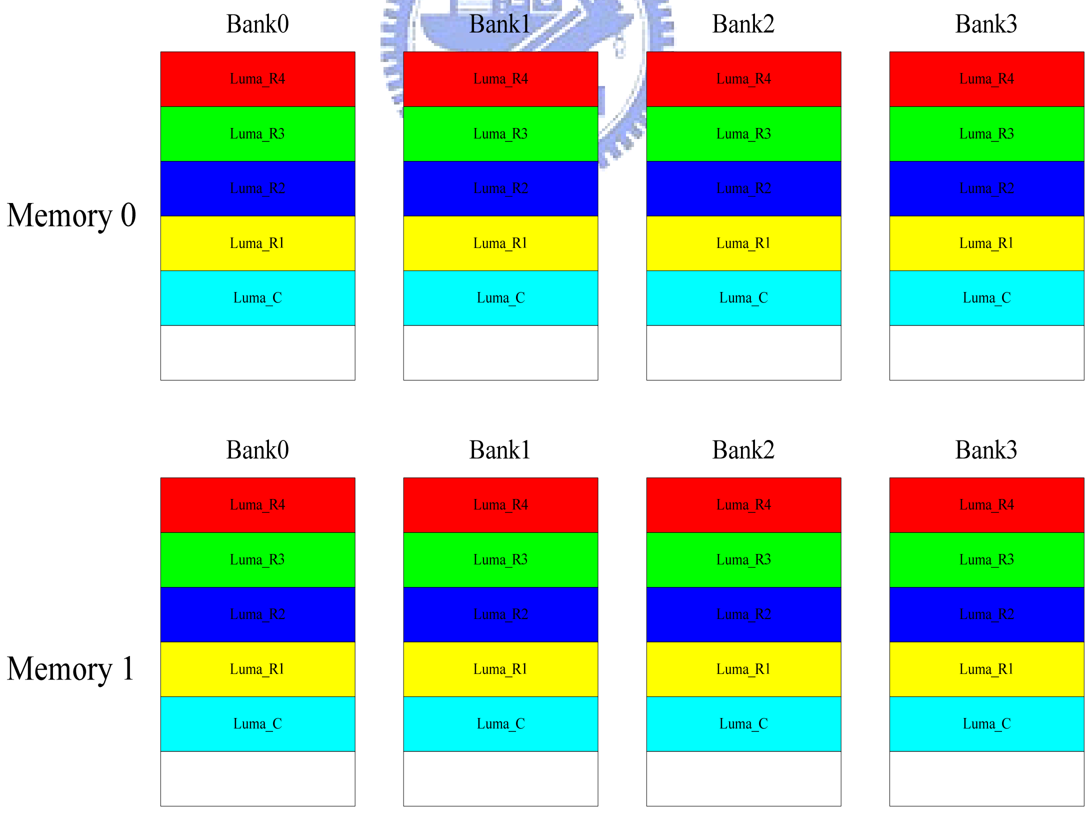


Fig. 5.5 Memory map to frame

Fig. 5.5 indicates the memory organization. There is one current frame and many reference frame need to be stored in the external memory. This is because this H.264 support multi reference frame. There are eight banks in two memories. Each frame is stored in the eight banks equally. This data arrangement leads to we can access data in memory 0 and memory 1 simultaneously. The proportion of data in each memory differs a lot will suffer a great memory bandwidth loss.

Memory mapping interlaced method: (one or two DRAMs)

If the number of memory is less than four we will interleave the luma block and chroma block. Fig. 5.6 shows the frame how to map to memory. The green block means luma block and the blue block means chroma block. We interleave the luma block and chroma block because that chroma block wiil be accessed after luma block. The active and pre-charge operation result from chroma block can be eliminated thus the latency can be decreased. Each 64 x 64 block means a particular row in the bank. We can see the probability of crossing bank increases thus the number of active operation and pre-charge operation for luma block increases but the active operation and pre-charge operation for chroma block will all be removed.

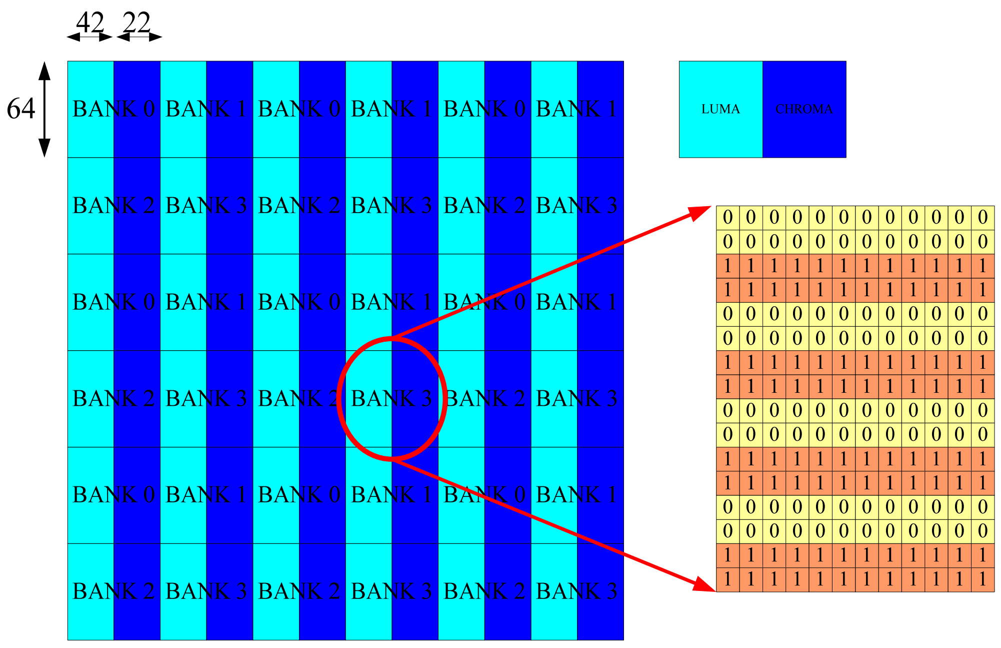


Fig. 5.6 Frame map to memory (1 or 2)

5.3.2 Latency Estimation

The architecture of our H.264 decoder is a pipelined design. Each module will process an 8 x 8 block in a stage. The amount of data each module will access will be described in this section. The cost of latency in each module will also be estimated here.

Inter prediction (motion compensation):

Inter Prediction module is used to decode the inter block by motion compensation. The Data Fetch module will receive the motion vectors from CABAC module. The block which is pointed by the motion vector in the reference frame will be scratched from external memory.

H.264 supports more flexibility in motion compensation, the following are a few example:

1. Selection of motion compensation block sizes (with a minimum luma motion compensation block size as small as 4 x 4).
2. Quarter-sample-accurate motion compensation.
3. Multiple reference picture motion compensation.
4. Motion vectors over picture boundaries.
5. Weighted prediction
6. Improved “skipped” and “direct” motion inference.

Due to the complexity of the motion compensation, the access latency to reference frame varies case by case. The worst case of accessing an 8 x 8 luma block for a P frame is shown in Fig. 5.7.

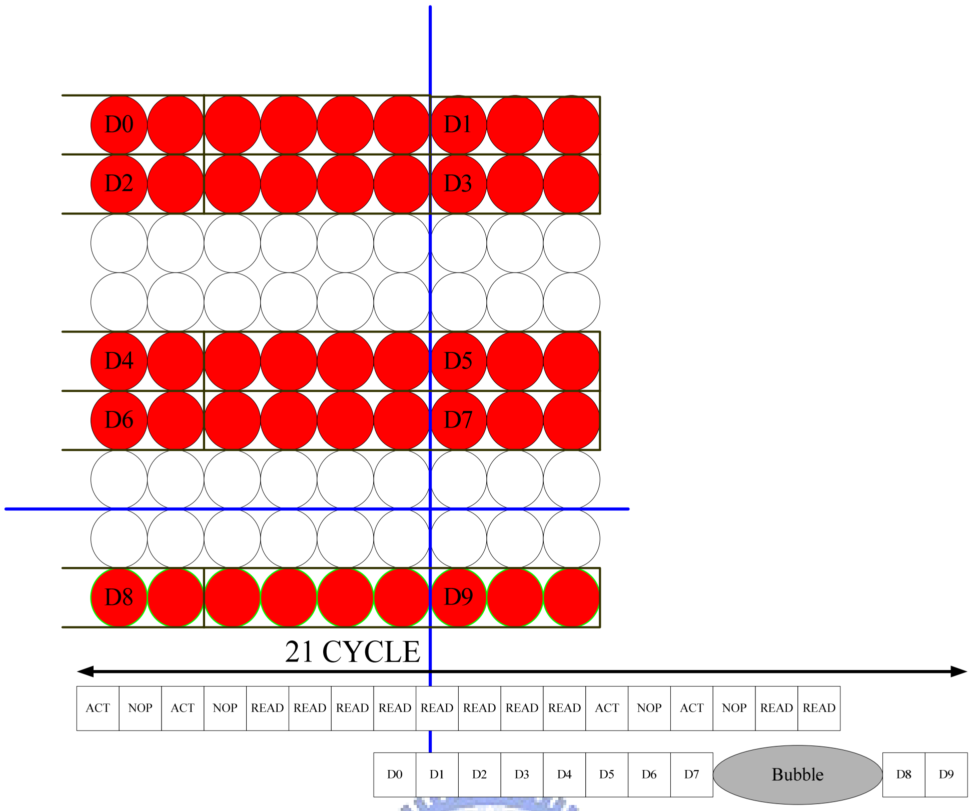


Fig. 5.7 Worst case of accessing a 4 x 4 luma block (P frame)

The worst case is when the block size is 4 x 4 so that we need to scratch four 4 x 4 blocks for an 8 x 8 block. The motion vector of the 4 x 4 block is 1/4 pixel thus it request to get a 9 x 9 block for interpolation. When this 9 x 9 block locates across four banks it would cost latency on activating the other three banks. The latency of scratching a 4 x 4 block is 21 cycles. As a result, it takes 21*4=80 cycles to read reference frame in worst case.

The worst case of accessing an 8 x 8 luma block for a B frame is shown in Fig. 5.8. It needs to read two 13 x 13 block in worst case. Reading a 13 x 13 block needs 25 cycles. The B frame uses bi-directional prediction. It need to read two reference block to do the motion compensation thus it takes 50 cycles for worst case.

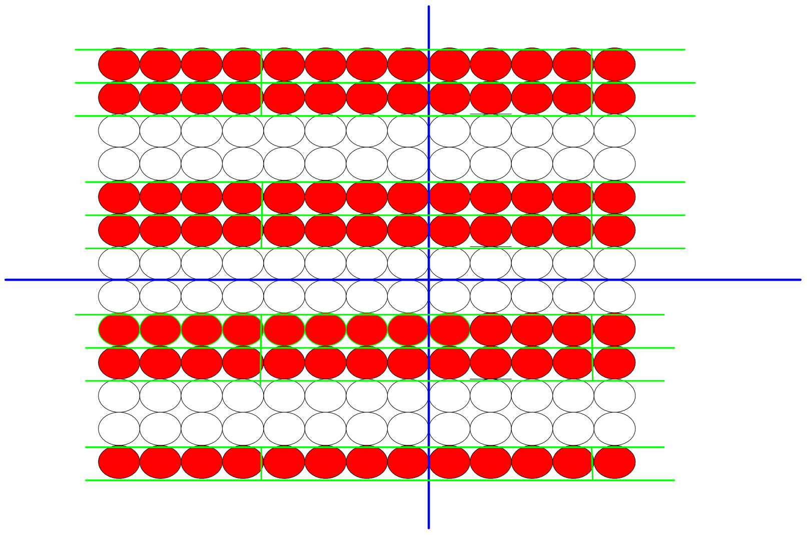


Fig. 5.8 Worst case of accessing a 8 x 8 luma block (B frame)

This memory system will read memory 0 and memory 1 for luma block and read memory 3 and memory 4 to for chroma block. The structure of reading reference frame for motion compensation is shown in Fig. 5.9. It costs 84 cycles to read reference frame for inter prediction block totally.

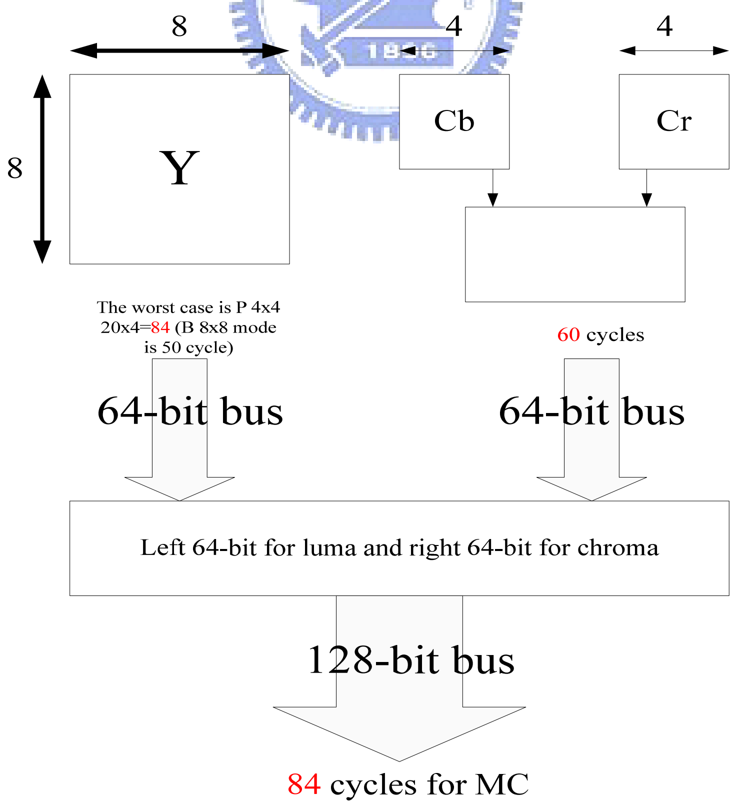


Fig. 5.9 Read operation for motion compensation

De-blocking:

De-blocking will write the decoded block into the external memory. The worst case of luma and chroma block is list in Table 5.2. We write luma block and chroma block to the four memories at the same time. As shown in the Tabl1 5.1, the luma block dominates the cycle counts.

De-interlace:

De-interlace module will de-interlace the fields that read from external memory. The cycle counts is shown in Table 5.3.

Table 5.3 Cycle count of each module in worst case

Module	Block size in worst case	Cycle count in worst case
Inter-Prediction	Luma : 4 (9x9) block Each is (8+10+3)=21 cycle	21* 4=84 cycles
	Chroma: 4 (6x3) block Each is (8+4+3)=15 cycle	15*4=60 cycles
De-blocking	Luma : 2 (4x4) block 3(8x4) block Each is (3+2+3)=8 cycle	8*5=40 cycles
	Chroma: 2 (2x2) block 3 (4x2) block Each is (8+1+3)=7 cycle	7*5=35 cycles

De-interlace	Luma : 1 (16x9) block 1(16x4) block Each is (4+10+3)=17 cycle Each is (3+4+3)=10 cycle Chroma: 1 (8x5) block 1 (8x2) block Each is (4+3+3)=10 cycle Each is (4+2+3)=9 cycle	17+10=27 cycles 10+9=19 cycles

5.3.3 Data Bus Schdule

In our design of H.264 decoder, our video pipe contains seven modules which are CABAC, IQ/IDCT, Data Fetch (DF), Intra-Inter prediction (IIP), De-Blocking, and De-Interlacer. CABAC is the first module in video pipe and the functionality of CABAC is decoding the bit-stream syntax below slice header. Because our decoder processes luma and chroma components in parallel and CABAC can not decode chroma components until all luma coefficients have decoded in one macroblock, it is more efficient to make CABAC operate in macroblock level. For saving the buffer size, all other modules operate in 8x8 block. The max macroblock processing rate is 245760 (MB/s). The operating frequency of our decoder is 162 MHz. In order to process an 8x8 block per stage, we have 165 cycles to deal with memory access. The cycle counts can be derived as follow.

(1) 1/245760 = 4.06 us / MB
(2) 1/162MHz = 6.17 ns / cycle
(3) 4060/6.17 = 658 cycles/MB
(4) 658/4 = 164.5 cycles/8x8 block

The cycle counts spend on accessing the external of each module is listed in Table 5.1. We can see the motion compensation dominate the time to access memory. The total cycle is shown as follow. Fig. 5.10 shows how we schedule the data bus for each master module.

84 (MC) + 40 (DB) + 27 (DEI) = 151 (cycles)

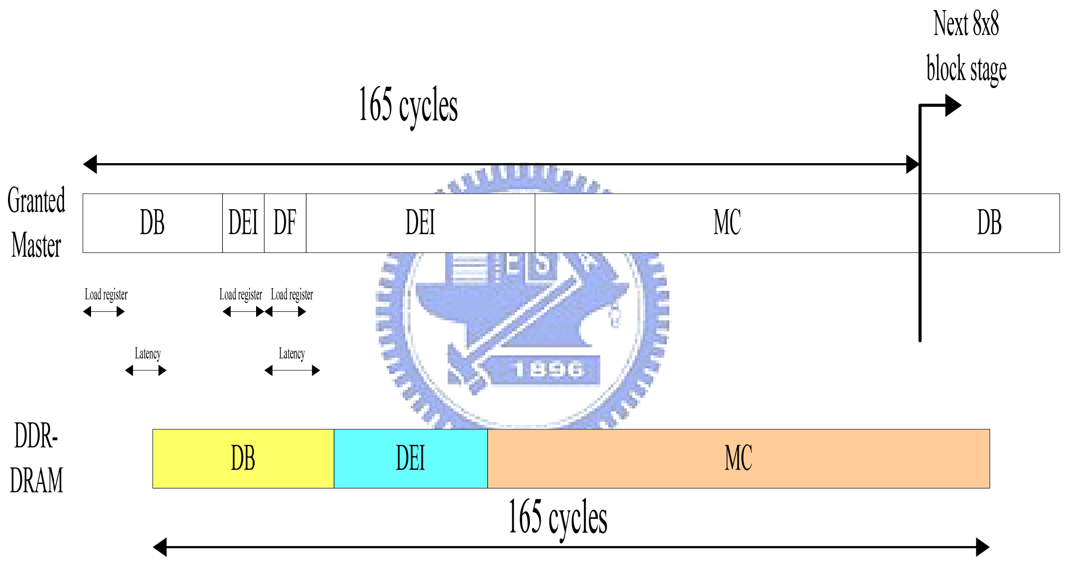


Fig. 5.10 Read operation for motion compensation

The DB master need to write control register first then it takes a short latency to generate address from layer 1 (ATM) of memory controller). When Layer 0 (EMI) receive the address it will translate addresses into appropriate commands and DRAM will start to work. After DB is completed, the master is turn to DEI module, it will write the control register and change to DF to write the control register. The master will return to DEI module to receive the data output from DRAM. After the data transfer is completed, the master will turn to MC to read the data.

5.4 Address Translation

The layer1 of our memory controller is address translation machine. It will generate the addresses by the data arrangement method we have discussed in previous section. The diagram of this layer is shown in Fig. 5.11

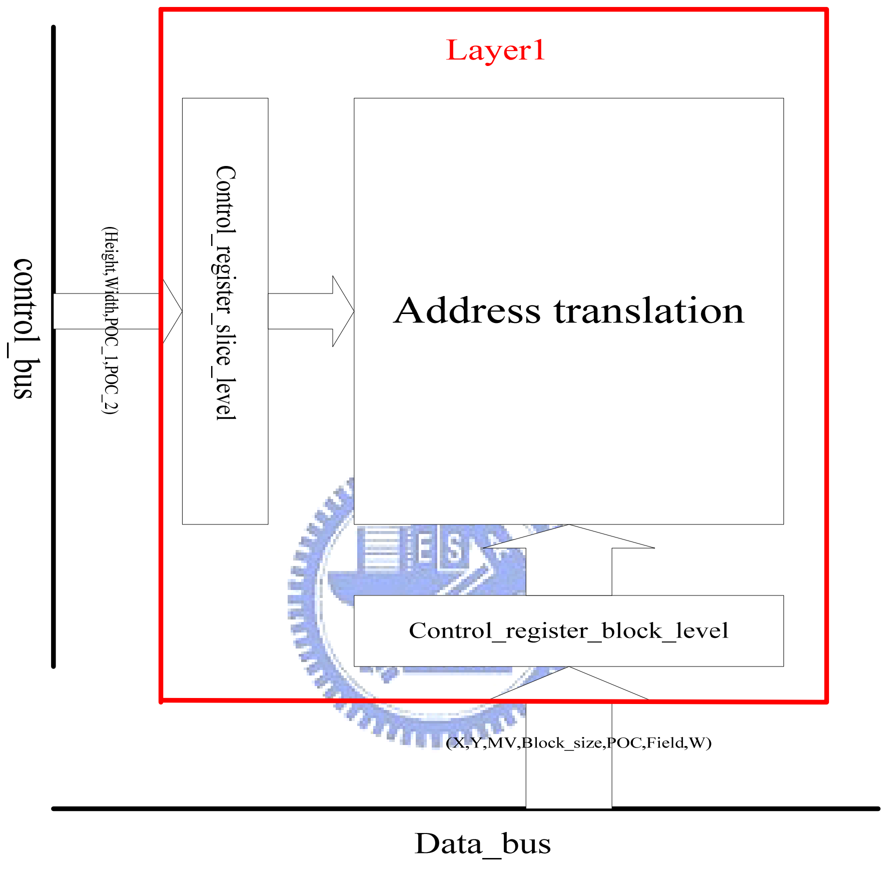

Layer1
control_bus
(Height,Width,POC_1,POC_2)
Control_register_slice_level
Address translation
Control_register_block_level
(X,Y,MV,Block_size,POC,Field,W)
Data_bus

Fig. 5.11 Layer 1 architecture

The data bus will transfer the control signal in block level. These control signals include initial location (X,Y), motion vector (MV), block size, frame or field, and read or write.

The control bus will send the information about slice level. These control signals include height, width, POC_1, and POC_2. The address translation will generate appropriate address based on the control signal which stored in the control register. The address will be send to EMI to generate commands.

5.5 Analysis & Simulation Result

5.5.1 Design for worst case

In this section, we present the experimental framework used to evaluate the considered DRAM controllers, we will measure the hit-rate and miss-rate of different type of frame. The latency and the bandwidth utilization of different picture will also be shown in this section. In the previous section, we have estimated that the granularity of the 8x8 block in worst case needs four DRAM. This memory subsystem design is for worst case, we use 8x8 block to be our granularity and use four external memories.

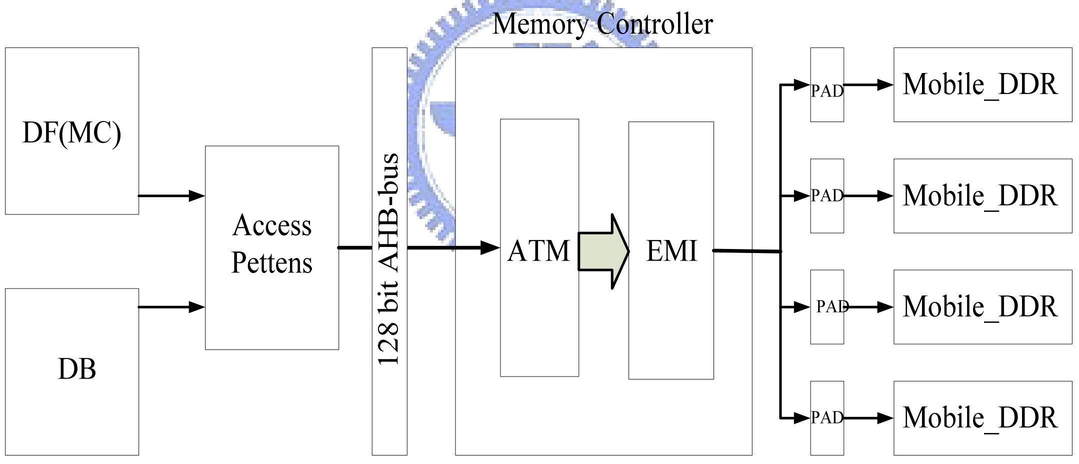


Fig. 5.12 Read operation for motion compensation

Fig. 5.12 shows the test-bed used to evaluate the considered memory controller. The descriptions of some key components of the test-bed are listed as followed.

The DF module and DB module are the functional blocks in the H.264 decoder. The DF module will generate the patterns that read reference frame for motion compensation. The DB module will generate the patterns that write the reconstructed data into memory. This pipelined system is operated on the 8x8 block thus the patterns will change module every 8x8 block.

The patterns that pass through address translation machine will be translated into external memory addresses. The external memory interface (EMI) will change these addresses to appropriate commands that are acceptable for the external memory.
The important settings of my EMI are listed in Table 5.4.

Table 5.4 EMI setting

EMI Setting	Option			
Data rate	SDR	DDR		
	N	Y		
Data bandwidth	x8	x16	x32	
	N	N	Y	
Buffer size	8-word	16-word	32-word	
	N	Y	Y	
Burst length	2	4	8	
	Y	N	N	
Auto-pre-charge method	Row-close	Row-open	Dynamic 1	Dynamic 2
	N	N	Y	N

In order to decrease the latency we choose DDR SDRAM as our external memory and select 32-bit to be the data bandwidth. The buffer is 16-word or 32-word. The burst length is two so that we can access four word data each read/write command. We don't choose four or eight to be our burst length is because the data usage will degrade when the burst length is increasing. We have shown the dynamic method 2 is suitable for video process in chapter 4 so we use dynamic method 1 to auto-pre-charge the memory.

We use Micron Mobile-DDR SDRAM to be our external memory. The ket parameter of this external memory model is shown in Table 5.5.

Table 5.5 Key parameters of micron Mobile-DDR SDRAM

Parameters	Values	Parameters	Values
Clock rate	162MHz	tRCD	3 cycles
Bus width	32bit	tRP	3 cycles
Internal bank	4	tRAS	7 cycles
Burst length	2,4,8	CAS latency	3 cycles

Simulation Result

The frame size we simulate is 1920 x 1080 (HD), this High-Definition sequence has 8160 macro-block (MB) per frame. The sequence name is rush hour. The frame display order we encode is shown in Fig. 5.13. The POC number means the display order. We will show the simulation result of this group of picture (GOP). There are seven pictures in this GOP.

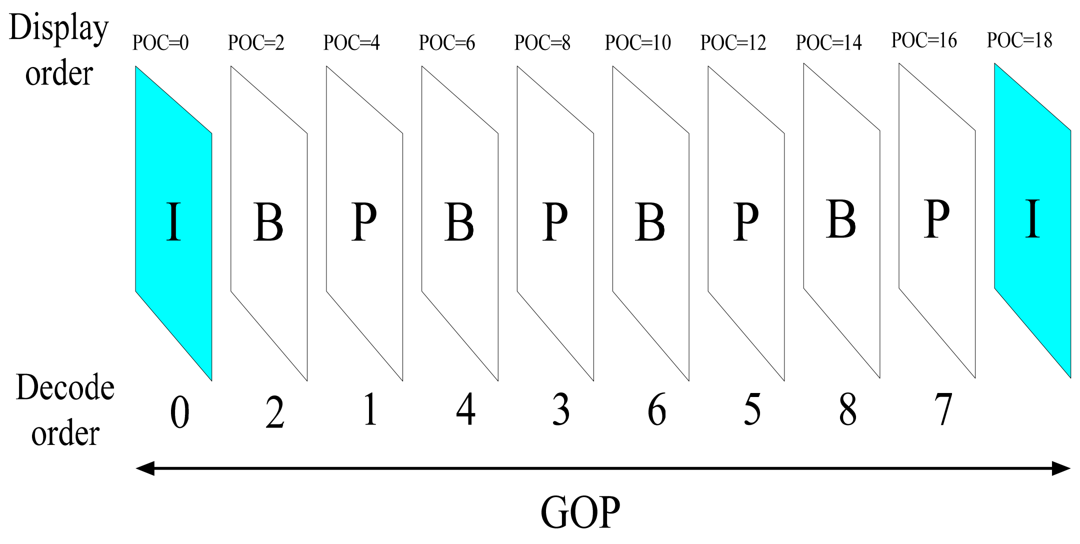


Fig. 5.13 Display order & Decode order

The commands of each frame are in each memory are shown in Table 5.6. We have four external memories in our memory subsystem. Table 5.6 shows the command number in the memory.

Table 5.6 Number of commands in each frame (Luma)

Mem / Frame	M0 commands	M1 commands	M2 commands	M3 commands	Intra block number
0I	265024	265024	0	0	8160
1P	67208	66031	265024	265024	6650
2B	432160	430483	101629	99731	411
3P	29511	29537	371233	371711	4916
4B	312654	316522	206543	204532	396
5P	12920	12863	431944	430092	3436
6B	303033	302932	201711	203026	374
7P	7094	6904	446479	445909	3254
8B	309217	309353	214379	203612	254

The macro-blocks in I frame is all intra MB, it only need to write the decoded MB into memory. I-frame doesn't need to read reference frame thus there is no command from DF (motion compensation) module. The second decoded frame is P frame. Because of the decoded frame is written to another two memories, memory 0 and memory 1 only receive the commands from DF (motion compensation) module. The third frame is B frame which can read reference frame from previous frames and next frames. We can see most of the MB in B frame is encoded as inter block so the commands from DF module is much more than the second frame.

Now we look at the special case in the frame which is shown in Table 5.7. The first three frames shows the number of command in memory 0, and the last four frames shows the number of commands in one of the memories which have the maximum commands.

Table 5.7 Some cases in the memory (Luma)

Case	Decode Order	Memory	commands	Intra block
Case1	0 I (P O C = 0)	M0	265,024	8160
Case2	1 P (P O C = 4)	M0	67,208	6650
Case3	2 B (P O C = 2)	M0	432,160	411
Case4	5 P (P O C = 1 2)	Worst case	431,944	3436
Case5	6 B (P O C = 1 0)	Worst case	303,033	374
Case6	7 P (P O C = 1 6)	Worst case	446,478	3254
Case7	8 B (P O C = 1 4)	Worst case	309,217	254

Case1 shows the operation of writing the reconstructed frame into memory. All commands are from DB module. Case 2 indicates the read operation for DF module. Memory 0 will only receive the commands from DF module. We can see the number of command in this P frame is much fewer than I frame. This is because of the most of the MB is encoded as intra block, only a few MB is encoded as inter block which need to read from external memory. Case 3 is a B frame. Memory 0 will receive commands from both DF module and DB module thus the number of command is more than the case 1 and case2.

Case 4 to case 7 shows the maximum command in the memory, other memories will wait until the commands in the worst case memory are all completed.

The proportion of row-hit status, row-miss status, and bank-miss status in the external memory for different cases are shown in Table 5.8.

Table 5.8 Some cases in the memory (Luma)

Frame	Command	Row-hit status	Row-miss status	Bank-miss status
I	265024	238723(90%)	6828(2.57%)	19471(7.34%)
P	67208	56039(83.38%)	375(0.56%)	10792(16%)
B	432160	357191(82.65%)	19924(4.6%)	55043(12.73%)
P	431944	355389(82.2%)	20601(4.76%)	55952(12.95%)
B	303033	261787(86.38%)	13381(4.4%)	27863(9.2%)
P	446478	365678(81.9%)	20621(4.61%)	60177(13.48%)
B	309217	266959(86.33%)	13317(4.3%)	28939(9.35%)

The first I frame only has write commands from DB module, and the second P frame only has read commands from DF module. We can if the frame doesn't need to change module the proportion of row-miss status is very rare.

The B frame will write decoded frame and read reference frame from different frame. It needs to change module every 8x8 block, we can see the proportion of row-miss status in the frame which need to change module is almost 5 %.

We know the row-miss status consumes most latency than any other status. The low proportion of row-miss status can help us to decrease the latency and increase the bandwidth utilization. On the other hand, the row-miss status will pre-charge the previous row and active the new row. The pre-charge operation and active operation consume a lot of power. If the proportion of row-miss status decreases, the power can be saved.

The different types of frame have different memory access time. The latency of each memory is shown in Fig. 5.14. The yellow part means the depth of FIFO size is 16-word long. The red part means the depth of FIFO size is 32-word long

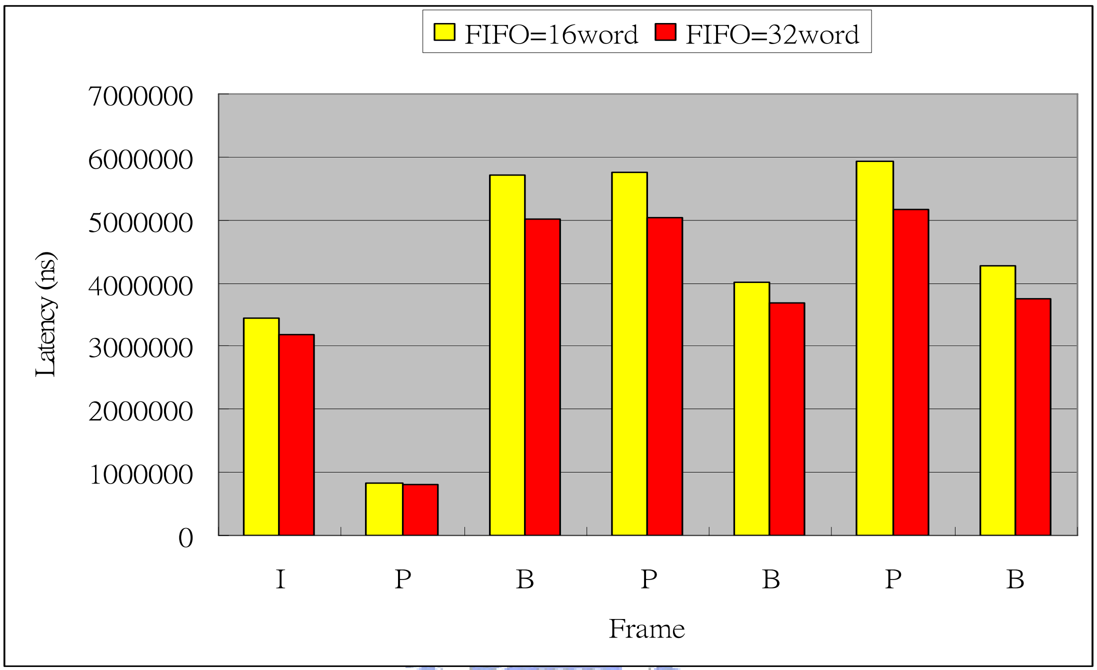


Fig. 5.14 Latency of different frame

We can see the latency rise at a rate proportionate to the number of commands. The first I frame and the second P frame have less commands thus the number of stall is small. When the FIFO size becomes bigger the improvement on latency is limited. From the third frame to the last frame the number of stall is bigger due to the numerous commands. We can see when there is a great deal of accessing to the external memory is an expended FIFO can reduce the latency apparently.

Now we look at the FIFO size effect to the power of memory. The number of active operation and pre-charge operation is shown in Fig. 5.15 and Fig. 5.16.

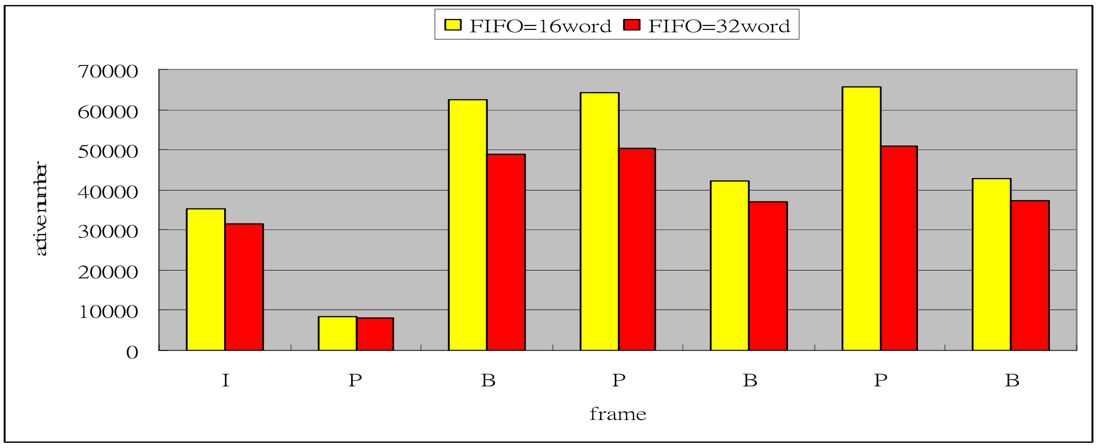


Fig. 5.15 Active number of different frame

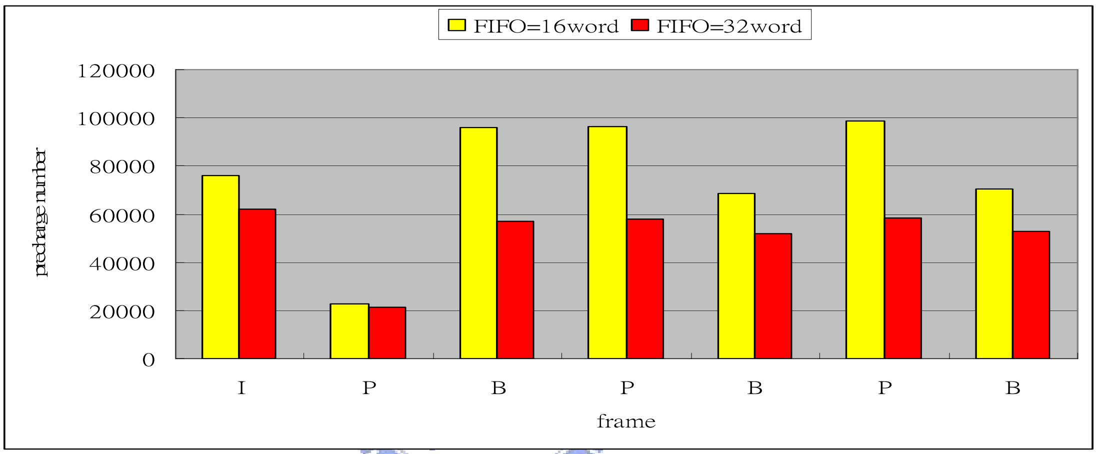


Fig. 5.16 Pre-charge number of different frame

The expanded FIFO size can help decrease the number of active operation and pre-charge operation. The active operation per command and pre-charge per command is listed in Table 5.9.

Table 5.9 Active frequency and pre-charge frequency

	I		P		B		P		B		P		B	
FIFO size	16	32	16	32	16	32	16	32	16	32	16	32	16	32
Active	0.13	0.11	0.12	0.11	0.14	0.11	0.15	0.12	0.14	0.12	0.15	0.11	0.14	0.12
Precharge	0.29	0.23	0.33	0.31	0.22	0.13	0.22	0.14	0.23	0.17	0.22	0.13	0.23	0.17

The higher bus utilization induces better throughput for our H.264 decoder. Due to the limited I/O pins of the external memory the memory data bus utilization is very important in the whole decoder system. Fig. 5.17 indicates the bandwidth utilization for each frame in different FIFO-size.

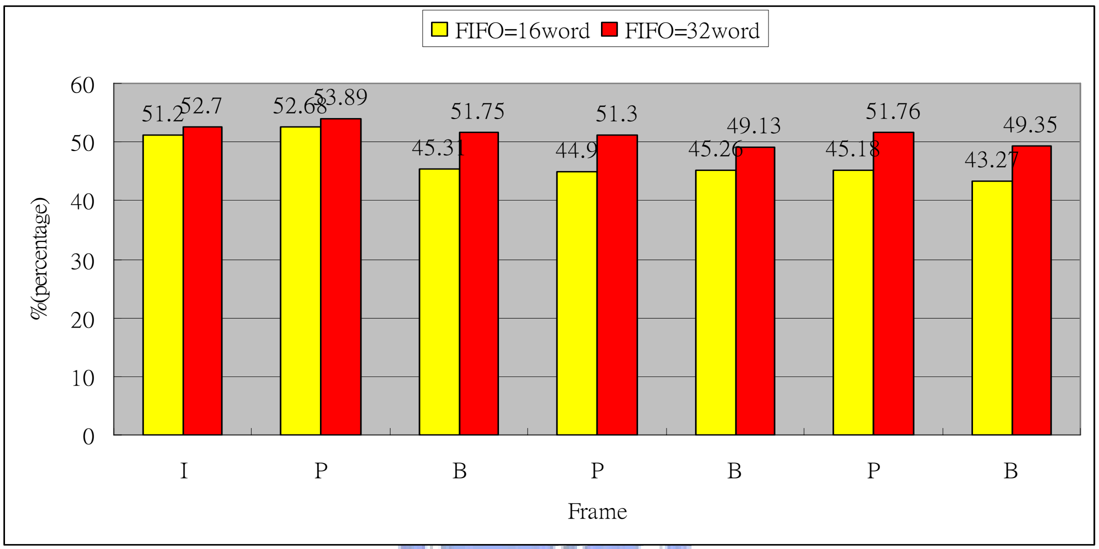


Fig. 5.17 Bus utilization of different frame

We can the first frame I only receive the commands from DB module. All the commands in I frame is write command. The second frame is P frame, it only receive the commands from DF module. All the commands in this memory is read command. The first two frames have higher bus utilization than other frames. The other frame need to change module every 8x8 block thus the bus utilization is not as good as the first two frame which need not to change module.

The expanded FIFO size can accelerate the bus utilization. It reduce the number of stall in the memory so the data can be continuously accessed without stall.The improvement is especially obvious in the frames that need to change module

5.5.2 Design for average case

In H.264, designing for the worst case may not be the best policy since the worst case could only occur in an irrational sequence. The probability of worst case may be very small. Moreover, the bandwidth increase by using more DRAMs could be a problem due to the limited pin counts. In the following, we provide more realistic numbers according to the statistical analysis. We use only one external memory to run different granularity including 4x4, 8x8, and 16x16.

In this section, we statistically analyze the DRAM access when the synchronization buffer is designed at different levels of granularities, including 4x4, 8x8, and 16x16. The analysis is conducted by using one DRAM. To capture the characteristics for different scenarios, the testing sequences include 4 HD sequences (Pedestrian, Rush hour, Sunflower, and Station2.). Each sequence is further coded at low, medium, and high bit rates by using the Qp of 10, 28, and 45 respectively. In addition, to minimize the number of active and pre-charge operations, our data placement in the external DRAM is based on the data arrangement introduced in previous section.

Fig. 5.18 shows the memory access efficiency of motion compensation, which dominates the access of DRAM. The memory efficiency is defined as (data read from DRAM)/ (actual data required for video process). As shown, the access of DRAM has higher efficiency when using larger block size. By firstly observing the block partition and motion vectors, we can prevent redundant data from transfer. Fig. 5.19 further shows the memory access efficiency from the system perspective; that is, the DRAM access for IIP, DB, DEI are simultaneously considered. Again, we have similar observation as in Fig. 2. However, the difference between the 8x8 and 16x16 is not significant at medium or low bit rates.

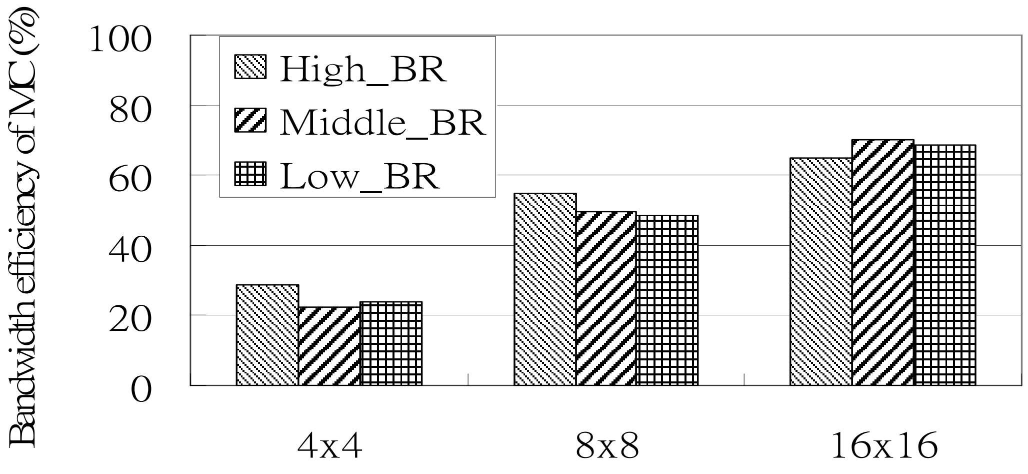


Fig. 5.18 The efficiency of DRAM access for MC

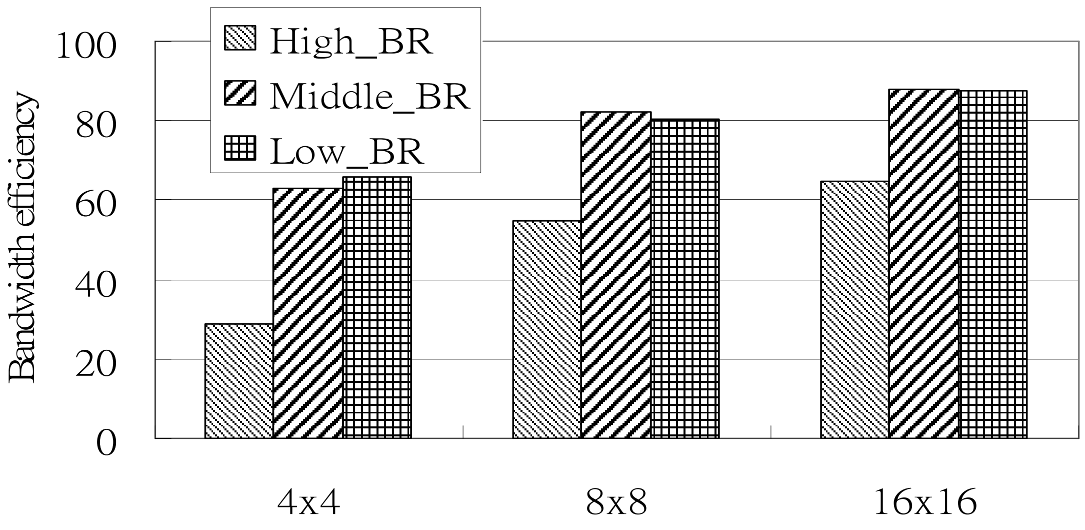


Fig. 5.19 The efficiency of DRAM access for MC,DB, and DEI

To evaluate the DRAM latency, Fig. 5.20 illustrates the average cycle counts for DRAM access per MB. Here, we consider all the data transfer for IIP, DB, and DEI. As expected, larger block size require less DRAM cycles because of higher memory efficiency and the higher bandwidth utilization. The latency is decided by the data utilization. The loss of data utilization cause longer latency. The definition of data utilization is list as follow

Data Utilization = Bandwidth utilization $\times$ Access efficiency

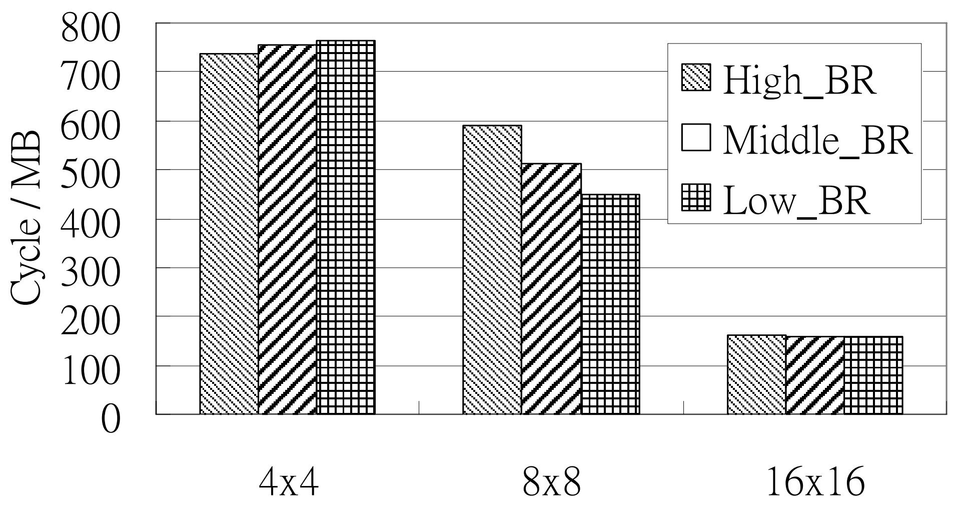


Fig. 5.20 Average cycle count of DRAM access per MB

The distribution of DRAM commands per MB is shown in Fig. 5.21. We can see larger block size has less memory overhead cycle, the smaller block has more r/w commands due to the redundant data. The granularity of 4x4 block size has the max number of active and pre-charge commands. This is because that when the modules which are in the different location are changed more frequently the number of active and pre-charge operations become more.

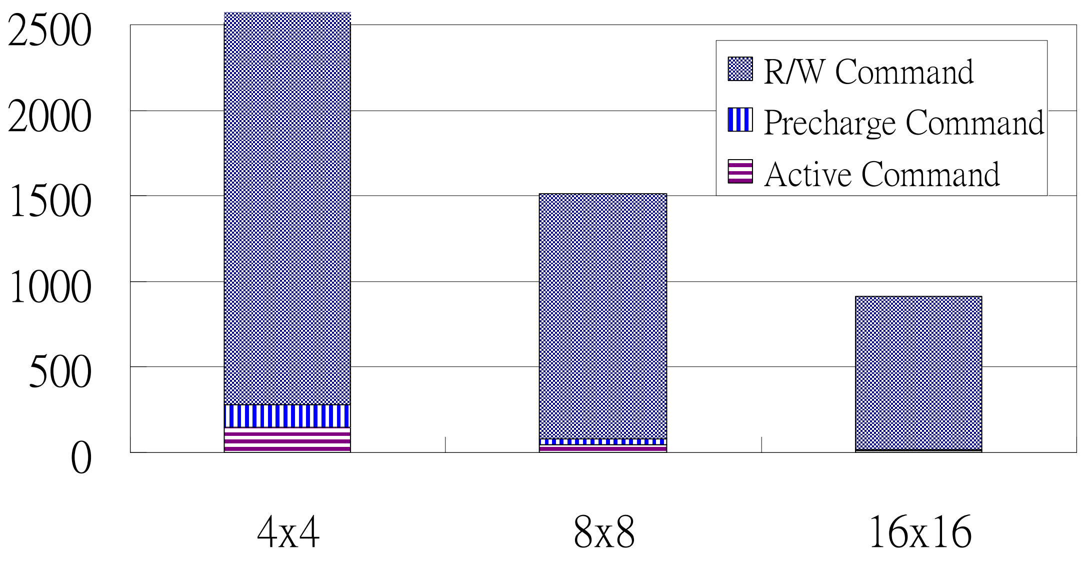


Fig. 5.21 The distribution of DRAM commands per MB

Fig. 5.22 compares the estimated power dissipation for different block granularities according to the DRAM specification. As shown, smaller granularity dissipates higher power due to more DRAM commands. Moreover, since each active or pre-charge command spends 7x power consumption than the read/write commands, larger block size, which has less active and pre-charge operations, is more efficient in terms of power dissipation.

In summary, the synchronization buffer with larger block size has higher memory efficiency, and thus, less DRAM access cycles and power dissipation. However, these advantages are gained at the cost of a larger on-chip buffer. From the statistical analysis, the 8x8 granularity provides better trade-off among cost, efficiency, power, and real-time requirement.

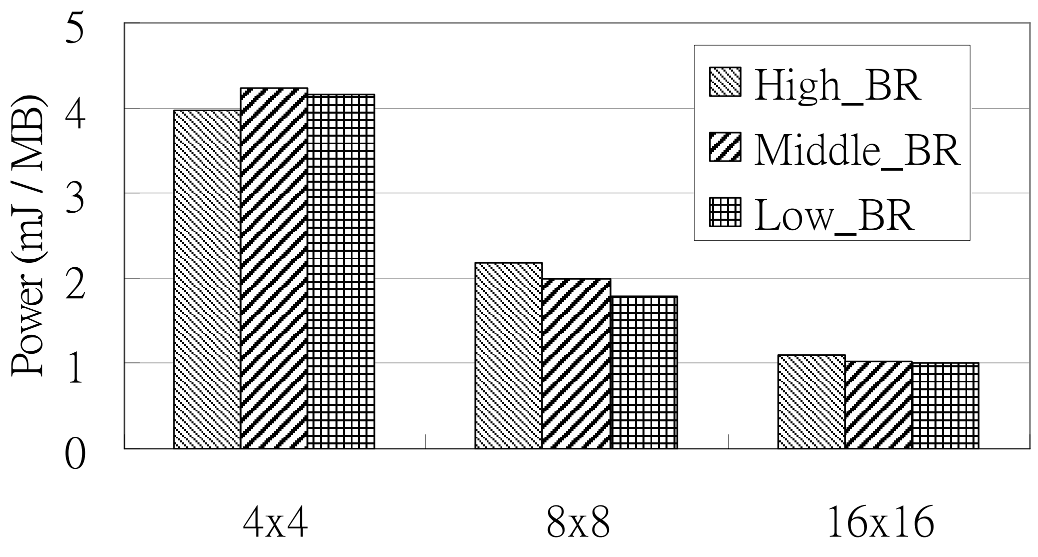


Fig. 5.22 Power consumption of DRAM per MB

5.6 Summary

We have discussed the row-miss status and bank-miss status will induce more latency than the row-hit status. In the chapter 5 we have used some technique to decrease the latency of row-miss status and bank-miss status such as auto-pre-charge method and schedule method. In this chapter, we proposed an address translation method which can reduce the number of row-miss status and bank-miss status. In the experimental result we can see the row-miss status is reduced to almost 5% and the bank-miss status is almost 10%. The latency can be greatly reduced after the address translation.

For the power issue, we expand the size of FIFO inside the memory controller. The expanded FIFO size can reduce the number of active operation and pre-charge operation. The active power and pre-charge power is much bigger than the read/write power. By decreasing the number of active operation and pre-charge operation the power can be saved in the external memory.

In this chapter, we also present the memory sub-system of H.264 decoder and analyze the DRAM access performance at different levels of granularities. From the experimental results, larger block size provides better efficiency of DRAM access at the cost of a larger local buffer. The larger block size can also save power due to the reduction of active and pre-charge operation. The most important of all is the larger block can bring about less latency to achieve the real-time requirement. However, the 8x8 block granularity is sufficient for the realtime requirement while minimizing the number of external DRAMs and local buffer size.

Chapter 6
Conclusions and Future work

6.1 Conclusions

In the video process there are large amount of data need to be fetched to/from the off-chip memory. In order to achieve real-time requirement in the multimedia system, the latency of accessing data is a critical issue. The limited performance of the off-chip memory is still the bottleneck of the video process due to the limited bandwidth. The real-time requirement in video process result in the request of a well designed memory sub-system.

In this thesis, a memory sub-system for the H.264/AVC decoder has been presented. The proposed memory sub-system contains a flexible memory controller, SRAM, AHB-bus, and DDR SDRAM.

This flexible controller contains two layers. Layer 0 is external memory interface (EMI). This EMI is used to decrease the latency and increase the bandwidth utilization. The other advantage of this EMI is its flexibility. The users can change the EMI setting to fit their process. This EMI is also suitable for variable different type of DRAM. The users can integrate this EMI and the external memory into their system rapidly.

Layer 1 of the memory controller is an address translation machine (ATM). This address translation machine is designed for H.264/AVC decoder. This is used to reduce the number of row-miss status and bank-miss status. By decreasing the number of row-miss status and bank-miss status the unnecessary active and pre-charge operation can be eliminated. This ATM bring a great improvement on bandwidth utilization, latency, power of external memory and on-chip bus traffic.

The whole memory subsystem is also discussed in this thesis. The memory subsystem is designed both in worst case and average case. We can see different granularity bring about different performance. The larger block size has the better performance than others but the trade off is the cost of synchronous buffer size. We find that design for worst case is not rational because of the low probability of worst case. We simulate different sequence (HD) and find that using only one external memory is sufficient in the granularity of 8x8 block size. Although using the higher granularity has the better performance, the synchronous buffer at the level of 16x16 is too big (11K byte) to cost the area. We assume the 8x8 block granularity is the adaptable for the real time requirement while minimizing the number of external DRAMs and local buffer size.

The experimental result of a H.264 decoder show that the proposed controller further reduces the access latency by approximately 30% and the memory utilization is accelerated from 42 % to 51%. The different granularity in the memory subsystem brings the power consumption from 1(mJ/MB) to 4 (mJ/MB) and the 4x4 block spend 5x latency than 16x16 block due to the bad access efficiency and bandwidth utilization.

6.2 Future Work

The DRAM technology still progresses and the DDR-3 SDRAM is a trend in the future. This memory controller can be improved to control any type of external memory. On the other hand in the SoC system there are many different kind of memory inside for different request. We wish to create a configurable memory controller that can control the DRAM, SRAM, flash memory simultaneously in the system. The performance of the SoC system would be improved by the efficient memory controller.

REFERENCE

[1] Micron Technology, Inc. product documents. Available: http://www.miron.com/products/

[2] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, " Memory access schedule " *in Proc. IEEE Int. Symp. Computer Architecture*, Vancouver, BC, Canada, Jun. 2000,pp. 128-138

[3] S. Miura, and T. Watanabe , " **A dynamic-SDRAM-mode-control scheme for low power systems with a 32-bit RISC CPU** " *in Proc. IEEE Int. Symp. Low Power Election. And Design,* Aug. 2001,pp.358-363

[4] K. B. Lee, T. C. Lin, and C. W. Jen, " **An efficient quality-aware memory controller for multimedia platform SoC**" *IEEE Trans. Circuits Syst. Video Techno.*,vol. 15, no. 5,pp.620-633, May 2005.

[5] H. Kim, and I. C. Park, **"High-performance and low-power memory-interface architecture for video processing applications**" *IEEE Trans. Circuits Syst. Video Techno.*, vol. 11, no 11,pp.1160-1170,Nov.2001.

[6] S. I. Park, Y. Yongseok, and I. C. Park, **"High performance memory mode control for HDTV decoders**" *IEEE Trans. Consumer Electron.*, vol.49, no.4, pp. 1348-1353, Nov.2003.

[7] J. Zhu, L. Hou, R. Wang, C. Huang, and J. Li, **"High performance syschronous DRAMs controller in H.264 HDTV decoder**" *in Proc. IEEE Int. Conf. Solid-stste and Integrated Circuits Technol.*, vol. 3, 2004, pp.1621-1624.

[8] S. Heithecker, A Carmo Lucas, R. Ernst, " **A mixed QoS SDRAM controller for FPGA-based high-end image processing**" *IEEE workshop signal processing System*, 2003 Page(s):322 – 327.

[9] H.Y. Kang, K. A. Jeong, J. Y. Bae , Y,S, Lee, S. H. Lee " **MPEG$ AVC/H.264 decoder with scalable bus architecture and dual memory controller**" *ISCAS Cicuits and Systems* Volume 2, 23-26 May 2004 Page(s):II - 145-8

[10] J. Tajime, T. Takizawa, S. Nogaki, and H. Harasaki, **"Memory compression method considering memory bandwidth for HDTV decoder"** *LSIs, in Proc. IEEE Int, Conf. Image Processing*, vol.2,1999,pp.779-782.

[11] T. Y. Lee **"A new frame-recompression algorithm and its hardware design for-MPEG-2 video decoder"** *IEEE Trans. Circuit Syst. Video Techni.*, vol. 13, no. 6,pp. 529-534, Jun.2003

[12] E. De Greef, F. Catthoor, and H. De. Man, **"Memory organization for video algorithms on programmable signal processor"** *in Proc. IEEE Computer Design: VLSI in Computer & Processors*, Oct. 1995, pp.552-557.

[13] L. Nachtergaele, F. Catthoor, B. Kapoor, S. Janssens, and D. Moolenaar, **"Low-power data transfer and storage exploration for H.263 video decoder system,"** IEEE J. Select. Areas Commun,. Col. 16, no. 1, pp. 120-129,Jan. 1998

[14] E. Brockmeyer, L. Nachtergaele, F. V. M. Catthoor, J. Bormans, H. J. De Man, **"Low-power memory storage and transfer organization for the MPEG-4 full pel motion estimation on a multimedia processor"** *IEEE Trans. Multimedia,* vol. 1, no. 2, pp. 202-216, June 1999.

[15] K. Denolf, C. De Vleeschouwer, R. Turney, G. Lafruit, and J. Bormans, **"Memory centric design of an MPEG-4 video encoder"** *IEEE Trans. Circuits Syst. Video Technol.,* vol. 15, no. 5, pp. 609-619, May 2005.

[16] S. Wuytack, J. Diguet, and F. V. M. Catthoor, **"Normalized methodology for data reuse exploration for low-power hierarchical memory mappings, "** *IEEE Trans. VLSI Syst.,* vol. 6, no. 4, pp. 529-537, Dec. 1998.

[17] Y. C. Chang Nelson, and T. S. Chang,**"Combined frame memory architecture for motion compensation in video decoding"** *in Proc. IEEE Int. Symp. Circuits and Sustems*. 2005, pp. 1806-1809.

[18] JEDEC Organization, http://www.jedec.org/

[19] Liz. King,**"RLDRAM memory: low-latency, high density, high bandwidth**

DRAM” ,May 2003, available on: http://www.rldram.com/presentation/

[20] Fujitsu,**“32 M bit/ 64 M bit mobile FCRAM**” Jun 2002, available on http://www.fma.fujitsu.com/

[21] Hskwon,**“Memory solution for network application**” Dec.2002,available on http://www.samsung.com/Products/

[22] Micron, **“Mobile SDRAM power-saving features**” Jun 2002 available on http://www.micron.com/products/

[23] Infineon,**“Infinenon specialty DRAMs-Mobile-RAM**” Feb. 2003, available on http://www.infineon.com/cgi/

[24] Churoo Park; HoeJu Chung; Yun-Sang Lee; Jaekwan Kim; JaeJun Lee; Moo-Sung Chae; Dae-Hee Jung; Sung-HoChoi; Seung-young Seo; Tae k-Seon Park; Jun-Ho Shin; Jin-Hyung Cho; Seunghoon Lee; Ki-Whan Song; Kyu-Hyoun Kim; Jung-Bae Lee; Changhyun Kim; Soo-In Cho; **“**A 512-mb DDR3 SDRAM prototype with C/sub IO/ minimization and self-calibration techniques**”** *Solid-State Circuits*, IEEE Journal of Volume 41, Issue 4, April 2006 Page(s):831 - 838

[25] Yamazaki, A.; Fujino, T.; Inoue, K.; Hayashi, I.; Noda, H.; Watanabe, N.; Morishita, F.; Ootani, J.; Kobayashi, M.; Dosaka, K.; Morooka, Y.; Shimano, H.; Soeda, S.; Hachisuka, A.; Okumura, Y.; Arimoto, K.; Wake, S.; Ozaki, H.; “**A 56.8 GB/s 0.18 μm embedded DRAM macro with dual port sense amplifier for 3D graphics controller**” *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International* 7-9 Feb. 2000 Page(s):394 - 395, 471

Vita

PERSONAL INFORMATION

Birth Date:	December. 06, 1982
Birth Place:	Taipei, Taiwan, R.O.C.
Address:	Department of Electronics Engineering National Chiao Tung University 1001 Ta-Hsueh Road Hsin-chu, Taiwan 30010, R.O.C.
E-Mail Address:	hsuan.ee89@nctu.edu.tw

EDUCATION

B.S. [2004]Department of Electronical Engineering, National Chiao-Tung University.

M.A. [2006] Institute of Electronics, National Chiao-Tung University.