國 立 交 通 大 學

電子工程學系 電子研究所

博 士 論 文

系統資料頻寬之研究

Study on System Data Bandwidth for Video and Vision Applications

研 究 生：張彥中

指導教授：張添烜 教授

王聖智 教授

中 華 民 國 九 十 八 年 八 月

系統資料頻寬之研究

Study on System Data Bandwidth for Video and Vision Applications

研 究 生：張彥中　　Student：Nelson Yen-Chung Chang

指導教授：張添烜　　Advisor：Tian-Sheuan Chang

王聖智　　Sheng-Jyh Wang

國 立 交 通 大 學

電子工程學系 電子研究所

博 士 論 文

A Dissertation
Submitted to Department of Electronics Engineering and
Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in
Electronics Engineering

August 2009
Hsinchu, Taiwan, Republic of China

中華民國 九 十 八 年 八 月

系統資料頻寬之研究

研究生：張彥中　　　　指導教授：張添烜 博士
王聖智 博士

國 立 交 通 大 學

電機學院 電子工程學系 電子研究所

摘　　要

資料頻寬問題長久以來就是嵌入式系統效能的瓶頸。若一個系統無法提供足夠的資料頻寬，或運算核心的頻寬需求過高，會造成運算核心缺乏足夠運算資料，因而使運算核心無法發揮全部的運算能力，進而影響到系統的效能。為了發揮系統的效能，本論文探討如何提高系統中可用頻寬使用率與降低頻寬需求。本論文以系統中資料傳輸的過程為主軸，針對傳輸源頭到終端所會遭遇的頻寬問題進行改善的研究。改善方法的核心概念是利用傳輸或存取資料之間的相關性，搭配系統中硬體裝置的工作特性提高頻寬使用率，或是重複利用已有資料減少頻寬需求。在資料傳輸的源頭，也就是記憶體控制器的部份，本論文探討如何透過存取排程，改善資料間的相關性，利用記憶體本身存取特性與系統匯流排協定的工作特性來提高的資料頻寬使用率。在改善資料傳輸源頭的頻寬後，瓶頸便落在系統匯流排上，因此本論文接著探討改善採用先進封包式協定匯流排頻寬使用率的方法。在資料傳輸末端的運算核心部份，本論文則是根據資料間存取的空間與時間的相關性，探討如何透過有效地資料重複使用，降低運算核心的頻寬需求。本論文研究的運算核心功能包含了視訊編解碼與早期視覺處理，這兩類運算的頻寬需求皆十分可觀。在視訊編解碼部份，本論文針對移動補償提出了一利用輸入移動向量與巨方塊種類之內容特性的運算核心硬體架構，可降低頻寬需求達72%。而在早期視覺處理部份，本論文針對平均飄移濾波演算法與立體視覺的立體匹配演算法，提出可減低頻寬需求的運算核心硬體架構。在平均飄移濾波架構中，本論文根據平均飄移向量大小的特

性，提出部分更新乒乓暫存記憶體架構，可減少八成畫面記憶體頻寬需求。另一方面，在立體匹配演算法中，本論文根據演算法本身資料存取在空間與時間上的侷限性與相關性，提出部份列資料重複使用與擴張視窗減少存取兩方法，來達到大幅減少立體匹配資料頻寬需求。本論文基於資料相關性的頻寬改善方法，可提高可用頻寬與降低頻寬需求，有效改善系統中資料頻寬的問題，進而幫助提高系統整體效能。

Abstract

Data bandwidth issue has long been the performance bottleneck of an embedded system. The computation cores in a system cannot maximize their utilization without enough data. This is usually a result of insufficient available data bandwidth or excessive data bandwidth requirement. Being aware of the importance of the data bandwidth issue, this dissertation addressed the bandwidth issue from the source to the destination of data transfers. The core concept was to facilitate the address and data correlation among accesses to solve the data bandwidth issue. Exploiting these correlations can increase bandwidth utilization given a device's access characteristics and can also reduce bandwidth requirement through data reuse. In particular, this dissertation explored methods to increase the bandwidth utilization of memory controllers by taking the advantage of the characteristics of external memories and new advanced data transfer protocol. After improving the bandwidth utilization at the source of data transfers, this dissertation focused on improving the bandwidth utilization of a bus interconnect adopting advanced protocol under the traditional share-link topology. Finally, bandwidth requirement reduction techniques have been studied at the destination of data transfers. For video coding and early vision tasks, these techniques performed data reuse based on algorithm's data access characteristics, such as the spatial ad temporal locality among data accesses. In video coding, this dissertation proposed a *combined frame memory motion compensation* (CFMMC) architecture that was capable of reducing the bandwidth requirement by up to 72% based on the characteristics of input motion vector and macroblock type data. In early vision tasks, this dissertation proposed a meanshift filtering architecture that used the proposed *partial-update ping-pong buffer* (PUPPB), which was based on the access locality due to intermediate meanshift vector characteristics, to reduce the bandwidth to the image memory by 81.6%. In stereo matching vision task, this dissertation proposed the *partial column reuse* (PCR) and *access reduction by expanding window* (AREW) techniques, which were based on the access locality due to the algorithm's flow, to significantly reduce bandwidth requirement for the proposed *mini-census adaptive support weight* (MCADSW) stereo matching architecture. The bandwidth utilization improving and bandwidth requirement reduction techniques studied in this dissertation can also be applied to other video coding or vision systems to improve system performance.

誌　謝

博士的研究生涯，得到了不少人的幫助。首先要感謝任建威博士，在我的碩士和博士初期給予的指導，同時感謝他提供良好的研究環境和資源。另外要感謝我的指導老師張添烜教授與王聖智教授，在我博士班期間毫無保留指導與建議。我要特別感謝張教授鼓勵我朝自己夢想與有興趣的題目進行研究，同時也要感謝王教授願意給我在這方面幫助，讓我受益良多。沒有這兩位教授的幫忙，我今天將不會有實現我自己的夢想的機會。另外，也要感謝我的口試委員：李鎮宜教授、蔡淳仁教授、楊家輝教授、張寶基教授、陳永昌教授、陳紹基教授、蔣迪豪博士，在忙碌中抽空齊聚一堂，參與我的論文口試，並給予十分有幫助的建議，讓我獲益良多。

另外，我也要感謝我的家人。首先，我要感謝我的父母，沒有你們的不間斷的督促與鼓勵，我將無法在論文連續被拒絕時，重新站起來。也要感謝父親，能夠理解並接受我改變研究方向。也要謝謝父母給我一個能後放心追求夢想的家庭。此外，也要感謝兄弟們(這包含友然在內)在博士班其間給我的鼓勵。

最後，要感謝在交大這幾年相遇的學長、同學、學弟妹們。首先要感謝李坤儐學長的用心帶領和指導，另外也要感謝林泰吉學長和李元仲學長願意適時給予有用的意見。同時還要感謝博士班重逢的林佑昆同學和約旦來的Esam同學，在這幾年間相互鼓勵，一起努力。接著要感謝和我一起合作的學弟妹們：浩雲、惠錚、旻奇、宗憲、宇晟、景竹。最後，要感謝張教授實驗室全體學弟妹，謝謝你們這段期間帶來的歡笑和幫忙。最後，也要謝謝我所有的朋友的鼓勵與幫忙。

謹將這本論文，獻給所有關心我的人們，也獻給不幸離開人世的蔡旻奇學弟。

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1. Background

Video coding technology has been developed over the past four decades and has made high quality portable video playback and recording a reality. Examples of portable video playback include the early suitcase TV [1], recent portable media players [2], and newly available iPhones [3]. On the other hand, portable video recording has also been developed after Le Prince invented moving picture recording technology since 1890's. Now, portable video recording can be found on small handheld camcoders to cell phones. These have been possible because of the advances in digital video coding algorithms, which have greatly improved the compression ratio while maintained high video quality. However, advanced digital video coding algorithms are often complex and computation power demanding.

Recently, real-time and portable vision-based applications have been receiving great attention. Vision algorithms, particularly early vision ones, have enabled traditional image/video consumer electronics to become smarter than ever. For instance, face detection and region of interest have improved a camera's focus capability [4] and made a camera seem "smarter" to human users. In addition to traditional image/video consumer electronics, vision algorithms have also been the core technology in robotics, autonomous vehicles, and intelligent surveillance. However, the drawback of using vision algorithms was the enormous computation requirement. The complexity of vision algorithms has been much higher than video coding algorithms. As a result, real-time implementations of vision algorithm were not as successful as in video coding technology.

One common approach to accommodate the computation requirement is to use multi-core embedded systems. A multi-core embedded system may include embedded processors, DSPs, high performance processors, GPUs, and dedicated hardware accelerators to satisfy the computation power requirement of video and vision algorithms. The large computation requirement issue can be easily solved by using a multi-core embedded platform. However, the hard part is the data bandwidth. Without the data to be processed, a computation core can only idle helplessly. Therefore, it is crucial to take care of the data bandwidth issue to ensure an embedded multi-core system works as expected.

1.2. Motivation

Being aware of the fact that the data bandwidth is one of the key factors that affects the system speed, this work explored methods to improve system speed under limited resource. One way to improve the system speed is to increase the available BW by investing more hardware resource. Examples of such approach were using higher clock rate, wider data port width, more buses, and complex crossbar interconnect. However, this approach often led to expensive systems.

Other approaches that demand less hardware resources were maximizing existing bandwidth utilization and reducing bandwidth requirement. Maximizing bandwidth utilization can be achieved by using new data transfer protocols and access methods. One example of new advanced data transfer protocols was AMBA AXI [7], which adopted a packet-based channel scheme to maximize the bandwidth utilization. Better access methods can increase the bandwidth utilization by taking access device's characteristics into account.

Reducing bandwidth requirement can be achieved by reusing data within computation cores so that less data transfer and access outside the cores are needed. Such data reuse often depends on finding data access characteristics of algorithms, such as data access spatial and temporal locality. Another approach in reducing bandwidth was by compressing the data to be transferred. However, this approach often introduced additional compression hardware. Hence compression-based methods were not discussed within the scope of this dissertation.

Motivated by the importance of bandwidth issue and the potential opportunities of the aforementioned approaches on maximizing bandwidth and reducing requirement, this work focused on the study of facilitating the features of new data transfer protocols and intrinsic characteristics of memory devices and algorithms to improve system performance. In this dissertation, the bandwidth bottlenecks in a multi-core embedded system were investigated from the source to the destination of data flow. In particular, the bandwidth issue at a memory controller, a system bus, and application specific cores were studied.

1.3. Dissertation Organization

The first chapter gave a brief introduction on the background, motivation, and the organization of this dissertation. The second chapter presented previous arts and related literatures to help readers better understand the detail background of this dissertation. Chapter 3 presented our study on the bandwidth issue at the source of data transfer, the memory controller. We have presented a method to improve the bandwidth utilization taking the advantage of the memory device's intrinsic characteristics as well as the new bus protocol's feature. Once the bandwidth performance at the source is improved, the system bus would become the data transfer bottleneck. Therefore Chapter 4 analyzed the bus bandwidth issue at system level and provided methods of enhancing the system bus performance by facilitating the features of new bus protocols. Chapter 5 studied the bandwidth issues at the destination of data transfers, the application specific computation cores. Finally, Chapter 6 gave a conclusion and some future work directions.

Chapter 2 Related Works

Related works were categorized into memory controller related, data transfer related, and application specific computation core related. In the memory controller part, previous works on DRAM memory controllers were presented. In the data transfer part, researches on analyzing and improving the bandwidth issue on system bus were described. Finally, in the application specific core part, we investigated previous work on motion compensation in video coding, mean-shift filtering, and stereo vision.

2.1. Memory Controller

Several works have been proposed to improve DRAM performance on bandwidth utilization and latency. It was initially discussed for single core environment with software based techniques to reorder access streams [8]. Instead of software approach, Ayukawa et al. proposed an access-sequence controller [9] which reorders the data input and output order to reduce access latency. However, neither transaction nor command scheduling were mentioned in their work. Later, Rixner et al. proposed a memory access scheduler architecture [10] capable of adopting various combinations of simple DRAM command arbitration policies and can perform bank-interleaving. In these works, they only focused on single core environment and neglected the impact of the arbitration policy on system buses.

For multicore environment, Takizawa et al. proposed a simple memory arbitration policy which reduces bank conflict and read/write turnaround in their MPEG-2 AV decoder system [11]. However, reducing bank conflict by increasing bank-interleaving would result in row precharge and activation increase, thereby increasing the memory power consumption. Recently, Lee et al. proposed an efficient quality-aware memory controller for multimedia platform SoC [12]. It utilizes a quality-aware scheduler to provide quality-of-service (QoS) guarantees. However, their scheduler requires multiple channels which may only be suitable for systems adopting star topology and will result in extra hardware cost for the system.

In the industry, Rambus proposed a pipelined memory controller [13] which includes a bank cache lookup and a command sequencer. Sonics Limited developed MemMax 2.0 memory controller [14] which improves the efficiency of DRAM but must be used their own MicroNetwork on-chip bus standard. For packet-based bus systems, ARM Limited has developed a configurable AXI compliant memory controller [15].

Despite improving bandwidth utilization and access latency, reducing memory energy consumption has also been discussed at system level [16][17][18]. These works essentially turn DRAMs into low power state during idle period by using either software or hardware approaches. However, reducing memory energy consumption in non-idle period should also be addressed to further reduce the system energy consumption. Meanwhile, the system timing constraint must also be met. Such issue has been briefly discussed in Burchardt's work [19]. However, thorough investigation on improving memory energy efficiency in non-idle period still remains rare.

2.2. System Bus

With the rapid progress of system-on-a-chip (SOC) and massive data movement requirement, on-chip system bus becomes the central role in determining the performance of a SOC. Two types of on-chip bus have been widely used in current designs, which are pipelined-based and packet-based bus.

For pipelined-based buses, such as ARM's AMBA 2.0 AHB [23], IBM's CoreConnect [24], and OpenCore's WishBone [25], the cost and complexity to bridge the communications among on-chip designs are low. However, pipeline-based bus suffers from bus contention and inherent blocking characteristics due to the protocol. The contention issue can be alleviated by adopting multilayer bus structure [26] or using proper arbitration policies [27][28]. However, the blocking characteristic, which allows a transfer to complete only if the previous transfer has completed, cannot be altered without changing the bus protocol. This blocking characteristic reduces the bus bandwidth utilization when accessing long latency devices, such as an external memory controller.

To cope with the issues of pipelined-based buses, packet-based buses such as ARM AMBA 3.0 AXI [7], OCP-IP's Open Core Protocol (OCP) [30], and STMicroelectronics STBus [31] have been proposed to support outstanding transfer and out-of-order transfer completion. We will focus on AXI here because of its popularity. AXI bus possesses multiple independent channels to support multiple simultaneous address and data streams. Besides, AXI also supports improved burst operation, register slicing with registered input, and secured transfer.

Despite the above features, AXI requires high cost and possesses long transaction handshaking latency. However, a shared-link AXI interconnect can provide good performance while requiring less than half of the hardware required by a crossbar AXI implementation. This work focused on the performance analysis of a shared-link AXI. The handshaking latency is at least two cycles if the interface or interconnect are designed with registered input. This would limit the bandwidth utilization to less than 50%. To reduce the handshaking latency, we proposed a hybrid data locked transfer mode. Unlike the lock transfer in [32] which requires arbitration lock over transactions, our data locked mode is based on a transfer-level arbitration scheme and allows bus ownership to change between transactions. This gives more flexibility to arbitration policy selection.

With the additional features of AXI, new factors that affect the bus performance are also introduced. The first factor is the arbitration combination. The multi-channel architecture allows different and independent arbitration policies to be adopted by each channel. However, Existing AXI related works often assumed a unified arbitration policy where each channel adopts the same arbitration policy [32][33][34]. Another key factor is the interface buffer size. A larger interface buffer usually implies more out-of-order transactions can be handled. The third factor is the task access setting, which defines how the transfer modes should be used by the devices within a system. Proper task access settings can yield better performance. However, the proper setting may be different under different circumstances, such as different buffer sizes.

Many works have been conducted on the communication architecture of pipelined-based bus. Earlier work used formal analytic approach [35][36] to explore the design space of communication architecture to evaluate the performance of a pipeline-based bus system. Although formal analytic approach can provide the average or best/worst case overall bus performance, such approach can hardly account for instantaneous changes of bus behavior. This limitation gave rise to high-level simulation-based approach which is capable of capturing the detailed instantaneous bus behavior with cycle accuracy [37]. Pasricha et al. [38] used the cycle count accurate transaction boundaries (CCATB) model in the architecture exploration of an MPEG AHB system. Later, Pasricha et al. also conducted experiment on bus architecture synthesis [39] under different given constraint. Their synthesis method yielded cost efficient bus matrices much faster and reliable than manual optimization.

Most of the techniques developed in the abovementioned works can be extended for the analysis of packet-based bus. Pasricha et al. extended their communication architecture synthesis framework to AXI [34]. Their work automatically generates the best bus topology, arbitration policy, and parameter settings driven by throughput requirements. Besides bus topology exploration, comparison between packet-based bus and pipelined bus has also drawn attention. Pasricha et al. [40] compared the performance of a shared-link AXI and a single-layered AHB. Their comparison showed that up to 30% of bandwidth utilization improvement can be achieved by AXI compared with AHB. They also investigated the impact of the transaction reordering buffer size in the memory controller. Lee et al. [41] built a crossbar AXI platform and a single-layered shared-link AHB platform to quantify the performance difference. They reported 40% communication efficiency improvement between AXI and AHB. Ruggiero et al. [42] studied the scalability of AHB, AXI, and STBus under shared bus topology. Their result showed that AXI is far more scalable to the number of master devices than AHB. When the number of processor reaches 8, AXI can achieve 60% bandwidth utilization improvement over AHB.

Comparison of bus connectivity configuration, such as shared-link, multilayer (partial crossbar), and full matrix (crossbar), has also been interested as well. Lahiri et al. [43] proposed a design space exploration methodology and compared the performance between single-layer and multilayer shared-link buses. Recently, Murali et al. [44] presented a bus communication architecture exploration method that finds the most power-efficient crossbar interconnect for a packet-based bus. They also briefly compared the performance and normalized cost ratio among shared-link, multilayer, and crossbar configurations.

Although the aforementioned works conducted analyses on communication architecture, the register slicing impact and multi-channel arbitration issues that arise with the features of packet-based bus have been overlooked. In addition, pervious performance comparison of multi-channel AXI and single-layer share-link AHB may not be fair since AXI requires much more hardware cost.

2.3. ASIP Data reuse related

2.3.1. Motion Compensation in Video Codecs

Motion compensation has been one of the most important tasks in a video encoder or decoder. Motion compensation reconstructs a predicted frame from reading frame data from a frame buffer. The reconstructed frame is written back into the frame memory. In many cases, the bandwidth requirement and size of the frame memory is usually large. To reduce the size of the frame memory in motion compensation, [48][49][50] adopted a merged-frame approach which stored the reference frame and the reconstructed frame together using one frame memory with the size slightly larger than one frame. Along with the reduced size frame memory and local buffers, these work claimed that the merged-frame approach is also capable of reducing the power consumption. Among these works, [48] and [49] proposed an in-place storage optimization for video decoders. The in-place storage used a buffer to store the reference frame data that are overlapped with the reconstructed current frame data in a snake-like manner. To handle the complex address generation and the control, they implemented a prototype using software. [50] also proposed a similar merged-frame memory architecture for motion estimation and compensation in an encoder. Although these works successfully reduced the frame memory size, none of them mentioned further improving the performance of motion compensation by exploiting the characteristic of MBs without motion and residue. Consequently, the bandwidth requirement could not be reduced.

Moshnyaga's works [47][51] on motion estimation reported the presence of block-data whose content remain unchanged between the adjacent frames. These unchanged block-data are facilitated to eliminate frame memory writes and computations during the motion estimation. In order to reduce memory writes for the unchanged block-data, Moshnyaga's work also adopted the merged-frame approach when the coding pattern has no B-frame.

Although the result shown was quite well for the test sequences listed in their works, the experiment result on video sequences with great amount of motion was absent.

2.3.2. Meanshift Filtering in Segmentation

Several hardware architectures have been proposed to improve the processing speed of image segmentation algorithms. However, very few architectures have been proposed for the Meanshift filter algorithm. For segmentation algorithms other than the Meanshift algorithm, Ranbabu et al. proposed a VLSI architecture for the well-known watershed segmentation algorithm [61]. Their architecture could speed up the speed by 3 times compared to their software implementation. Neuenhahn et al. also implemented the watershed algorithm on an FPGA with optimal parameter settings [62]. Their work could segment 576x720 resolution images at a frame rate of 50 FPS. Yamaoka et al. proposed a VLSI architecture that implemented an image-scan based region-growing segmentation algorithm [63]. The image-scan based region growing algorithm had regular process flow and was therefore more suitable for hardware implementation. Their architecture was capable of segmenting up to 230 segments in QVGA images at a frame rate of 30 FPS.

For clustering-based segmentation algorithms, the K-means algorithm has received great attention by hardware designers. Leeser et al. proposed one of the earliest architecture for K-means color clustering algorithm [64] one decade ago. However, Leeser's work did not adapt to the characteristics of the image being processed. Maliatski and Yadid-Pecht proposed a hardware-driven architecture for adaptive K-means algorithm [65]. Their architecture was capable of processing with 64 cluster centers in QCIF images at a frame rate of 15 FPS. Later, Hernandez proposed a "global-quasi-systolic local-hyper-connected" VLSI architecture for histogram peak-climbing image segmentation algorithm [66]. Their work could segment images of 702x576 resolution at a maximum frame rate of 50 FPS. However, the cost of Hernandez' architecture was very large and required a tremendous amount of internal memories. Maruyama and Saegusa also proposed an architecture for filtered K-means color image segmentation algorithms [67]. They used KD-tree to filter out redundant cluster computations. Their FPGA implementation could perform image segmentation on VGA images at an average frame rate of 35 FPS. However, the maximum segment count in their K-mean implementation was limited to 256. Moreover, the performance of K-means algorithm is heavily dependent on good initial guess of the cluster centers. Recently, Chen et al. presented a design for K-means color segmentation [68]. Their work conducted detailed architectural design space analysis and provided a prototype system that can perform segmentation of QVGA images with a maximum segment count of 16.

In contrast to the attention received by the K-means algorithm, the popular Meanshift clustering algorithm seems to have been over sighted by architecture designers despite the Meanshift's well recognized performance in image segmentation. Only one architecture has been proposed in the past. Park et al. proposed a systolic array architecture [69] that implemented the dynamic Meanshift (DMS) [70] filter algorithm. The DMS computes a new mode from the old modes in the Meanshift window instead of the original pixel data in the window. The DMS can achieve super-linear convergence and can reduce the execution time by at least 30%. Park et al. modified the DMS to map the modes onto a regular 2-D grid graph to make the computation less irregular. This mapping was suitable for the systolic array architectures and increased the parallelism. To reduce the cost and memory of the array architecture, Park et al. adopted a sliding window approach. However, their DMS architecture would still need a memory to store the mode at each node in the grid graph. If this mode memory is located externally, the bandwidth requirement between the sliding window and the mode memory would be very large.

2.3.3. Stereo Matching

A. Stereo Matching Algorithms

Disparity estimation algorithms can be categorized into local and global approaches [72]. Local approach determines the disparity of a pixel based on the content similarity between the support windows of this pixel and its candidate pixel in the other image. The local approach usually has low computation complexity and storage requirement, and has been frequently adopted by real-time implementations [74]-[84]. Global approach determines the disparity of all the pixels in an image as a whole by optimizing a global energy function. However, the optimization is usually complex and extremely computation intensive. Hence, we will focus on local approaches.

Early works on local approach studied the impact of different similarity measures [85][86]. Their work pointed out that census, rank [87], and mutual information [88] achieved better disparity estimation performance and were more robust to radiometric distortion. Later, [89] investigated the performance of using different color representation. Recently, [90] investigated the performance and speed jointly of different similarity measure and color representation combinations. The result showed that census-based combination achieved better performance, but also takes more time to compute.

Another important research topic that has been studied is the support window size. The simple fixed size rectangular window adopted in early local approaches suffered from incorrect disparity estimation in occlusion, textureless, and repeating pattern regions. To remedy this, [91][92] proposed variable window size algorithms. Later, [93] also proposed a variable window size algorithm that adaptively adjusted the window size based on a reliability measure. The variable window size could effectively improve the disparity estimation performance in textureless and repeating pattern regions, but not in occlusion regions. Being aware of this, [94][95] proposed shiftable windows algorithms to improve the performance. Kang et al. [96] combined both the concept of variable window size and shiftable window together. However, the qualitative result of their work still showed great room for improvement.

The reason for not being able to completely improve the performance in the occlusion region is because the assumption of same disparity in the window does not hold in occlusion and slanted surface regions. Understanding this, Veksler [97] proposed a compact window class method which could model non-rectangular support windows. Although their result showed significant performance improvement compared to previous algorithms, the performance near the boundary region was still inferior to complex global approaches. Yoon et al. proposed an adaptive support weight (ADSW) [73] algorithm that assigned different weights to the pixels in a support window based on the proximity and color distances to the center pixel. As a result, the ADSW could achieve the effect of using a support window of arbitrary size and shape. With multiple iterations of aggregations, the performance of ADSW was comparable to some of the complex global algorithms. Later, segmentation-based support methods were also proposed [98][99]. The outlier rejection [98] used a binary weight based on the segmentation region instead of the weight used in the ADSW. Tombari et al.'s segment support algorithm [99] only assign weight to the pixels in the same segment the center pixel is in. Recently, [100] conducted a detailed comparison on the performance and processing speed of local algorithms. Their result showed that the segment support has the highest performance but is two times slower than the ADSW. The performance of the ADSW is only slightly inferior to the segment support algorithm.

B. Real-time Implementations

Real-time stereo matching implementations can be categorized into general purpose processor solutions, digital signal processor (DSP) solutions, graphic processing unit (GPU) solutions, and dedicated hardware solutions.

The general purpose processor solutions rely on the great computation power in state-of-the-art processors to accommodate the high computation complexity of stereo matching algorithms. Early works [76][101] tried to implement real-time stereo matching on general purpose processors, however they could only achieve non-video rate real-time performance due to limited computing power at their times. As the processor technology advances, [102][103][104] implemented real-time stereo matching algorithms on general purpose processors. They managed to achieve real-time processing, but the performance of their disparity map was not very high because of using simple local algorithms. Although simple local algorithms have been adopted by most general purpose solutions, Forstmann et al. [105] proposed a real-time implementation of the less complex global algorithm, the dynamic programming, on general purpose processors. Their performance is higher than most of the previous local algorithms, but their real-time processing speed is limited to images smaller than VGA.

The DSPs have better processing speed on signal processing algorithms because of the SIMD and MIMD architectures than general purpose processors. In addition, they are often less expensive and less power consuming than the state-of-the-art general purpose processors. Hence, DSP solutions are more favorable in embedded stereo vision applications. Konolige's Small Vision System [76] is one of the most famous early real-time DSP solutions. Recently, [84] also proposed a real-time DSP implementation with jigsaw matching templates. Although the DSP solutions may have more computation power than general purpose processors, the data word alignment and bandwidth issue often limit their capability. As a result, the DSP solutions are usually limited to local algorithms and cannot provide high performance result in real-time.

Another powerful solution is the GPU solutions. The GPUs have extremely high memory bandwidth that ranges from 6.5 GB/sec to 128 GB/sec and can have up to 256 stream processors. With so much hardware resource, the GPU solutions [106]-[109] could implement high performance complex stereo matching algorithms. However, GPUs are too expensive and power consuming to be used in embedded applications currently.

The dedicated hardware solutions can also provide great computation power while allowing the computation resource to be optimized for utilization by designing the architecture in a customized way. This enables the dedicated hardware solutions to be more cost efficient than the GPU solutions. The dedicated hardware includes both FPGA/PLD and applications specific integrated circuit (ASIC). Faugeras et al.'s PeRLe-1 board [110] and Nishihara's PRISM-3 based stereo system [111] are two of the earliest dedicated hardware solutions. Later, other early dedicated hardware solutions [75][77][78][93] have also been proposed. Among these works, [77] and [78] are two of the first real-time implementation adopting the census matching. However, these early solutions only implemented simple local algorithms. Consequently, their performance is not high. Being aware of the performance limitation of local algorithms, hardware architectures have been proposed for dynamic programming [112] and hierarchical belief propagation (HBP) [113] algorithms [114]. Their performance is very high since they are based high performance global methods. However, their hardware cost is also very high compared to other dedicated hardware implementations. Recently, Tsai et al. [115] studied data reuse techniques in aggregation-based algorithms to reduce the internal storage size, computation resource, and bandwidth requirement.

Chapter 3   AXI Memory Controller

3.1. Memory Controller's Role in a System

With the rapid progress of VLSI technology, system-on-a-chip (SoC) [1] emerges and becomes feasible with on-chip bus compliant IPs. These SoCs possess sufficient computing power to implement complex and bandwidth demanding multimedia systems for various embedded applications. In such an embedded design, the bandwidth of memory subsystem is one of the major issues that have to be evaluated and optimized first to ensure the system's success.

We proposed a high bandwidth utilization memory controller which worked with a packet-based bus interface. In which, we have chosen AXI bus [3] as a representing case for packet-based bus that supported flexible out-of-order transaction completion. Packet-based bus not only eliminated the need to access data in request order, the additional transaction ID tag also provided valuable information about the source of an access request that can help in scheduling memory accesses. Thus, we proposed a memory controller with a two-level scheduling framework using such source information. The first level is the transactions-level scheduling which adopts a limited temporal source prioritized (LTSP) policy that used the temporal source correlation of accesses in a system. The second-level was the DRAM command-level scheduling that issues commands based on command age and type to hide access latency. The experimental result of a multimedia platform running a video phone application provided quantitative result of bandwidth usage, memory power consumption, and energy efficiency for different memory scheduling policies.

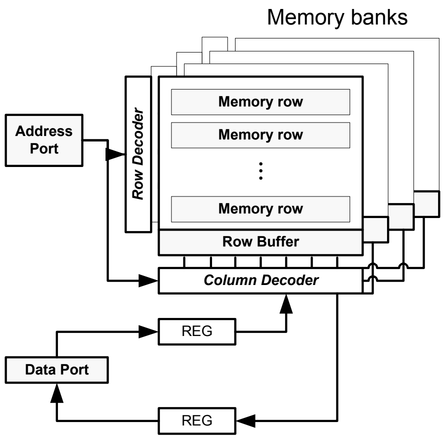


Fig. 1. Simplified block diagram of a DRAM

3.2. DRAM Basics

Fig. 1 illustrates a simplified block diagram of a DRAM. A DRAM usually consists of four memory banks. For each bank, it includes a row buffer, several memory rows. A DRAM usually has only one data port and is shared for both read and write. The address port is also shared for both read and write.

To access a data located in a DRAM, the row with the data must be first "opened" using an ACTIVATE command. The opened row is read from the memory and written to the row buffer. After the activation, the bank with the row being opened cannot be access for a period of time defined as the active to column access delay. However, other banks can still accept commands. Once the row is opened and the memory bank is ready to accept a command, a column-access READ/WRITE command is issued to access the data in the row buffer. If it is a READ column-access, the data would be available at the data port after a column access latency. Unlike the ACTIVATE command, there can be no other column-access command during the column-access latency because the data port is currently being used. After the current data have been accessed, the memory row can be "closed" by issuing a PRECHARGE command to this bank. Once the row is closed, it would take a period of time called the precharge command period before the next activation command can be issued.

Similar to ACTIVATE command, other banks can accept commands within the percharge command period.

Fig. 2 illustrates different DRAM bank state transitions. In Fig. 2(a), if an access happens on an idle bank in which no opened row is available, it is called a bank-miss access. In contrast, if an access happens on an opened row as shown in Fig. 2(b), it is called a row-hit access, which introduces the least access latency. This is because for a row-hit access, only a READ/WRITE column-access command is needed. However, if the access row in an active bank is closed as shown in Fig. 2(c), it is called a row-miss access. For a row-miss access, an additional PRECHARGE command must be issued before the ACTIVATE and READ/WRITE column-access command can be issued. Hence, a row-miss access results in

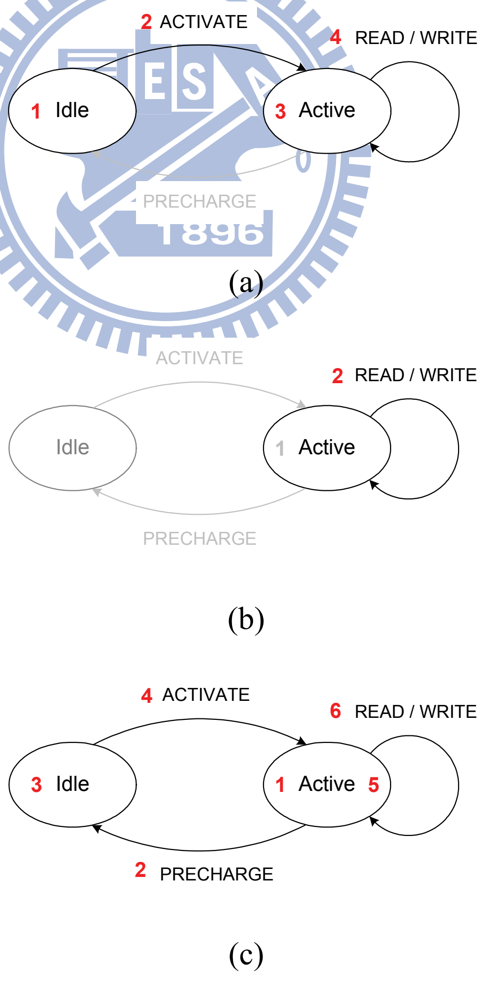


Fig. 2. Bank state transition and related commands when (a) current bank state is idle (b) current bank state is active and row-hit (c) current bank state is active and row-miss

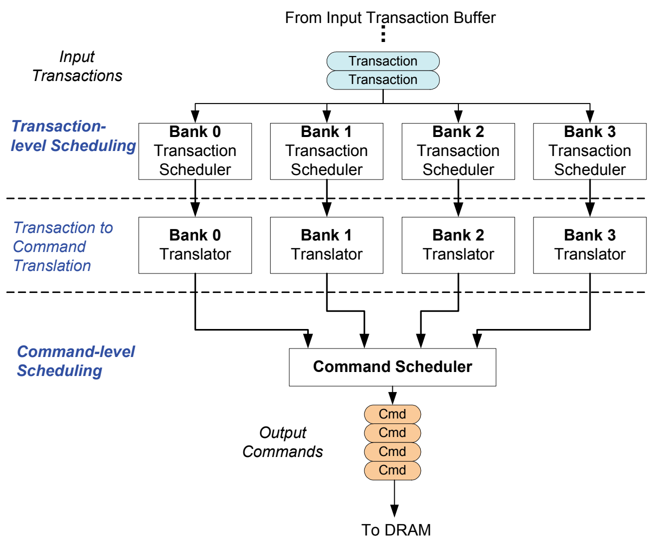

From Input Transaction Buffer
Input Transactions
Transaction
Transaction
Transaction-level Scheduling
Bank 0 Transaction Scheduler
Bank 1 Transaction Scheduler
Bank 2 Transaction Scheduler
Bank 3 Transaction Scheduler
Transaction to Command Translation
Bank 0 Translator
Bank 1 Translator
Bank 2 Translator
Bank 3 Translator
Command-level Scheduling
Command Scheduler
Output Commands
Cmd
Cmd
Cmd
Cmd
To DRAM

Fig. 3. Overview of the two-level scheduling flow

the longest access latency.

3.3. AXI Memory Controller Policy

3.3.1. Overall Scheduling Framework

Fig. 3 illustrates the scheduling flow of the proposed two-level scheduling framework. Input transactions are dispatched to each bank for transaction-level scheduling. The goal of transaction-level scheduling is to increase memory row-hit opportunity. After transaction-level scheduling, reordered transactions are translated into DRAM commands with the status of each memory bank taken into consideration. Once the DRAM commands are available, command-level scheduling issues command to DRAM based on command age and the type information. The details of the two-level scheduling are explained as follows.

3.3.2. First-level: Transaction-Level Scheduling

Two transaction scheduling policies are investigated: first-in first-serve (FIFS) and limited temporal source prioritized (LTSP). The first policy is similar to the first-ready scheduling policy mentioned in Rixner's work and has been widely adopted. The second one is proposed and recommended in this work. The details of each scheduling policy are described below.

A. Baseline First-In First-Serve

The FIFS policy issues transactions based on transaction input order. The advantage of FIFS policy is its fairness with respect to input transactions because every transaction would eventually be issued. However, FIFS policy is highly dependent on the bus arbitration policy, which determines the transaction input order. This characteristic would make the memory performance sensitive to bus arbitration policy.

B. Limited Temporal Source Prioritized

The proposed LTSP policy sets higher priority to transaction which has the same source as the last issued transaction. In other words, LTSP policy groups transactions from the same source device together. The transactions within a group are issued consecutively based on their relative temporal order. If no such transaction exists at that moment, LTSP policy gives higher temporal priority to earlier transactions and issues the transaction with highest temporal priority. The pseudo code of LTSP scheduling policy is listed below.

```
Initialize {
    prev_source_id = null;
    consecutive_cnt = 0;
}

LTSP Loop {
    if( transaction_buffer not empty ) {
        if( consecutive_cnt < limit_threshold ) {
            next_transaction = get_transaction_with_source_id( prev_sorce_id );
            if( next_transaction not null ) {
                consecutive_cnt ++;
            }else{
                next_transaction                                         =
get_transaction_with_highest_temporal_priority( );
                consecutive_cnt = 0;
            }
        }else{
            next_transaction                                             =
get_transaction_with_highest_temporal_priority( );
            consecutive_cnt = 0;
        }
        prev_master_id = next_transaction_source_id;
        issue next_transaction;
    }
}
```

We designed LTSP policy based on the observation that multimedia applications, such as video or audio processing, often involve massive amount of vector and block data access. These data are often correlated in their access location. Such correlation can be observed in conjunction with temporal and source locality. Therefore, the type (read or write) of transactions from the same master are likely the same, and the address of these transactions are also likely to be sequential in real multimedia application. Hence bundling transactions from the same source together provides more chances to achieve row-hits, which is relatively less energy consuming than row-misses. However, issuing multiple consecutive transactions from the same source increases the latency of transactions from other source. To avoid such starvation effect, we set a threshold to limit the maximum number of transactions in a group.

Although LTSP is suitable for multimedia applications, it needs additional request source information, such as the transaction ID tag provided in AXI, to identify a transaction's source. Unfortunately, such transaction source information is absent in traditional system bus such as AHB. Hence, scheduling transaction using transaction source information in multicore environment has not been possible with traditional bus under shared-link topology.

3.3.3. Second-Level: Command Scheduling

Command scheduling determines which bank can issue commands to DRAM based command age and type. Before selecting a command to be issued, DRAM status and timing constraint must be checked first. If the DRAM timing constraint inhibits any regular access command from being issued, a NOP command would be issued instead. If access commands are allowed to be issued, the oldest command is selected. However, if there are two commands with the same age, their command type are considered. We assign the command type priority as PRECHARGE>ACTIVATE>READ>WRITE. PRECHARGE is given the highest priority because PRECHARGE should be issued as soon as possible to avoid

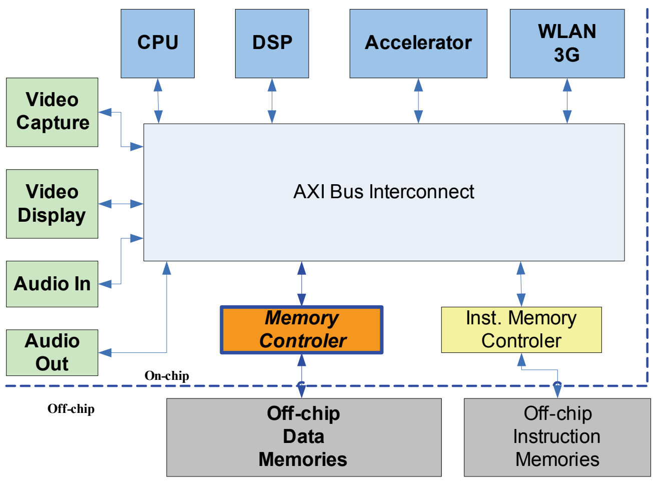


Fig. 4. The target platform

increasing the already very long access latency due to a row-miss. Similarly, ACTIVATE is given the second highest priority because of the same reason. READ is given higher priority than WRITE to minimize read-write turnarounds. Although there are other possible priority assignments, their bandwidth utilization is usually inferior to the assignment described here and the differences are within 3%.

3.4. Simulation Result

To evaluate the performance of a memory controller within a system, we model a simplified dual core platform system using SystemC [18] with different degrees of abstraction for different parts. For the AXI bus, each channel is modeled at transaction-level, whereas the memory controller is modeled at behavior-level. Note that the AXI bus and the memory controller models are all cycle accurate on the interface ports.

3.4.1. Multimedia Platform Architecture

Fig. 4 illustrates the target platform from the memory controller's point of view. The memory controller only connects with the AXI bus and the DRAM. Memory access requests are sent by the master devices connected to the AXI bus interconnect. The memory model is

Table 1 The task, access pattern, bandwidth, and completion time requirement of each master device

Master	**Task**	**Memory Access Pattern**	**Bandwidth Requiremen t**	**Timing Constrain t**
CPU	Audio codec OS	Read bitstream and PCM data Write bitstream and PCM data Random reads and writes for OS	16.14MB/se c	24 ms
DSP	Video decoding	Read bitstream Read reference macroblock (YUV) Write reconstructed macroblock (YUV) Write reconstructed macroblock (RGB) Random reads and writes	72.48 MB/sec	33 ms
Accelerat or	Video encoding	Read input macroblock (RGB) Read reference macroblock (YUV) Write reconstructed macroblock (YUV) Write reconstructed macroblock (RGB) Write bitstream	70.94 MB/sec	33 ms
Network	Tx/Rx bitstream	Read bitstream Write bitstream	2.30 MB/sec	33 ms
Audio In	Audio input	Write PCM data	8.46 MB/sec	24 ms
Audio Out	Audio output	Read PCM data	8.46 MB/sec	24 ms
Video In	Video input	Write captured video (RGB)	36.86 MB/sec	33 ms
Video Out	Video output	Read display video (RGB)	36.86 MB/sec	33 ms
Total	**Video phone**	N/A	**252.5 MB/sec**	

based on Micron's MT46V8M16 DDR SDRAM [21]. Note that only data memory is considered. Instruction memory and access are excluded because instruction memory can often achieve high bandwidth utilization due to predictable access behavior and pattern.

3.4.2. Videophone Application

The target application adopted in our simulation platform is a video phone application. In this video phone application, the system must deliver both audio and video communication at the same time. The system supports 44.1 Khz stereo audio capture/output and audio compression/decompression. As to video, the system provides 4CIF sized video capture, compression/ decompression, and display with a frame rate of 30 FPS. Table 1 lists the task description, minimal bandwidth requirement, and task completion time constraint of each master device in the video phone application. These system tasks are arranged in a pipelined fashion so that inter task dependency is minimized. The minimal memory bandwidth requirement for target performance is 252.5 MB/sec. If a memory controller can deliver more bandwidth, more data can be transferred within a second and better system speed can be achieved.

3.5. Simulation Result

In this section, we evaluate the performance of different memory controllers on our simulation platform. The memory scheduling policies for comparison are No-Scheduling Nor-Bank-Interleaving (NSNBI), FIFS with bank-interleaving (FIFS), and LTSP. NSNBI represents the simplest memory controller without bank-interleaving support. FIFS represents the most common memory controllers with bank-interleaving support.

We investigated the impact of burst length, bus interface buffer size, and bus arbitration policy on bandwidth usage, memory power consumption, and memory energy efficiency.

3.5.1. Evaluation Metrics

Table 2 lists the performance evaluation metrics and their physical meaning. The bandwidth usage (BU) evaluates how much data a memory controller can access within a second; it also implies shorter effective transaction latency from a system's point of view. The memory power consumption evaluates scheduling policy impact on memory's power consumption. However, scheduling policy that delivers higher bandwidth usage may also result in higher power consumption. Hence, we also use the memory energy efficiency to evaluate how much data a scheduling policy can deliver per unit energy. With memory energy consumption being the dominant part in a system, higher memory energy efficiency represents longer battery life and hence longer device operation time.

Table 2　Performance evaluation metrics

Evaluation Metric	Description	Unit
Bandwidth Usage	The ratio between the total amount of data transferred and the total time taken to transfer the data.	MB/sec
Memory Power Consumption	The power consumed by DRAM estimated with Micron's DRAM power calculator [16].	mW
Memory Energy Efficiency	The amount of data that can be accessed for a given amount of memory energy consumption.	KB/mJ

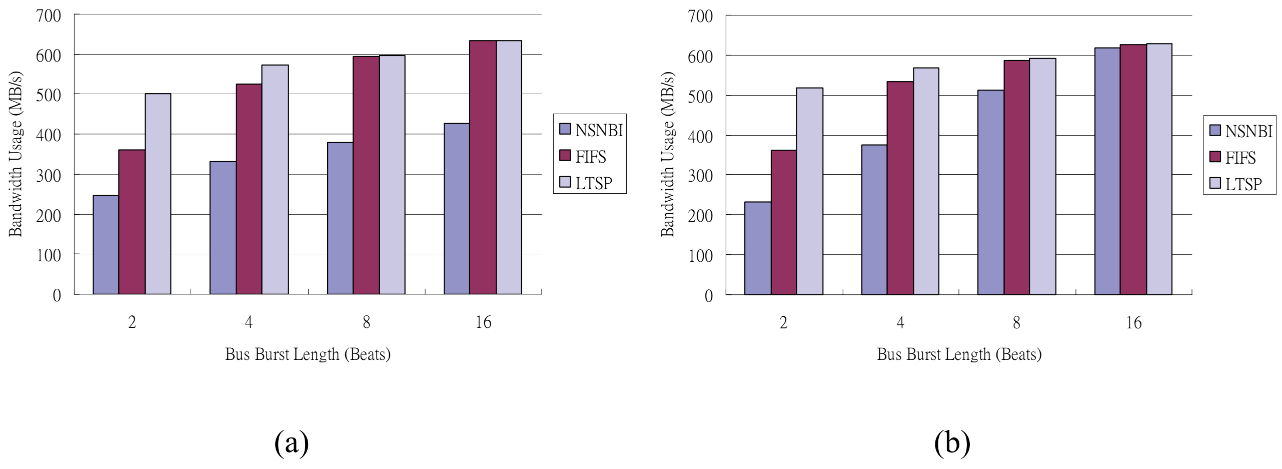


Fig. 5.　Bandwidth usage of different memory scheduling policies with (a)FP and (b)RR bus arbitration

3.5.2. Burst Length Impact

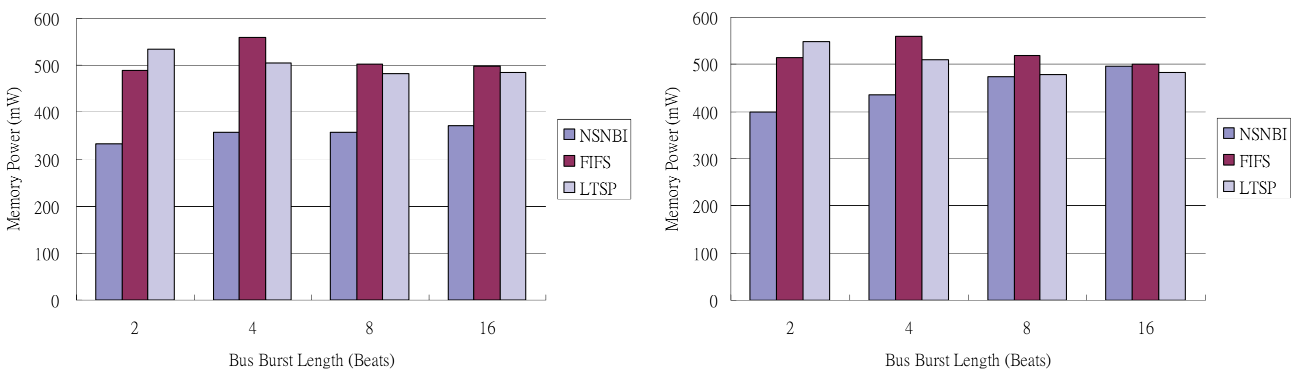
The effect of bus burst length 2, 4, 8, and 16 on DRAM performance is investigated. These bus burst length corresponds to memory burst length 4, 8, 16, and 32 on DDR memories because of the double data rate. The buffer size in both the masters and the memory controller are 8 entries.

A. Bandwidth Usage (BU)

Fig. 5 shows the bandwidth usage of using LTSP scheduling policies is the highest among the compared scheduling policies. The bandwidth usage improvements compared with FIFS can reach up to 19.7% for burst length 2. However, for burst length 8 and 16, the bandwidth usage of using LTSP scheduling policy is slightly higher than that of FIFS. This is because long burst length not only reduces the amount of row activation but also hides the long precharge and activation latency with the long data access time.

B. Memory Power Consumption

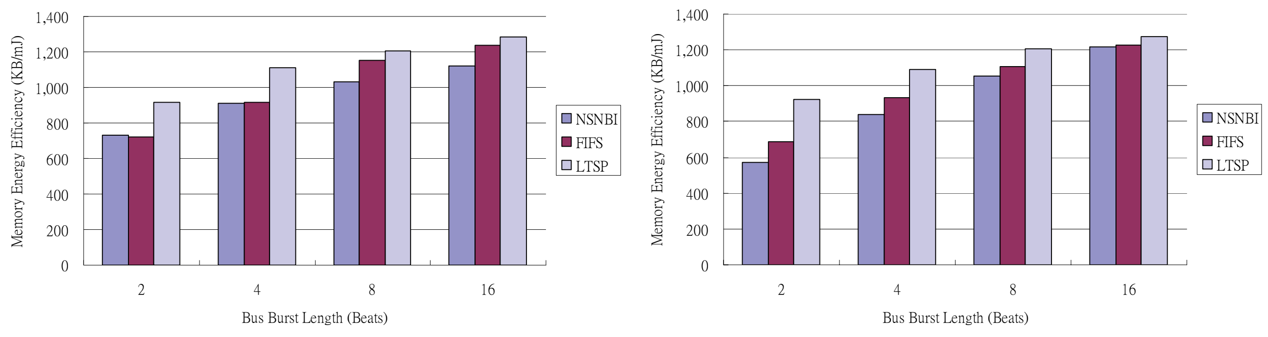
Fig. 6 shows that LTSP achieves the second lowest memory power consumption for burst length longer than 2. For burst length 2, however, LTSP has the highest memory power consumption. This is mainly due to the fact that the total time taken to transfer all the data when using LTSP policy is much shorter than others, hence decreasing the denominator in the power consumption formula. For burst length 4 and 8, at least 22 mW of power consumption can be reduced compared with burst length 2. However, the memory power consumptions of burst length 8 are almost the same as the power consumption of burst length 16. In contrast to LTSP's second lowest memory power consumption, FIFS has the highest power consumption for burst length longer than 2. NSNBI can achieve the lowest memory power consumption in most cases because it performs fewer accesses within unit time than other policies. Note that the power consumptions are around 500mW and are at least an order larger than the power consumption of a memory controller itself. Hence, the power consumption overhead of the memory controller is insignificant compared with that of the memory module, which



(a) (b)

Fig. 6. Power consumption of different memory scheduling policies with (a)FP and (b)RR bus arbitration

dominates the power consumption of a multimedia platform.



(a) (b)

Fig. 7. Memory energy efficiency of different memory scheduling policies with (a)FP and (b)RR bus arbitration

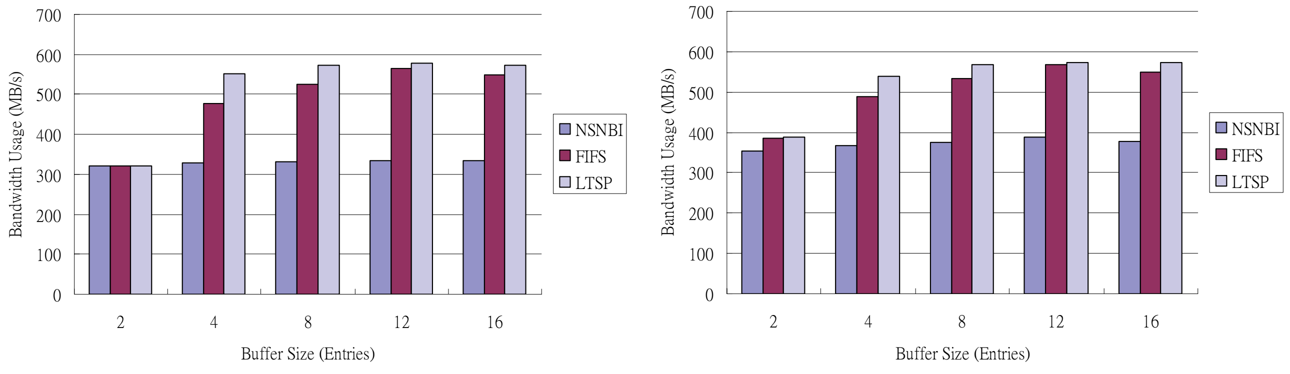
C. Memory Energy Efficiency

Fig. 7 shows that LTSP scheduling policy provides the highest energy efficiency in all cases. The efficiency improvements between LTSP and FIFS are 34.1%~4.0%. The improvement decreases as the burst length becomes longer due to the same reason why bandwidth usage saturates.

In summary, LTSP scheduling policy can provide both higher bandwidth utilization and memory energy efficiency, which is suitable for high performance and high energy efficiency applications. On the other hand, FIFS is an option to provide fair bandwidth usage when long burst length is available. However, FIFS has very high memory power consumption compared with other scheduling policies. Therefore FIFS is recommended for medium performance applications in which power consumption is less of an issue.

3.5.3. Transaction Buffer Size Impact

This sub section presents the impact of using different transaction buffer size. The buffer size determines the number of transactions that can be scheduled. We investigate the scheduling policies used in previous section with buffer size 2, 4, 8, 12, and 16 entries. Each entry stores a transaction. The bus burst length is set to 4 for all cases.



(a) (b)

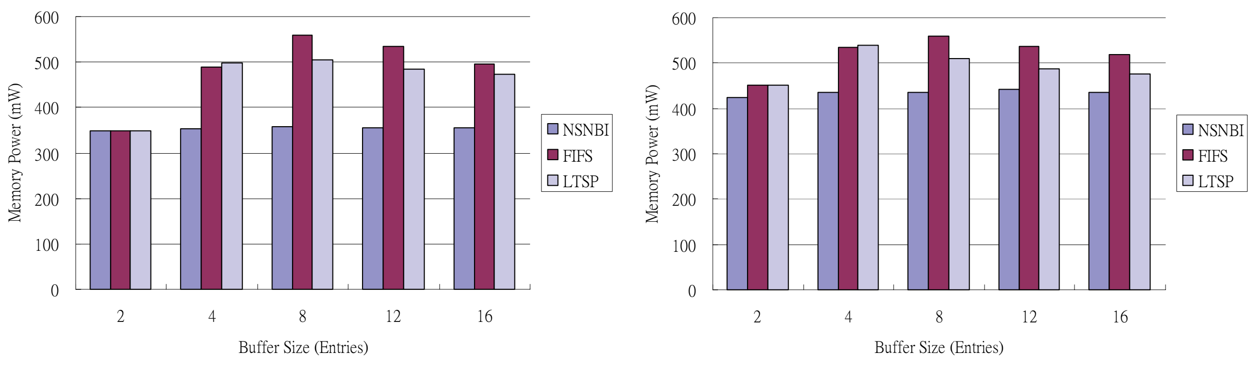
Fig. 8. Bandwidth usage of different buffer size with (a)FP and (b)RR bus arbitration

A. Bandwidth Usage (BU)

Fig. 8. illustrates the bandwidth utilization using different buffer size. For buffer size 2, the bandwidth utilizations of LTSP scheduling policy and FIFS are almost the same. For buffer size 4, the bandwidth utilization of LTSP is at least 6% higher than that of FIFS. In general, increasing the buffer size increases the bandwidth utilization because larger buffer size allows more transactions to be scheduled. Moreover, larger buffer size also reduces the possibility of the buffer being occupied by transactions accessing to only one particular bank. If all the transactions within the buffer try to access the same bank, only one transaction scheduler and command translator can be utilized. However, the improvement of bandwidth usage saturates as buffer size increases over 12. This is because all the transaction schedulers and command translators are already fully in use and hence scheduling capacity is reached. In addition, the utilization limit of the memory command bus also limits the maximum number of access commands that can be issued to DRAM.

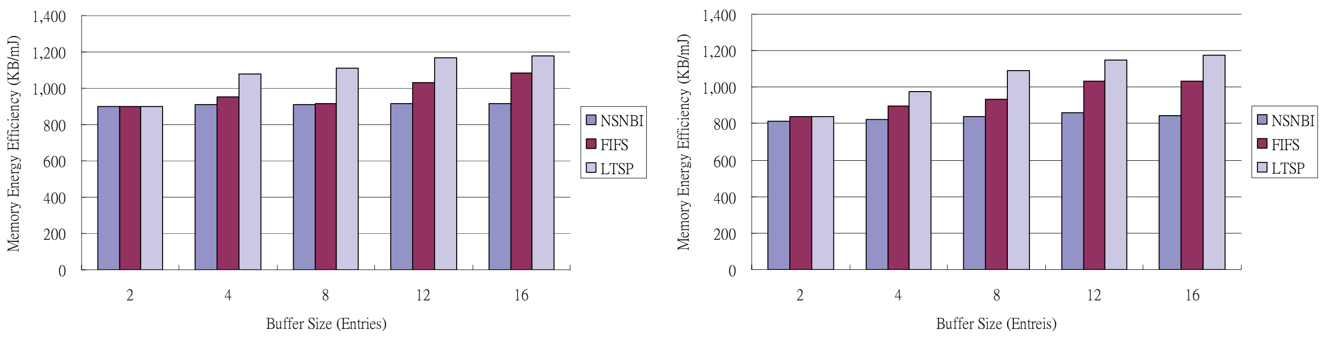
B. Memory Power Consumption

Fig. 9. illustrates the memory power consumption using different buffer size. For buffer size 2, the memory power consumptions of LTSP and FIFS are almost the same. For buffer size 4, the memory power consumption of LTSP is the highest. However, for buffer size larger than 4, the memory power consumption of FIFS becomes the highest. For buffer size



(a) (b)

Fig. 9. Memory power consumption of different buffer size with (a)FP and (b)RR bus arbitration



(a) (b)

Fig. 10. Memory energy efficiency of different buffer size with (a)FP and (b)RR bus arbitration

larger than 4, the power consumptions of LTSP and FIFS decrease gradually as the buffer size increases.

C. Memory Energy Efficiency

Fig. 10 reveals that the energy efficiency increases as buffer size increases for LTSP scheduling policy in general. Although there is still energy efficiency improvement between buffer size 12 and 16 using LTSP, the improvements is only 2.1% due to the saturation effect explained earlier. The energy efficiency improvement of FIFS has similar trend. However, the energy efficiency of FIFS is inferior to that of LTSP.

In summary, given that the system bus is adopting packet-based protocol, increasing the buffer size is an alternative way to increase the bandwidth usage and memory energy efficiency for memory scheduling policies supporting bank-interleaving. However, the buffer size should not exceed 12 because of the saturation effect. Although with larger buffer size, the bandwidth usage of FIFS is only slightly lower than the bandwidth usage of LTSP. LTSP scheduling policy is still the better choice when memory energy efficiency is also an issue. From the simulation result, buffer size of 8 or 12 are suggested as the best trade-off on performance and buffer size.

3.5.4. Bus Arbitration Policy Impact

This sub section discusses the impact of using fixed-priority (FP) or round-robin (RR) bus arbitration policies on performance.

A. Bandwidth Usage (BU)

The bus arbitration impact on bandwidth usage result in at most 1% difference for LTSP policy. FIFS is also less sensitive to bus arbitration policy when the buffer size is large enough; however, for buffer size 2, FIFS achieves higher bandwidth usage in RR than in FP. In contrast to LTSP being independent of bus arbitration policy, NSNBI has 25%~ 7% higher bandwidth usages in RR bus arbitration than in FP bus arbitration. This is because RR bus arbitration would result in higher access locality than FP bus arbitration would. If FP bus arbitration is used with longer burst, a row which was previously accessed by another device with higher bus arbitration priority would be re-opened. As a result, the access pattern would jump back and forth from one memory row to another from time to time. In contrast, RR bus arbitration has higher chances to group accesses from different devices together. Thus result in more row-hits and fewer row re-openings.

B. Memory Power Consumption

In both bus arbitration policies, the power consumptions of using LTSP are almost the same. This is due to the fact LTSP can reorder the transactions and is thus independent of which bus arbitration is being used. For NSNBI, the power consumption in FP bus arbitration

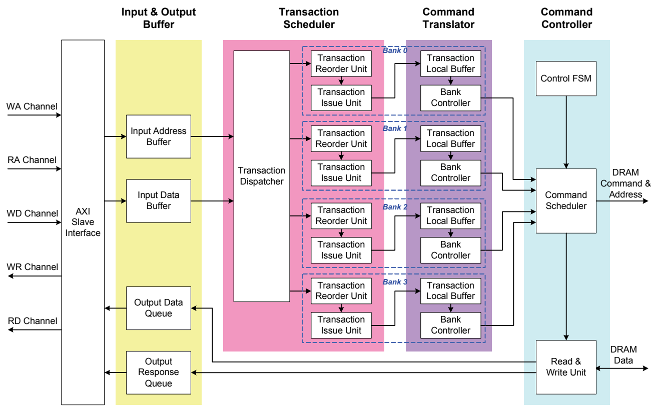


Fig. 11. Architecture of the AXI-compatible memory controller with the two-level scheduling policy

is at least 60 mW lower than that in RR bus arbitration. This is because NSNBI can achieve more accesses in RR bus arbitration than in FP bus arbitration as explained earlier.

C. Memory Energy Efficiency

The impact of bus arbitration policy is insignificant when using LTSP scheduling policy; the energy efficiency in FP bus arbitration is only 0.8%~1.7% higher than the energy efficiency in RR bus arbitration.

LTSP scheduling policy is relatively independent of which bus arbitration policy is being used. This allows system designer to exploit bus arbitration policy without having to take the memory's scheduling policy into account.

3.6. AXI Memory Controller Architecture

Fig. 11 illustrates the proposed memory controller. The AXI slave interface is the first stage of the memory controller; it handles the communication between the memory controller and the AXI system bus. The AXI bus has multiple channels that enable transferring read and write transactions at the same time. In addition, each channel has a transaction ID bus to enable out-of-order data transfer. The transaction ID is also used inside our memory controller to identify the source of a transaction.

Once a transaction is received from the bus, the transaction would be stored in the input buffer and queued before being issued to transaction scheduling. If a transaction has finished its access, this transaction is discarded from the buffer and queue. The input transactions from input buffer are reordered by transaction reorder units according to the policies mentioned earlier. Each transaction reorder and issue unit corresponds to a DRAM bank. After a transaction has been reordered and sent to a local transaction buffer, this transaction would then be translated into DRAM commands based on the status of the bank to be accessed. Each bank's status is tracked by one bank controller.

Finally, the translated input commands are scheduled by a command controller. The command controller determines which command can be issued to DRAM based on the aforementioned priority assignment and DRAM timing constraints. If a transaction's write data has been written or read data has been retrieved, the write response or read data are sent back to the output buffer and response queue.

The synthesized gate count of the proposed memory controller using UMC 0.18 μm technology is 47.6K. The memory controller is clocked at 166 MHz and has 32-bit data port.

3.7. Summary

We proposed a packet-based bus compatible memory controller with two-level scheduling scheme. By facilitating the flexibility and additional transaction source information available in packet-based bus protocol, the proposed memory controller can achieve relatively higher bandwidth usage and memory energy efficiency. In addition, the proposed LTSP transaction scheduling policy is independent of which bus arbitration is being used. The simulation result which includes the bus arbitration impact shows that the proposed two-level scheduling policy improves bandwidth usage and memory energy efficiency by up to 19.4% and 34.1% respectively.

Chapter 4 AXI Shared-link Bus

4.1. System Bus' Role in a System

With the rapid progress of system-on-a-chip (SOC) and massive data movement requirement, on-chip system bus has become the central role in determining the performance of a SOC. Two types of on-chip bus have been widely used in current designs, which are pipelined-based and packet-based bus.

For pipelined-based buses, such as ARM's AMBA 2.0 AHB [23], IBM's CoreConnect [24], and OpenCore's WishBone [25], the cost and complexity to bridge the communications among on-chip designs are low. However, pipeline-based bus suffers from bus contention and inherent blocking characteristics due to the protocol. The contention issue can be alleviated by adopting multilayer bus structure [26] or using proper arbitration policies [27][28]. However, the blocking characteristic, which allows a transfer to complete only if the previous transfer has completed, cannot be altered without changing the bus protocol. This blocking characteristic reduces the bus bandwidth utilization when accessing long latency devices, such as an external memory controller.

To cope with the issues of pipelined-based buses, packet-based buses such as ARM AMBA 3.0 AXI [29], OCP-IP's Open Core Protocol (OCP) [30], and STMicroelectronics STBus [31] have been proposed to support outstanding transfer and out-of-order transfer completion. We will focus on AXI here because of its popularity. AXI bus possesses multiple independent channels to support multiple simultaneous address and data streams. Besides, AXI also supports improved burst operation, register slicing with registered input, and secured transfer.

Despite the above features, AXI requires high cost and possesses long transaction handshaking latency. However, a shared-link AXI interconnect can provide good performance while requiring less than half of the hardware required by a crossbar AXI implementation. This work focused on the performance analysis of a shared-link AXI. The handshaking latency is at least two cycles if the interface or interconnect are designed with registered input. This would limit the bandwidth utilization to less than 50%. To reduce the handshaking latency, we proposed a hybrid data locked transfer mode. Unlike the lock transfer in [32] which requires arbitration lock over transactions, our data locked mode is based on a transfer-level arbitration scheme and allows bus ownership to change between transactions. This gives more flexibility to arbitration policy selection.

With the additional features of AXI, new factors that affect the bus performance are also introduced. The first factor is the arbitration combination. The multi-channel architecture allows different and independent arbitration policies to be adopted by each channel. However, Existing AXI related works often assumed a unified arbitration policy where each channel adopts the same arbitration policy [32][33][34]. Another key factor is the interface buffer size. A larger interface buffer usually implies more out-of-order transactions can be handled. The third factor is the task access setting, which defines how the transfer modes should be used by the devices within a system. Proper task access settings can yield better performance. However, the proper setting may be different under different circumstances, such as different buffer sizes.

Being aware of the performance factors mentioned above, we conducted a detailed simulation-based analysis on the performance impact of the factors. The analysis is carried out by simulating a multi-core platform with a shared-link AXI backbone running a video phone application. The performance is evaluated in terms of bandwidth utilization, average transaction latency, and system task completion time. In addition to the analysis on the performance impact of the aforementioned factors, the performance of a corresponding 5-layer AHB-lite bus, which has a cost comparable to a 5-channel shared-link AXI, is also included for comparison.

The rest of the chapter is organized as follows. Section 4.2 presents the proposed transfer modes and the corresponding arbitration framework. Section 4.3 presents the simulation platform and evaluation metrics for performance comparison. The comparison of the simulation result is available in Section 4.4. Finally, Section 4.5 concludes this work.

4.2. Proposed AXI Scheme

4.2.1. Transfer Modes

A. Normal

This mode is the basic transfer mode in an AXI bus with registered interface. In the first cycle of a transfer using normal mode, initiator sets the valid signal high and sends it to the target. In the second cycle, the target receives the high valid signal and sets the ready signal high for one cycle in response. Once the initiator receives the high ready signal, the initiator resets the valid signal low and this transfer is completed. As a result, at least two cycles are needed to complete a transfer in an AXI bus with registered interface. Fig. 12 illustrates the transfer of two normal transactions with a data burst length of four. It takes 16 bus cycles to complete the eight data transfer in the two transactions. This means 50% of the bus available bandwidth is wasted.

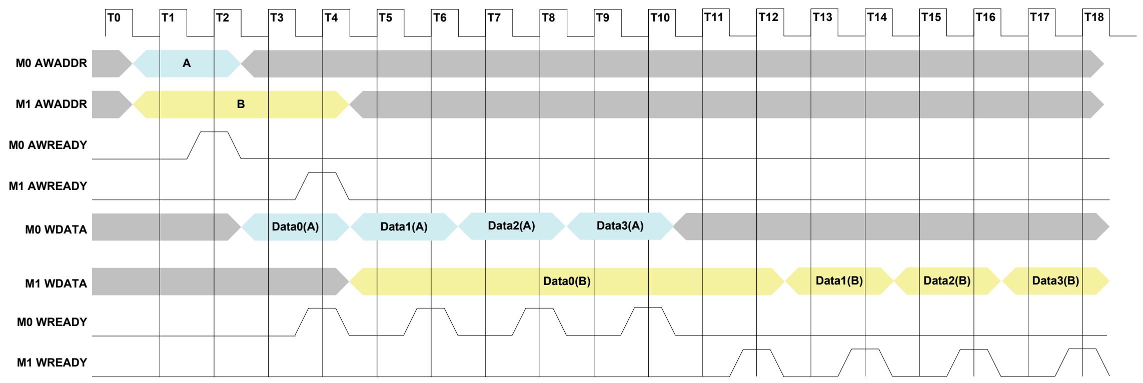


Fig. 12. Normal mode transfer example

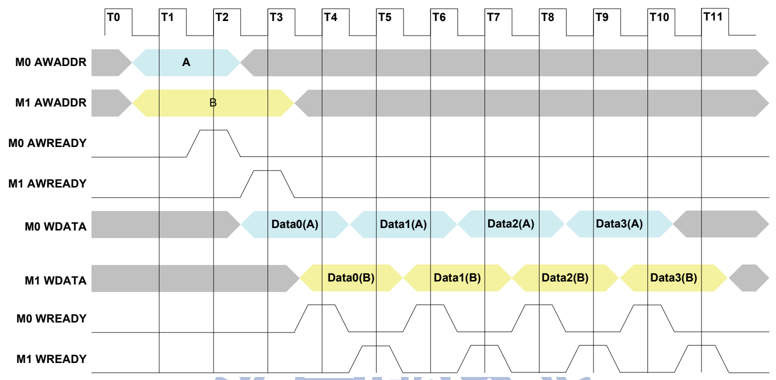


Fig. 13. Interleaved mode transfer example

B. Interleaved Mode

The interleaved mode [32][42] hides transfer latency by allowing two transactions from different initiators to be transferred in an interleaved manner. Fig. 13 illustrates the transfer of the two transactions mentioned earlier using interleaved transfer mode. The one cycle latency introduced in the normal mode for request B is hidden by the transfer of request A. Similarly, the interleaved transfer mode can also be applied to data channels. As a result, transferring the data of the two transactions only takes 9 cycles.

To support the interleaved mode, only the bus interconnect needs additional hardware. No additional hardware in device interface or modification on bus protocol is required. Hence, an AXI interconnect that supports the interleaved mode can be used with standard AXI device.

C. Proposed Data Locked Mode

Although the interleaved mode can increase bandwidth utilization when more than one initiator is using the bus, the interleaved mode can not be enabled when only one standalone initiator is using the bus. To handle this, we proposed the data locked mode. In contrast to the locked transfer implemented in [33] that can only perform when the bus ownership is locked across consecutive transactions, the proposed data locked mode locks the ownership of the bus only within the period of burst data transfers. During the burst data transfer period, the ready signal is tied high and hence the handshaking process is bypassed. Unlike the interleaved mode, which can be applied to both request and data channels, the proposed data locked mode only supports burst data transfer.

Fig. 14 illustrates an example of two transactions using data locked mode to transfer data. Device M0 sends a data locked request A and device M1 sends a data locked request B. Once the bus interconnect accepted request A, the bus interconnect records the transaction ID of request A. When a data transfer with the matched ID appears in the data channel, the bus interconnect uses data locked mode to transfer the data continuously. For a transaction with a data burst of n, the data transfer latency is n+1 cycles.

There are two approaches to signal the bus interconnect to use the data locked mode for a transaction. One is using ARLOCK/AWLOCK signal in the address channels to signal the bus of an incoming transaction using data locked transfer. However, doing so requires modifying the protocol definition of these signals and the bus interface. To avoid modifying

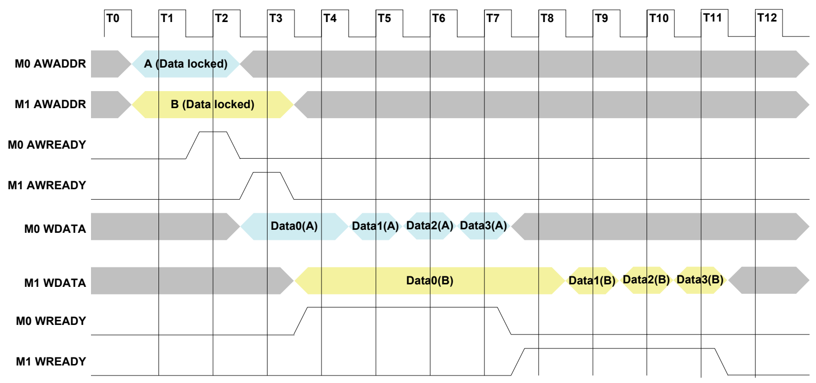


Fig. 14. Data locked mode transfer example

the protocol, the other approach is to assign the devices that can use the data locked mode in advance. The overhead of this approach is that the bus interconnect must provide mechanisms to configure the device transfer mode mapping. Note that these two approaches can be used together without conflict.

To support the proposed data locked mode, the bus interconnect needs an additional buffer, called data locked mode buffer, to keep record of the transactions using the data locked mode. Each entry in the buffer stores one transaction ID. If all the entries in the data locked mode buffer are in use, no more transaction can be transferred using the data locked mode.

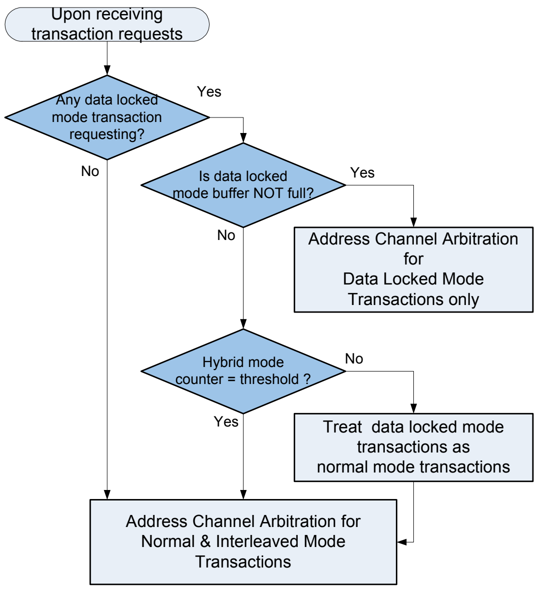
D. Proposed Hybrid Data Locked Mode

The hybrid data locked mode is proposed to allow additional data locked mode transaction requests to be transferred using the normal or interleaved mode when the data locked mode buffer is full. This allows more transactions to be available to the scheduler of the devices that support transaction scheduling. With the additional transactions, the scheduler of such devices may achieve better scheduling result.

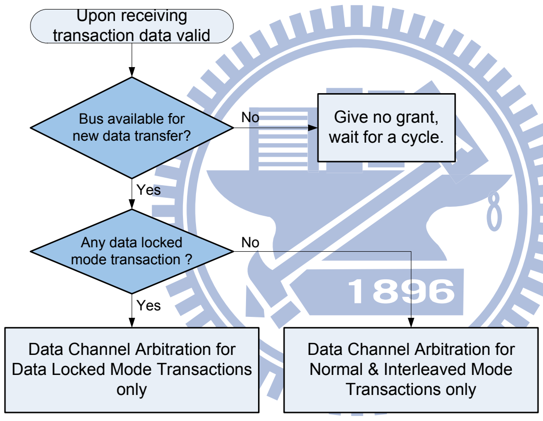
However, only a limited number of additional transactions using the data locked mode can be transferred using the normal or interleaved mode. This avoids bandwidth-hungry devices from occupying the bus with too many transactions. A hybrid mode counter is included to count the number of additional transactions transferred. If the counter value reaches the preset threshold, no more data locked mode transactions can be transferred using the normal or interleaved mode until the data locked mode buffer becomes not full again. Once the data locked mode buffer is not full, the hybrid mode counter is reset.

4.2.2. Arbitration Scheme

With the introduction of the multi-channel architecture and the proposed transfer modes, traditional arbitration framework that was based on single-channel architecture must be


Upon receiving transaction requests
Any data locked mode transaction requesting?
Yes
No
Is data locked mode buffer NOT full?
Yes
No
Address Channel Arbitration for Data Locked Mode Transactions only
Hybrid mode counter = threshold ?
No
Yes
Treat data locked mode transactions as normal mode transactions
Address Channel Arbitration for Normal & Interleaved Mode Transactions

(a) address channel arbitration


Upon receiving transaction data valid
Bus available for new data transfer?
No
Give no grant, wait for a cycle.
Yes
Any data locked mode transaction ?
No
Yes
Data Channel Arbitration for Data Locked Mode Transactions only
Data Channel Arbitration for Normal & Interleaved Mode Transactions only

(b) data channel arbitration

Fig. 15. Arbitration framework for a share-link AXI bus

revamped. Therefore, we propose an arbitration framework that supports different arbitration flows for address channels and data channels. In contrast to existing works which used unified arbitration, each independent channel in the proposed arbitration framework is allowed to have its own arbitration policy. This framework allows different arbitration policies to be combined together in a simple plug-and-play manner.

Fig. 15 (a) illustrates the arbitration flow for address channels. Upon receiving multiple data locked mode transaction requests from different initiators, the arbitration flow first checks if the data locked mode buffer is full or not. If there is an available empty entry in the data locked mode buffer, the data locked mode transaction requests are arbitrated according to the arbitration policy adopted for the address channel. If the data locked mode buffer is full and the hybrid mode counter has not reached the threshold, all data locked mode transaction requests are treated as normal and interleaved mode transaction requests. As a result, all transaction requests are arbitrated together according to the adopted arbitration policy. On the other hand, if the hybrid mode counter has already reached the threshold, only the original normal and interleaved mode transaction requests are arbitrated. This arbitration flow gives higher priority to data locked mode transactions than normal or interleaved mode transactions.

Fig. 15 (b) illustrates the arbitration flow for data channels. The arbitration flow first checks if there is already a transaction transferring data using the data locked mode. If there is already a transaction transferring data using the data locked mode, no other transaction would be granted. If no transaction is transferring, data locked mode transactions would be arbitrated according to the arbitration policy adopted by the data channel. If there is no data locked mode transaction requesting to transfer data, normal and interleaved mode transactions are arbitrated.

The reason for giving higher priority to the transactions using the data locked mode is that these transactions are often latency sensitive. To minimize the latency of these transactions, the transactions must be transferred using the data locked mode and given the highest arbitration priority.

4.3. Simulation Setup

To properly evaluate the performance of the proposed transfer modes and arbitration framework on a shared-link AXI bus, we built a high-level model of a simplified multi-core platform system using SystemC [45]. The simulation accuracy of this model depends on modeling methodology, platform architecture authenticity, and application traffics accuracy.

The bus and components in the platform were modeled using transaction-level and behavior-level modeling method respectively. Transaction-level modeling uses a transaction instead of a cycle as the basic simulation unit. Since a transaction takes a fixed number of cycles to complete in each channel, transaction-level modeling ensures bus cycle accuracy in our simulations. More detail on transaction-level modeling can be found in [37]. To pursue platform architecture authenticity, the multi-core platform model was built based on a real multi-core platform [46]. The real platform has been verified with portable media player and smartphone applications. This ensures the simulation result from our platform model to be practical. The application traffics were derived based on the behavior and algorithm of the platform components to ensure traffics accuracy. The details of the platform architecture and bus traffics are provided in the following subsections.

4.3.1. Multimedia Platform Architecture

Fig. 16 illustrates the target platform from the system bus point of view. Note that when the platform is used for AHB simulation, the bus interconnect is replaced with a 5-layer AHB-lite interconnect with each master port having one dedicated AHB-lite bus. Since we only focus on the transaction behavior on the bus, the devices are modeled to only exhibit transaction behavior and pattern. However, the CPU does generate transactions related to interrupt service routines (ISR) upon receiving an interrupt request (IRQ). In addition, the DMA controller is also programmed to carry out different data moving tasks to mimic the behavior of its real counterpart. Including such more detailed behavior enables us to include the inter task dependency between devices. Note that the memory controller has two slave ports to allow more transactions to be seen by the scheduler of the memory controller. Among all the devices, the memory controller is the only one with access latency ranging from 0 to 16 cycles.

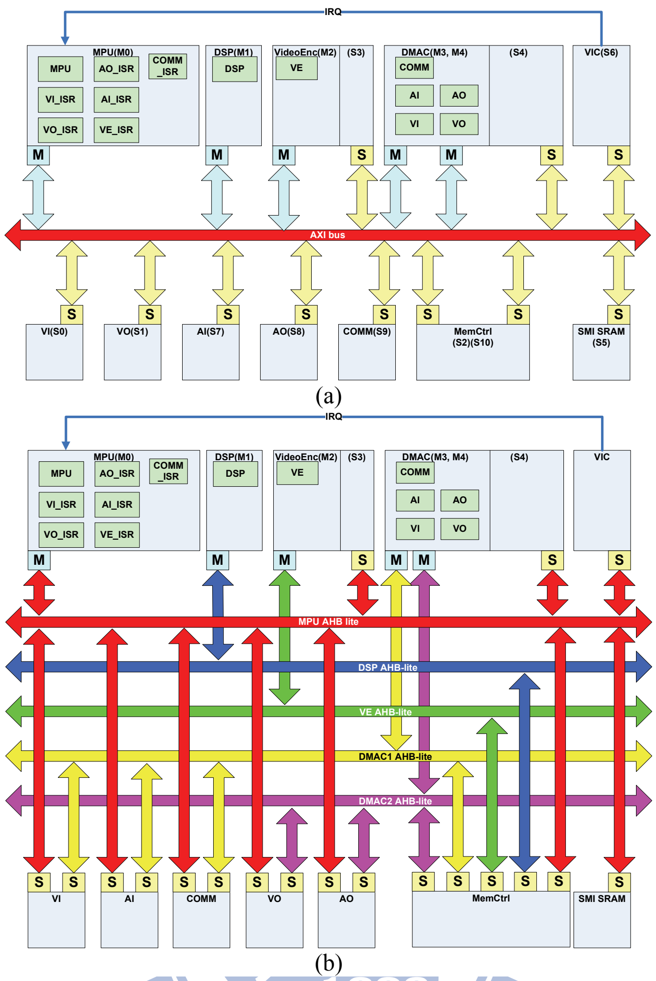


Fig. 16. Block diagram of the target platform using (a)AXI, (b) AHB-lite

The AXI bus is clocked at 40MHz with both the address and data widths being 32-bit wide. This would yield an ideal total bandwidth of 320 MB/sec with the read and write bandwidths being 160 MB/sec each.

4.3.2. Video Phone Scenario

We have selected the video phone application for analysis because it covers a variety of devices and traffics that are common in most multimedia consumer electronic products. The bandwidth requirement of the video phone application is heavier than other applications such as portable media player, video recording, MP3 player, and regular phone service. This heavy bandwidth requirement also makes the video phone application a perfect application to test the performance limit of a bus.

Table 3 Port task description and bandwidth requirement

Master Port	Task	Required Read BW (MB/sec)	Required Write BW (MB/sec)	Total Required BW (MB/sec)
MPU	Audio codec	1.467	1.467	2.934
	OS routine	0.001	0.001	0.002
	Total ISR	0.172	0.493	2.935
	Total Bandwidth Requirement	1.640	1.961	3.601
DSP	Video decode	14.836	42.473	57.309
Video Encoder	Video encode	59.927	14.255	74.182
DMAC0	Video in to MEM	27.927	27.927	55.855
	Audio in to MEM	0.176	0.176	0.353
	3G communication	0.132	0.132	0.265
	Total Bandwidth Requirement	28.236	28.236	56.472
DMAC1	MEM to video out	27.927	27.927	55.855
	MEM to audio out	0.176	0.176	0.353
	Total Bandwidth Requirement	28.104	28.104	56.208
System Total Bandwidth Requirement		**132.743**	**115.028**	**247.771**

In the video phone application, the system must deliver both audio and video communication at the same time. The system supports 44.1 KHz stereo audio capture/output and audio compression/decompression. As to video, the system provides VGA sized video capture, compression, decompression, and display with a target frame rate of 30 FPS. Table 3 lists the task description, bandwidth requirement, and task completion time constraint of each master device in the video phone application. Although more devices may be included in a system, the bus traffic is usually dominated by the master devices listed in Table 3. The total bus bandwidth requirement is 247.8 MB/sec, which occupies 77.5% of the 320 MB/sec available bus bandwidth. If the bus can achieve a bandwidth utilization higher than 77.5%, all the system tasks are more likely to complete within the specified timing constraints.

4.3.3. Evaluation Metrics

The definition and physical meaning of the evaluation metrics are explained as follows.

A. Bandwidth Utilization (BWU)

The *bandwidth utilization (BWU)* is defined as the percentage of available ideal bus bandwidth being used to actually transfer data, i.e.

$$BWU = \frac{B_{used}}{B_{ideal}} \times 100\%, \tag{1}$$

where B_{used} and B_{ideal} are the actually used bandwidth and available ideal bandwidth respectively. A higher BWU implies more data can be transferred within a period of time. It also implies shorter effective transaction latency from the system's point of view.

B. Transaction Latency

The transaction latency we used is defined as the average of read and write transaction latencies. The latency of a read or write transaction is measured from the time a transaction request is sent from a master till the time the read data or write response is returned to the master. The average transaction latency, denoted as *TL*, can be defined as

$$TL = \frac{\sum TL_{read} + \sum TL_{write}}{N_{read} + N_{write}}, \tag{2}$$

where $\sum TL_{read}$ and $\sum TL_{write}$ are the sums of all read and write transaction latencies respectively. N_{read} and N_{write} are the total number of read and write transactions respectively. In contrast to the bandwidth, which increases as more data can be transferred, the transaction latency may remain the same even if the bandwidth utilization has been increased.

C. System Task Completion Time

The system task completion time is defined as the time when all tasks in the video phone application have been completed. We believe it is crucial to minimize the system task completion time so that the task-level timing constraint can be met. In the video phone application, all tasks must be done within 33 ms, otherwise we say the system violates the real-time constraint.

Table 4 Combinations of arbitration policies

Arbitration policy of channels			
Combination Name	Address channel	Data channel	Write response channel
FF	Fixed priority	Fixed priority	Round-Robin
FT	Fixed priority	TDMA	Round-Robin
FR	Fixed priority	Round-Robin	Round-Robin
FL	Fixed priority	Lottery	Round-Robin
TF	TDMA	Fixed priority	Round-Robin
TT	TDMA	TDMA	Round-Robin
TR	TDMA	Round-Robin	Round-Robin
TL	TDMA	Lottery	Round-Robin
RF	Round-Robin	Fixed priority	Round-Robin
RT	Round-Robin	TDMA	Round-Robin
RR	Round-Robin	Round-Robin	Round-Robin
RL	Round-Robin	Lottery	Round-Robin
LF	Lottery	Fixed priority	Round-Robin
LT	Lottery	TDMA	Round-Robin
LR	Lottery	Round-Robin	Round-Robin
LL	Lottery	Lottery	Round-Robin

Table 5 Weight allocation of TDMA and Lottery arbitration schemes

Channel	**Slots/Tickets of each initiator port**				
Read Address	**CPU**	**DSP**	**Video Enc.**	**DMAC0**	**DMAC1**
	4	8	24	24	24
Write Address	**CPU**	**DSP**	**Video Enc.**	**DMAC0**	**DMAC1**
	4	24	8	24	24
Read Data	**Video In**	**Mem. Ctrl. 0**	**Mem. Ctrl. 1**	**Others**	
	8	24	16	4	
Write Data	**CPU**	**DSP**	**Video Enc.**	**DMAC0**	**DMAC1**
	4	24	8	24	24

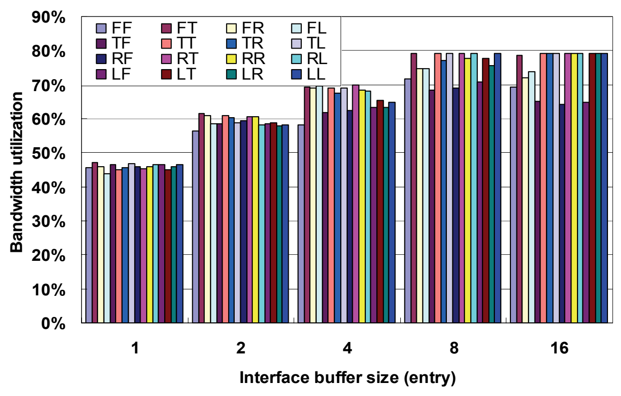
4.4. Experiment Result

4.4.1. AXI Interface Buffer Size and Bus Arbitration Impact

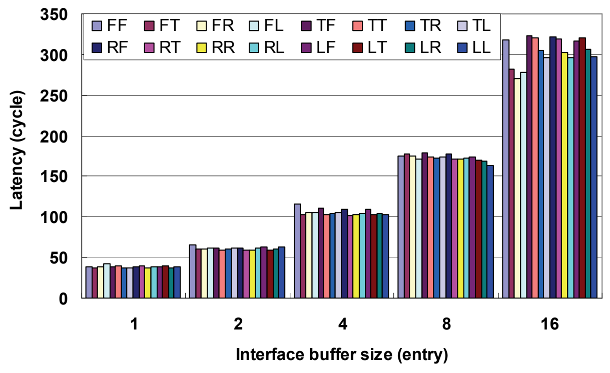
The effect of the bus interface buffer size and the combination of arbitration policies are investigated first. The investigated buffer sizes are 1, 2, 4, 8, and 16. Each entry keeps the record of a transaction. Table 4 lists the abbreviations of the investigated arbitration policy combinations. The weighting parameter, which is slots in the TDMA and tickets in the

Lottery, are tuned to match the bandwidth requirement of the video phone application. Table 5 lists the detail weight parameter of each channel. Since write response channel does not require high bandwidth, round-robin arbitration is selected for write response channel due to its fairness. Note that in this experiment, only the normal and the interleaved modes are used.

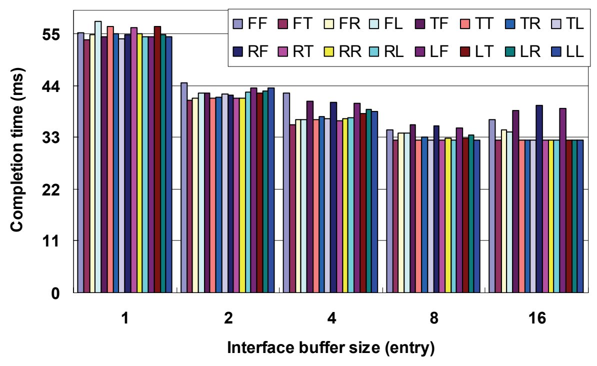
Fig. 17 shows the bandwidth utilization, average transaction latency, and completion time respectively. In general, the bandwidth utilization increased as the interface buffer size increased. However, the bandwidth utilization stopped increasing when the buffer size is



(a) Bandwidth utilization



(b) Average transaction latency



(c) System task completion time

Fig. 17. Performance of different interface buffer size and arbitration policy combinations

greater than 8 because of the required bandwidth limit. The average transaction latency is also proportional to the interface buffer size. This is because with a larger buffer, a transaction would spend more time pending in the interface buffer before this transaction finishes transferring data. In contrast, the completion time decreases as the buffer size increases.

It is interesting that the transaction latency did not reflect the result in bandwidth utilization. This is because the outstanding and out-of-order transfer capabilities, which are related to the buffer size, allow multiple transactions to be transferred on the bus in an overlapped manner. As a result, the latency of a transaction becomes longer, but the transfers on the bus can be arranged in a more compact way. This is one of the characteristics in a packet-based bus that is different from a traditional pipeline-based bus.

For the arbitration policy, the impact on the performance was not significant when the buffer size is 1 and 2. After the buffer size becomes larger than 2, the combinations with data channels using fixed-priority, such as FF, TF, RF, and LF, usually achieved lower bandwidth utilization than other combinations. On the other hand, the combinations with data channels using TDMA, such as FT, TT, RT, and LT, usually achieved the highest bandwidth utilization. Similar trend is also observed in the execution time. However, this trend is weak in the transaction latency, which doest not reflect the result of the bandwidth utilization.

The completion time comparison shows that when TDMA is used for data channel, the 33 ms timing constraint can be satisfied in buffer size 8 and 16. On the other hand, no arbitration policy combination satisfied the 33 ms timing constraint with the buffer size smaller than 8

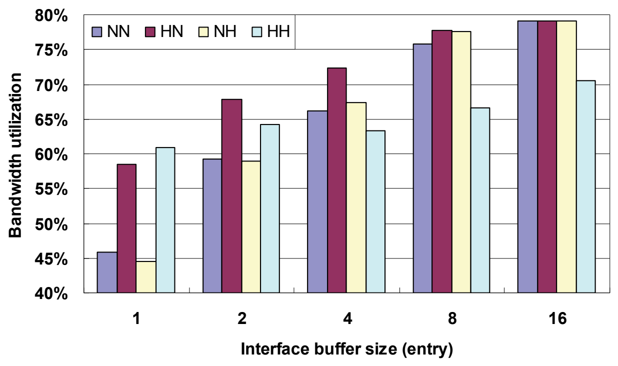
In summary, bandwidth and completion time improved sub-linearly as the interface buffer size increased. However, the buffer size increase also increased transaction latency near linearly. When smaller interface buffer is used, the arbitration combination had less impact on performance. On the other hand, for larger buffer size, the best arbitration combination could yield up to 23.3 % performance gain over the worst arbitration combination. The result suggests that using a fair arbitration policy on data channels should be more promising. For address channels, a simpler arbitration policy is good enough because address channel arbitration had less impact on bus performance.

4.4.2. Task Access Setting Impact

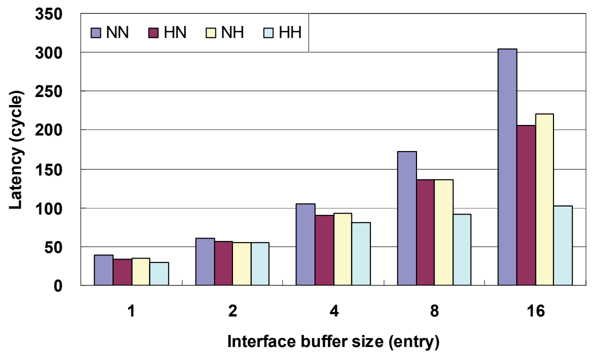
This subsection finds out how the hybrid data locked mode should be used and shows the performance impact delivered by using the hybrid data locked mode. A task access setting defines how the transfer modes should be used by the devices in a system. Table 6 lists the four settings investigated here. In which, the memory device is singled out because it is the only device with non-zero access latency. All other devices have zero access latency and hence are treated the same. Note that the results here are the average over all 16 arbitration policy combinations. The buffer size of data locked mode and hybrid counter threshold are both set to one.

Table 6 Task access settings

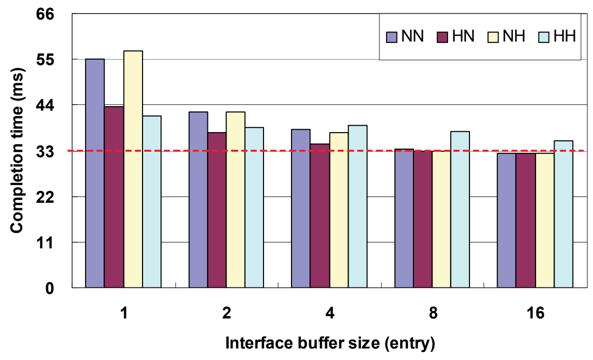
Settings	Memory Access Tasks	Other Tasks
NN	Normal & Interleaved	Normal & Interleaved
HN	Hybrid	Normal & Interleaved
NH	Normal & Interleaved	Hybrid
HH	Hybrid	Hybrid



(a) Bandwidth utilization



(b) Average transaction latency



(c) System task completion time

Fig. 18. Performance of different task access settings and interface buffer size

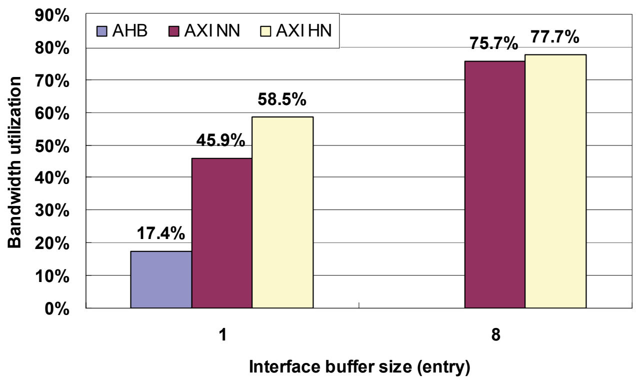
Fig. 18 shows the average bandwidth utilizations, average transaction latency, and completion time of different task access settings. In general, HN setting achieved the highest bandwidth utilization among all the task access settings except in buffer size 16. This is because when the buffer size is 16, the bandwidth utilization of NN, HN, and NH settings is already high enough to handle all the data transfer. If the buffer size is small, the use of the hybrid data locked mode could reduce the completion time by up to 26.8% compared with NN. Although HN achieved the highest performance in most cases, HH achieved the highest bandwidth utilization in buffer size 1 because no out-of-order transfer can be carried out with the interface buffer having only 1 entry. Consequently, HH took shorter time to transfer a transaction and less bandwidth would be wasted. In contrast to HH's highest bandwidth utilization in buffer size 1, HH setting achieved the lowest bandwidth utilization when the buffer size is larger than 2. This is because HH had less opportunity to enable interleaved transfer mode on normal transactions when the interface buffer size increases. Unlike the result in the bandwidth utilization, HH setting achieved the shortest average transaction latency among the four settings. The transaction latency of HH did not increase significantly as the buffer size increased. On the other hand, NN had the expected longest average transaction latency. The trend in completion time matches the trend in bandwidth utilization in general. HN setting achieved the shortest completion time and met the 33ms timing constraint with buffer size larger than 4.

In summary, HN was the best task access setting in most cases in terms of bandwidth utilization and completion time. This suggests the hybrid data locked mode would best to be used by long access latency devices, but not by zero access latency devices. From the transaction latency perspective, HH achieved the shortest transaction latency. This suggests that processors or devices that require short latency should use the hybrid mode.

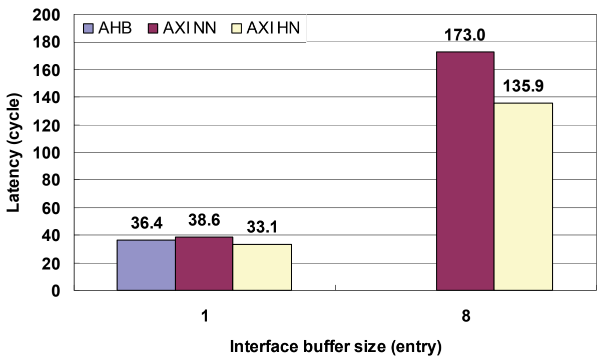
4.4.3. Single-Layer Shared-link AXI vs. 5-Layer AHB-lite

This subsection compares the performance between a share-link 5-channel AXI interconnect and a cost equivalent 5-layer AHB-lite interconnect. The 5-layer AHB-lite interconnect is capable of providing a maximum bandwidth of 800 MB/sec. We used two task access settings for the AXI case, one is NN setting and the other is HN setting. The interface buffer sizes we investigated are 1 and 8. Since the interface buffer size has no effect in AHB, only the result of buffer size 1 is available for AHB.

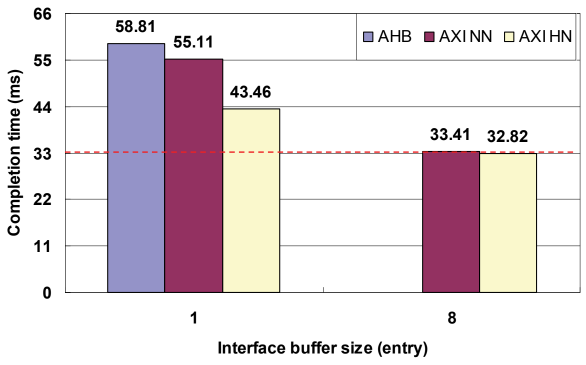
Fig. 19 compares the bandwidth utilization, average transaction latency, and completion time of the AHB and AXI platforms. The bandwidth utilization of shared-link AXI is significantly higher than that of AHB. If the interface buffer size is 8, the bandwidth utilization of AXI outperformed AHB by at least 58.3%. However, AXI's transaction latency can reach up to 4.7 times of AHB's in buffer size 8. The completion time comparison shows that despite the long latency in AXI, the completion time in AXI reduced up to 44.2% when



(a) Bandwidth utilization



(b) Average transaction latency



(c) System task completion time

Fig. 19. Performance of 5-layer AHB-lite and single-layer shared-link AXI

compared with AHB.

The result shows that a single share-link AXI outperforms a 5-layer AHB-lite interconnect in the videophone case study. Given that the hardware cost of a 5-layer AHB-lite interconnect is comparable to a shared-link AXI interconnect, using a shared-link AXI interconnect may be more efficient than using a multi-layer AHB interconnect.

4.5. Summary

The analysis in this work provides some insights for multimedia system design involving a shared-link AXI interconnect. If the buffer cost and transaction latency are not the primary concerns, system designers can consider using larger interface buffer to take the full advantage of out-of-order and outstanding transfer capabilities. However, some care must be taken in selecting a proper arbitration combination when using a channel-independent arbitration framework, especially when the interface buffer is large. The analysis showed that the arbitration combination could affect bus performance by up to 23.2 %. Moreover, the arbitration combination that yields the best system performance may vary depending on the interface buffer size, task access setting, and application traffic characteristic. In general, using a fair arbitration policy such as the TDMA is preferred for data channels. As to address channels, system designers can select a simpler arbitration policy to reduce the cost since the address channel arbitration has less impact on system performance. On the other hand, if the buffer cost and transaction latency do matter, system designers can use the hybrid data locked mode to achieve a performance similar to the case that uses only the interleaved mode, but with only half the interface buffer size. When the hybrid data locked mode is adopted for only long access latency devices, the simulation showed up to 21.1% completion time and 14.3% transaction latency reduction with respect to the setting without the hybrid data locked mode. With the short transaction latency, system designers can also consider adopting the hybrid data locked mode for latency-sensitive devices, such as CPUs, to reduce transaction latency. Although the analysis was conducted using AXI, we believe the experience can be extended and applied to other shared-link packet-based bus as well.

Chapter 5 Bandwidth Reduction Techniques in Computation Cores

5.1. Bandwidth Reduction Methods

Bandwidth requirement can be reduced in two major ways. The first way is to take the advantage of data characteristics. Some video or vision algorithms access data according to input data and intermediate data. These input and intermediate data often exhibit special patterns that can be used to reduce the number of data access. Take motion compensation for instance, its data access address is determined by motion vectors and macroblock types. Facilitating the motion vector and macroblock type characteristics can effectively reduce the bandwidth requirement. Another example is Meanshift filtering, the data to be accessed is determined by Meanshift vectors. However, the characteristic of the Meanshift vector magnitude can also be facilitated to reduce bandwidth requirement. The details of the bandwidth reduction methods adopted in these two examples are described in the following two subsections.

The other way to reduce bandwidth requirement is to reuse data based on an algorithm's data access spatial and temporal locality. By paying the price of extra small buffers, the bandwidth requirement can be greatly reduced. This dissertation took stereo matching to demonstrate how data reuse can be used to significantly reduce the bandwidth requirement. The detail of the stereo matching case is presented in subsection 5.4.

5.2. CFMMC

5.2.1. Motion Compensation's Role in a Video Decoder System

Over the past two decades, the development of the video coding standard has been undergoing great progress. Even though the latest video coding standard provides much better compression performance as well as extra functionalities, all video coding standards are still consist of motion compensation, transform, and entropy coding. Among these common video coding tasks, motion compensation interacts with the frame memory most, and is often the bottleneck of the speed, area, and energy in a video decoder. The operation in motion compensation can be regarded as to copy the predicted macroblock (MB) from the reference frame first, and then add the predicted MB with the residual MB to reconstruct the MB in the current frame. This operation involves extensive amount of frame memory access. Consequently, the bandwidth to the frame memory would become a performance bottleneck. On the other hand, the frame memory access is also the dominating part in the energy consumption of a video decoder. In addition, the requirement of storing the great amount of the reference frame data and the reconstructed current frame data results in a frame memory which would occupy most of the silicon area in motion compensation. Therefore, the optimization of the frame memory architecture is of great significance in reducing the bandwidth requirement, cost, and energy consumption of motion compensation.

The most common frame memory architecture for video coding without bidirectional prediction is the ping-pong frame memory (PPFM), which stores the reconstructed current frame and the reference frame in two memories. The PPFM swaps the role of the reconstructed current frame memory and the reference frame memory upon the completion of each frame’s motion compensation. Hence, the reconstructed current frame memory of a previous frame (t-1) would become the reference frame memory of a current frame (t). As a result, the PPFM requires a memory size of two frames, which is a considerable amount. In addition, the read and write accesses to the frame memory result in high bandwidth requirement. Furthermore, the energy consumption due to accessing the large sized memories often accounts for approximately half of the energy in a video decoder [47].

Motivated by the fact that the PPFM is very bandwidth hungry, area costly, and energy consuming, this paper proposes to use the statistical characteristics found from the video data to reduce the bandwidth, area, and energy consumption. We noticed that from the statistical analysis on various test sequences, the percentage of MBs with zero motion vector and no residue in a P-frame ranges from 7%~96%. Essentially, this type of MB is identical in the reference frame and the reconstructed current frame. By using this characteristic, this paper proposes the combined frame memory (CFM) architecture, which combines the reconstructed current frame memory and the reference frame memory into one single memory. Unlike other merged-frame approaches, the CFM includes an additional table to keep track of MBs with zero motion vector and no residue. For each MB without motion and residue, no further memory access is necessary for copying the MB from the reference frame to the reconstructed current frame. This is because the reconstructed current frame data and the reference frame data are the same and is already in the same memory. Thus, it is possible to reduce the bandwidth requirement and energy consumption due to frame memory accesses. In addition, the memory size is also reduced compared with that of the PPFM. Consequently, the cost for the motion compensation can be reduced. For QCIF resolution with the vector range of [-16:+15], the total memory size of the CFM architecture is reduced to 56.6% of the total memory size in the PPFM architecture.

There are two major contributions in this work. First of all, the statistical analysis and the concept of the combined frame memory motion compensation (CFMMC) are presented. This can serve as a reference for designing a CFMMC for other video decoder in different standards. The other contribution is the investigation result on the hardware implementations. The implementations of the CFMMC and the ping-pong frame memory motion compensation (PPFMMC) are compared to each other. The comparison over various test sequences shows that the hardware architecture can achieve lower bandwidth requirement, less silicon cost, and can reduce the energy consumption by -32% ~ 18%. To achieve the best benefit in throughput and energy consumption from using the CFMMC, the video being processed must exhibit enough percentage of MBs without motion and residue. This suggests that the CFMMC is more suitable for applications with much still background, such as video surveillance, video telephony, and video conference.

5.2.2. Combined Frame Memory Motion Compensation

A. Statistics of Perfect-Matched MB

The percentage of perfect-matched MBs within a frame determines the bandwidth reduction and energy consumption of the frame memory in motion compensation. A *perfect-matched MB* is one that has zero-valued MV and no residual. The reconstruction of such MB does not require the summation of the motion compensated (predicted) MB and the residual MB. For instance, a *NOT-CODED* MB in MPEG-4 [52] is a MB with zero-valued MV and no residual; hence a NOT-CODED MB is a perfect-matched MB. If a MB is a perfect-matched MB, the MB data read from the reference frame memory is the same as the MB data written to the reconstructed current frame memory in PPFM. Since the perfect-matched MB would be read and written with the same content at the same location, there is an opportunity to eliminate the redundant memory access for a perfect-matched MB. To eliminate the repeat accesses for a perfect-matched MB, the content of the perfect-matched MB must be already in the reconstructed frame memory before performing the motion compensation. The only way to achieve this requirement without performing extra memory access is to merge the reconstructed frame memory with the reference frame. Therefore, it is necessary to use the merged-frame approach so that the memory accesses of a

Table 7 Percentage of perfect-matched MBs when QP=16

Test sequences	**QCIF (%)**	**CIF (%)**
container (A)	91.74	88.91
mother_daughter (A)	81.42	77.65
hall (A)	86.21	83.86
akiyo (A)	91.32	89.09
coastguard (B)	10.35	2.69
foreman (B)	24.49	23.38
news (B)	82.53	83.01
stefan (C)	15.71	20.90
mobile	10.93	3.39

perfect-matched MB can be eliminated. Since the memory access reduction depends on the percentage of perfect-matched MBs within a frame, the reduction of bandwidth requirement and energy consumption is also highly dependent on this percentage.

Table 7 lists the average percentage of perfect-matched MBs within one frame. Both the results for QCIF and CIF sized sequences are listed. The statistics were gathered from running MPEG-4 VM18 [53] with the quantization parameter (QP) set to 16. The parenthesis next to each sequence represents the class it belongs as classified in [53]. Class "A" to "C" represents different levels of spatial detail and amount of movement, where class "A" is the lowest class and class "C" is the highest class. The statistics shows that lower class test sequences, such as akiyo, container, mother_daughter, news, and hall, which exhibit large portion of static background have more than 70% of perfect-matched MBs in average. Other test sequences with more motion, such as foreman, stefan, coastguard, and mobile, have less than 30% of perfect-matched MBs.

The QP used in Table 7 was 16, this QP value was relatively lower than the typical QP values of 16~24 adopted in practical MPEG-4 applications. Fig. 20 illustrates the impact of different QP values on the percentage of perfect-matched MBs. It can be seen that for most sequences with high percentage, the highest percentage of NOT-CODED MB appeared when QP=16. However, for most sequences with low percentage, the percentage of NOT-CODED MB significantly increased until QP=24. After QP>24, the increase became insignificant. It is suspected that after the QP is larger than 24, the reconstructed frame's quality would be so bad that the residue becomes increasingly larger, thus resulting the decrease in the percentage of NOT-CODED MB. Nevertheless, for the sequences which have low percentage, since the percentage of NOT-CODED MBs increases when QP>16, practical video applications should result in higher percentage of NOT-CODED MBs than those listed in Table 7 for these sequences.

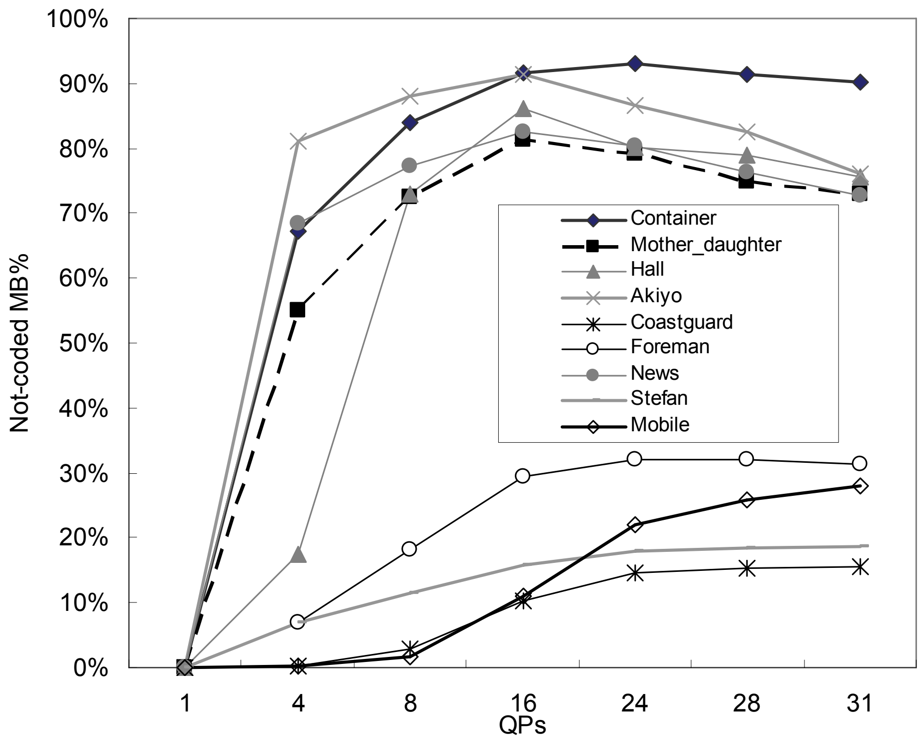


Fig. 20. Impact of different QP values on percentage of NOT-CODED MB in MPEG-4

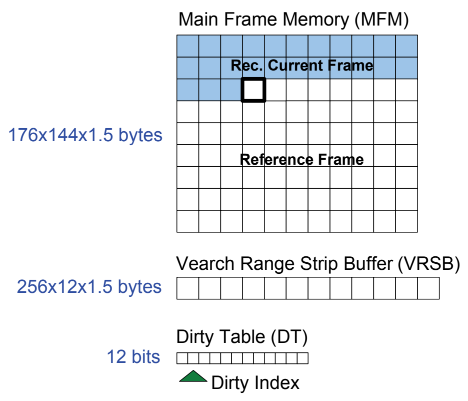


Fig. 21. Memory components for QCIF with vector range of [-16:+15] in the CFM

B. Combined Frame Memory

The CFM architecture adopts the merged-frame approach with an additional look-up table. The reconstructed current frame data and the reference frame data are mapped to one single frame memory with the size of one single frame. Unlike the merged-frame approach in [48][49], we introduced the additional look-up table to indicate whether the predicted pixel data are in the frame memory or in the local buffers. There are three major parts in the proposed CFM architecture: the main frame memory (MFM), the vector range strip buffer (VRSB), and the dirty table (DT), as illustrated in Fig. 21 for QCIF size with the vector range of [-16:+15]. The function of each component is explained as follows.

- Main frame memory (MFM): The MFM stores the reference frame data and the reconstructed frame data together. The reconstructed current frame data are stored at the upper part of the MFM whereas the reference frame data are stored at the lower part of the MFM. The size of MFM is as large as one single frame, i.e. 176x144x1.5 bytes for QCIF.

- Vector range strip buffer (VRSB): The VRSB is a rectangular strip of memory which works as an exchange buffer for the reference frame data. If one reference MB in the

MFM is to be updated by a reconstructed current MB, this reference MB would be copied into the VRSB as a backup in case the subsequent MB needs it. This avoids the reference frame data from being ruined by the reconstructed current frame data. The size of the VRSB is determined by the height of the vector range and the width of a frame, i.e. 16x(176+16)x1.5 bytes for QCIF with the vector range of [-16:+15].

- Dirty table (DT): The DT is the look-up table that keeps record of which pixels in the MFM are updated. If a MB in the MFM is updated by the reconstructed current frame data, the corresponding dirty bits of this MB will be set. This indicates that the

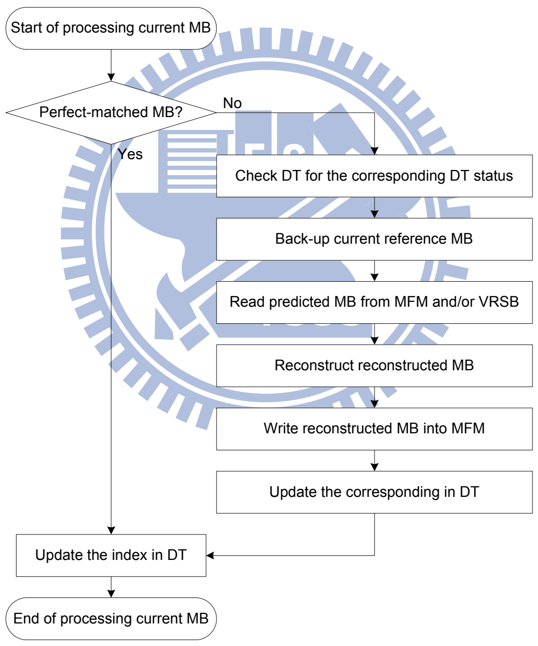
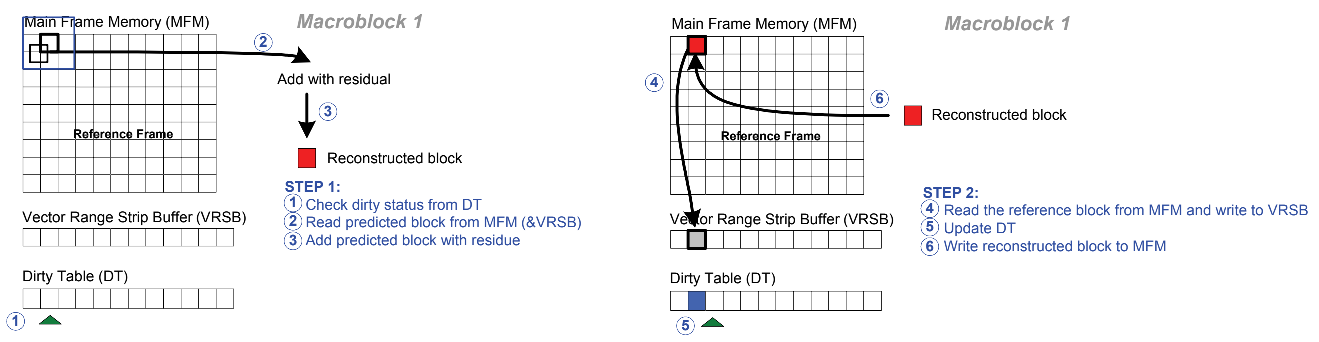
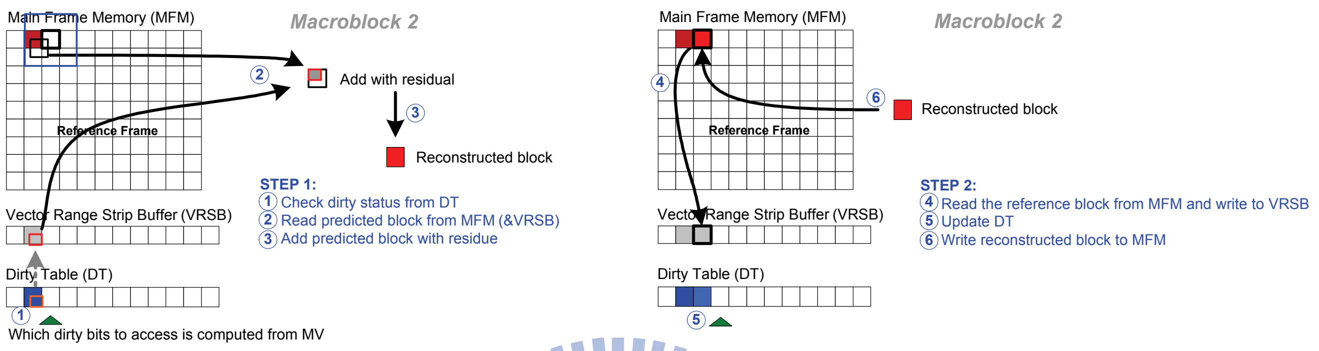

Start of processing current MB
Perfect-matched MB?
No
Yes
Check DT for the corresponding DT status
Back-up current reference MB
Read predicted MB from MFM and/or VRSB
Reconstruct reconstructed current MB
Write reconstructed MB into MFM
Update the corresponding in DT
Update the index in DT
End of processing current MB

Fig. 22. Flow chart of motion compensation process in the CFMMC

reference pixels in that MB are stored in the VRSB for backup as mentioned earlier. If the subsequent MB requires the reference pixels of this MB, these reference pixels will be read from the VRSB instead of the MFM. The size of the DT varies according to the size of the VRSB, i.e. 16x(176+16) bits for QCIF with the vector range of [-16:+15].


Main Frame Memory (MFM)
Macroblock 1
Add with residual
Reference Frame
Reconstructed block
STEP 1:
1 Check dirty status from DT
2 Read predicted block from MFM (&VRSB)
3 Add predicted block with residue
Vector Range Strip Buffer (VRSB)
Dirty Table (DT)
STEP 2:
4 Read the reference block from MFM and write to VRSB
5 Update DT
6 Write reconstructed block to MFM

(a) MB 1


Main Frame Memory (MFM)
Macroblock 2
Add with residual
Reference Frame
Reconstructed block
STEP 1:
1 Check dirty status from DT
2 Read predicted block from MFM (&VRSB)
3 Add predicted block with residue
Vector Range Strip Buffer (VRSB)
Dirty Table (DT)
Which dirty bits to access is computed from MV
STEP 2:
4 Read the reference block from MFM and write to VRSB
5 Update DT
6 Write reconstructed block to MFM

(b) MB 2

Fig. 23. The processing of non-perfect-matched MBs

Fig. 22 illustrates the flow chart of the CFMMC. The process is very simple for a perfect-matched MB, but is complex for a non-perfect-matched MB. When processing a perfect-matched MB, since the reference MB and the reconstructed MB are the same and resides within the MFM at the same location, no memory access is performed. The only operation carried out is the updating of the index in the DT. For the non-perfect-matched MB case, the DT is checked first to determine where the predicted MB pixels are stored, each pixel in the predicted MB is either read out from the MFM or the VRSB according to the corresponding dirty bit. After the predicted MB is read out, it is summed with the residual to reconstruct the reconstructed MB. Then the current reference MB in the MFM must be copied into the VRSB before the reconstructed MB is written back to the same location. Finally, the reconstructed MB is written back to the MFM, and the DT and its index are updated at the end. Fig. 23 illustrates the motion compensation process for two consecutive non-perfect-matched MBs.

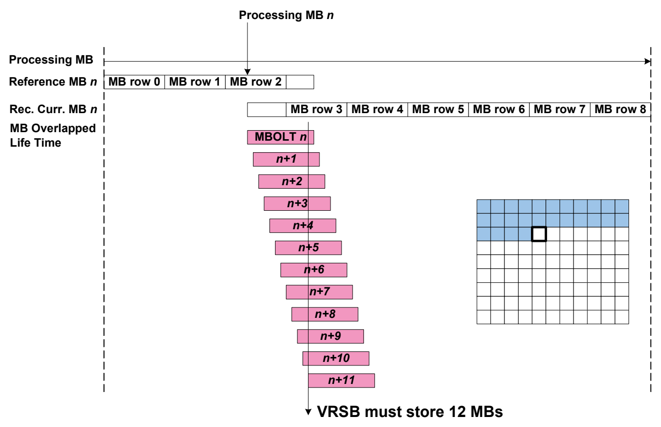


Fig. 24. Life time analysis of MBs

C. Analytic Estimation of Memory Size, Energy, and Latency

The memory requirement of the CFMMC can be determined through the life time analysis of the collocated MB in the reconstructed current frame and the reference frame, as illustrated in Fig. 24. For each MB, the life time of the reconstructed current frame data and the reference frame data overlaps for a portion of period during the processing of one single frame. This overlapped lifetime of a collocated MB would be referred as *MB overlapped life time (MBOLT)* here on. The length of MBOLT is determined by the vector range's height and width. For instance, the reconstruction of the current MB requires the reference pixels from the most upper-left corner of the vector range in the worst case; thus the reference MB having the required reference pixels has to remain in VRSB until the reconstruction of the current MB is complete. MBOLT is proportional to the raster-scan MB distance between the reference MB and the current MB. The larger the vector range is, the longer the MBOLT is.

The maximum number of MBs having overlapped MBOLT determines the size for VRSB and DT. In another word, the VRSB size must be large enough to store all the reference MBs who are currently alive. By the term alive we mean that a reference MB may still be needed by further motion compensation of subsequent MBs. For example, consider the case of QCIF with the vector range of [-16:+15], the maximum number of MBs having overlapped MBOLT is 12 MBs. This means there are at most 12 reference MBs alive simultaneously, hence the VRSB size is 12 MBs and the DT size is 12 bits. Comparing the memory requirement with other merged-frame approach [48][49], the VRSB size is 1 MB smaller than their LIFO buffer size. We generalized the formulation of memory size requirement for the MFM, VRSB, and DT and listed them in Table 8. Note that this formulation can be applied to any given frame size and vector range. The overall memory size was also compared with that of the most commonly used PPFM. For the aforementioned QCIF case, the memory size of the CFMMC architecture is 56.6% compared to that of the PPFM architecture.

Table 8 Memory sizes required in CFM

Memory	**Memory Size Formula (bytes)**	**Size for QCIF with SR of [-16, +15] (bytes)**
MFM	height_frame x width_frame x 1.5	38,016
VRSB	floor(height_VR/height_MB) x height_MB x (width_frame + (floor(width_VR/width_MB) x width_MB)) x 1.5	4,608
DT	floor(height_VR/height_MB) x height_MB x (width_frame + (floor(width_VR/width_MB) x width_MB)) x 0.125	384
Combined Total	size_of_MFM + size_of_VRSB+size_of_DT	43,008
Ping-pong Total	(height_frame x width_frame x 1.5) x 2	76,032

Table 9 Memory access energy consumption and access latency of processing one frame

Memory	Access Bandwidth Requirement	Access Energy Consumption	Access Latency
MFM	M x 3 x (1-P_0) x D_{MB}	M x 3 x (1-P_0) x E_{MFM}	M x (1-P_0) x 3 x C_{MB}
VRSB	M x (1-P_0) x D_{MB}	M x (1-P_0) x k^{-1} x E_{MFM}	M x (1-P_0) x C_{MB}
DT	M	M x k^{-1} x E_{MFM} x 0.125 (neglected)	M x 0.125 x C_{MB} (neglected)
Combined Total	M x 3 x (1-P_0) x D_{MB}	M x (3+ k^{-1}) x (1-P_0) x E_{MFM}	M x (1-P_0) x 4 x C_{MB}
Ping-pong Total	M x 2 x D_{MB}	M x 2 x E_{MFM}	M x 2 x C_{MB}

Table 10 Average memory access energy consumptions and latencies for various QCIF test sequences with k=4

K=4	**Average Bandwidth Requirement**			**Average Energy Consumptions**			**Average Access Latency**		
Test sequences (QCIF)	**Ping-pong (MxD_{MB})**	**Combined (MxD_{MB})**	**Reduced bandwidth (%)**	**Ping-pong (MxE_{MFM})**	**Combined (MxE_{MFM})**	**Reduced energy (%)**	**Ping-pong (MxC_{MB})**	**Combined (MxC_{MB})**	**Reduced latency (%)**
container (A)	2	0.25	87.6	2	0.27	86.6	2	0.33	83.5
mother_daughter (A)	2	0.56	72.1	2	0.60	69.8	2	0.74	62.8
hall (A)	2	0.41	79.3	2	0.45	77.6	2	0.55	72.4
akiyo (A)	2	0.26	87.0	2	0.28	85.9	2	0.35	82.6
coastguard (B)	2	2.69	-34.5	2	2.91	-45.7	2	3.59	-79.3
foreman (B)	2	2.27	-13.6	2	2.45	-22.7	2	3.02	-51.0
news (B)	2	0.52	73.8	2	0.57	71.6	2	0.70	65.1
stefan (C)	2	2.53	-26.4	2	2.74	-37.0	2	3.37	-68.6
mobile	2	2.67	-33.6	2	2.89	-44.7	2	3.56	-78.1

Table 9 lists the analytic model of average bandwidth requirement, energy consumption, and latency due to memory accesses. The model is evaluated for processing one P-frame. In Table 8, D_{MB} represents the amount of data to be read or write for one macroblock. The total bandwidth requirement accounts only the access with the MFM since it is common to implement MFM using external memories. We model the energy consumption of accessing one MB in the MFM and the VRSB as E_{MFM} and E_{VRSB} respectively. This assumes that the energy consumption of a memory read and a write are the same. Based on this assumption, the average memory energy consumption of processing a frame is listed in Table 8, where M represents the number of MBs in a frame, P_0 represents the percentage of perfect-matched MBs, and k represents the ratio of E_{MFM} to E_{VRSB}. The energy consumed in the MFM includes the energy of reading predicted MBs from the MFM, reading the reference MBs for backup, and writing the reconstructed MBs into the MFM. Since the accesses to the MFM only occurs when processing a non-perfect-matched MB, only Mx(1-P_0) MBs would read the MFM twice and write it once. The energy consumption of the VRSB is mainly due to the backup of reference MBs, which writes the reference MBs of non-perfect-matched MBs into VRSB. Although some predicted pixels may have been stored in the VRSB, the worst case for energy consumption happens when all the predicted pixels are read from the MFM. This is the reason we account the energy of reading predicted pixels to the MFM's energy consumption.

The memory access latencies of processing one frame are also listed in Table 9. The access latencies are modeled based on the assumption that the access latencies of read and write to either the MFM or the VRSB are all the same, hence the memory access latency of accessing one MB is denoted as C_{MB} . A typical scenario for such assumption to hold is when SRAM is adopted for both the MFM and the VRSB. In the CFMMC, extra memory access latency is introduced for a non-perfect-matched MB whereas the memory access latency for a perfect-matched MB is eliminated. For each non-perfect-matched MB, the predicted MB is first read from the MFM or the VRSB, and then the content of the current MB which resides in the MFM is read and written into the VRSB for reference MB backup; the reconstructed current frame is then written back to the MFM at the end. As a result, the memory access latency of four MBs is needed for each non-perfect-matched MB. However, overlapping the latencies of reading the reference MB from MFM and writing the reference MB into VRSB may reduce the total latency to $\mathrm{M}\times(1-\mathrm{P}_0)\times 3\times \mathrm{C}_{\mathrm{MB}}$. According to the formulas in Table 8, the memory access latency in the CFMMC can be less than that of the PPFMMC when P_0 is larger than 50%.

The reduction of energy consumption in the CFMMC depends on the adopted memory type and the contents of video sequences. For instance, if on-chip SRAM [54] is used for both the MFM and the VRSB, k would be about 4. Note that this SRAM case may represent the worst case reduction of energy consumption. If external memory is adopted, such as

Mobile SRAM [55], *k* might be even larger, and the energy reduction should also be larger. The reduction of bandwidth requirement, memory energy consumption, and access latency in different test sequences when $k = 4$ is listed in Table 10. The CFMMC may reduce 72.1% ~ 87.0% of memory access bandwidth compared to that of the PPFMMC for QCIF test sequences *container, akiyo, news, hall,* and *mother_daughter*. However, for test sequences with small P_0, such as *foreman, stefan, coastguard,* and *mobile*, the estimated bandwidth requirement may increase by 13.6% ~ 34.5% compared to that of the PPFMMC. This analytic evaluation disregards the impact of memory banking because the memory organization is beyond the scope of interest in this work.

Table 10 also lists the estimated memory access energy and latency for different test sequences. For test sequences with larger P_0 (>70%), the memory access energy consumption and latencies in a QCIF frame can be reduced by 71.6% ~86.6% and 62.8% ~ 83.5% compared with that of the PPFMMC respectively. However, for other test sequences with smaller P_0, such as *foreman, stefan, coastguard,* and *mobile*, the access energy consumption and latency are increased by 22.7% ~ 45.7% and 41.0% ~ 79.3%. However, this extra latency can be hidden by overlapping these latencies with the computation time of motion compensation.

5.2.3. Architecture

The VLSI architecture of the proposed CFMMC is designed and synthesized. Although hardware implementation enables overlapping of memory accesses, the complex control of the CFMMC would introduce extra hardware and energy consumption overheads. This extra hardware includes logics for block and pixel offsets computation, address generation, and dirty table management. The additional operations also consume extra energy consumption, which would reduce the energy consumption reduction of the CFMMC. To find out the impact of these overheads, the hardware implementations of both the CFMMC and the

PPFMMC are compared.

The implementations are targeted for mobile devices, so the video format of QCIF is considered. Since the format size is small, SRAM is used for both the MFM and the VRSB. However, the CFMMC is not limited to any frame size, nor is it limited by only using SRAMs. The details on the architecture of the CFMMC and the PPFMC are explained in the following subsections.

A. Architecture of the Combined Frame Memory Motion Compensation

The architecture of the CFMMC is illustrated in Fig. 25, which consists of five major parts. The first part includes the *mvprocessor*, the *pblk* and *inblk offsets generators*, and the *dirty table*. The second part is the *memory accessor*, which includes the *address generators* for MFM and VRSB, the *memory multiplexer*, and the *predicted row buffer*. The third part is the *motion compensation controller* which coordinates the tasks among the modules and also interfaces the control signals. And the fourth part is the *filter and reconstructor*. The last part is the memories, which includes the MFM and the VRSB. Each of the key parts will be explained in this subsection.

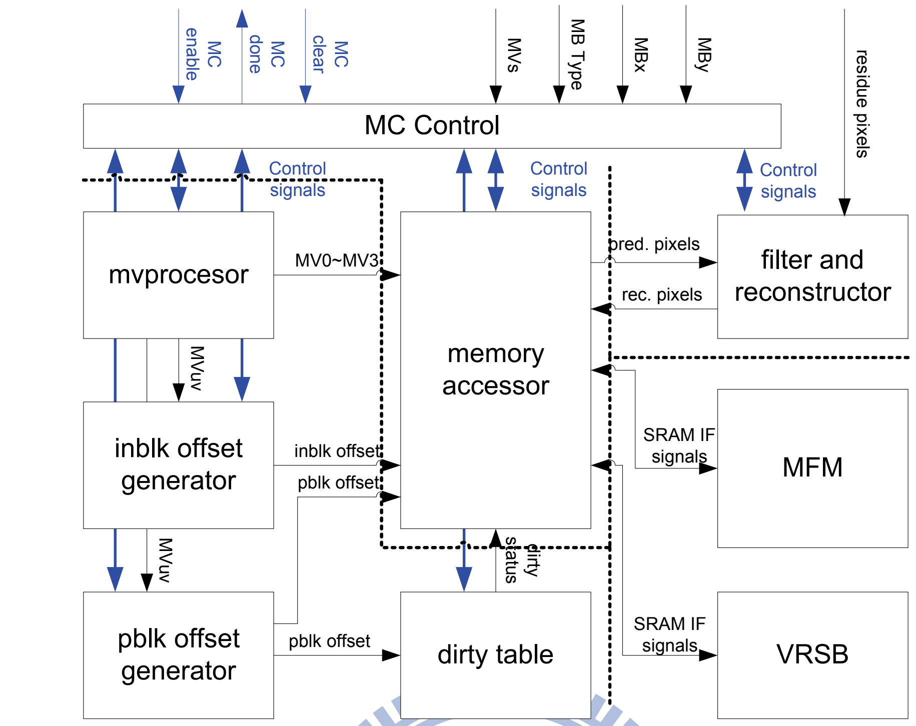


Fig. 25. Block diagram of the CFMMC hardware

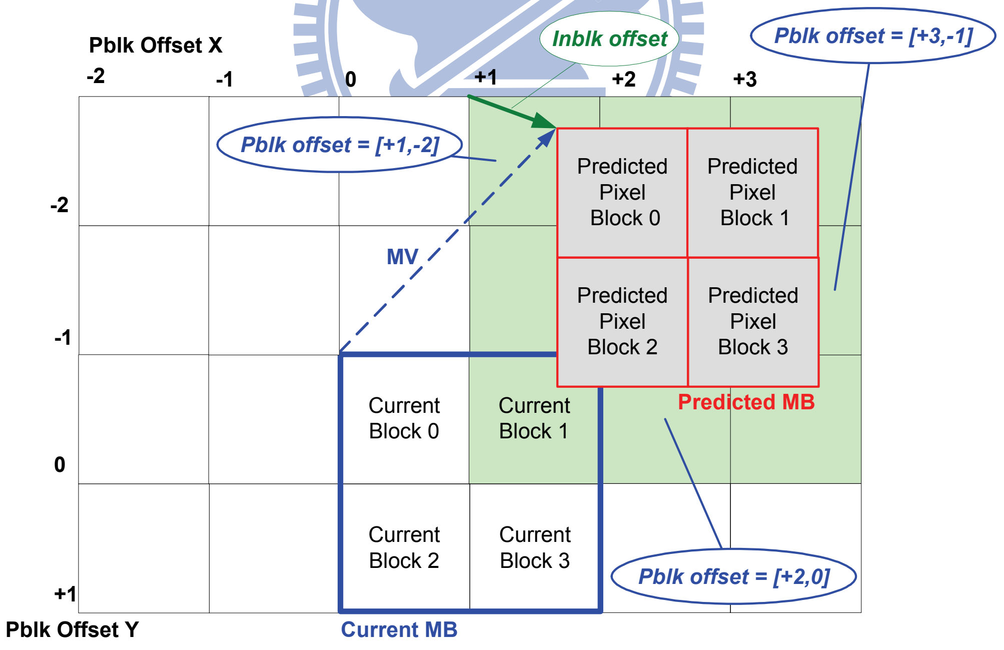


Fig. 26. Pblk offsets and inblk offsets

Before introducing each part of the CFMMC architecture, the address generation will be explained first. Fig. 26 illustrates the relation between the motion vector and the offsets used for the address generation. For each block with a motion vector, the blocks which contain the predicted pixels are referred as *pblk*. A predicted pixel block can cover up to 4 *pblks*. The offsets between the current block and the *pblks* are the *pblk offsets*. The base address of *pblks* can be computed using *pblk offsets* and the block position of the current block. The distance between the top of the predicted pixel block and the top boundary of the *pblk* within a *pblk* is the vertical *inblk offset*. Similarly, the distance between the left side of the predicted pixel block and the left boundary of the *pblk* within a *pblk* is the horizontal *inblk offset*. The base address of the predicted pixel block is computed using the *inblk offsets* and the base address of the *pblk*.

The first part of the CFMMC architecture pre-processes the key information that is needed by the memory accessor, such as the motion vector of the chroma component, the *pblk offsets*, the *inblk offsets*, and the dirty status. The components are explained below.

- *MVprocessor:* This module computes the motion vector of the chroma components from the motion vectors of the luma component.
- *Pblk offset generator:* This module computes the *pblk offset* from the motion vector. The *pblk offset* is used to determine which entry in the dirty table is to be accessed.
- *Inblk offset generator:* This module computes the *inblk offset* of each *pblk* according to the motion vector. The *inblk offset* is used by the memory accessor together with the *pblk offset* to compute the base address within the MFM and the VRSB.
- *Dirty Table:* This module provides the dirty status of each *pblk* and updates the dirty table. The dirty status is needed by the memory accessor to determine whether the MFM or VRSB should be accessed. At the start of processing each frame, all entries in the dirty table are reset. The update procedure is done for every MB(4 blocks) at the end of the motion compensation process. In addition, the *dirty index*,

which is computed from the *pblk offset* to determine the current entry in the dirty table, is also provided for the memory accessor to generate the address to the VRSB. Each entry is one bit and represents the dirty status of its corresponding block. To support the VRSB with the size of 12 MBs(48 blocks), the dirty table has 48 entries, which is implemented using a 48 bits register array.

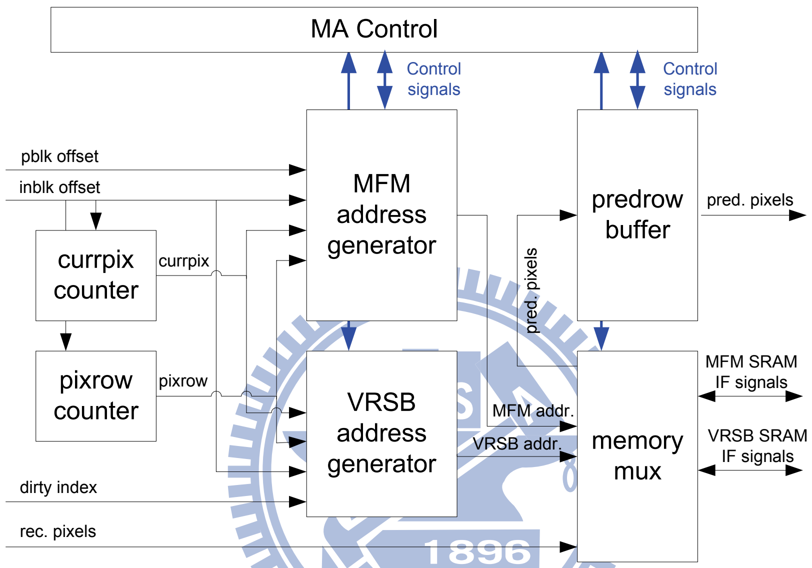


Fig. 27. Block diagram of the memory accessor

The second part is the memory accessor, which plays the major role of generating the addresses to the MFM and the VRSB and multiplexing data among the memories and buffers. The block diagram is shown in Fig. 27. Each block is explained as follow.

- *MFM Address Generator*: This address generator generates the memory addresses for the MFM. The address includes those for reading the predicted pixels, reading the reference pixels for backup, and writing the reconstructed pixels. The address is generated from the *pblk offset*, *inblk offset*, and the counter values within the memory accessor.

- *VRSB Address Generator:* The address generator for the VRSB is similar to the MFM's except it generates addresses for the VRSB. This module generates the address for reading the predicted pixels and the address for writing the backup reference pixels. The addresses are generated using the *dirty index* from the dirty table, *inblk offset* and the counter values.

- *Memory Multiplexer*: The memory multiplexer handles the data traffic among the MFM, the VRSB, the *predicted row buffer*, and the reconstructed buffer. When the predicted pixels are read from the MFM and the VRSB, the pixels would be written into the predicted row buffer for reconstruction. If the reference pixels need to be backup, the pixels would be read from the MFM and then written into the VRSB. After the reconstruction is done, the reconstructed pixels would be read from the reconstructed buffer, which is located in the filter and reconstructor, and then written into the MFM. The control of the memory multiplexer is given from the memory accessor control.

- *Predicted Row Buffer*: The predicted row buffer stores the predicted data before sending them to the filter and reconstructor. The filter and reconstructor is designed to handle one row of predicted pixels at a time, hence the predicted row buffer only stores one row of pixels.

- *Memory Accessor Control*: The memory accessor control coordinates the operations of different modules. An internal FSM determines whether to idle, to read the predicted pixels, to backup the reference pixels, or to write the reconstructed pixels. The memory accessor is only activated when the CFMMC is enabled and is in the processing state. The processing state is defined as the state in which the CFMMC actually needs to perform memory accesses. If the CFMMC encounters a NOT-CODED MB, which is a perfect-matched MB by definition, the

CFMMC would not enter the processing state. This means the memory accessor would remain in the idle state.

The filter and the reconstructor part are in charge of generating sub-pel samples and adding the predicted pixels with the residual pixels. The reconstructed pixels would be stored in the local reconstructed buffer, which is implemented with registers. Although implementing a 256-pixel buffer using registers is not economic, the implementation of the filter and reconstructor is beyond the focus of this work. To be fair during the architecture comparison, both the CFMMC and the PPFMMC use the same filter and reconstructor design.

The memories of the MFM and the VRSB are implemented using SRAM. There are two reasons for using SRAM instead of external DRAM. The first reason is that the design is targeted for QCIF format, hence the memory size of one QCIF frame is considered to be acceptable. The other reason is that the SRAM model being used includes a power model which is more convenient for evaluating power consumption than using DRAM models. Since using SRAM is likely to decrease the energy consumption ratio between the MFM and the VRSB, which is referred as the k value, the result acquired using SRAM can be considered as a lower bound for the amount of energy reduction. If k is larger, which is likely to be the case when the MFM is implemented using DRAM, the energy reduction should be larger. The detailed implementation and organization of the memories are described below.

- *MFM Memory*: The MFM is implemented to store both the luma and the chroma components. We used three 8,192 bytes and one 768 bytes single-port SRAM for the luma component. The total memory size of the luma component in the MFM is 25,344(256x99) bytes. For the chroma component, we used two 6,336(64x99) bytes single-port SRAM. The port data width of the MFM is 1 byte because the bandwidth is sufficient for processing QCIF at 30 frames per second.

- *VRSB Memory*: The VRSB is implemented to backup the reference MBs for both the luma and the chroma components. We used a 3,072(256x12) bytes single-port SRAM for the luma component and two 768(64x12) bytes single port SRAM for the chroma components. The port data width of the VRSB is also 1 byte because of the same reason used in the MFM.

B. Architecture of the Ping-pong Frame Memory Motion Compensation

A prototype of the PPFMMC is implemented based on the prototype of the CFMMC. Therefore, some of the modules are the same between the two prototypes; this would help us observe the differences easier. The main differences between the PPFMMC and the CFMMC are that there is no offsets generation, no dirty table, and no address generator for the VRSB. Another major difference is that the PPFMMC needs extra frame memory to store another frame.

In the PPFMMC, there's no need to compute the *pblk* and *inblk offsets* because the pixel data are not partitioned into blocks to correspond to the dirty bits. Without this partitioning and mapping, the PPFMMC not only reduces the logics for the offsets computation, it also eliminates the need of the dirty table. Moreover, the address generation is also simplified to only compute the logical coordinate of the pixels to be accessed. This address computation only requires the current coordinate and the motion vector. The simplified address generation and the elimination of the VRSB address generator reduce the area of the memory accessor. As a result, the area cost of the logics in the PPFMMC should be smaller than the CFMMC's.

The PPFMMC requires a memory to store both the reference frame and the reconstructed current frame. To achieve this, two MFM modules are used in the CFMMC to implement the ping-pong frame memory. Each MFM module is consists of three 8,192 bytes SRAMs, four 768 bytes SRAMs, and two 6,336 bytes SRAMs. As a result, the total frame memory size is 50,688 bytes.

Table 11 Latencies of different MB modes

MB Modes	Description	PPFMMC Latency (cycles)	CFMMC Latency (cycles)
INTRA	Intra MB in I-frames	385	385
INTER_INTRA	Intra MB in P-frames	385	770
INTER	Inter MB with only 1 MV	770	1157
INTER4V	Inter MB with 4 MVs	770	1157
NOT-CODED	Inter MB with perfect-match	770	1

C. Architecture Latency Comparison

The latency of the CFMMC and PPFMMC hardware architectures are both dominated by the memory access time. The memories adopted in our hardware architectures are single-port SRAMs with data port width of 8-bit. However, the latencies of processing different MB modes, such as INTRA, INTER_INTRA, NOT_CODED, INTER, and INTER4V, are different. The definition of different MB modes and their processing latencies in the two prototypes are listed in Table 11.

Essentially, the processing operations of INTRA MBs and INTER-INTRA MBs are similar. Both the CFMMC and PPFMMC check the MB mode first, which requires one cycle. After that, the PPFMMC directly write the residue into the frame memory according to the MB's position. This is similar to the CFMMC when processing an INTRA MB. As a result, the total latencies of processing an INTRA MB in both architectures and processing an INTER-INTRA MB in the PPFMMC are the same, which is 1+(256+64+64) = 385 cycles. However, for an INTER-INTRA MB, the CFMMC must back-up the collocated reference MB into VRSB and update the dirty table, which is not necessary for an INTRA MB. Consequently, the total latency of processing an INTER-INTRA MB is 1 + (256+64+64) + 1 + (256+64+64) = 770 cycles.

The latency to process an INTER or INTER4V MB in the CFMMC includes the latencies of identifying the MB mode, computing the chroma's motion vectors, reading the predicted pixels, backing up the reference pixels, and writing back the reconstructed pixels. The computation of chroma's motion vectors takes 3 cycles to complete. The reconstruction computation, which adds the residue with the predicted pixels (Fig. 23, Step1 (2)) is performed while the backup of the reference pixels (Fig. 23, Step2 (4)) is taking place. Therefore, the latency of processing an MB is 1 + 3 + (256+64+64) + (256+64+64) +1 + (256+64+64) = 1,157 cycles. In contrast, the latency of PPFMMC only accounts the latencies of reading the predicted pixels, writing the reconstructed pixels, and some preprocessing time. The difference is that no backup of the reference pixels is needed in the PPFMMC. As a result, the total latency of processing an MB in the PPFMMC is 1 + (256+64+64) + 1 + (256+64+64) = 770 cycles.

For NOT-CODED MBs, they are perfect-matched MBs. Thus, the CFMMC does not perform any memory access; the total latency of processing a NOT-CODED MB only takes one cycle. The one cycle latency is used to update the dirty table. On the contrary, the PPFMMC has to perform the operations of reading and writing the ping-pong frame memory. As a result, the latency of processing a NOT-CODED MB is exactly the same as that of processing an INTER or INTER4V MB, which is 770 cycles.

Considering the processing latency for the worst case scenario, which is no perfect-matched (NOT-CODED) MB is found in a P-frame, the clock rate of the CFMMC would have to be 1.5 times of that in the PPFMMC. With the increased clock rate and the increased memory access to backup the reference pixels, the energy consumption in the CFMMC operating under the worst case scenario might increase by more than 50%. However, the negative impact of the clock rate increase in the CFMMC architecture can be minimized when P_0 is higher than 33.4%.

5.2.4. Implementation Result

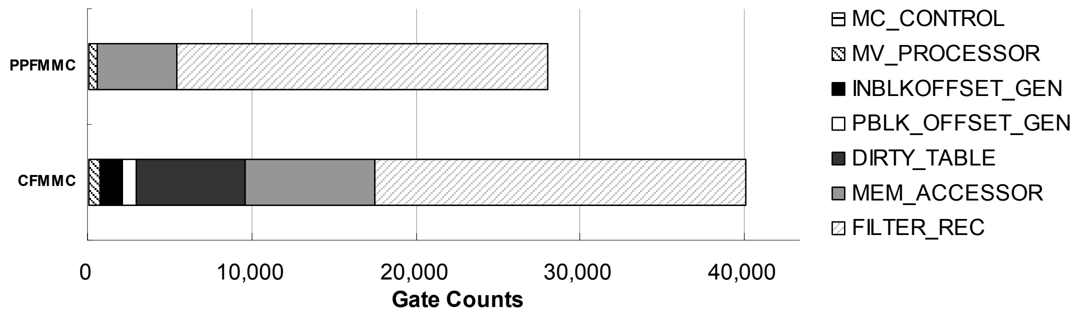


Fig. 28. Gate count distribution and comparison of the logics part in the PPFMMC and the CFMMC

A. Architecture Cost Comparison

The prototypes of the CFMMC and the PPFMMC architectures are both synthesized from Verilog RTL design using UMC 0.18μm 1P6M CMOS technology [56]. Both designs are synthesized with the clock constrained at 50 MHz, which is more than enough to perform real-time decoding. The logics of the CFMMC prototype (exclude memory) have a gate count of 40,065. The MFM and the VRSB have the equivalent gate count of 280,749, which occupies 87.5% of the total cell area. As to the prototype of the PPFMMC, the logics part has a gate count of 28,039. The ping-pong frame memory has an equivalent gate count of 486,643. It is obvious that the extra logics used in the CFMMC increases the gate count by 43.4% compared with the gate count of the logic part in the PPFMMC. However, the total equivalent gate count, which takes the memory area into consideration, shows that the cell area of the CFMMC is actually 37.7% smaller compared with that of the PPFMMC. In other words, the total cell area of the CFMMC is only 72.3% of PPFMMCS's total cell area. The detailed distributions of the gate counts in different modules excluding the memories are compared in Fig. 28. It can be seen that the dirty table, the *pblk offset* generator, the *inblk offset* generator, and the extra logics in the memory accessor are responsible for the area increase. Despite the increase in the area of the CFMMC's logics part, which seems to be relatively large, the total cell area is significantly smaller than that of the PPFMMC.

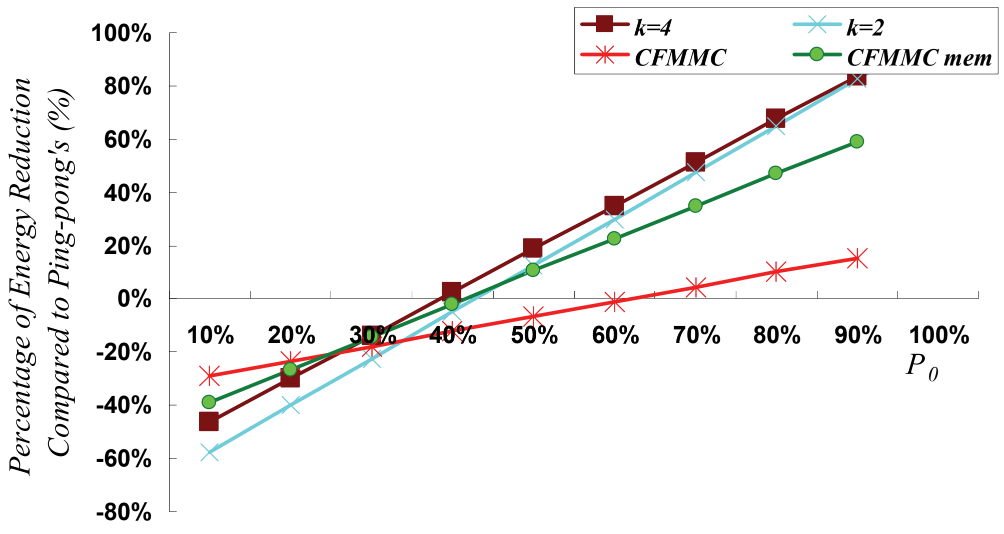


Fig. 29. Plot of the energy reduction percentages of the CFMMC at different P_0

B. Architecture Energy Consumption Comparison

The gate-level power is reported by using Power Compiler [57]. The signal switching activities are gathered by running at 50 MHz for both the CFMMC and the PPFMMC. The reason to use such a high clock rate is to increase the numerical order of the reported power, which corresponds to the energy of processing 109 CIF frames in one second. This can make the comparison of the energy consumption between the CFMMC and the PPFMMC easier.

Fig. 29 plots the energy consumption comparison between the CFMMC and PPFMMC running the manually created test patterns. There are two lines related to the CFMMC architectures. One is the line for the memories themselves, the other one is for the overall CFMMC architecture. The line of the CFMMC has smaller slope than the slope of the CFMMC memories line. This is due to the energy consumed by the logics which reduced the energy reduction. The line of the CFMMC memories has a slightly smaller slope compared with the slopes of the theoretical lines with k=4 and 2. This may be explained by that the logics and memories also consume energy even when there's no memory access, thus compromising the energy model derived merely based on memory access energy. The detail distribution of the power consumption in the CFMMC when P_0=20% and 80% are illustrated in Fig. 30; the power distribution of the PPFMMC is also illustrated for comparison. The *memory accessor* accounts for the second most power consumption among the logics in the CFMMC. This part of the energy consumption was not part of the energy consumption model to evaluate the percentage of reduction.

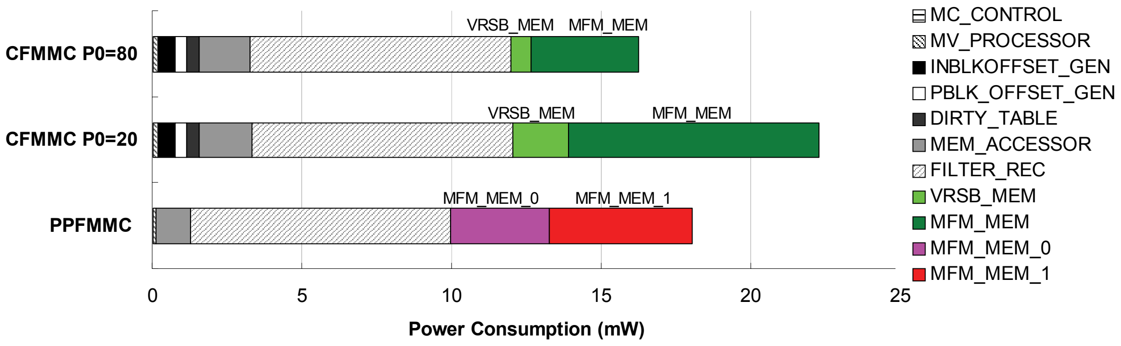


Fig. 30. Power consumption distribution and comparison of the PPFMMC and the CFMMC

The energy reduction percentages of the real test patterns are listed in Table 12. The percentages of NOT-CODED MB found in the first 15 frames of each test sequences are also listed. For those test sequences with more than 70% of MBs being NOT-CODED MBs, the energy consumptions are reduced by 11%~18%. For sequences with much less MBs being NOT-CODED MBs, the energy consumption are increased by 18%~32%. Fig. 31 illustrates the detail distribution of the power consumption evaluated for running test sequences *mobile* and *akiyo*, which are the sequences with the least and the most energy consumption reduction percentages. The result gathered from the real test patterns verifies the conclusion made in

Table 12 Energy reduction percentage of the real test patterns

Test sequences	***P_0* of the first 15 frames (%)**	**Energy reduction percentage (%) compared with the PPFMMC**
container (A)	92.93	15.45
mother_daughter (A)	92.83	15.80
hall (A)	95.96	17.19
akiyo (A)	96.87	18.44
container (A)	92.93	15.45
coastguard (B)	34.95	-18.06
foreman (B)	29.39	-21.89
stefan (C)	15.66	-30.66
mobile	7.47	-31.67

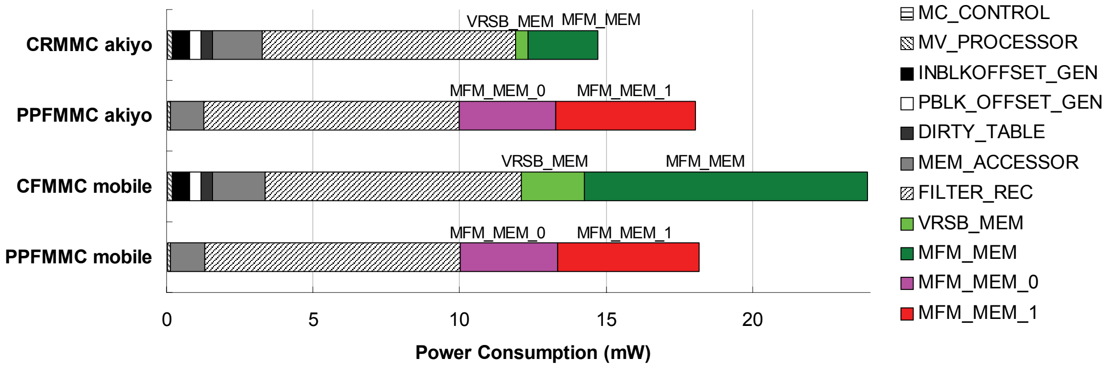


Fig. 31. Power consumption distribution and comparison of the PPFMMC and the CFMMC for mobile and akiyo.

previous section which states that the CFMMC should be more suitable for applications with more static background and less motion.

The hardware implementation of the CFMMC proves the feasibility of the architecture. The comparison with the most commonly used PPFMMC shows some advantages and limitations. From the cost perspective, the CFMMC has been proven to reduce the silicon area compared with the PPFMMC. From the latency perspective, the CFMMC prototype can achieve a comparable throughput when P_0 is more than 33.4%. The limitation, however, is in the worst case scenario in which the CFMMC would need a clock which is 1.5 times faster than that of the PPFMMC. Nevertheless, such issue can be alleviated with the use of dynamic frequency scaling. In the energy consumption aspect, the CFMMC prototype can reduce energy consumption when P_0 is high enough. Similar to the latency issue, the energy consumption would increase if P_0 is not high enough. This issue originates from the architecture’s data-dependent nature, and cannot be solved. With this limitation, the suitable applications of the CFMMC hardware are limited to video surveillance, video telephony, and video conferencing.

5.2.5. Summary

We proposed a combined frame memory motion compensation (CFMMC) architecture which did not only reduce the frame memory size, but also is potential in reducing bandwidth requirement, access latency, and energy consumption. The statistics on perfect-matched MB were investigated for the well known video sequences. Based on the statistical result, we derived the latency and the energy consumption model for evaluation. During the exploration, we found that when the percentage of perfect-matched MBs (P_0) was higher than 50%, the CFMMC could reduce both the latency and the energy consumptions due to memory accesses.

To investigate the cost of extra computation and control logics for achieving the aforementioned benefits, hardware architectures of the CFMMC and the most commonly used PPFMMC were both implemented. The hardware implementation of the CFMMC only required 75% of the silicon area used to implement the PPFMMC. The CFMMC architecture was also capable of reducing the bandwidth requirement and energy consumption up to 72% and 16% respectively when P_0>70%. However, when P_0 is not high enough , the CFMMC suffered from bandwidth requirement, energy consumption, and latency increases. Consequently, these limitations limited the application of CFMMC into video surveillance, video telephony, and video conferencing. For these applications, the CFMMC shall guarantee its bandwidth requirement, energy consumption, and latency reduction capability.

5.3. Meanshift

5.3.1. Meanshift's Role in Vision Applications.

Image segmentation has been widely adopted in applications such as intelligent surveillance, autonomous vehicles, and mobile robotics. These applications often require not only segmentation performance, but also processing speed. Among most of the image segmentation algorithms, the Meanshift algorithm [58][59] has been one of the most commonly used because of its good performance and speed. However, the processing speed of the Meanshift segmentation is still not fast enough for real-time (not necessarily video rate) applications. Table 13 lists the execution time taken by the optimized Meanshift segmentation program, EDISON [60], to process a VGA image. The total execution time took more than 2 seconds and the Meanshift filter operation occupied more than 78% of the total execution time. Thus, to reduce the execution time, a VLSI implementation is necessary for these real-time applications.

Table 13 EDISON's execution time of VGA image "Raincoat Sam" on a PC with PentiumIV 2.8GHz processor and 1GB memory.

Operation	Execution time (sec.)	Percentage (%)
Meanshift Filter	1.58	78.6
Connected Component	0.09	4.5
Transitive Closure	0.20	9.9
Region Pruning	0.14	7.1
Total	2.01	100.0

Motivated by the need of speed in the Meanshift filter operation, we propose a VLSI architecture for the Meanshift filter. However, VLSI design of the Meanshift filter encounters severe challenges such as huge image data access, large temporary storage, and limited parallelism due to the nature of the Meanshift algorithm. To conquer these challenges, this dissertationr presents several novel design approaches. First, for the huge image data access, we proposed a partial-update ping-pong buffer that significantly reduces the amount of data read from the input image memory. Second, for the large temporary storage, we use a ping-pong local Meanshifted status buffer that reduces the size of the Meanshifted status memory by 50%. Third, for the limited parallelism, we propose a 4-pixel parallel 9-stage pipelined Meanshift vector computation unit to utilize the limited parallelism in the standard Meanshift algorithm. The proposed architecture can process 9.8 FPS of VGA image in average when clocked at 110 MHz.

The contribution of this work is two fold. First, the proposed architecture is the first VLSI architecture implementing the standard Meanshift filter algorithm. The speed of the proposed architecture is at least 3 times higher than the software solution. This makes the Meanshift filter available to non-video rate real-time applications. Second, this is the first work that investigated the amount of data read from the image memory needed by the Meanshift algorithm. We believe this bandwidth requirement information would help implementing video rate real-time Meanshift filtering in the future.

5.3.2. Meanshift Algorithm

The Meanshift algorithm is a non-parametric clustering algorithm which finds the cluster belonging of each feature point in the feature space. The Meanshift algorithm clusters the feature points by computing a Meanshift vector (MSV) of each feature point. The Meanshift vector of a feature point points toward the cluster center that this feature point belongs to.

In image filtering, the Meanshift vector **m** at a point **x** in the feature space can be computed by

$$\mathbf{m}(\mathbf{x}) = \frac{\sum_{i \in w(\mathbf{x})} g(\mathbf{x}_i^{color} - \mathbf{x}^{color}) k(\mathbf{x}_i^{space} - \mathbf{x}^{space}) \mathbf{x}_i}{\sum_{i \in w(\mathbf{x})} g(\mathbf{x}_i^{color} - \mathbf{x}^{color}) k(\mathbf{x}_i^{space} - \mathbf{x}^{space})} - \mathbf{x}\,, \qquad (3)$$

where $\mathbf{x_i}$ are points within the color range h_c and spatial range h_s of point **x**, the superscript *color* and *space* represents the color and spatial subspace components of a point, and $g()$ and $k()$ are weight functions based on Euclidean color and spatial distance respectively. If $g()$ and $k()$ are uniform and linearly increasing respectively, the first part in eq. (1) is similar to finding the mass center of the space defined by h_c, h_s, and $\mathbf{x}$. We call the first part of eq. (1) as the *temporary mode* and the spatial window defined by hs as the *Meanshift window*. The vector difference between the new temporary mode and the original point is the *Meanshift vector*. We call the computation of a Meanshift vector described above as a *Meanshift search iteration*. The Meanshift algorithm takes the new temporary mode as the center of the next Meanshift window. If the length of a Meanshift vector is smaller than a very small threshold and stops to decrease further, the Meanshift vector is said to converge. The temporary mode of a converged Meanshift vector is a *final mode*. The pixels with their Meanshift vector pointing toward the same mode are part of the same cluster, and are assigned with the same final mode's color in Meanshift filtering.

Supposedly, each pixel in an image should be associated to a mode through the Meanshift search session described above. However, the pixels or modes that are close enough to a mode are very likely to be associated to this mode. Using this property, a pixel that is within very small color and spatial distances to a mode is directly associated to this mode and no Meanshift search is performed. On the other hand, if a temporary mode is within this very small color and spatial distances to a final mode, the pixels associated to this temporary mode is changed to associate to this final mode. As a result, the subsequent Meanshift search iterations in this Meanshift search session can be skipped. We call this operation as the *basin collection*. With the basin collection, both the number of Meanshift search iteration and session can be reduced, thus increasing the speed of the Meanshift search.

For more details on Meanshift filtering algorithm, please refer to [59][60].

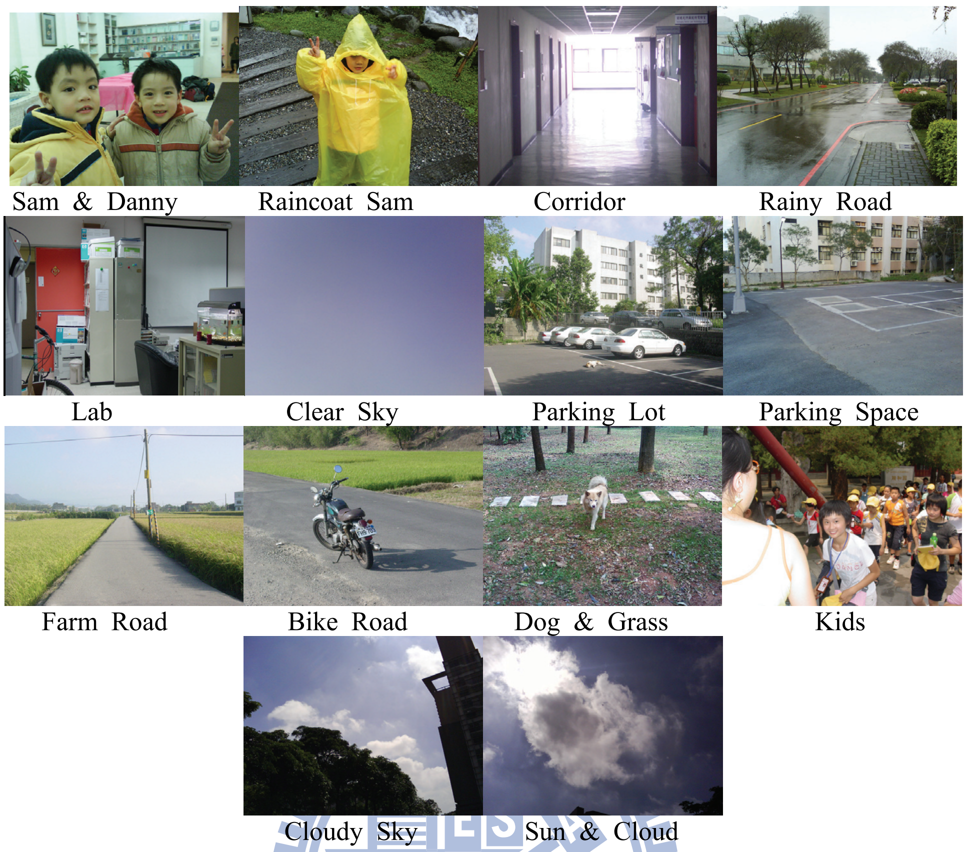


Fig. 32. VGA test images

5.3.3. Test Images and Mean Shift Filter Parameter Settings

Fig. 32 shows the VGA test images used in this work. These images include simple and complex indoor and outdoor scenes. We believe these scenes are common to real-time applications such as human recognition, robot navigation, and autonomous vehicles. Test image "clear sky" is the simplest image which has the minimal number of regions. Test image "dog & grass" is the most complex image that has the maximal number of regions.

This parameter setting described as follows is used in this work. The spatial and color range of the mean shift filter are 7 and 6 respectively. The color range used in basin collection is 3. The mean shift search iteration limit is 100.

5.3.4. Meanshift Architecture

A. Architecture Design Challenges

The nature of the Meanshift algorithm gives rise to three main architecture design challenges: the huge amount of data read from the image memory, the large storage requirement, and the limited parallelism.

The huge amount of data read limits the processing speed of the Meanshift algorithm. In a direct implementation, the pixels within a Meanshift window should be read and ready in the beginning of each Meanshift search iteration. Otherwise, the computation cannot be started and thus the processing speed is limited. For example, the amount of data read requirement for a VGA sized image with h_c and h_s being 6 and 7 respectively can reach up to 97.56 million pixels, which is about 372.19 MB if each pixel is 32-bit wide. If the data port width to the image memory is only 32-bit, 97.56 million cycles would be spent on only reading the input data. Therefore, it is necessary to reduce the amount of data read to improve the processing speed.

The second challenge is the large storage requirement. The Meanshift algorithm needs an input image memory, an output filtered image memory, and a *Meanshifted status* memory which keeps the record of whether a pixel is associated to a mode or not. The size of these memories is proportional to the image size and would be very large for VGA size images. If it is possible to reduce the data width of these memories, the cost of the storages in the Meanshift filter architecture can be reduced.

The limited parallelism is a result of the iterative Meanshift search process. The next Meanshift search iteration cannot start before the previous Meanshift search iteration ended. This is because the Meanshift window center of the new iteration is determined by the temporary mode computed from the previous iteration. Such dependency that originated from the algorithm's iterative nature makes it difficult to be accelerated by using parallel

Meanshift search hardware. Therefore, it is necessary to exploit the available parallelism within each Meanshift search iteration to increase the processing speed.

B. Architecture Overview

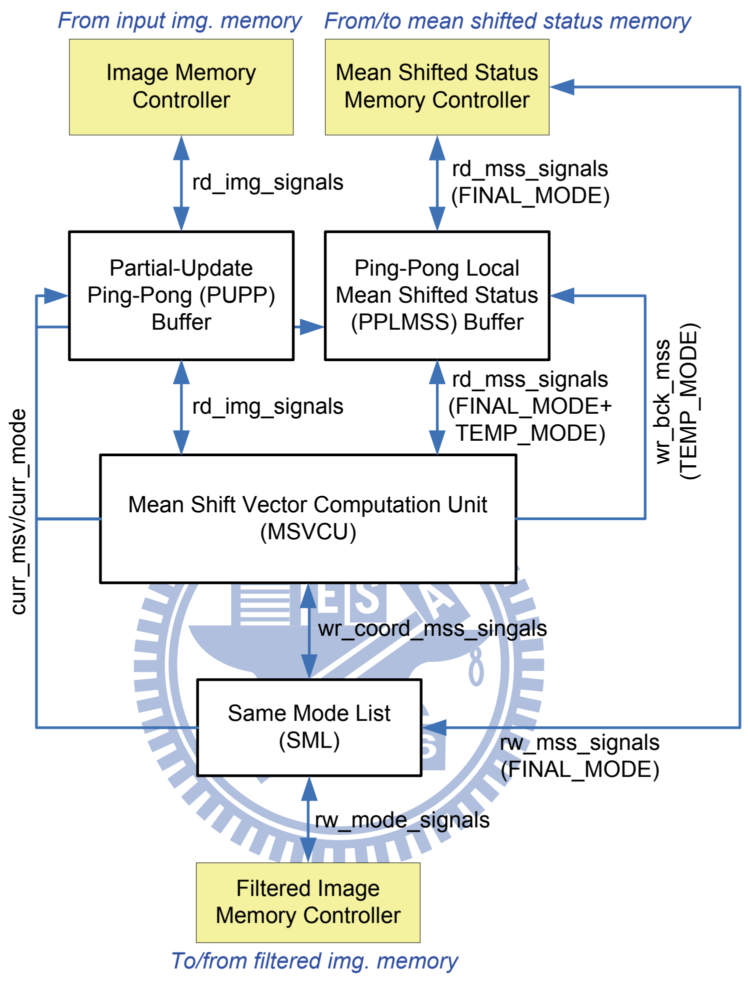

From input img. memory
From/to mean shifted status memory
Image Memory Controller
Mean Shifted Status Memory Controller
rd_img_signals
rd_mss_signals (FINAL_MODE)
Partial-Update Ping-Pong (PUPP) Buffer
Ping-Pong Local Mean Shifted Status (PPLMSS) Buffer
rd_img_signals
rd_mss_signals (FINAL_MODE+ TEMP_MODE)
wr_bck_mss (TEMP_MODE)
curr_msv/curr_mode
Mean Shift Vector Computation Unit (MSVCU)
wr_coord_mss_singals
Same Mode List (SML)
rw_mss_signals (FINAL_MODE)
rw_mode_signals
Filtered Image Memory Controller
To/from filtered img. memory

Fig. 33. Block diagram of the proposed Meanshift filter architecture

Fig. 33 illustrates the block diagram of the proposed Meanshift filter architecture. The core architecture consists of a partial-update ping-pong buffer (PUPPB), ping-pong local Meanshifted status buffer (PPLMSSB), Meanshift vector computation unit (MSVCU), and same mode list (SML). In which, the PUPPB reuses the data in the Meanshift window to reduce the amount of data read from the image memory. The PPLMSSB stores part of the Meanshifted status internally in small local ping-pong buffers to reduce the storage size. This reduces the size of the Meanshifted status memory by 50%. Finally, to increase the processing speed, the parallelism in the Meanshift vector computation and basin collection is exploited by the 4-pixel parallel MSVCU and the SML. The details of these components are presented in the following sub sections.

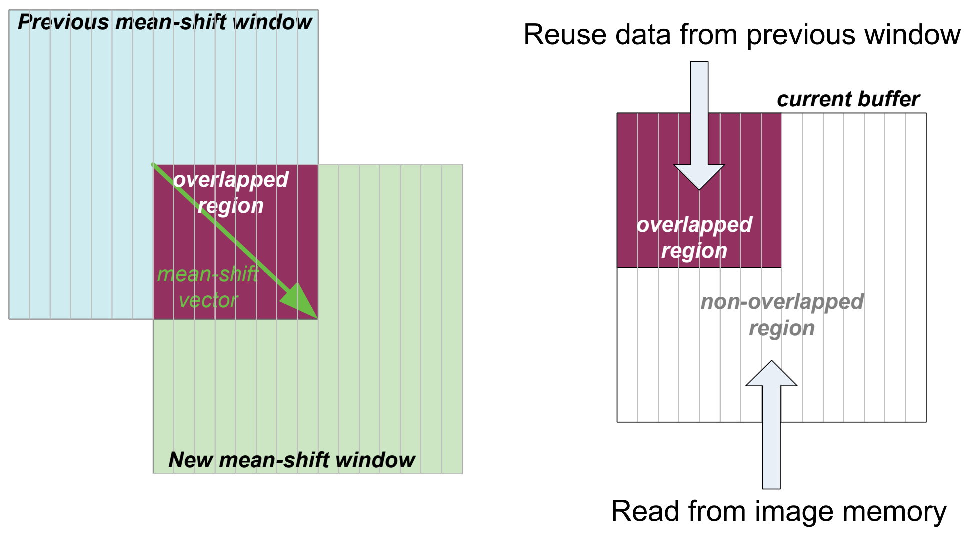
The work flow of the proposed Meanshift filter is explained as follows. The Meanshift filter processes each pixel in an image in raster-scan order. Before finding the mode of a pixel, the Meanshifted status of this pixel is checked first. The Meanshifted status of this pixel is read from the Meanshifted memory and the PPLMSSB. If the current pixel already has a mode, the next pixel in the image is processed. For each pixel without a mode, the pixel data and the Meanshifted status in the current Meanshift window are read into the PUPPB and PPLMSSB.

Once the pixel data in the Meanshift window are ready, the MSVCU computes the new Meanshift vector and the new mode. The MSVCU also performs basin collection. The image coordinates and updated Meanshifted status of the pixels belonging to the same mode are stored in the SML. Note that the Meanshifted status in the PPLMSSB is also updated during the basin collection. At the end of each MSV computation, the MSVCU also checks the MSV's length for convergence.

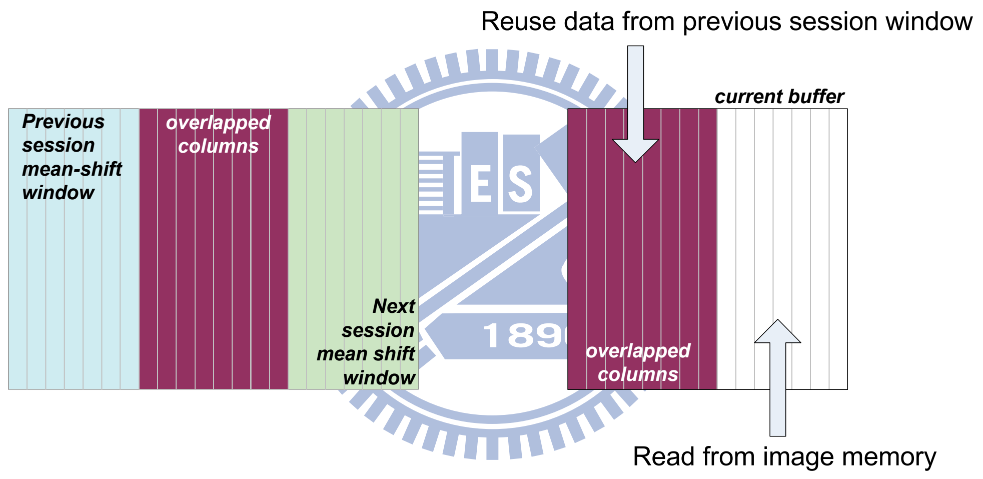
If the MSV converges, all the pixels in the SML are written back to the filtered image memory with the value of the current converged mode. Their corresponding Meanshifted status in the Meanshifted status memory is also updated. Then the next pixel in the image will be processed. If the MSV did not converge, the temporary mode of this MSV would be the center of the new Meanshift window, and another iteration of MSV computation process would be performed. The Meanshift filter flow ends when the last image pixel is processed.

C. Partial-Update Ping-Pong Buffer (PUPPB)

The PUPPB reuses the image pixels in the overlapped portion of the new and old Meanshift windows to reduce the bandwidth requirement. The PUPPB explores two types of reuse: intra Meanshift search reuse and inter Meanshift search reuse.



(a) Intra Meanshift search reuse


Reuse data from previous session window
current buffer
Previous session mean-shift window
overlapped columns
Next session mean shift window
overlapped columns
Read from image memory

(b) Inter Meanshift search reuse

Fig. 34. Concept of PUPP reuse

Fig. 34 (a) illustrates the concept of the intra Meanshift search reuse. Within a Meanshift search session, the Meanshift window of two consecutive iterations is partially overlapped. The pixel data in the overlapped part are read by the previous iterations. These pixel data can be reused and avoid being read from the image memory again in this iteration. The size of the overlapped part is at least one quarter of a Meanshift window. This is because the maximum Meanshift vector can only point to the corner pixel positions. As a result, at least 25% of the bandwidth can be saved by reusing the pixel data in the overlapped part.

Fig. 34 (b) illustrates the concept of the inter Meanshift search reuse. Given the image pixels are processed in raster-scan order, the Meanshift window of two pixel positions is often overlapped when they are near each other. The pixel data in the horizontal overlapped part between the two Meanshift windows are reused. Only the pixel data in the non-overlapped part are read from the image memory column by column. Note that the distance between the centers of the two windows must be smaller than half the width of a Meanshift window.

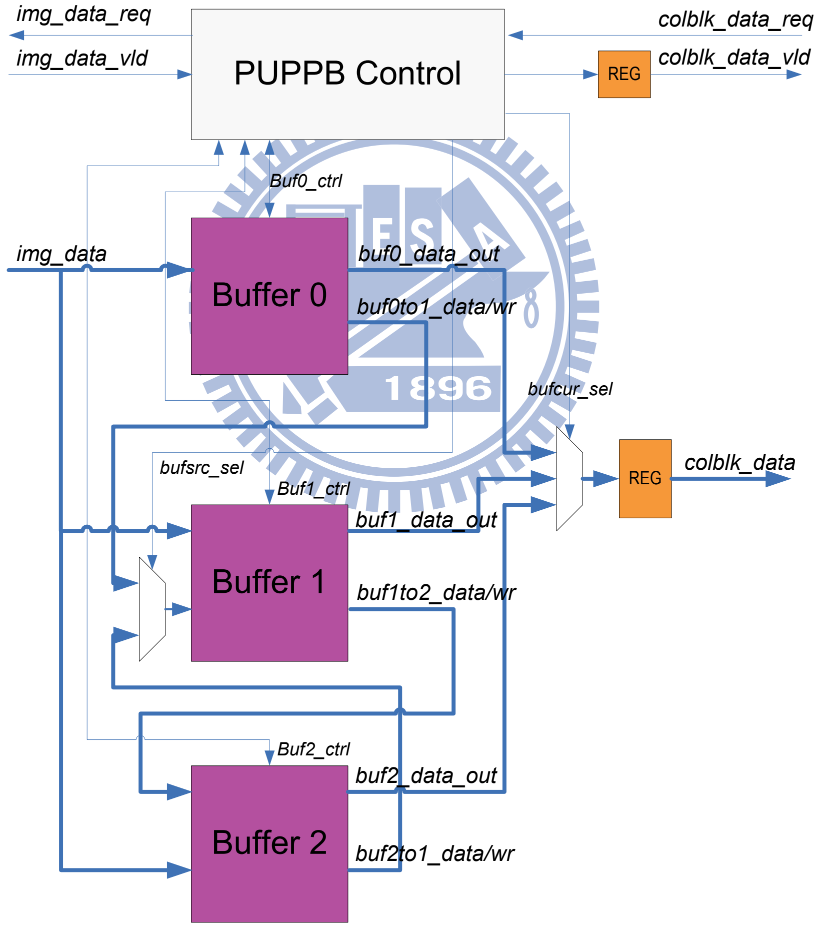


Fig. 35. Block diagram of PUPP

Fig. 35 illustrates the block diagram of the PUPPB and the interconnect among the buffers. The PUPP consists of three buffers. Buffer 0 is used for the inter Meanshift search reuse whereas buffer 1 and buffer 2 are used for the intra Meanshift search reuse.

Buffer 0 is only used in the first iteration of a Meanshift search session. The pixel data from the image memory are written into buffer 0 column-wise. We call this process as "fetch". During the fetch process, each read reads four pixels simultaneously through a 128-bit data port. It takes four reads to read a pixel column, which consists of 15 pixels, in a Meanshift window. The last pixel of the fourth read is not used.

Buffer 1 and buffer 2 are the ping-pong buffers that are used in the iterations after the first iteration. The pixel data of the overlapped part in a Meanshift window are updated from the "old" buffer that holds the data of the previous iteration. The pixel data of the non-overlapped part are read from the image memory. The pixel data from the old buffer or the image memory are written into the "new" buffer that should hold the data for the current iteration. We call this process as the "update" process. The "old" buffer can be any of the three buffers, but the "new" buffer can only be buffer 1 or 2.

The buffers are connected to allow buffer 1 to be updated by buffer 0 or buffer 2, and to allow buffer 2 to be updated by buffer 1. All the buffers have two outputs. One output is for the MSV computation and the other is for the ping-pong update. The data width of the output for the MSV computation is 4 pixel wide because of the 4-pixel parallel design. The data width of the output for the ping-pong update is 15 pixels which correspond to a column in the Meanshift window. The update output of buffer 0 and buffer 2 is multiplexed to buffer 1's overlapped data input. Similarly, the update output of buffer 1 is connected to buffer 2's overlapped data input. In addition to the overlapped data input, all the buffers have an input from the image memory. For buffer 0, this input is for the fetch process; for buffer 1 and buffer 2, this input is for the update process to read the pixel data in the non-overlapped part.

Fig. 36 illustrates the image memory pixel read count and the read count reduction in different test images. The PUPPB can reduce the average image memory read count by 81.6%. For ”dog & grass”, the image memory read count can be reduced by 91.1%. This is because the MSVs in complex images are mostly short. Therefore, the overlapped part is usually large. Consequently, more data can be reused and less data are read from the image memory. The smallest read count reduction happened in ”clear sky”, the reduction is only 50.9%. This is because the MSVs are mostly long in simple scene. Therefore, the overlapped part is usually small. As a result, less data can be reused. This image complexity dependent characteristic of MSVs enables the PUPPB to save more bandwidth in more complex scene.

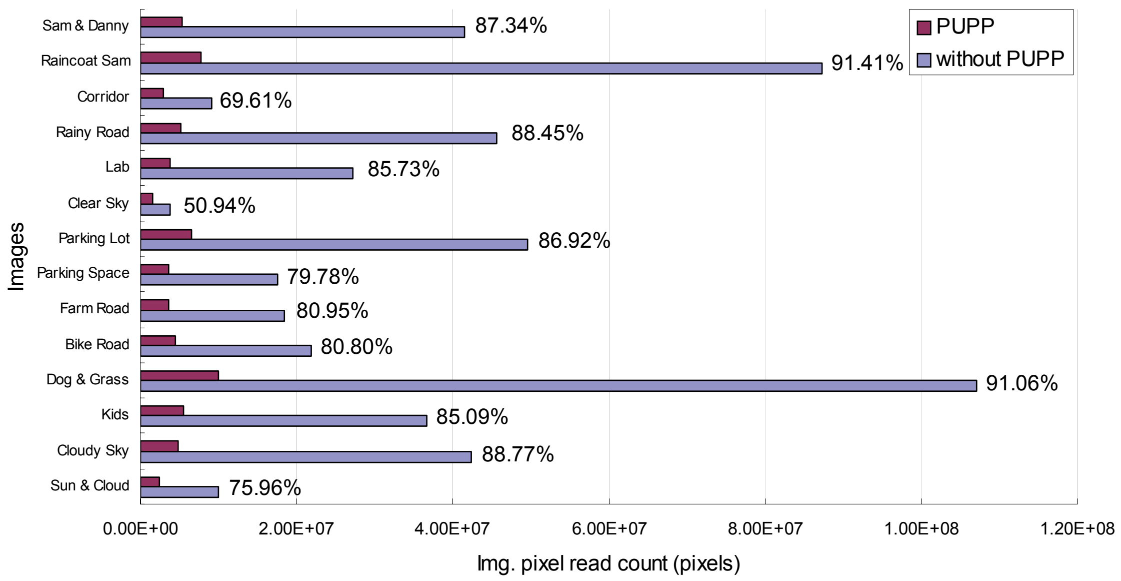


Fig. 36. Image memory pixel read count comparison between using PUPP and not using PUPP

D. Ping-Pong Local Meanshifted Status Buffer (PPLMSSB)

Meanshifted status represents whether a pixel has been associated to a mode or not. If a pixel is already associated to a mode, either a temporary or final mode, we do not need to find its Meanshift vector. The Meanshifted status has three states, which are NO-MODE, TEMP-MODE, and FINAL-MODE. NO-MODE means the current pixel is not associated to a mode yet. A NO-MODE pixel would be associated to a mode through either finding its Meanshift vector or the basin collection. If a pixel is associated to a temporary mode through the basin collection, the Meanshifted status of this pixel is TEMP-MODE. If a pixel is associated with a final mode, the Meanshifted status of this pixel is FINAL-MODE.

At first glance, a Meanshifted status must be represented by at least two bits. One bit represents the TEMP-MODE while the other represents the FINAL-MODE. If both bits are 0, they represent NO-MODE. Since each pixel has a Meanshifted status, storing all the Meanshifted status of a VGA-sized image takes 75.0 KB of memory space, which is quite large in VLSI design. However, we notice that the lifetime of TEMP-MODE bit only lasts through the iterations in a Meanshift search session. Once the final mode is found and the pixels are associated to it, the TEMP-MODE bit of these pixels is no longer needed. Based on this observation, we decided that only the FINAL-MODE bit is stored in the external Meanshifted status memory. When the FINAL-MODE bit is needed, it is read from the memory into a local buffer. The TEMP-MODE bit is stored in a pair of local ping-pong buffers. Doing so reduces the storage requirement from 75.0 KB to only 37.5 KB plus the size of the three local buffers. The buffer size is as large as the Meanshift window size and each pixel is represented by only one bit. As a result, for a Meanshift window of 15x15, only 225 bits are needed for each buffer.

The local ping-pong buffer operates similarly to the PUPPB. The overlapped part is copied from the old buffer to the new one in the same manner as the PUPPB does. However, the non-overlapped part does not need to be updated with the Meanshifted status from the external Meanshifted status memory. Instead, the pixels within the non-overlapped region are updated from the MSVCU if they are associated with a temporary mode through the basin collection.

Fig. 37 illustrates the block diagram of the PPLMSSB. Similar to the PUPPB, the

PPLMSSB consists of three buffers. MSSBuf 0 is used to store the FINAL-MODE bit read from the Meanshifted status memory, whereas MSSBuf 1 and MSSBuf 2 are the ping-pong buffers for the TEMP-MODE bit.

Unlike the buffer 0 in the PUPPB, MSSBuf 0 is independent of the other two because their content is different. The width of the data port connecting to the external Meanshifted status memory is 4-bit, which corresponds to the Meanshifted status of 4 pixels. In contrast, MSSBuf 1 and MSSBuf 2 are connected similar to the PUPPB. However, the write path from the external memory is replaced by the write back path from the MSVCU. The write back path updates the TEMP-MODE bit in the current buffer after the basin collection. The width of the input and output data port connected to the MSVCU is also both 4-bit.

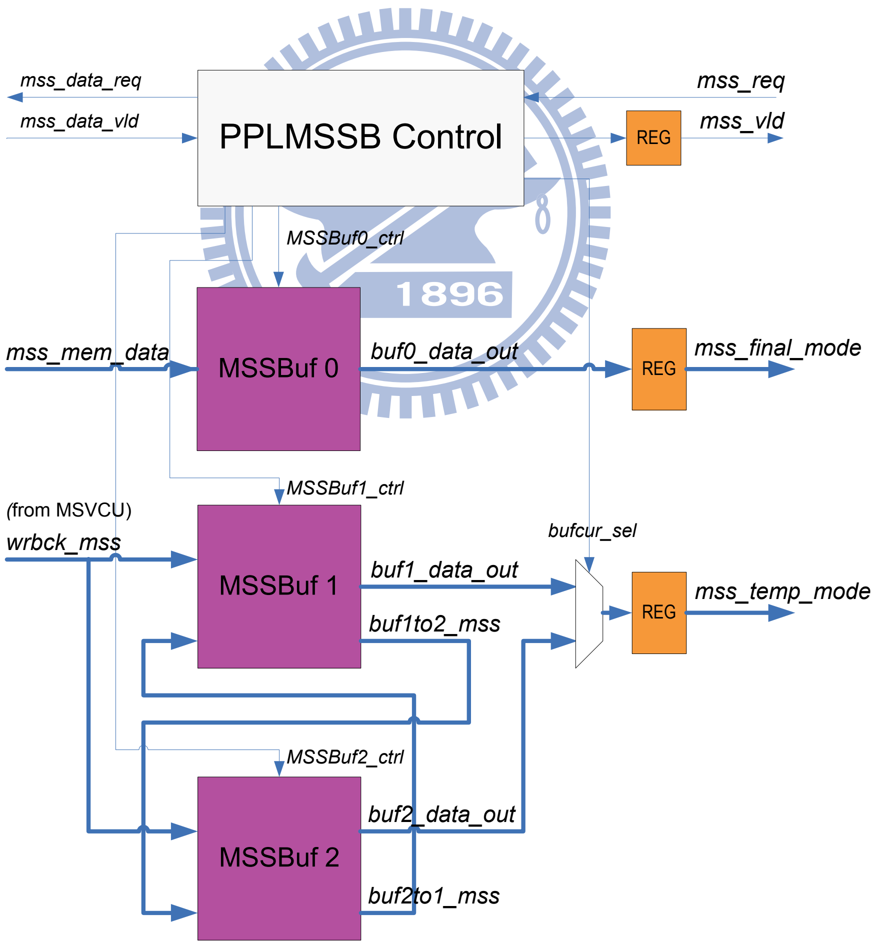


Fig. 37. Block diagram of the PPLMSSB

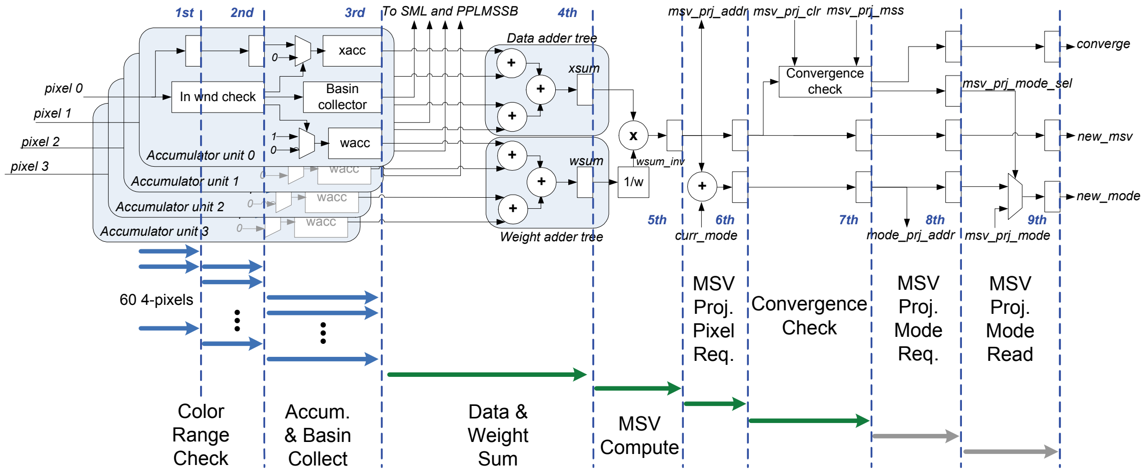


Fig. 38. Datapath and pipeline schedule of the MSVCU

E. MSV Computation Unit (MSVCU)

Fig. 38 illustrates the pipelined datapath and schedule of the MSVCU. The MSVCU is a 4-pixel parallel and 9-stage pipelined architecture. The 4-pixel parallel design enables higher throughput and matches the width of the data port connected to the image memory. Together with the 9-staged pipelined architecture, the computation resource can be better utilized.

In the first and second stage, the pixels are checked if their color is within the color range. If a pixel is within the color range, it would be passed to the next stage and its weight would be one; otherwise, the weight would be zero. This weight assignment reflects a uniform kernel. In addition, this stage also checks if the pixel is within the spatial and color range of basin collection.

The third stage is the accumulation stage. The pixel data and weight are both accumulated in accumulator *xacc* and *wacc* respectively. In addition to the accumulation, the coordinates of the pixels being collected are written to the SML by the basin collector. Note that all the pixels in the 15x15 window are checked and accumulated in the first and second stage before the being processed by the third stage. Since our architecture is 4-pixel parallel, it takes 62 cycles to accumulate all the pixels in a window.

Once the color and weight of all the pixels within the window are accumulated, the accumulated sums from the four parallel accumulator units are added together through a small adder tree. The final sum of pixel data and weight are denoted as *xsum* and *wsum* in the figure.

In the fifth stage, the inverse of *wsum* is approximated by a piecewise-linear model. Then *xsum* is multiplied to the inverse of *wsum* to give the new MSV.

The sixth and seventh stage checks for MSV convergence. The reason for using two stages is to reduce the critical path and account for the one cycle PUPPB and PPLMSSB access latency. The color and final Meanshifted status of the pixel pointed by the MSV projection, which we would refer as the MSV projected pixel, is read from the PUPPB and PPLMSSB. If the color difference between the MSV projected pixel and the new mode is too large, the length of the MSV is checked for convergence. If the color difference is smaller than the color range of the basin collection, the Meanshifted status of the MSV projected pixel is checked. If the MSV projected pixel is not associated to a final mode. In this case, there is only 7 cycles of pipeline delay. If the MSV projected pixel is associated to a final mode, the mode of the projected pixel is the new mode and the Meanshift search session ends. This would introduce two additional pipeline stages to read the mode of the MSV projected pixel from the filtered image memory. As a result, the total pipeline delay would be 9 cycles.

In the case of 9 cycles pipeline delay, the total cycles taken to generate one MSV is 74. This includes one request cycle from the PUPPB and PPLMSSB to send read request to the input image memory and Meanshifted memory, 5 cycles to read the pixels of the first column into the PUPPB and PPLMSSB, 62 color range check and accumulation cycles, and 6 cycles for the rest of the pipeline.

F. Same Mode List (SML)

The SML stores the spatial coordinates of the pixels associated to the same mode during

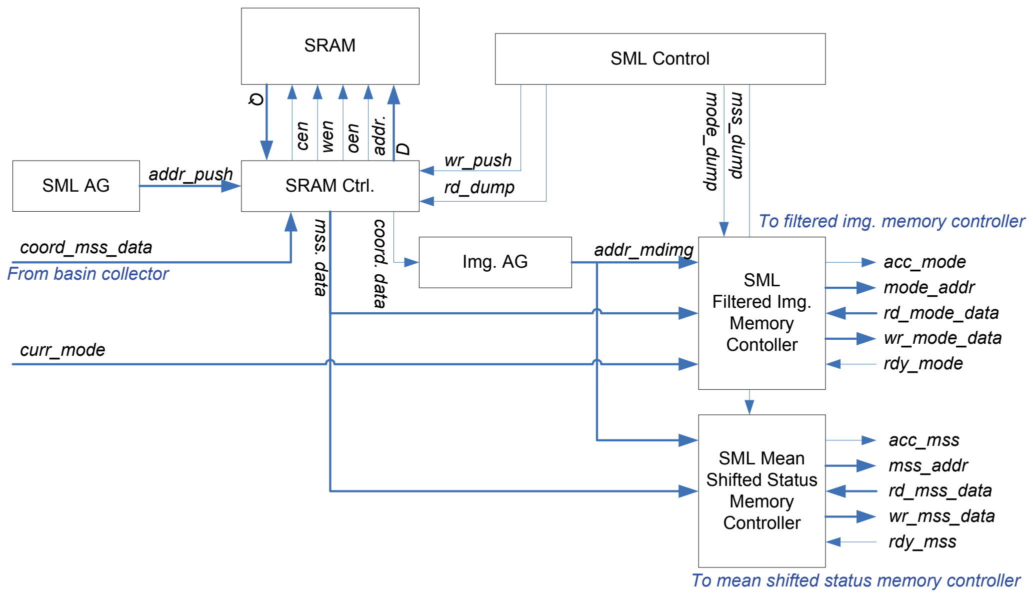


Fig. 39. Block diagram of the SML

a Meanshift search session. Once the final mode is found, all the pixels stored in the SML are written to the filtered image memory with the current mode color and their corresponding Meanshifted status in the Meanshifted memory are also updated. We call this process as the “SML dump” process.

Fig. 39 illustrates the block diagram of the SML. The SML mainly consists of two address generators, a SRAM controller and a SRAM memory, a filtered image memory controller, and a Meanshifted status memory controller. The SML address generator generates the address for accessing the SML memory. The other address generator generates the address for accessing the external filtered image and Meanshifted status memory. The address is generated from the data in the SRAM. Each entry in the SRAM stores the pixel coordinates of the first pixel in a 4-pixel chunk and their updated Meanshifted status. As a result, the width of a word in the SRAM is 23-bit, in which 19 bits are for the image address and 4 bits are for the Meanshifted status. The SRAM has 384 entries and is sufficient when the Meanshift search iteration limit is 100. During the SML dump process, the SML writes 4 pixels to the filtered image and the Meanshifted status memories simultaneously because of the 4-pixel parallel architecture.

5.3.5. Implementation result

The proposed Meanshift filter design is synthesized using UMC 90 nm technology with standard cell and SRAM libraries. The equivalent gate-count, excluding the SRAMs, of the synthesized netlist is 516,533. The equivalent gate-count of each component is listed in Table 14. The PUPPB and PPLMSSB occupy the majority of the gate-count. Despite the control and miscellaneous logics, the SML occupies the smallest portion of the gate-count; however, most of the SML's area is contributed by the SRAM cells. Although the maximum clock rate is 110 MHz, there is still room to increase the clock rate by using more pipeline stages.

Table 14 Synthesized gate count of each component

Component name	Equivalent gate-count	Percentage %
PUPPB	437,064	84.61
PPLMSSB	21,702	4.20
MSVCU	51,016	9.88
SML	5,194	1.01
Control	780	0.15
Misc.	778	0.15
Total	516,533	100.00

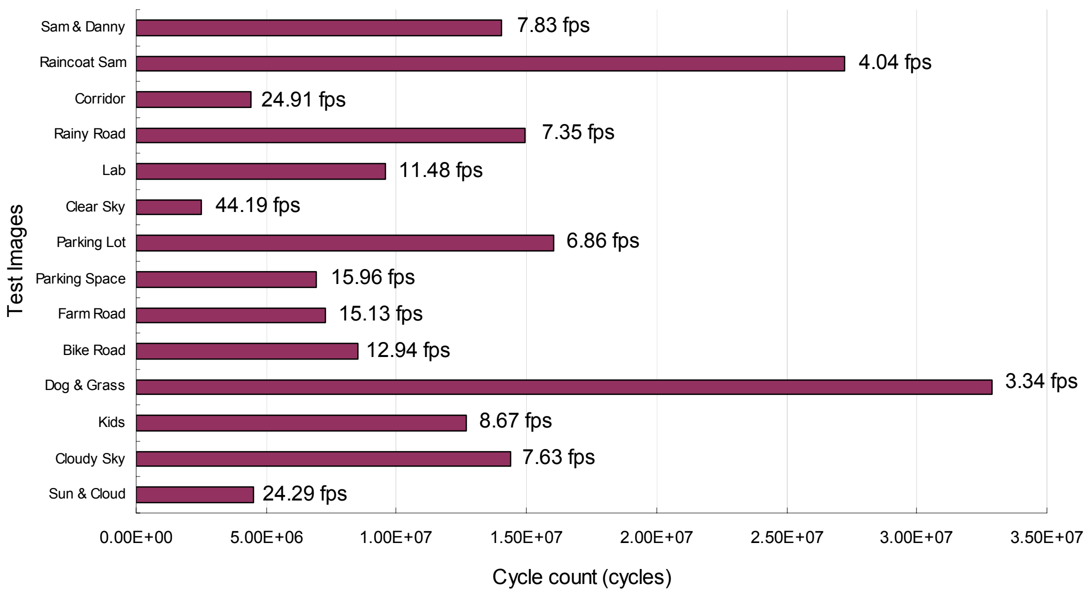


Fig. 40. Processing cycles and estimated frame rate of each test images

Fig. 40 shows the clock cycles needed to filter the test images. The estimated frame rate is also marked in the figure assuming a 110 MHz operating frequency. For the worst case, which is the “dog & grass” case, the total cycle count is 32,914,220. If the clock is 110 MHz, our Meanshift filter can process VGA images at a frame rate of at least 3.34 FPS. The average frame rate for our test images is 8.75 FPS. For QVGA images, the minimal frame rate is 14.20 FPS.

Table 15 lists the specification the proposed Meanshift filter architecture and other existing segmentation implementations. We also include the EDISON software implementation for comparison. Although Park et al.’s dynamic Meanshift filter architecture is the first architecture for the Meanshift algorithm; they only outlined their systolic array architecture and have yet to provide further implementation detail. The internal memory usage of the proposed architecture is the smallest among the compared implementations. Although the throughput of other segmentation algorithm implementations is higher than the proposed Meanshift filter architecture, the segmentation performance of different algorithms

Table 15 Comparison of existing segmentation implementation

Implementation	**Proposed Meanshift Filter**	**Dynamic Meanshift Architecture [69]**	**Real-time K-means Architecture [67]**	**Region Growing Architecture [63]**	**Histogram Peak Climbing Architecture [66]**	**EDISON Software [60]**
Implementation Type	ASIC	FPGA	FPGA	FPGA	ASIC	PC
Max. Processing Region Count	2^{24}	2^{24}	256	260	24,882	2^{24}
Process/FPGA	UMC 90 nm	Not Available	Xilinx Virtex-II XC2V6000	Altera Stratix EP2S180	0.13 um	0.13 um
Operation frequency	110 MHz	Not Available	82.7 MHz	20MHz	7 MHz	1.8GHz
Throughput	*3.34 FPS (VGA) 14.20 FPS (QVGA)	Not Available	35 FPS (VGA)	30 FPS (QVGA)	50 FPS (DVD)	<1 FPS (VGA) <3 FPS (QVGA)
Hardware Cost	517 K gates	Not Available	85% slices	86 K ALUTs	9,379 K gates	~11 M gates
Memory Requirement	1.08 KB (internal) 1.03 MB (external)	Not Available	76% block ram	230 KB (embedded)	2.52 MB (internal)	2.05 MB

* worst case frame rate

is not the same. For instance, Meanshift based algorithm can support up to 2^{24} segments whereas other algorithms can support less than 261 segments except the histogram peak climbing architecture. This means under segmentation is likely to happen in other algorithms. Moreover, the Meanshift algorithm does not rely on initial guess and does not need to give the number of cluster center in advance, making the Meanshift algorithm more convenient to use than K-means algorithms. When compared to the EDISON software, the proposed Meanshift filter architecture can provide a 3 times speedup with relatively much less hardware and lower clock rate.

5.3.6. Summary

In the proposed VLSI Meanshift filter architecture, the partial-update ping-pong buffer reduced the bandwidth to the image memory by 81.6%. In addition, the proposed ping-pong local Meanshifted status buffer reduced the size of the Meanshifted status memory by 50%. Finally, a 4-pixel parallel 9-stage pipelined MSV computation unit is also proposed to exploit the limited parallelism in the Meanshift algorithm. The synthesized gate count of the proposed architecture is 517K. When clocked at 110 MHz, the proposed architecture can perform Meanshift filtering of VGA images at the average frame rate of 8.75 FPS. We anticipate this work to encourage real-time implementations of complex but high performance vision algorithms so that more vision-based real-time application would be possible in the future.

The bandwidth to external memory remained as a speed limiting issue. Therefore, further reducing the bandwidth requirement of the Meanshift algorithm is very important. Another worthy future research topic is to reduce the power consumption. The proposed architecture involved a lot of buffer and memory accesses which would make the architecture very power consuming. Therefore it is necessary to explore low power Meanshift architecture in the future.

5.4. MCADSW

5.4.1. Stereo Matching's Role in Vision Applications

Stereo vision is an important early vision tool that has been widely adopted by applications such as multiview video coding, freeview TV, 3D video conferencing, intelligent surveillance, autonomous vehicles, and mobile robots. Stereo vision finds the depth in a scene based on the stereo image pair of the scene. The depth of a pixel is inversely proportional to the disparity of this pixel. The disparity of a pixel is the distance of this pixel and the corresponding pixel in the other image. The process of finding the corresponding pixel is often referred as disparity estimation or stereo matching. The resulting disparity of each pixel in an image forms a disparity image or disparity map. For more detail on stereo vision, please refer to [71].

5.4.2. Stereo Matching Issues

The applications adopting stereo vision often require high performance and real-time processing speed. The performance is usually defined by the error rate of a disparity map when compared to the ground truth disparity map [72]. Lower error rate implies higher performance. Complex disparity estimation algorithms usually achieve much better performance than simple algorithms. However, simple algorithms are usually much faster than complex algorithms. As a result, most real-time applications have adopted simple algorithms to trade the performance for speed. For applications that cannot accept trading performance for speed, complex algorithms have been adopted and implemented using powerful computation devices such as DSPs, GPUs, and dedicated hardwares. However, the computation power of DSPs is not high enough to enable complex disparity estimation algorithms to support real-time processing. The GPUs can support real-time disparity estimation, but is too expensive for embedded real-time applications. The dedicated hardware approach that uses FPGAs/ASICs can provide high computation power with relatively less expensive hardware cost. This makes the dedicated hardware approach suitable for implementing complex disparity estimation algorithms for real-time applications. However, complex disparity estimation algorithms are often not hardware-friendly and bandwidth hungry.

5.4.3. MCADSW algorithm

A. Algorithm Overview

Fig. 41 shows the overall flow of the proposed mini-census adaptive support weight (MCADSW) algorithm. The MCADSW algorithm consists of four major steps. First, the mini-census transform and matching step performs mini-census transform on the left and right images and computes the initial matching cost of each pixel. The second step is the weight generation which generates the weight coefficients needed in the cost aggregation step. Once both the initial matching cost and weight coefficients are available, the matching cost will be aggregated through a two-pass cost aggregation step. Finally, the best disparity can be obtained by finding the disparity with the minimum aggregated matching cost through a Winner-Takes-All method.

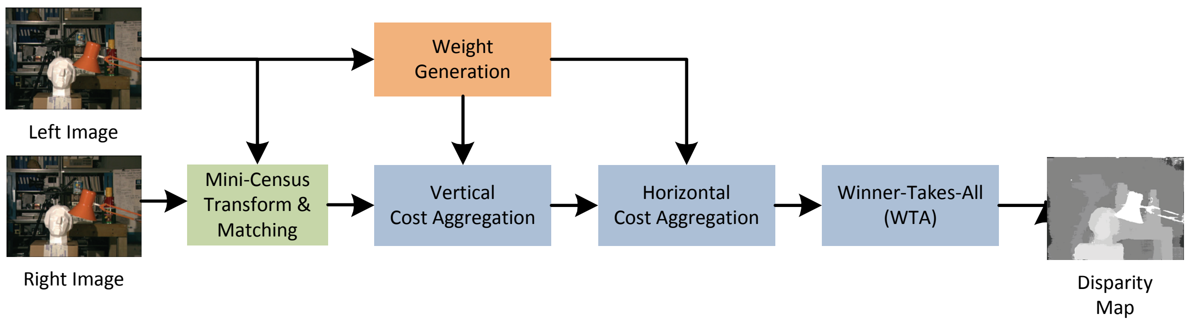

Left Image
Right Image
Weight Generation
Mini-Census Transform & Matching
Vertical Cost Aggregation
Horizontal Cost Aggregation
Winner-Takes-All (WTA)
Disparity Map

Fig. 41. Overall flow of the proposed mini-census adaptive support weight algorithm

B. Mini-Census Transform & Matching

The mini-census transform, a modified and simplified version of census transform [87], compares the luminance of the six pixels within a support window with the center pixel. The six pixels template is marked in Fig. 38. If a pixel's luminance is larger than the center pixel's luminance, it is given the label 0, otherwise the label 1. After the comparison of the six pixels, a binary bitstream is obtained which characterizes the luminance relation between the center pixel and its surrounding six pixels. With the mini-census bitstream, we can represent each pixel using only 6-bit.

The mini-census matching cost between two pixels is defined as the hamming distance between the mini-census bitstreams. We would refer the mini-census matching cost as the census cost hereon for brevity, which is defined as

$$E_{i,d} = H(b_{L,i}, b_{R,i,d}), \tag{4}$$

where $E_{i,d}$ is the census cost of pixel i at disparity d; $b_{L,i}$ is the bitstream of pixel i in the left image and $b_{R,i,d}$ is the bitstream of pixel i at disparity d in the right image; H is the hamming distance function.

Fig. 42 illustrates an example of the mini-census transform and census cost. After the transform, the mini-census bitstreams of the two pixels in the figure are 111000 and 111011

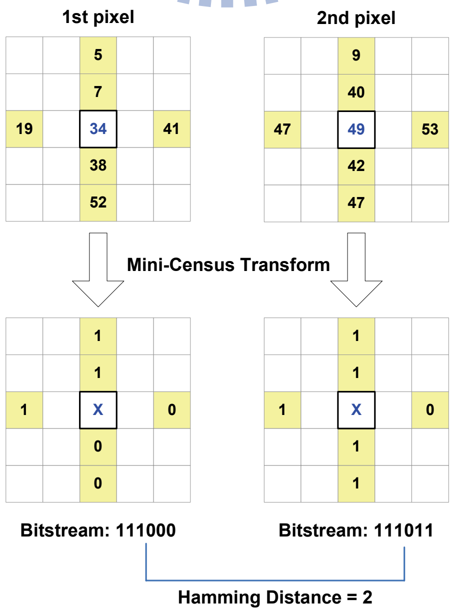


Fig. 42. The census transform and matching

respectively. The hamming distance between the bitstreams is 2; hence, the census cost is 2.

Since the bitstream represents relative information, the census cost is therefore much less sensitive to brightness bias and exposure gain. In addition, the census cost preserves the depth boundary in disparity maps better than the traditional SAD cost does.

C. Weight Generation

The adaptive weight generation was based on the proximity and color distances in the original adaptive support weight algorithm. However, we removed the proximity weight based on our observation that it mainly benefits the performance when the support window is larger than 19x19. We have compared the average disparity error rate of MCADSW with and without the proximity weight. The average disparity error rate was averaged over the overall error rate of tsukuba, venus, teddy, and cones stereo image pairs from the Middlebury stereo vision evaluation website [72]. Note that we have decided to use the overall error rate instead of the non-occlusion error rate because in most applications, such as robotics, freeview TV, and 3D modeling, prefer overall error rate over non-occlusion error rate. This is due to the fact these application do not just rely the depth at non-occlusion, the depth at depth discontinuities is also critical to the performance of these applications. From our experiment result, the difference of the disparity estimation error rate between using and not using the proximity weight was less than 3% for window size of 31x31. Therefore, we decided to trade the very small performance loss for the computation complexity reduction.

The color weight was originally defined as a Gaussian function of the color distance between a pixel i in the support window and the center pixel c of the window. The color weight $w_{i,c}$ of pixel i with respect to pixel c is

$$w_{i,c} = \exp(-\frac{\Delta C_{i,c}}{\gamma_c}), \tag{5}$$

where $\Delta C_{i,c}$ is the color distance between pixel i and c; γ_c is a tuning constant. This weight allows the pixel with color similar to the center pixel to have more influence on the final matching cost. Note that since we use two-pass aggregation, the weight for vertical and horizontal aggregations are generated separately. The vertical weight is generated with respect to the center pixel of each column in the support window, whereas the horizontal weight is generated from the center row with respect to the center pixel of the support window.

To reduce computation complexity and make the algorithm more hardware-friendly, we proposed three simplifications to the weight generation.

First, we adopted the YUV color representation instead of the L*a*b* color representation, which was originally adopted by the ADSW, in our MCADSW. Using YUV color representation allowed us to use hardware-friendly positive integer numbers instead of complex hardware-unfriendly signed floating-point numbers during the weight generation. Fig. 43 shows the average error rate of MCADSW using different color representations. The error rate was averaged over the overall error rates of the four stereo image pairs from the Middlebury stereo vision evaluation website [72]. From Fig. 3 we noticed that the error rate difference between using YUV and L*a*b* color representations was less than 2%. The error rate in the case of using Y-only and RGB color representations was significantly larger than using L*a*b*. Therefore, we have decided to adopt YUV color representation in MCADSW.

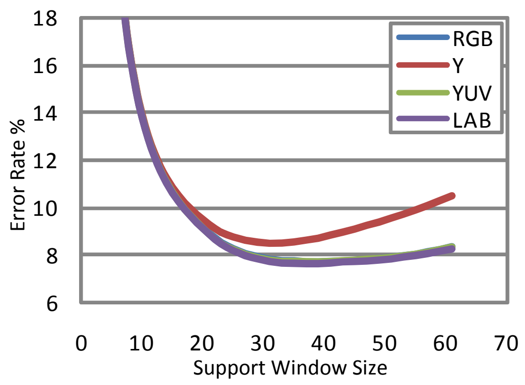


Fig. 43. Disparity estimation error rate of using different color spaces

Second, we used Manhattan color distance instead of Euclidean color distance to eliminate the three square and one square root computations that was necessary for computing Euclidean distance. Table 16 lists the disparity estimation error rate and execution time on a Pentium IV 2.8GHz machine when Euclidean and Manhattan distances were used in MCADSW. Interestingly, the result using Manhattan color distance slightly outperformed using Euclidean color distance. One possible explanation is that YUV color representation is not perceptually uniform and linear like L*a*b* color representation is. Therefore, Euclidean color distance could no longer reflect the actual color distance. After using Manhattan color distance, the execution time of the MCADSW was reduced by 34.3%.

Table 16 Performance comparison between using Euclidean and Manhattan color distances

Method	**Average Error Rate (%)**	**Error Rate (%)**				**Tsukuba Execution Time (seconds)**
		TSUKUBA	**VENUS**	**TEDDY**	**CONES**	
Euclidean	7.47	3.47	0.91	14.3	11.2	4.75
Manhattan	6.94	3.08	0.59	14.0	10.1	3.12

Third, we proposed a scale-and-truncate approximation of the color weight function. The scale-and-truncate approximation approximated the exponential function by scaling it up by 64 then truncate it to leave only one non-zero most significant bit (MSB). The reason for scaling up by 64 is because the error rate stopped decreasing after the scale factor exceeds 64. The reason for preserving only one non-zero MSB is because the error rate difference

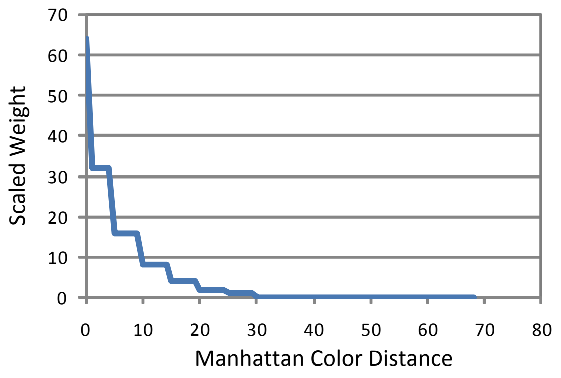


Fig. 44. Scaled, truncated, and non-zero MSB preserved weight function

between preserving one bit and two bits was less than 0.5%. With only one bit in the color weight being one, the multiplication between the color weight and the initial cost can be implemented with only one simple shift operation. The curve of the final scale-and-truncated weight is shown in Fig. 44. After applying the proposed approximation, the execution time could be reduced by 41.0%.

D. Vertical and Horizontal Cost Aggregation

The final matching cost was aggregated from the weighted cost within the support window using a two-pass approach proposed by Wang et al. [109]. The two pass approach first aggregates vertically to give a vertical aggregated cost of each column, then the vertical aggregated costs are aggregated horizontally together to give the final matching cost. The vertical aggregated cost $E_{v,col,d}$ of a column *col* within the support window at disparity *d* can be defined as

$$E_{v,col,d} = \sum_{i \in col} E_{i,d} w_{v,i,c}, \tag{6}$$

where $E_{i,d}$ is the initial census cost of the pixel *i* in the column, $w_{v,i,c}$ is the vertical weight of pixel *i* with respect to the center pixel *c* of the column. The horizontal aggregated cost $E_{h,row,d}$ at disparity *d*, which is also the final matching cost $E_{finall,d}$, is defined as

$$E_{final,d} = E_{h,row,d} = \sum_{col \in W} E_{v,col,d} w_{h,j,c}, \tag{7}$$

where $E_{v,col,d}$ is the vertical aggregated cost of the column *col* within the aggregation window *W*; $w_{h,col,c}$ is the horizontal weight of column *col*'s center pixel *j* with respect to the center pixel *c* of the window.

The two-pass approach can reduce computation complexity when compared to the direct approach. If the window size is $(r+1)$ x $(r+1)$ and the disparity range is *D*, the complexity of the two-pass approach is $O(2rD)$, whereas the complexity of the direct approach is $O(r^2D)$. In addition to the computation complexity reduction, the two-pass approach also reduces the internal bandwidth of the hardware design. However, [109] have reported observable quality drop of the disparity map after applying the two-pass approach to the original ADSW. Our own experimental result on the ADSW showed approximately less than 3% average error rate increase after applying the two-pass approach.

E. Performance Evaluation

Table 17 lists the error rate and desktop PC execution time of the proposed MCADSW and other state-of-the art algorithms. The error rate evaluates the performance of a disparity estimation algorithm and is independent of which computing platform an algorithm is being implemented. The execution time evaluates the computing complexity of an algorithm based on desktop PC-based implementations. Although the execution time of an algorithm depends on the clock rate of the target processor and code optimization, comparing the execution time of different algorithms gives a rough figure of their computation complexity. The error rate and the execution time of other algorithms in the table were acquired from their published works. For ADSW, we have included the execution time provided from their published work as well as the execution time acquired from us running their software. The error rate and execution time we gathered is labeled as "Our ADSW". For algorithms that did not provide the execution time of the tsukuba stereo image pair, such as RealTimeBP and RealTimeGPU, we estimated their execution time based on their best disparity estimation speed, which is usually represented in terms of *million disparity estimation per second* (MDE/s). For instance, RealTimeGPU reported a disparity estimation speed of 0.36 MDE/s on an Intel Pentium IV 3GHz machine in their work; by dividing the number of disparity estimations needed in the tsukuba stereo image pair, which is 384x288x16 = 1.77 MDE, by the disparity estimation speed, we get an estimated execution time of 4.91 seconds.

The error rate of the proposed MCADSW was comparable to RealTimeBP and was slightly inferior to the original ADSW. When compared to the state-of-the-art CoopRegion [118] method, the error rate of the MCADSW was 0.46%~4.67% higher. However, the execution speed of the MCADSW was at least 10 times faster than CoopRegion, 30 times faster than the original ADSW, and near 2 times faster than the RealTimeBP. This implies the MCADSW is likely to have a much lower computation complexity than the compared high performance disparity estimation algorithms. Only SSF+MF, EffectiveAggr, and RealDP were faster than the MCADSW. However, these algorithms had significantly higher error rate than the MCADSW as well.

Table 17 Performance comparison of the MCADSW and other algorithms

Method	**Error Rate %**				**PC Spec. (Brand, processor, clock rate)**	**Tsukuba Exec. Time(sec)**
	TSUKUBA	**VENUS**	**TEDDY**	**CONES**		
SSD+MF[72]	7.07	5.16	24.8	19.8	Intel CoreDuo 2.99 GHz	0.64
EffectiveAggr[116]	2.11	4.75	15.2	12.6	Intel CoreDuo 2.14GHz	0.20
RealDP[105]	2.85	6.42	N.A.	N.A.	AMD AthlonXP 2800+	0.02
RealTimeBP[113]	3.40	1.90	13.2	11.6	Intel Pentium IV 3.0G	3.39
RealTimeGPU[109]	4.22	2.98	14.4	13.7	Intel Pentium IV 3.0G	4.91
CoopRegion[118]	1.13	0.18	9.03	7.80	Intel Pentium M 1.6G	~20.00
Original ADSW[73]	1.85	1.19	13.3	9.79	AMD AthlonXP 2700+	~60.00
Our ADSW	4.18	3.41	20.6	16.0	Intel Pentium IV 2.8GHz	95.65
MCADSW	2.80	0.64	13.70	10.1	Intel Pentium IV 2.8GHz	1.84

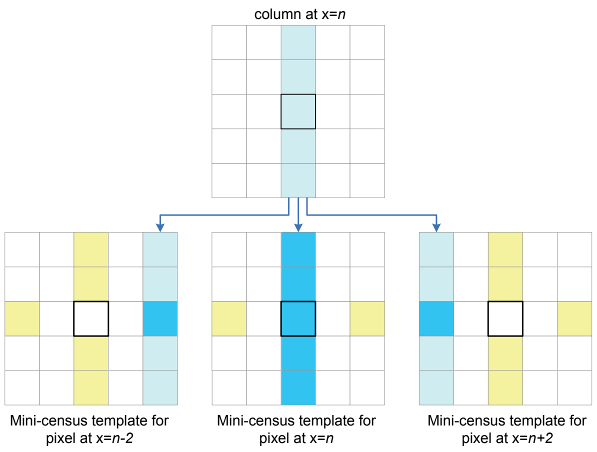
5.4.4. Bandwidth Reduction Techniques for MCADSW Architecture

Reducing bandwidth requirement is important because available bandwidth is limited. We proposed *partial column reuse* (PCR) and *access reduction with expanded window* (AREW) techniques to reduce the bandwidth requirement.

A. Partial Column Reuse (PCR)

The PCR reuses the data in each column to reduce the memory bandwidth and computation requirements. A column is usually a part of multiple horizontally overlapped windows. Therefore, the data of each column can be shared by the computation of the final result for these windows. The data could be original pixel data or temporary intermediate results. By storing these data, the number of memory access and computation can be reduced. As a result, each column is only read and computed once.

Fig. 45 illustrates how the PCR is applied to the mini-census generation. The pixels in column x=n contribute to the generation of three mini-censuses. Fig. 46 illustrates how a vertical aggregated cost is shared by 18 horizontally overlapped aggregation windows. This reduced the read count of a pixel column from 18 to 1 for these 18 windows. Since PCR can reuse computation as well, PCR may also be applied to other types of implementations, such


column at x=*n*
Mini-census template for pixel at x=*n-2*
Mini-census template for pixel at x=*n*
Mini-census template for pixel at x=*n+2*
Fig. 45. Partial column reuse in mini-census transform

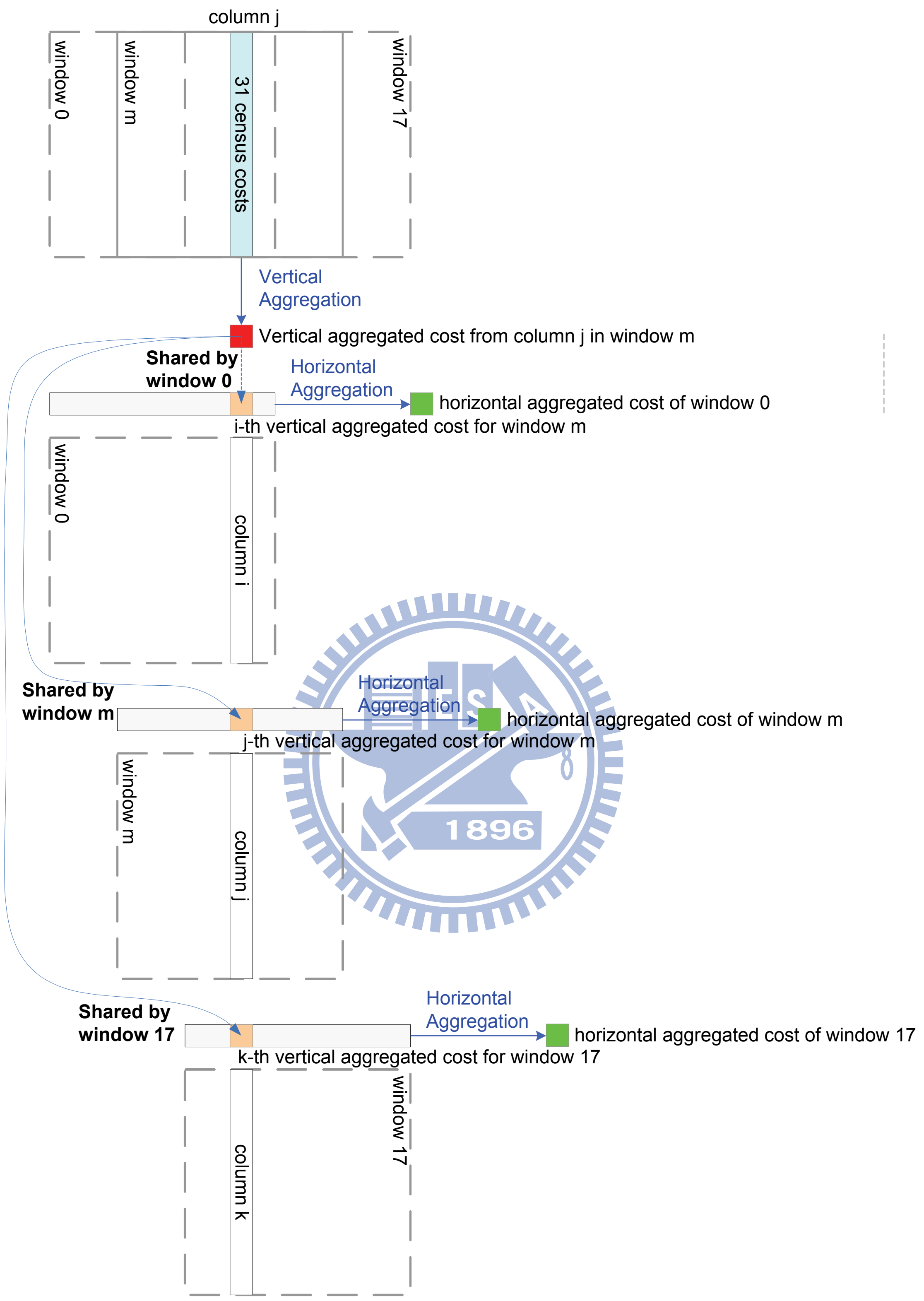

column j
window 0
window m
31 census costs
window 17
Vertical Aggregation
Vertical aggregated cost from column j in window m
Shared by window 0
Horizontal Aggregation
horizontal aggregated cost of window 0
i-th vertical aggregated cost for window m
window 0
column i
Shared by window m
Horizontal Aggregation
horizontal aggregated cost of window m
j-th vertical aggregated cost for window m
window m
column j
Shared by window 17
Horizontal Aggregation
horizontal aggregated cost of window 17
k-th vertical aggregated cost for window 17
column k
window 17

Fig. 46. Partial column reuse in cost aggregation

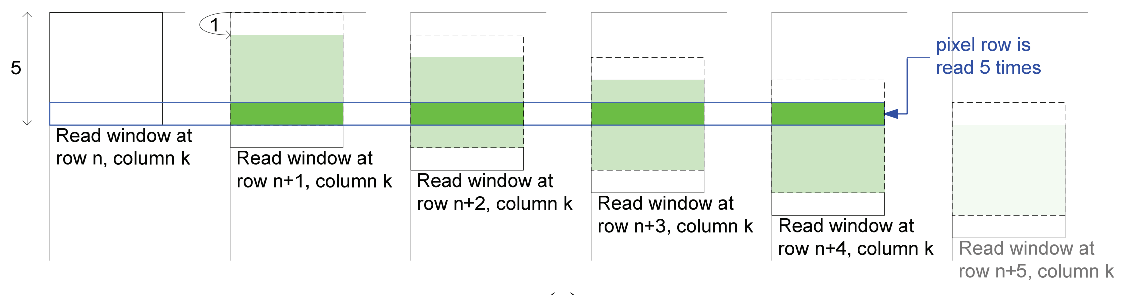
as processor-based, DSP-based, GPU-based, and FPGA-based, to reduce computation requirement.

B. Access Reduction with Expanded Window (AREW)

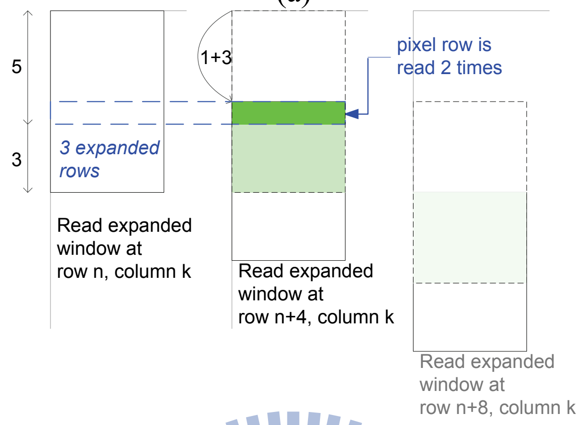
The AREW reduces the bandwidth requirement by deliberately expanding the size of the read window. The expanded window reduces the read count of a pixel by reducing the number of overlapping window containing this pixel. We will explain this using an example of vertically expanded window shown in Fig. 43. Note that we have ignored considering horizontal overlap for the sake of clarity. In this example, the original window size is 5x5 pixels and the number of vertical expanded row is 3.

Fig. 47 (a) illustrates how windows are overlapped vertically without expanding rows. The first window is located at row *n* and column *k*. When the window changes the row position at the end of a horizontal scan, the new window would be vertically overlapped with the old window. As a result, the second window is located at row *n+1* and column *k*. The position of the first window is shown by the box with dashed line. The overlapped region is marked by darker color. Since we only buffer the pixel data within the read window due to cost consideration, the vertically overlapped region must be re-accessed. Consequently, the access count of a pixel is determined by the number of overlapping window containing this pixel. The maximal access count of a pixel is five in this case as shown in the figure.

Fig. 47 (b) illustrates the case with expanded window. With the expanded rows, the vertically enlarged window would result in farther vertical jump distance when a row change happens. As a result, the second window is located at n+4. The maximal access count of each pixel is only 2, which is much smaller than in the case without expanded window. Horizontal expansion also reduces the access count in the same way as the vertical expansion. If we could enlarge the window to the size of the image, the read count of each pixel would be only one. However, expanding the window would also require larger internal storage size and more hardware resource. Therefore, the number of expanded row and column should be carefully selected. In our case, the number of expanded row and the number of expanded


5
1
pixel row is read 5 times
Read window at row n, column k
Read window at row n+1, column k
Read window at row n+2, column k
Read window at row n+3, column k
Read window at row n+4, column k
Read window at row n+5, column k

(a)


5
3
1+3
pixel row is read 2 times
3 expanded rows
Read expanded window at row n, column k
Read expanded window at row n+4, column k
Read expanded window at row n+8, column k

(b)

Fig. 47. Example of access count reduction with expanded window, (a) without expanded rows, (b) with 3 expanded rows

column are both 17. The AREW is applied to the mini-census generation and weight generation.

The bandwidth requirement to external frame memory can be estimated based on the read count of each pixel. The read count of a pixel is determined by the number of times it is overlapped by mini-census transform windows and aggregation windows. In a direct implementation without any data reuse, a pixel is overlapped by 3 mini-census transform windows in the horizontal direction, 4 mini-census transform windows in the vertical direction, 31 aggregation windows in the horizontal direction and 31 aggregation windows in the vertical direction. The read data width of a pixel in the mini-census transform is one byte, whereas the read data width of a pixel in the aggregation is three bytes. As a result, the total bandwidth requirement for a CIF size base image at 30 FPS is about ((7x31x31)x1byte+31x31x3byte)x(352x288)x30FPS = 27.22 GB/s. neglecting the boundary case. If we assume the pixel data read for the weight generation already included the pixel data read for the mini-census transform, the total bandwidth can be reduced to (31x31x3byte)x(352x288)x30FPS = 8.17 GB/s. After applying the PCR and AREW bandwidth reduction techniques, the average read count of a pixel can be reduced to 5.17 times. The bandwidth requirement can therefore be reduced to 5.17x3bytex(352x288)x30FPS= 44.99 MB/s. The proposed bandwidth reduction can also be applied to other aggregation based stereo matching architectures to reduce their bandwidth requirement.

5.4.5. Real-time Architecture for MCADSW

A. Architecture Overview

Fig. 48 shows the architecture of the MCADSW. The architecture consists of a memory controller, mini-census transformer, weight generator, and a cost aggregator and WTA

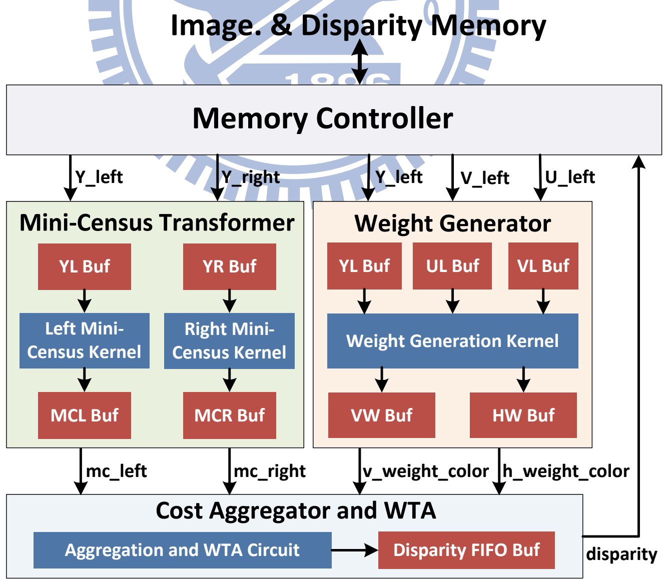


Fig. 48. Block diagram of the MCADSW

module. The details of each module are explained in the following subsections.

B. Mini-Census Transformer

Fig. 49 shows the architecture of one of the two (left and right) identical mini-census

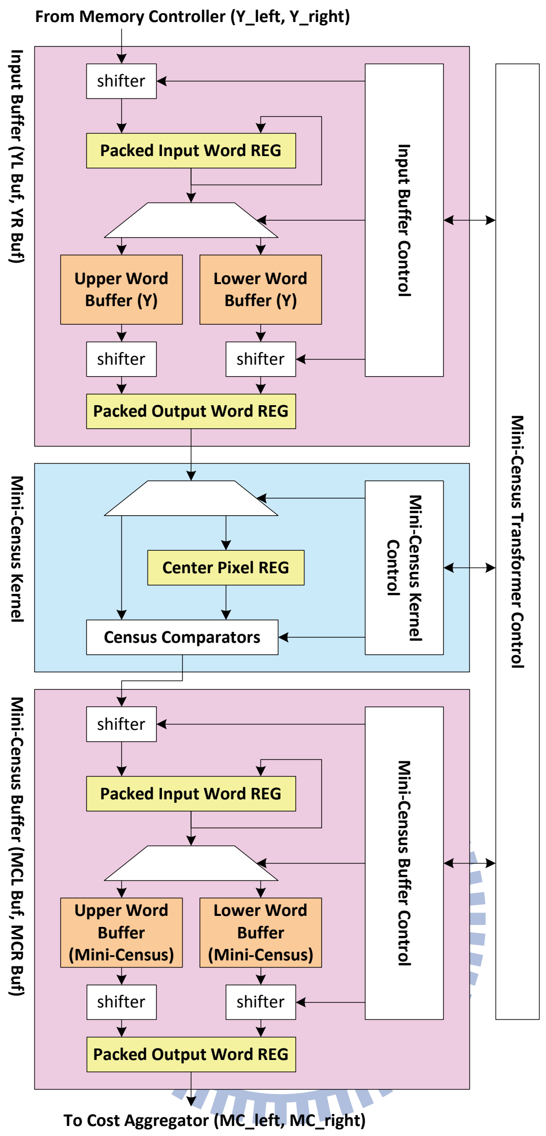

From Memory Controller (Y_left, Y_right)
Input Buffer (YL Buf, YR Buf)
shifter
Packed Input Word REG
Upper Word Buffer (Y)
Lower Word Buffer (Y)
shifter
shifter
Packed Output Word REG
Input Buffer Control
Mini-Census Transformer Control
Mini-Census Kernel
Center Pixel REG
Census Comparators
Mini-Census Kernel Control
Mini-Census Buffer (MCL Buf, MCR Buf)
shifter
Packed Input Word REG
Upper Word Buffer (Mini-Census)
Lower Word Buffer (Mini-Census)
shifter
shifter
Packed Output Word REG
Mini-Census Buffer Control
To Cost Aggregator (MC_left, MC_right)

Fig. 49. Architecture of the mini-census transformer

transform units. Each unit consists of an input buffer, a mini-census kernel, and a mini-census buffer. The input buffer stores the input image data read from the external image. The input data are first packed into data words and stored into the word buffers. Once the data are ready for mini-census transform, the data in the input buffer are read into the mini-census kernel. The center pixel is stored in the register and compared with its surrounding six pixels in the mini-census template. The comparison result, the 6-bit mini-census bitstream, is then written into the mini-census buffer.

C. Weight Generation

Fig. 50 shows the architecture of the weight generator. The architecture is similar to the mini-census transformer. However, there are three sets of input buffer because the weight generation needs all three color components. The mini-census kernel is replaced by a weight generation kernel which reads the input pixels column by column to generate vertical weight. The color distances between the center pixel and others are computed in the Manhattan color distance computer. Once the color distance is available, it is used to look up the corresponding weight from the weight table. During the column by column read from the input buffer, the center pixel of each column is also stored in the horizontal row buffer. After the vertical weight is generated, the horizontal weight is generated from the pixel data in the

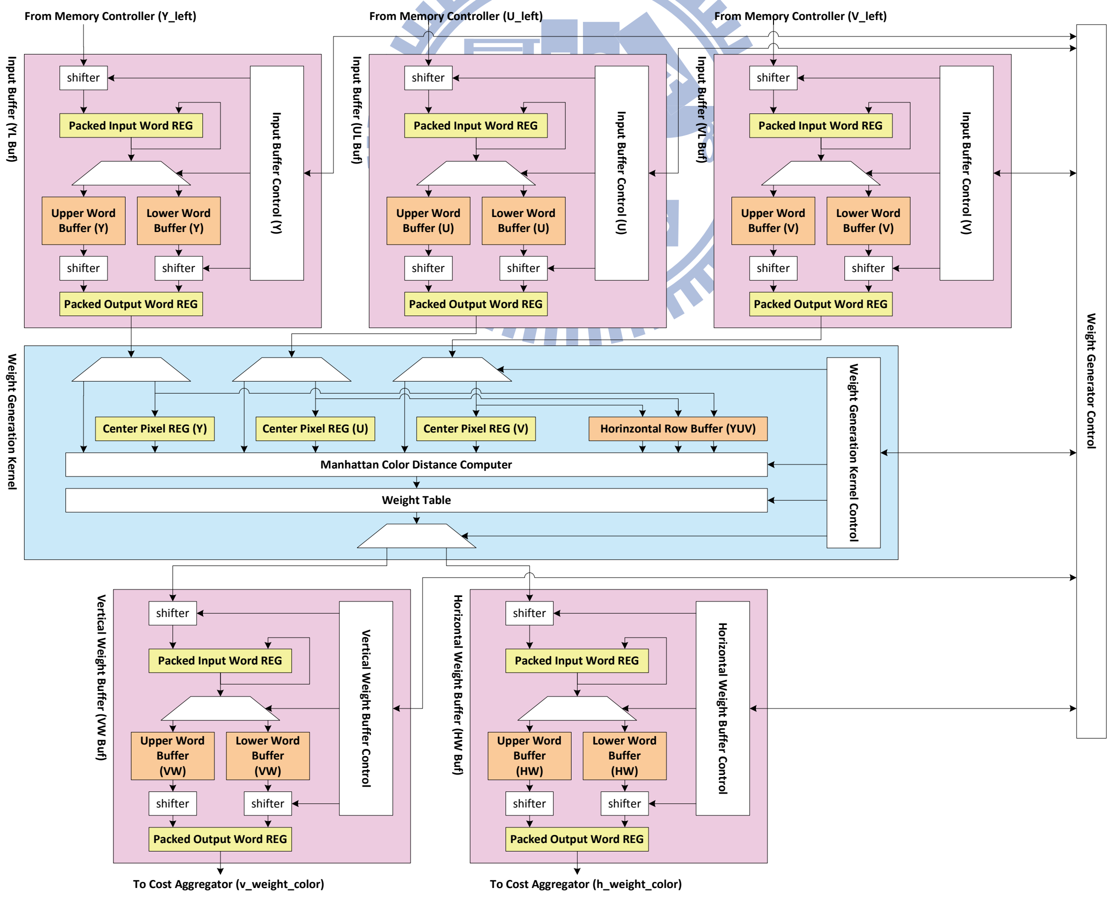
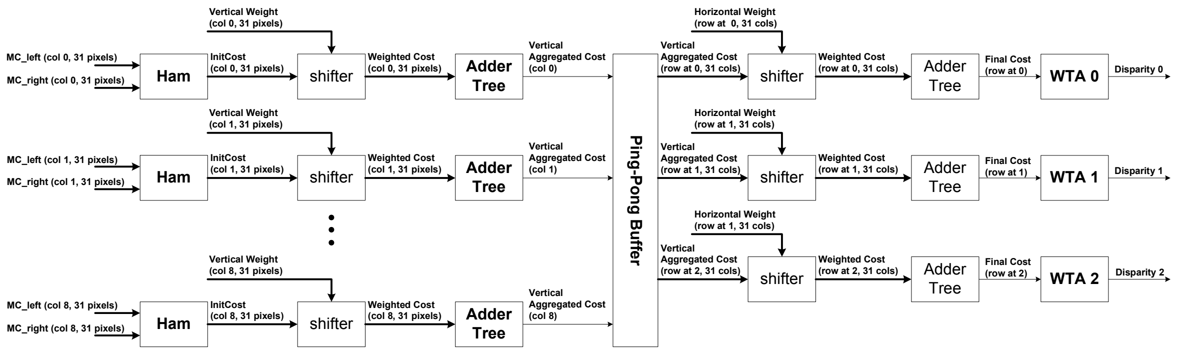


Fig. 50. Architecture of the weight generator

horizontal row buffer. This avoids reading the input buffer again during the horizontal weight generation. The vertical weight buffer and horizontal weight buffer store the resulting vertical and horizontal weight.

D. Cost Aggregator and WTA

Fig. 51 shows the architecture of the cost aggregator and winner-takes-all (WTA). The mini-census cost computation and vertical cost aggregation are shown to the left of the ping-pong buffer, whereas the horizontal cost aggregation and WTA are shown to the right. The census costs between the left and right mini-census bitstreams are computed by the hamming distance computation unit. To increase processing speed, each unit computes the census cost of all the pixels within a column in parallel. After the census costs are available, they are multiplied with the vertical weights and summed together to give the vertical aggregated cost. The vertical aggregated costs are stored in the ping-pong buffer. The ping-pong buffer is scheduled as shown in Fig. 52 to ensure that the aggregation can be performed continuously without any pause. The ping-pong buffer outputs 33 vertical aggregated costs to the three shifters used for horizontal aggregation. We have applied the PCR technique so that the first 31 costs in the 33 costs are sent to the first shifter, the second 31 costs are sent to the second shifter, and the third 31 costs are sent to the third shifter. This



Vertical Weight (col 0, 31 pixels)
MC_left (col 0, 31 pixels)
MC_right (col 0, 31 pixels)
Ham
InitCost (col 0, 31 pixels)
shifter
Weighted Cost (col 0, 31 pixels)
Adder Tree
Vertical Aggregated Cost (col 0)
Vertical Weight (col 1, 31 pixels)
MC_left (col 1, 31 pixels)
MC_right (col 1, 31 pixels)
Ham
InitCost (col 1, 31 pixels)
shifter
Weighted Cost (col 1, 31 pixels)
Adder Tree
Vertical Aggregated Cost (col 1)
Vertical Weight (col 8, 31 pixels)
MC_left (col 8, 31 pixels)
MC_right (col 8, 31 pixels)
Ham
InitCost (col 8, 31 pixels)
shifter
Weighted Cost (col 8, 31 pixels)
Adder Tree
Vertical Aggregated Cost (col 8)
Ping-Pong Buffer
Horizontal Weight (row at 0, 31 cols)
Vertical Aggregated Cost (row at 0, 31 cols)
shifter
Weighted Cost (row at 0, 31 cols)
Adder Tree
Final Cost (row at 0)
WTA 0
Disparity 0
Horizontal Weight (row at 1, 31 cols)
Vertical Aggregated Cost (row at 1, 31 cols)
shifter
Weighted Cost (row at 1, 31 cols)
Adder Tree
Final Cost (row at 1)
WTA 1
Disparity 1
Horizontal Weight (row at 1, 31 cols)
Vertical Aggregated Cost (row at 2, 31 cols)
shifter
Weighted Cost (row at 2, 31 cols)
Adder Tree
Final Cost (row at 2)
WTA 2
Disparity 2

Fig. 51. Architecture of the cost aggregator and WTA

reduces the output bandwidth requirement of the ping-pong buffer. Once the final matching costs are available, they are compared with the current minimal final costs in the WTA modules. After the cost of all the disparities are compared, the disparity with the minimal cost

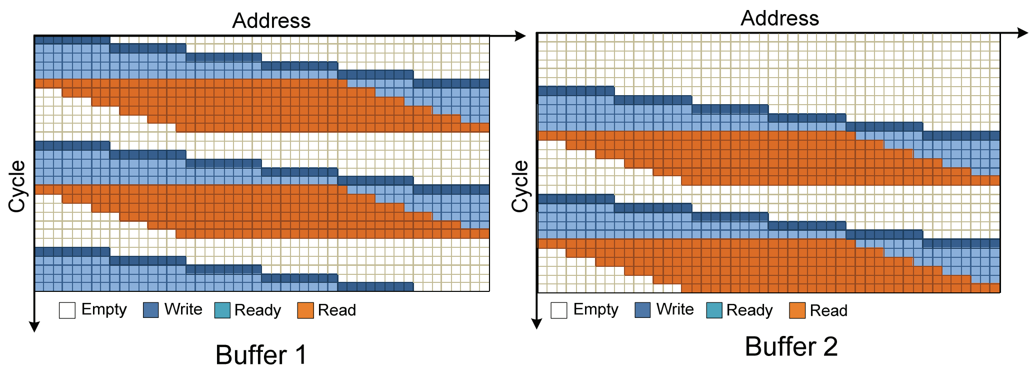


Fig. 52. Schedule of the ping-pong buffer

is the final output disparity.

Fig. 53 illustrates the processing order of the cost aggregator. The cost aggregator is capable of processing eight columns of 31 pixels simultaneously. The initial pipeline delay is 7 cycles. After the initial pipeline delay, it takes 6 cycles to process all 31x48 pixels to give

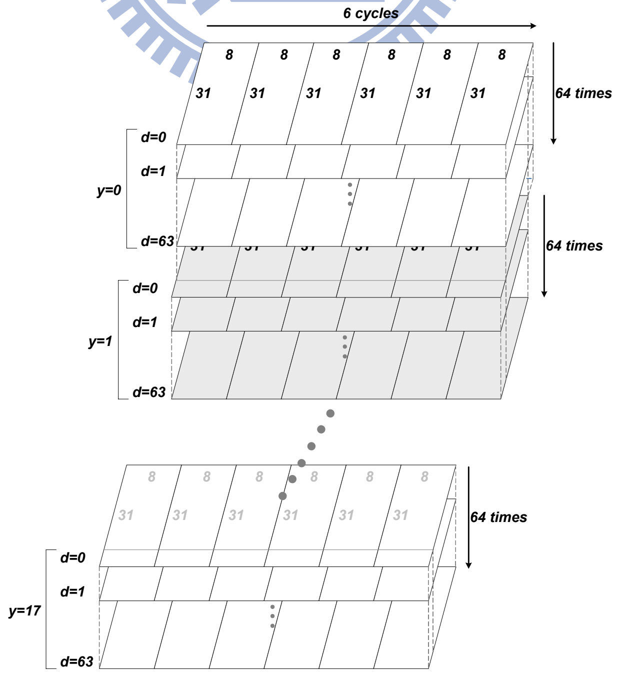


Fig. 53. Processing schedule of the cost aggregator

18 final matching costs. These 18 matching costs are of the same disparity. After the final matching costs of a disparity are computed and compared, the final matching costs of the next disparity are computed and compared. Once the final disparity of an 18-pixel row is determined, the cost aggregator and WTA start processing the next 18-pixel row. For example, the disparity of y=0 is estimated first, then the disparity of y=1 is estimated. After (7+6x64)x18 = 7,038 cycles, the disparity of an 18x18 block are determined.

E. Memory Controller

The memory controller interfaces to external memory and arbitrates the external memory access requests from the mini-census transformer, weight generator, and cost aggregator and WTA. The data port width between the image memory and the memory interface is 32-bit. The arbitration is a hybrid of round-robin and fixed priority strategy. The depth FIFO always has the highest priority due to the high penalty of suspending of the cost aggregator and WTA. The priority of the mini-census transformer and weight generator are determined by round-robin.

F. Scheduling

Fig. 54 illustrates the scheduling of the MCADSW architecture. The cost aggregator starts processing data after all the 31x48 mini-censuses and weights are available. This takes 470 cycles to prepare the censuses and 1,536 cycles to prepare the weights. Since the cost aggregator and WTA take 7,038 cycles to finish, the mini-census transform and weight generation of the next 18x18 block can be performed at the same time. Based on this

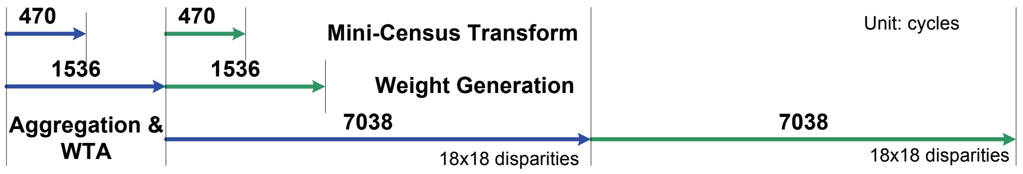


Fig. 54. Schedule of the MCADSW architecture

scheduling, it takes approximately 2.5 million cycles to complete the disparity estimation of all the 320 18x18 blocks in a CIF sized stereo image pair.

5.4.6. Implementation Result and Comparison

A. Gate Count and Memory Size

The proposed MCADSW architecture was synthesized using UMC 90 nm standard cells. Table 18 lists the core characteristics of the synthesized design. The total equivalent

Table 18 Core characteristics of the proposed MCADSW

Technology	UMC 90 nm
Max clock rate	95 MHz
Equivalent gate-count (excluding memories)	562,642
Memory size	21.3 KB
Image size	352x288 (CIF)
Disparity range	64
Maximal frame rate	42 FPS@95MHz

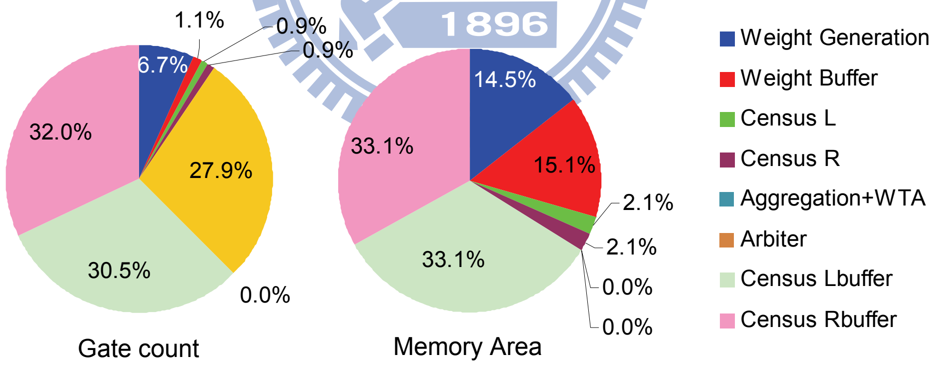


Fig. 54. Percentage of the memory area and combinational gate counts

gate-count is about 563K excluding the memories and the maximum operation frequency is 95 MHz. The equivalent gate-count and the memory area distributions are shown in Fig. 51. The total gate-count was dominated by the cost aggregator and WTA, census left buffer, and census right buffer. This was due to the high computation resource requirement and complex demultiplexing circuits. The memory area was dominated by the weight generation, weight buffer, census left buffer, and census right buffer.

B. Performance Comparison

Table 19 and Table 20 compare the disparity estimation speed and performance of the MCADSW architecture with other existing high performance real-time implementations quantitatively. In TABLE IV, the performance was evaluated using Middleburry's stereo image pairs and their evaluation method [72]. The error rate is the overall error rate with the tolerance of one disparity level. In addition to the quantitative performance evaluation, we also included the disparity maps generated by different implementations in Fig. 52 for qualitative performance comparison.

Table 19 Speed comparison of different implementations

Design	Implementation	Image Size	Disparity Range	FPS	MDE/s
Proposed MCADSW	**UMC 90nm Std. Cell**	**352x288**	**64**	**42**	**272.5**
TrellisDP[112]	Xilinx Virtex II Pro-100	320x240	128	30	294
HBP[114]	Xilinx Virtex II Pro-100 x2	320x240	32	30	73.7
EffectAggr[116]	Intel Core 2 Duo 2.14 GHz	463x370	75	1.67	18.9
RealDP[105]	AMD AthlonXP 2800	384x288	100	18.9	209
CBiased[107]	Nvidia Geforce 7900	512x512	96	24	605
SepLaplacian[108]	Nvidia Geforce 7900	256x256	96	87	547
RealTimeBP[113]	Nvidia Geforce 7900	320x240	16	16	19.6
RealTimeGPU[109]	ATI Radeon 9800	320x240	16	16	19.6
ReliableGPU[106]	ATI Radeon 9800	N.A.	N.A.	16.6	N.A.
GradientGuided[117]	ATI Radeon 9800XT	512x384	40	14.7	117

Table 20 Performance comparison of different implementations

Design	Tsukuba	Venus	Teddy	Cones	Saw Tooth	Map
Proposed	**2.80**	**0.64**	**13.7**	**10.1**	**2.11**	**3.21**
TrellisDP[112]	2.63	3.44	N.A.	N.A.	1.88	0.91
HBP[114]	2.85	1.92	N.A.	N.A.	6.25	6.45
EffectAggr[116]	2.11	4.75	15.2	12.6	N.A.	N.A.
RealDP[105]	2.85	6.42	N.A.	N.A.	6.25	6.45
CBiased[107]	4.77	10.2	N.A.	N.A.	0.82	0.65
SepLaplacian[108]	13.0	N.A.	N.A.	N.A.	N.A.	N.A.
RealTimeBP[113]	3.40	1.90	13.2	11.6	N.A.	N.A.
RealTimeGPU[109]	4.22	2.98	14.4	13.7	N.A.	N.A.
ReliableGPU[106]	1.36	1.09	N.A.	N.A.	2.35	0.55
GradientGuided[117]	2.48	3.91	N.A.	N.A.	1.63	0.73

From the speed perspective, the TrellisDP, CBiased, and SepLaplacian outperformed the MCADSW architecture in terms of the MDE/s. The TrellisDP was faster because it adopted a fully parallel systolic array architecture with 128 PEs. Their systolic architecture maximized the utilization of the PEs to achieve a processing speed of 294 MDE/s. However, the systolic architecture would require very high bandwidth and large storage to keep the 128 PEs working without idling. The CBiased and SepLaplacian were at least two times faster than the MCADSW because they were implemented using high performance programmable GPUs. These high performance GPU had extremely high bandwidth and computation hardware resource available. For instance, Nvidia's Geforce 7800GTX GPU had 256MB of GDDR3 DRAM clocked at 600MHz, with a data port of 256-bit, the maximum available peak bandwidth can reach up to 38.4 GB/s [119]. Together with an operating clock of 430 MHz and 8 vertex and 16 pixel shaders, it is reasonable for GPU-based implementation to achieve such high processing speeds. In contrast, the MCADSW architecture required much lower clock rate, less bandwidth, and smaller silicon area. This makes the MCADSW more applicable to embedded vision applications. As to the disparity estimation performance, only the TrellisDP was comparable to the MCADSW in terms of the error rate, whereas both the performance of the Cbiased and SepLaplacian were inferior. This can also be observed in the disparity maps shown in Fig. 55. The disparity map of the MCADSW had more accurate depth-discontinuity than others in most regions except in the camera region. The reason for the camera region being blurry was probably due to the removal of the proximity weight. All in all, the MCADSW provided high disparity estimation speed and performance that was comparable to other high performance real-time implementations while requiring less bandwidth and being more suitable for embedded vision systems.

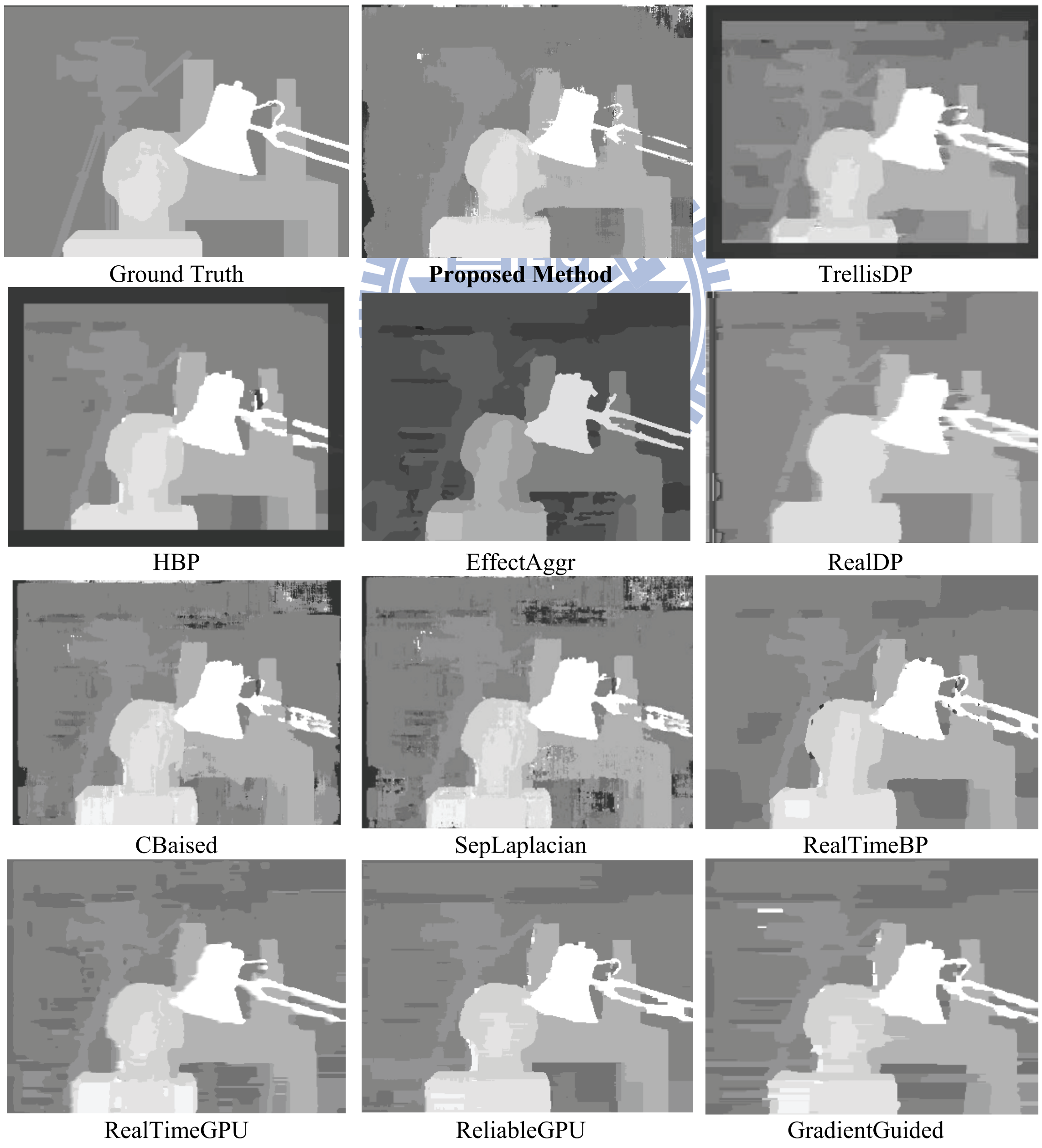


Fig. 55. Disparity map of different implementations

5.4.7. Summary

This section presented a hardware-friendly high performance disparity estimation algorithm, the MCADSW, and its corresponding architecture for real-time stereo matching. The proposed hardware-friendly simplifications not only made the MCADSW more hardware-friendly, but also reduced the execution time of the MCADSW algorithm by 61.3%. In the design of the MCADSW architecture, we proposed the PCR and AREW techniques to significantly reduce bandwidth requirement. The proposed architecture was synthesized using UMC 90 nm standard cells. At the operation frequency of 95 MHz, the proposed architecture can achieve 42 FPS of CIF size disparity map with 64 disparity levels. The equivalent gate-count and total memory size are 562 K and 21.3KB respectively.

Chapter 6 Conclusion

This dissertation addressed the bandwidth issue from the source to the destination of data transfers based on the core concept of facilitating the address and data correlation among accesses. At the source of data transfers, this dissertation proposed a memory controller that increased the bandwidth utilization by facilitating access address correlation, taking the advantage of new advanced data transfer protocol, and the characteristics of external memories. After improving the bandwidth utilization at the source of data transfers, this dissertation focused on improving the bandwidth utilization of a bus interconnect adopting advanced protocol under the traditional share-link topology. Finally, the bandwidth requirement reduction techniques based on data and access characteristics have been studied at the destination of data transfers. The bandwidth requirement can be reduced in two major ways. The first approach is to take the advantage of data characteristics. The other approach is to reuse data based on an algorithm's data access spatial and temporal locality. In video coding, the CFMMC architecture was capable of reducing the bandwidth requirement and energy consumption up to 72% and 16% respectively when the percentage of perfect matched macroblocks is higher than 70%. In early vision tasks, the proposed PUPP reduced the bandwidth to the image memory by 81.6% in the proposed meanshift architecture. Both CFMMC and PUPP were examples of the first approach to reduce bandwidth requirement. In the MCADSW stereo matching architecture, the proposed the PCR and AREW techniques, which were examples of the second bandwidth requirement reduction approach, could reduce bandwidth requirement by an order.

Although this dissertation has proposed methods to increase system's effective bandwidth and to reduce core's bandwidth requirement for video and vision applications, systematic integration of these techniques into advanced ESL design flow tools has not been available. With the proposed bandwidth issue solutions, future researches can consider integrating these solutions into an automatic bandwidth optimizing tool. Doing so would enable more complex bandwidth demanding but extremely useful video and vision algorithms to be accelerated for real-time applications.

References

[1] Sentinel 400 7-inch television receiver, 1948, Science and Society Picture Library Search, available online: http://www.scienceandsociety.co.uk/results.asp?image=10463744

[2] Portable Media Player on Wikipedia, available online: http://en.wikipedia.org/wiki/Portable_media_player

[3] Apple iPhone homepage, available online: http://www.apple.com/iphone/

[4] Canon Ixus 850IS 產品介紹, available online: http://www.rainbowphoto.com.tw/ixus850is/main_850is.htm

Memory Controller

[5] R. Saleh et al., "System-on-Chip: Reuse and Integration," in Proceedings of the IEEE, vol. 94, pp. 1050-1069, June 2006.

[6] S. Przybylski, "Sorting out the new DRAMs," in Hot Chips Tutorial, Stanford, CA, 1997.

[7] ARM Limited, "AXI Protocol, " [online]. Available: http://www.arm.com/products/solutions/AMBA3AXI.html

[8] S. MacKee et al., "Dynamic access ordering for streamed computations," IEEE Trans. Computers, vol. 49, no. 11, pp. 1255-1271, Nov. 2000.

[9] K. Ayukawa, T. Watanabe, and S. Narita, "An access-sequence control scheme to enhance random-access performance of embedded DRAM's," IEEE J. Solid-State Circuit, vol. 33, no. 5, pp. 800-806, May 1998.

[10] S. Rixner, et al., "Memory access scheduling," in Proc. 27th Annual Int'l Symp. Computer Architecture, Vancouver, Canada, June 2000, pp. 128–138.

[11] T. Takizawa and M. Hirasawa, “An efficient memory arbitration algorithm for a single chip MPEG2 AV decoder,” IEEE Trans. Consumer Electronics, vol. 47, no. 3, pp. 660-665, August 2001.

[12] K. –B. Lee, T. –C. Lin, and C. –W. Jen, “An efficient quality-aware memory controller for multimedia platform SoC,” IEEE Trans. Circuits and Systems for Video Technology, vol. 15, pp. 620-633, May 2005.

[13] P. K. Nizar and M. W. Williams, “Method and apparatus for providing a pipelined memory controller,” U.S. Patent, no. 621261, April 2001.

[14] MemMax 2.0 Multi-threaded DRAM access scheduler, Sonics Limited, http://www.sonicsinc.com/documets/MemMax_2.0_Data_Sheet.pdf

[15] PrimeCell AXI SDRAM Controller PL340, ARM Limited, [online] available http://www.arm.com/products/solutions/PL340AXIController.html

[16] V. De La Luz, et al., “Hardware and software techniques for controlling DRAM power modes,” IEEE Trans. Computers, vol. 50, pp. 1154–1173, Nov. 2001.

[17] Y. Joo et al., “Energy exploration and reduction of SDRAM memory systems,” in Proc. Design Automation Conf., June 2002, pp. 892-897.

[18] N. –Y. Ker and C. –H. Chen, “An effective SDRAM power mode management scheme for performance and energy sensitive embedded systems,” in Proc. Asia and South Pacific Design Automation Conf., Jan. 2003, pp. 515–518.

[19] A. Burchardt et al., “A real-time streaming memory controller,” in Proc. Design, Automation and Test in Europe, March 2005, vol. 3, pp. 20-25.

[20] Jeff Janzen, “Calculating memory system power for DDR SDRAM,” in Micron Designline, quarter 2, 2001.

[21] Micron Technology Inc., MT46V8M16 128Mb DDR SDRAM [online]. Available: http://download.micron.com/pdf/datasheets/dram/ddr/128MBDDRx4x8x16D.pdf

[22] Open SystemC Initiative (OSCI), “SystemC” [online]. Available: http://systemc.org

System Bus

[23] ARM Limited, “AMBA 2 Specification,” [online]. Available: http://www.arm.com/products/solutions/amba2overview.html

[24] IBM Microelectronics, “CoreConnect bus architecture,” [online]. Available: http://www-306.ibm.com/chips/products/coreconnect/

[25] OpenCore, “SoC Interconnect: Wishbone,” [online]. Available: http://www.opencores.org/projects.cgi/web/wishbone/wishbone

[26] ARM Limited. “Multi-layer AHB Overview,” May 2004.

[27] F. Poletti, D. Bertozzi, L. Benini, and A. Bogliolo, “Performance Analysis of Arbitration Polices for SoC Communication Architectures,” Design Automation for Embedded System, vol. 8, no. 2-3, June 2003, pp. 189-210.

[28] K. Lahir, A. Raghunathan, and G. Lakshminarayana, “LOTTERYBUS: a new high-performance communication architecture for system-on-chip designs,” in Proc. 38th Design Automation Conf., June 2001, pp. 15-20.

[29] ARM Limited, “AXI Protocol, “ [online]. Available: http://www.arm.com/products/solutions/AMBA3AXI.html

[30] Open Core Protocol International Partnership (OCP-IP), “Open Core Protocol,” [online]. Available: http://www.ocpip.org/

[31] ST Microelectronics, “STBus Interconnect,” [online]. Available: http://www.st.com/stonline/products/technologies/soc/stbus.htm

[32] Arm Limited, “PrimeCell AXI Interconnect (PL300) Technical Reference Manual,” October 2005.

[33] Synopsis Inc. “DesignWare IP solutions for AMBA Interconnect,” [online]. Available: http://www.synopsys.com/products/designware/amba_solutions.html

[34] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Automated Throughput driven Synthesis of Bus-based Communication Architectures", In Proc. of ASPDAC Feb. 2005.

[35] K. Richter, M. Jersak, and R. Ernst., "A Formal Approach to MpSoC Performance Verification," IEEE Computer, 36:60–67, April 2003.

[36] G. Madl, S. Pasricha, Q. Zhu, L. Bathen, N. Dutt, "Formal performance evaluation of AMBA-based system-on-chip designs," in Proc. 6th ACM & IEEE Int'l Conf. Embedded Software, 2006, pp.311-320.

[37] L. Cai, D. Gajski, “Transaction Level Modeling: An Overview”, in Proc. 2nd IEEE/ACM/IFIP Int'l Conf. Hardware/Software Codesign, Oct. 2003, pp. 19-24.

[38] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Fast exploration of bus-based on-chip communication architectures," in Proc. 2nd IEEE/ACM/IFIP Int'l Conf. Hardware/Software Codesign and System Synthesis, Sep. 2004, pp.242-247.

[39] S. Pasricha, N. Dutt, and M. Ben-Romdhane, “Constraint-driven bus matrix synthesis for MPSoC,” in Proc. ASPDAC, Jan. 2006, pp. 30–35.

[40] S. Pasricha, N. Dutt, M. Ben-Romdhane, "High Level Design Space Exploration of Shared Bus Communication Architectures", CECS Technical Report 04-06, March, 2004.

[41] S. Lee, C. Lee, H.-J. Lee, “A new multi-channel on-chip-bus architecture for system-on-chips,” in Proc. of IEEE Int’l SOC Conf., September 2004, pp. 305-308.

[42] M. Ruggiero et al., “Scalability Analysis of Evolving SoC Interconnect Protocols,” in Proc. Int’l. Symp. System-on-Chip, Nov. 2004, pp.169-172.

[43] K. Lahiri, A. Raghunathan, S. Dey, "Design space exploration for optimizing on-chip communication architectures," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systmes, vol. 23, no. 6, June 2004, pp.952-961.

[44] S. Murali, L. Benini, G. De Micheli, "An Application-Specific Design Methodology for On-Chip Crossbar Generation," IEEE Trans. Coputer-Aided Design of Inegrated Circuits and systems, vol. 26, no. 7, July 2007, pp. 1283-1296.

[45] Open SystemC Initiative (OSCI), “SystemC” [online]. Available: http://systemc.org

[46] T.-J. Lin, C.-N. Liu, S.-Y. Tseng, Y.-H. Chu, A.-Y. Wu, “Overview of ITRI PAC project - from VLIW DSP processor to multicore computing platform,” in Proc. IEEE Int’l Symp. VLSI Design, Automation, and Test, April 2008, pp.188-191.

Motion Compensation

[47] V. G. Moshnyaga, "Reducing Energy Dissipation of Frame Memory by Adaptive Bit-Width Compression" IEEE Transactions on Circuits and Systems for Video Technology, Vol. 12, no.8, pp.713-718, August 2002.

[48] F. Catthoor et al., "Low power storage exploration for H.263 video decoder," in 4th Workshop on VLSI Signal Processing, pp.115 - 124, 30 Oct.-1 Nov. 1996.

[49] L. Nachtergaele et al., "Low-power data transfer and storage exploration for H.263 video decoder system," in IEEE Journal on Selected Areas in Communications, Vol. 16 , Issue 1 , pp.120 - 129, Jan. 1998.

[50] K. Denolf et al., "Memory Centric Design of An MPEG-4 Video Encoder," in IEEE Trans. Circuit and System for Video Technology, vol. 15, pp.609-pp.619, May 2005.

[51] V. G. Moshnyaga, K. Masunaga, and N. Kajiwara, "A data reusing architecture for MPEG video coding," Proc. Int'l Conf. Circuits and Systems (ISCAS'04), vol. 3, pp.797-pp.800, May 2004.

[52] ISO/IEC 14496-2, "Information technology - Coding of audio-visual objects," 2nd edition, Switzerland, Dec. 2001.

[53] ISO/IEC JTC1/SC29/WG11 N3908, MPEG-4 Video Verification Model version 18.0, Jan. 2001.

[54] Artisan Components, Inc. "UMC 0.18um Process High-Speed single Port SRAM Generator User Manual," Release 4.0, August 2000.

[55] NEC Inc., "16M-bit CMOS Mobile Specified RAM Datasheet," [online] http://www.necel.com/memory/pdfs/M15085EJ5V0DS00.pdf

[56] UMC Free-of-Charge Libraries, [online] http://www.umc.com/english/design/b_3.asp.

[57] Synopsy Inc., "Power Compiler User Guide," Release W-2004.12, January 2005.

Meanshift Filtering

[58] Y. Cheng, "Meanshift, mode seeking, and clustering," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 17, no. 9, pp.790-799, Aug. 1995.

[59] D. Comaniciu and P. Meer, "Meanshift: a robust approach toward feature space analysis," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 24, no. 5, pp. 603-619, May 2002.

[60] C. Christoudias, B. Georgescu, and P. Meer, "Synergism in Low Level Vision," in Proc. Int'l Conf. Pattern Recognition, vol. 4, August 2002, pp.150-155.

[61] C. Rambabu, I. Chakrabarti, and A. Mahanta, "Flooding-based watershed algorithm and its prototype hardware architecture," in IEE Proc. Vision, Image and Signal Processing, vol. 151, pp.224-234, June 2004.

[62] M. Neuenhahn, H. Blume, and T. G. Noll, "Pareto optimal design of an FPGA-based real-time watershed image segmentation," in 15th Annual Workshop on Circuits systems on Signal processing, Nov. 2004.

[63] K. Yamaoka, T. Morimoto, H. Adachi, K. A. A. K. Awane, T. A. K. T. Koide, and H. J. A. M. H. J. Mattausch, "Multi-object tracking VLSI architecture using image-scan based

region growing and feature matching," in Proc. IEEE Int'l Symp. Circuit and System, 2006, p. 4.

[64] M. Leeser, N. Kitaryeva, and J. Crisman, "Spatial and color clustering on an FPGA-based computer system," in Proc. SPIE, vol. 3526, Nov. 1998, pp. 25-33.

[65] B. Maliatski and O. Yadid-Pecht, "Hardware-driven adaptive k-means clustering for real-time video imaging," IEEE Tran. Circuit and System for Video Technology, vol. 15, no. 1, pp. 164-166, Jan. 2005.

[66] O. J. Hernandez, "A high-performance VLSI architecture for the histogram peak-climbing data clustering algorithm," IEEE Trans. Very Large Scale Integration Systems, issue 2, vol. 14, pp. 111-121, Feb. 2006.

[67] T. Maruyama, "Real-time K-Means Clustering for Color Images on Reconfigurable Hardware," in 18th Int'l Conf. Pattern Recognition, 2006, pp. 816-819.

[68] T. –W. Chen, C. –H. Sun, J. –Y. Bai, H. –R. Chen, and S. –Y. Chien, "Architectural Analyses of K-Means silicon intellectual property for image segmentation," in IEEE Int'l Symp. Circuit and System, May 2008, pp.2578-2781.

[69] S. Park, Y. Ha, and H. Jeong, "A Parallel and Memory-Efficient Meanshift Filter on a Regular Graph," in Proc. Int'l Conf. Intelligent Pervasive Computing, Oct. 2007, pp. 254-259.

[70] K. Zhang, J. T. Kwok, and M. Tang, "Accelerated convergence using dynamic Meanshift," in Proc. European Conf. Computer Vision, vol. 2, May 2006, pp. 257-268.

Stereo Matching

[71] M. Brown, D. Burschka, and G. Hager, "Advances in computational stereo," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 25, no. 8, August 2003.

[72] D. Scharstein and R. Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," Int'l Journal on Computer Vision, vol. 47, no. 1-3, pp.7-42, April 2002.

[73] K.-J. Yoon and I.-S. Kweon, “Adaptive support-weight approach for correspondence search,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 28, issue 4, pp.650-656, April 2006.

[74] T. Kanade, M. Okutomi, and T. Nakahara, “A multiple-baseline stereo method,” in Proc. ARPA Image Understanding Workshop, 1992, pp. 409-426.

[75] S. Kimura, T. Kanade, H. Kano, A. Yoshida, E. Kawamura, and K. Oda, “CMU Video-Rate Stereo Machine,” Proc. Mobile Mapping Symp. 1995.

[76] K. Konolige, “Small vision systems: hardware and implementation,” in Proc. Eighth Int’l Symp. Robotics Research, Oct. 1997, pp.203-212.

[77] J. Woodfill and B. Von Herzen, “Real-time stereo vision on the PARTS reconfigurable computer,” in Proc. IEEE Workshop FPGAs for Custom Computing Machines, 1997, pp. 240-250.

[78] P. Corke and P. Dunn, “Real-time stereopsis using FPGAs,” in Proc. IEEE TENCON – Speech and Image Tech. for computing and Telecommunications, Apr. 1997, pp. 235-238.

[79] S. Kimura et al. “A convolver-based real-time stereo machine (SAZAN),” in Proc. Conf. Computer Vision and Pattern Recognition, vol. 1, 1999, pp. 457-463.

[80] M. Hariyama et al., “FPGA implementation of a stereo matching processor based on window-parallel-and-pixel-parallel architecture,” in Proc. 48th Midwest Symp. Circuit and System, vol.2, August 2005, pp. 1219-1222.

[81] M. Gong and Y. -H. Yang, “Near real-time reliable stereo matching using programmable graphics hardware,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition, vol. 1, June 2005, pp. 924-931.

[82] M. Hariyama, H. Sasaki, and M. Kameyama, ”Architecture of a stereo matching VLSI processor based on hierarchically parallel memory access”, IEICE Trans. Info. and Syst., vol. E88-D, no. 7, pp.1486-1491, 2005.

[83] D. Masrani and J. MacLean, “A real-time large disparity range stereo-system using FPGAs,” in Proc. IEEE Conf. Computer Vision Systems, Jan, pp. 13-13. 2006.

[84] N. Chang, T.-M. Lin, T.-H. Tsai, Y.-C. Tseng, "Real-time DSP implementation on local stereo matching," in Proc. IEEE Conf. Multimedia and Expo, July 2007, pp.2090-2093.

[85] J. Banks and P. Corke, “Quantitative evaluation of matching methods and validity measures for stereo vision,” Int’l Journal of Robotics Research, vol. 20, no. 7, pp.512-532, July 2001.

[86] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in IEEE Proc. Conf. Computer Vision and Pattern Recognition, June 2007, pp.1-8.

[87] R. Zabih and J. Woodfill, “Non-Parametric Local Transforms for Computing Visual Correspondence,” in Proc. Third European Conf. Computer Vision, pp. 150-158, 1994.

[88] H. Hirschmuller, “Accurate and efficient stereo processing by semi-global matching and mutual information,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition, vol. 2, June 2005, pp.807-814.

[89] M. Bleyer, S. Chambon, U. Poppe, and M. Gelautz, "Evaluation of different methods for using colour information in global stereo matching approaches," The Congress of the International Society for Photogrammetry and Remote Sensing, July 2008.

[90] N. Chang, Y.-C. Tseng, and T.-S. Chang, "Analysis of color space and similarity measure impact on stereo block matching," to be appeared in Proc. IEEE Asia Pacific Conf. Circuits and Systems, Dec. 2008.

[91] M. Okutomi and T. Kanade, "A locally adaptive window for signal matching," Int'l Journal of Computer Vision, vol. 7, pp. 143-162, Jan. 1992.

[92] T. Kanade and M. Okutomi, " A stereo matching algorithm with an adaptive window: theory and experiment," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 16, issue 9, pp.920-930, Sept. 1994.

[93] M. Hariyama, T. Takeuchi, and M. Kameyama, "Reliable stereo matching for highly-safe intelligent vehicles and its VLSI implementation," in Proc. IEEE Intelligent Vehicles Symposium, Oct. 2000, pp.128-133.

[94] R. Arnold, "Automated stereo perception," Technical Report AIM-351, Artificial Intelligence Laboratory, Stanford University, 1983.

[95] A. F. Bobick and S. S. Intille, "Large Occlusion Stereo," Int'l Journal on Computer Vision, vol. 33, no. 3, pp.181-200, Sep. 1999.

[96] S.-B. Kang, R. Szeliski, and J. Chai, "Handling occlusions in dense multi-view Stereo," in Proc. IEEE Conf. on Computer Vision and Pattern Recognition, vol. 1, 2001, pp.103-110.

[97] O. Veksler, "Stereo matching by compact windows via minimum ratio cycle," in Proc. IEEE Int'l Conf. Computer Vision, vol. 1, July 2001, pp.540-547.

[98] M. Gerrits and P. Bekaert, "Local stereo matching with segmentation-based outlier rejection," in Proc. 3rd Canadian Conference on Computer and Robot Vision, June 2006, pp.66-66.

[99] F. Tombari, S. Mattoccia, and L. Di Stefano, "Segmentation-based adaptive support for accurate stereo correspondence," Lecture Notes in Computer Science, vol.4872, p.427-438, Dec. 2007.

[100] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda, "Classification and evaluation of cost aggregation methods for stereo correspondence," in Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition, June, 2008, pp.1-8.

[101] Point Grey Research Inc., Digiclops, 1997.

[102] Point Grey Research Inc. , Triclops, 2001.

[103] H. Hirschmuller, "Improvements in real-time correlation-based stereo vision," in Proc. IEEE Workshop on Stereo and Multi-Baseline Vision, Dec. 2001, pp. 141-148.

[104] H. Hirschmuller, P. R. Innocent, and J. Garibaldi, "Real-time correlation-based stereo vision with reduced border errors," Int'l Journal of Computer Vision, vol. 47, no. 1-3, pp.229-246, Nov. 2004.

[105] S. Forstmann, Y. Kanou, O. Jun, S. Thuering, and A. Schmitt, "Real-time stereo by using dynamic programming," in Proc. Computer Vision and Pattern Recognition Workshop on Real-Time 3D Sensor and Their Use, June 2004, pp.29-29.

[106] G. Minglun and Y. Yee-Hong, "Near real-time reliable stereo matching using programmable graphics hardware," in Proc. IEEE Conf. Computer Vision and Pattern Recognition, vol.1, June 2005, pp.924-931.

[107] L. Jiangbo, G. Lafruit, and F. Catthoor, "Fast variable center-biased windowing for high-speed stereo on programmable graphics hardware," in Proc. IEEE Int'l Conf. Image Processing, vol. 6, Oct. 2007, pp.568-571.

[108] L. Jiangbo, S. Rogmans, G. Lafruit, and F. Catthoor, "Real-time stereo correspondence using a truncated separable Laplacian kernel approximation on graphics hardware," in Proc. IEEE Int'l Conf. Multimedia and Expo, July 2007, pp.1946-1949.

[109] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister, "High-quality real-time stereo using adaptive cost aggregation and dynamic programming," in Proc. 3rd Int'l Symp. 3D Data Processing, Visualization, and Transmission, June 2006, pp.798-805.

[110] O. Faugeras, B. Hotz, H. Matthieu, T. Vieville, Z. Zhang, P. Fua, E. Theron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy, “Real time correlation-based stereo: algorithm, implementations and applications,” INRIA Technical Report 2013, 1993.

[111] H. K. Nishihara, “Real-time stereo- and motion-based figure-ground discrimination and tracking using LOG sign-Correlation,” in Proc. 27th Asilomar Conf. Signals, Systems, and Computers, 1993, pp. 95-100.

[112] S. Park, H. Jeong, "Real-time stereo vision FPGA chip with low error rate," in Proc. Int'l Conf. Multimedia and Ubiquitous Engineering, April 2007, pp.751-756.

[113] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nister, "Real-time global stereo matching using hierarchical belief propagation," in Proc. The British Machine Vision Conference, 2006.

[114] S. Park, C Chen, and H Jeong. “VLSI architecture for MRF based stereo matching,” Lecture Notes in Computer Science, vol. 4599, pp.55-64, Aug. 2007.

[115] T.-H. Tsai, N. Chang, and T.-S. Chang, “Data reuse analysis of local stereo matching,” in Proc. Int’l Symp. Circuits and Systems, May 2008, pp. 812-815.

[116] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda. “Near real-time stereo based on effective cost aggregation,” in Proc. Int’l Conf. Computer Vision and Pattern Recognition, 2008.

[117] M. Gong and R. Yang, "Image-gradient-guided real-time stereo on graphics hardware," in Proc. Fifth Int’l Conf. 3-D Digital Imaging and Modeling, 2005, pp. 548-555.

[118] Z. Wang and Z. Zheng. A region based stereo matching algorithm using cooperative optimization. CVPR 2008.

[119] S. Wattson, "NVIDIA's GeForce 7800 GTX graphics processor," The Tech Report, June 2005. [online] http://techreport.com/articles.x/8466/5