

國立交通大學

資訊工程學系

碩士論文



視訊編碼器在雙核心平臺上的最佳化

**Video Codec Optimization for Dual-core
Architectures**

研究生：曾建堂

指導教授：蔡淳仁 教授

中華民國九十三年六月

視訊編碼器在雙核心平臺上的最佳化

Video Codec Optimization for Dual-core Architectures

研究生：曾建堂

Student : Chien-Tang Tseng

指導教授：蔡淳仁

Advisor : Chun-Jen Tsai

國立交通大學

資訊工程學系



A Thesis

Submitted to Department of Computer Science and Information Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science and Information Engineering

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

博碩士論文授權書

本授權書所授權之論文為本人在交通大學(學院)資訊工程系所
組 九十三學年度第二學期取得碩士學位之論文。

論文名稱：視訊編碼器在雙核心平臺上的最佳化

指導教授：蔡淳仁 教授

1. 同意 不同意

本人具有著作財產權之上列論文全文(含摘要)資料，授予行政院國家科學委員會科學技術資料中心(或改制後之機構)，得不限地域、時間與次數以微縮、光碟或數位化等各種方式重製後散布發行或上載網路。

本論文為本人向經濟部智慧財產局申請專利(未申請者本條款請不予理會)的附件之一，申請文號為： ，註明文號者請將全文資料延後半年再公開。

2. 同意 不同意

本人具有著作財產權之上列論文全文(含摘要)資料，授予教育部指定送繳之圖書館及國立交通大學圖書館，基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會及學術研究之目的，教育部指定送繳之圖書館及國立交通大學圖書館得以紙本收錄、重製與利用；於著作權法合理使用範圍內，不限地域與時間，讀者得進行閱覽或列印。

本論文為本人向經濟部智慧財產局申請專利(未申請者本條款請不予理會)的附件之一，申請文號為： ，註明文號者請將全文資料延後半年再公開。

3. 同意 不同意

本人具有著作財產權之上列論文全文(含摘要)，授予國立交通大學與台灣聯合大學系統圖書館，基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會及學術研究之目的，國立交通大學圖書館及台灣聯合大學系統圖書館得不限地域、時間與次數，以微縮、光碟或其他各種數位化方式將上列論文重製，並得將數位化之上列論文及論文電子檔以上載網路方式，於著作權法合理使用範圍內，讀者得進行線上檢索、閱覽、下載或列印。

論文全文上載網路公開之範圍及時間 -

本校及台灣聯合大學系統區域網路： 年 月 日公開

校外網際網路： 年 月 日公開

上述授權內容均無須訂立讓與及授權契約書。依本授權之發行權為非專屬性發行權利。依本授權所為之收錄、重製、發行及學術研發利用均為無償。上述同意與不同意之欄位若未鈎選，本人同意視同授權。

研究生簽名：

學號：9117559

(親筆正楷)

(務必填寫)

日期：民國 年 月 日

國家圖書館博碩士論文電子檔案上網授權書

本授權書所授權之論文為本人在交通大學(學院)資訊工程系所
組 九十三 學年度第 二 學期取得 碩士 學位之論文。
論文名稱：視訊編碼器在雙核心平臺上的最佳化

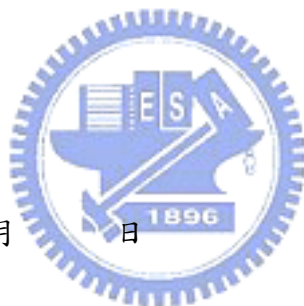
指導教授：蔡淳仁 教授

同意 不同意

本人具有著作財產權之上列論文全文(含摘要)，以非專屬、無償授權國家圖書館，不限地域、時間與次數，以微縮、光碟或其他各種數位化方式將上列論文重製，並得將數位化之上列論文及論文電子檔以上載網路方式，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

上述授權內容均無須訂立讓與及授權契約書。依本授權之發行權為非專屬性發行權利。依本授權所為之收錄、重製、發行及學術研發利用均為無償。上述同意與不同意之欄位若未鈎選，本人同意視同授權。

研究生簽名：
(親筆正楷)



學號：9117559
(務必填寫)

日期：民國 年 月 日

視訊編碼器在雙核心平臺上的最佳化

學生：曾建堂

指導教授：蔡淳仁 教授

國立交通大學資訊工程學系（研究所）碩士班

摘要

在本篇論文中，我們提出一個方法使 MPEG-4 Simple Profile 視訊編碼器在雙核心(RISC 以及 DSP)平臺上的執行能更具效能。在目前視訊編碼器對於 RISC 核心以及 DSP 核心的使用，運算重心是以 DSP 核心為主。但隨著 RISC 運算能力的強化，未來 RISC 核心也將有足夠的能力來處理繁重的運算。因此，我們藉著評估分析視訊編碼器中各主要元件的運算特性，建立出一套能動態分配工作至各運算單元使之平行運算的雙核心視訊架構。而為了解決雙核心之間資料傳輸的負擔，該架構中也使用 DMA 的機制來改進效能。而從實作結果證實，在使用我們的雙核心視訊架構後，視訊編碼器效能將因此提升。

Video Codec Optimization for Dual-core Architectures

student : Chien-Tang Tseng

Advisors : Dr. Chun-Jen Tsai

Department (Institute) of Computer Science and Information Engineering
National Chiao Tung University

Abstract

In this paper, we propose a dynamic task partitioning framework on dual-core architecture (RISC and DSP) for the MPEG-4 Simple Profile video codec. Using a dynamic task scheduler, an efficient dynamic partitioning framework of video encoder algorithm on dual-core architecture are developed. Existing practices of embedded software development on a dual-core platform either assign a subtask to the RISC core or the DSP core. However, since new generations of RISCs are powerful enough for computationally intensive task as well, the proposed framework will invoke both the RISC and the DSP cores in parallel to complete a single subtask in a tightly-coupled manner. To alleviate the communication overhead between the two cores, DMA is used to transfer data between the MCU and the DSP. From the experiments, it is shown that the proposed approach achieves higher performance than the conventional approach where only one of the cores (either MCU or DSP) is used for each subtask.

Acknowledgement

能完成這篇論文，要感謝的人很多，首先要感謝的是我的指導老師 蔡淳仁 教授，感謝教授提供了豐富的資源以及經驗、技術上的協助，讓我的研究能順利完成。再來要感謝的是家人經濟上的支援，讓我在研究所生涯中能努力學習，無後顧之憂。也感謝同學以及朋友的及時雨，讓我在研究上遭遇瓶頸時，能勇於面對不被打倒。最後，感謝交大資工，這裡的研究風氣讓我成長，讓我更有信心來面對未來的挑戰。



Index of content

摘要.....	i
Abstract.....	ii
Acknowledgement	iii
Index of content	iv
List of figures.....	v
List of tables.....	vii
Chapter 1 Introduction	1
1.1 Introduction to the OMAP 1510 platform.....	1
1.2 Introduction to MPEG Video Codec.....	5
Chapter 2 Previous work.....	9
Chapter 3 The Proposed Framework	13
3.1 Dual-core processing architecture.....	14
3.2 Pre-processing and Post-processing.....	16
3.3 Intra-frame processing	17
3.3.1 Intra macroblock encoding	18
3.4 Inter-frame processing	26
3.4.1 Interpolation processing.....	27
3.4.2 Motion estimation.....	30
3.4.3 Inter macroblock encoding	37
Implementation.....	41
Chapter 4 using DSP Hardware Extension for Video Coding	41
4.1 FDCT module and IDCT module	41
4.2 Interpolation module.....	44
4.3 Motion estimation module	46
Chapter 5 Experimental results	49
5.1 Experiment of Intra frame processing.....	49
5.1.1 Overall result.....	49
5.1.2 Profile for dual-core I macroblock encoding module.....	53
5.2 Experiment of Inter frame processing.....	55
5.2.1 Overall results	55
5.2.2 Profile for dual-core Interpolation module	59
5.2.3 Profile for dual-core Motion Estimation module.....	61
5.2.4 Profile for dual-core P macroblock encoding module	63
Chapter 6 Conclusions and Future Works.....	66
Chapter 7 References	67

List of figures

Fig 1 The Innovator OMAP development board made by PSI.....	2
Fig 2 OMAP 1510 Architecture	3
Fig 3 OMAP 1510 function block diagram	4
Fig 4 MPEG4 encoding scheme	6
Fig 5 MP3 decoding system on dual-core architecture.....	9
Fig 6 DMA and dual buffer mechanism on DSP	10
Fig 7 Multi-core architecture	12
Fig 8 Our MPEG4 encoding scheme	13
Fig 9 Fetch data on ARM and DSP.....	14
Fig 10 Main phases of dual-core execution flow.....	15
Fig 11 Pre-processing and Post-processing	17
Fig 12 Intra macroblock encoding	19
Fig 13 Dual-core intra macroblock encoding scheme	20
Fig 14 Control module	22
Fig 15 Protection of control module	22
Fig 16 DSP interface architecture	23
Fig 17 Improvement from DMA support.....	24
Fig 18 DMA architecture for dual-core intra macroblock encoding.....	25
Fig 19 Dual-core interpolation processing scheme.....	27
Fig 20 Example of interpolation processing on DSP.....	29
Fig 21 DMA architecture for dual-core interpolation processing	30
Fig 22 Dual-core motion estimation scheme	31
Fig 23 Mode decision module.....	33
Fig 24 Four step hierarchy search algorithm	34
Fig 25 DMA architecture for motion estimation processing	35
Fig 26 Motion estimation with on-the-fly architecture on DSP	36
Fig 27 Inter macroblock encoding scheme	37
Fig 28 Execution flow in Inter macroblock encoding scheme	39
Fig 29 DMA architecture for dual-core P MB encoding.....	40
Fig 30 Format conversion	43
Fig 31 Macroblock encoding with built-in hardware extension support	44
Fig 32 Interpolation processing with built-in hardware extension support ...	46
Fig 33 Motion estimation with built-in hardware extension support.....	48

Fig 34 Profile for dual-core I macroblock encoding module..... 53
Fig 35 profile for dual-core Interpolation module 60
Fig 36 Profile for dual-core Motion Estimation module 61
Fig 37 Profile for dual-core P macroblock encoding module 64
Fig 38 Architecture of future work 66



List of tables

Table 1	Combination of transform coding	10
Table 2	Intra frame encoding result	18
Table 3	Description of dual core I MB encoding module.....	20
Table 4	Specification of DSP I macroblock encoding	21
Table 5	Inter-frame processing result.....	26
Table 6	Description of DSP interpolation processing module.....	28
Table 7	Specification of DSP interpolation processing.....	28
Table 8	Description of dual core motion estimation processing module.....	31
Table 9	Specification of DSP motion estimation processing	32
Table 10	Specification of DSP ME processing with interpolation on the fly	36
Table 11	Description of dual-core P MB encoding module.....	38
Table 12	Specification of DSP P MB encoding	38
Table 13	Specification of FDCT with HW extensions.....	42
Table 14	Specification of IDCT with HW extensions	42
Table 15	Specification of Interpolation with HW extensions	45
Table 16	Specification of ME in HW extensions.....	47
Table 17	Setup of experiment environment	49
Table 18	Experiment result of pure ARM core	49
Table 19	Experiment result of pure DSP core.....	50
Table 20	Experiment result of pure DSP core, FIQ	50
Table 21	Experiment result of pure DSP core, FIQ, HW extensions.....	51
Table 22	Experiment result of dual-core.....	51
Table 23	Experiment result of dual-core with DMA	52
Table 24	Performance comparison.....	52
Table 25	Experiment result of profile I MB encoding.....	53
Table 26	Calculated result of profile P MB encoding.....	54
Table 27	Experiment result of pure ARM core	55
Table 28	Experiment result of pure DSP core.....	56
Table 29	Experiment result of pure DSP core, FIQ	56
Table 30	Experiment result of pure DSP core, FIQ, HW extensions.....	57
Table 31	Experiment result of dual-core core.....	57
Table 32	Experiment result of dual-core with DMA	58

Table 33 Experiment result of dual-core, DMA, enhanced ME	59
Table 34 Performance comparison.....	59
Table 35 Experiment result of profile interpolation module.....	60
Table 36 Calculated result of profile interpolation module	61
Table 37 Experiment result of profile Motion estimation module.....	62
Table 38 Calculated result of profile Motion estimation module	62
Table 39 Experiment result of profile Motion estimation module.....	63
Table 40 Calculated result of profile Motion estimation module	63
Table 41 Experiment result of profile P MB encoding	64
Table 42 Calculated result of profile P MB encoding.....	65
Table 43 Experiment result of profile P MB encoding	65
Table 44 Calculated result of profile P MB encoding.....	65



Chapter 1 Introduction

MPEG-4 Simple Profile (SP) is a visual coding standard which is suitable for low bitrate, low delay applications such as those for mobile phones. The hardware architecture of a mobile phone is usually composed of a MCU for light-weight control tasks plus a DSP for computationally intensive tasks. DSP has been a crucial component in mobile devices due to its excellent power/performance ratio for signal processing tasks. However, new generations of mobile multimedia applications involves complex blending of sophisticated control tasks and data-processing task. In the meanwhile, the capability of the RISC core in the MCU has become more and more powerful. Therefore, executing the whole signal processing tasks on DSP alone may not be the most cost efficient ways anymore.

In this thesis, a dynamic task-partitioning dual-core framework for MPEG-4 video SP encoder is presented. The TI OMAP processor comprised of a 16-bit DSP and a 32-bit RISC ARM core is used as the target architecture. The efficiency of the proposed system is obtained by utilizing both processor cores in parallel to complete each codec task (e.g. motion estimation). The ratio of task division between the RISC core and the DSP core is determined dynamically at runtime by a control module. In order to reduce the transfer overhead between the RISC core and the DSP core, a DMA is used to move the data among various memory banks.

The organization of this thesis is as follows. In the rest of chapter 1, the architecture of the TI-OMAP 1510/5910 processor and the PSI Innovator development board used in this thesis is introduced. Chapter 2 presents some previous works on dual core implementation of media codecs. The proposed dynamic task-partitioning dual-core framework will be described in chapter 3. Chapter 4 discusses the implementation of some of the codec modules using the hardware extension of the TI C5510 DSP core. In chapter 5, some experiment results are reported. Finally, chapter 6, conclusions and future work will be given.

1.1 Introduction to the OMAP 1510 platform

As the wireless industry moves into a new century of differentiated services, developer are seeking for better platforms for developing 2.5G and 3G wireless applications. The Open Multimedia Application Platform (OMAP) which combines

high-performance, power-efficient processor cores with easy-to-use, open software architecture is an intriguing platform for new mobile multimedia applications. These features provide a powerful hardware and software foundation for the development of innovative applications, and they help simplify development and thus save time-to-market for new embedded system products.



Fig 1 The Innovator OMAP development board made by PSI

The OMAP1510 device is efficient in performance and power consumption for wireless multimedia devices. The ARM core is well suited for handling control code, such as user interface, OS and applications. The DSP core is better suited for real-time multimedia signal-processing.

The C55x DSP core architecture includes some extensions of the core functions for multimedia-specific operations. C55x devices are the first family of TI DSPs with such core-level multimedia extensions. The extensions include, Motion estimation, discrete cosine transforms (DCT/IDCT), and pixel interpolation. Software developers access the multimedia extensions through coprocessor-specific instructions that have been added to the general C5500 instruction set. The combination of coprocessor and general arithmetic instructions will get efficient execution and better performance. These features will be examined in our implementation.

The OMAP1510 device, shown in Fig. 1, is based on two integrated microprocessor cores: a C55x DSP and a high-performance ARM 9 core. There are on chip caches for both processors which can reduce average fetch times to external memory and eliminate the power consumption of unnecessary external accesses. And the memory management units (MMU) for both cores provide virtual-to-physical

memory translation and task-to-task protection. Low-power operating modes are available to conserve power during periods when the OMAP device is not used frequently or is not in use.

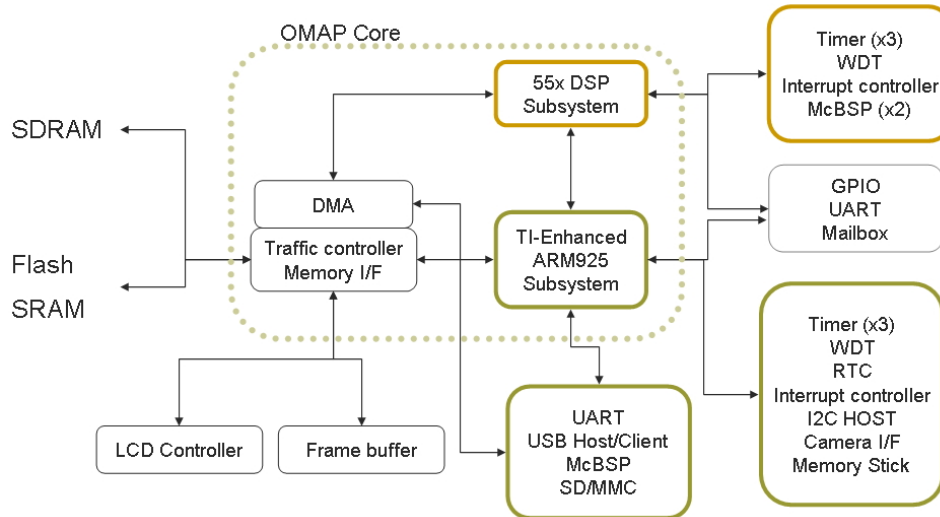


Fig 2 OMAP 1510 Architecture

In embedded systems, the I/O bottleneck is an important issue. The OMAP platform adopts many techniques for improved I/O performance. There are two external memory interfaces and one single internal memory port in the OMAP platform. The first external interface supports a direct connection to synchronous DRAM at up to 100 MHz; and the second external interface supports standard asynchronous memories systems such as SRAM, FLASH or burst Flash devices. This interface is typically used for program storage and can be configured as either 16- or 32-bit wide. The internal memory port allows direct connection to on chip SRAM or embedded Flash memory and can be used to save time and power for frequently accessed data, such as critical OS routines or the LCD frame buffer. Finally, all three interfaces are completely independent and allow concurrent access from either cores or the DMA units.

The OMAP platform also contains many interfaces for connecting to peripherals or external devices. Each processor has its own external peripheral interface, which supports both direct connections to peripherals and DMA from the processor's DMA unit. The local bus interface is high-speed and bi-directional, and the controller of the bus can be used to connect to external peripherals or additional OMAP devices. Additionally, a high-speed access bus is available to allow an external device to share the main OMAP system memory, both external and internal. And in order to support common OS requirements, the OMAP platform includes on-chip peripherals such as timers, general-purpose I/O, a UART, and watchdog timers. A color LCD controller is

also included to support a direct connection to the LCD panel. The ARM DMA unit contains a dedicated channel used to transfer data to the LCD controller from the frame buffer, which can be allocated in either the external SDRAM or the internal SRAM. The follow functional block diagram shows the detail architecture of OMAP architecture.

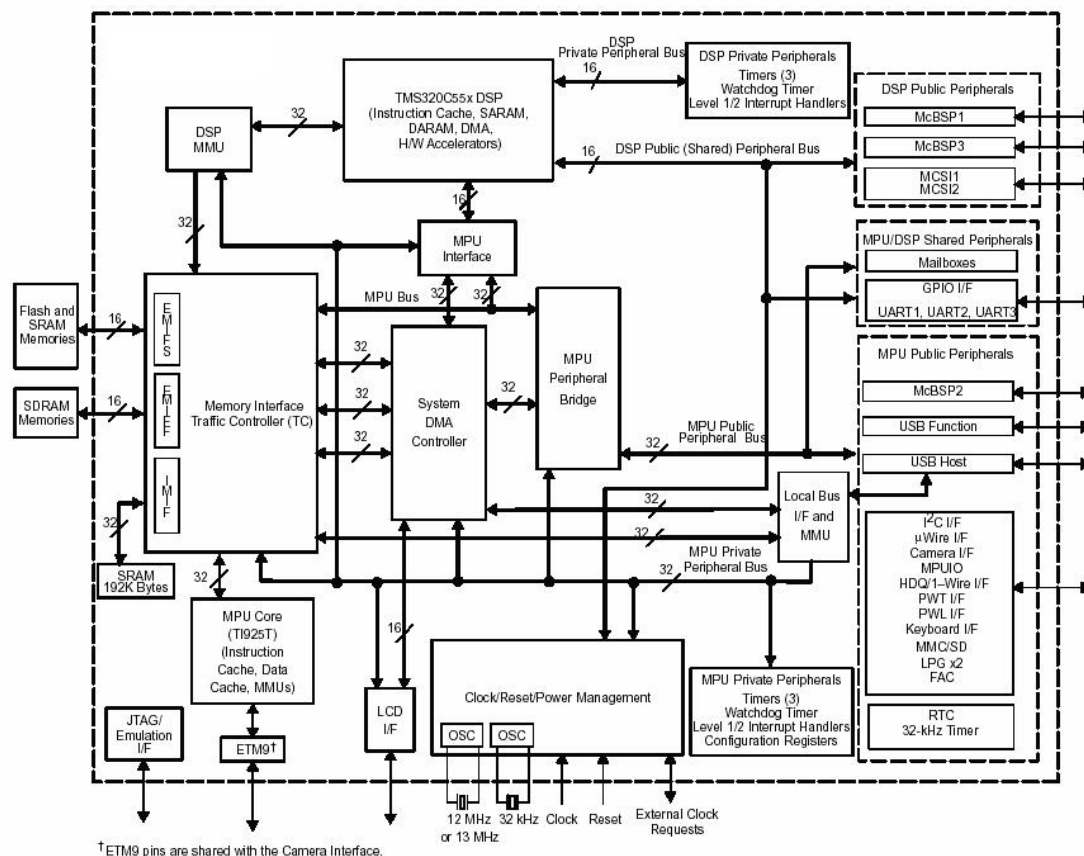


Fig 3 OMAP 1510 function block diagram

The OMAP platform includes an open software architecture that is needed to support application development and provide a dynamic upgrade capability for heterogeneous multiprocessor system designs. This architecture includes a framework for developing software, which targets system design and application programmer interfaces (API) for executing software on the target system. Additionally, in order to simplify software development, the DSP software architecture was abstracted from the general-purpose programming (GPP) environment. In the OMAP platform, this abstraction is accomplished by defining an architectural interface that allows the GPP to be the system master. And the DSPBridge interface consists of a set of APIs that contain device driver interfaces which called DSPBridge interface. In our own design, the architecture of DSPBridge interface has been referenced and thus implement our proposed codec on dual-core architecture OMAP platform.

1.2 Introduction to MPEG Video Codec

MPEG stands for "Moving Pictures Experts Groups". It is a group working under the directives of the International Standards Organization (ISO) and the International Electro-Technical Commission (IEC). The groups work concentrates on defining standards for the coding of moving pictures, audio and related data. MPEG video compression is used in many current and emerging products. It is at the heart of digital television set-top boxes, DSS, HDTV decoders, DVD players, video conferencing, Internet video, and other applications. These applications benefit from video compression in the fact that they may require less storage space for archived video information, less bandwidth for the transmission of the video information from one point to another or a combination of both.

MPEG-1 defines a framework for coding moving video and audio, significantly reducing the amount of storage with minimal perceived difference in quality. In addition a System specification defines how audio and video streams can be combined to produce a system stream. This forms the basis of the coding used for the VCD format. MPEG-2 builds on the MPEG-1 specification, adding further pixel resolutions, and support for interlace picture, better error recovery possibilities, more chrominance information formats, non-linear macroblock quantization and the possibility of higher resolution DC components.

MPEG-4 is good for both low and high bit-rate applications, since it has good error resilient coding and is capable of handle high quality video. The error resilient coding and low bit-rate capabilities can be utilized for instance in mobile phones or handheld computers with a wireless network connected to it. Other areas of usage can be where you want both high and low bit-rate video, like on the web, where you want to show a movie to the visitors.

Following MPEG-4 is the not yet finalized standard MPEG7. Metadata is added to the content of the multimedia, that is names or labels are added to the objects introduced in MPEG-4. This will allow advanced searching for certain content within an MPEG-7 encoded media. With the ever so fast growing amounts of available media this will become necessary sooner or later. There are drafts and proposals for the standard but it has not yet been finalized. MPEG-21, the work on this standard started in June 2000. It is aimed to identify and define the key elements needed to support the multimedia delivery chain.

MPEG-4 provides a large and rich set of tools for the coding of audio-visual objects. In order to allow effective implementations of the standard, subsets of the MPEG-4 Systems, Visual, and Audio tool sets have been identified, that can be used for specific applications. These subsets, called ‘Profiles’, limit the tool set a decoder has to implement. For each of these Profiles, one or more Levels have been set, restricting the computational complexity.

Generally speaking, MPEG-4 Simple Visual Profile provides efficient, error resilient coding of rectangular video objects, suitable for applications on mobile networks. It consists of several modules, including intra coding (I-VOP), inter coding (P-VOP), motion compensation, resynchronization, variable length coding (VLC). And it is compatible of H.263 baseline coding.

Fig 4 is the block diagram of a MPEG-4 encoder. The details functionality of the blocks is described in the following paragraph.

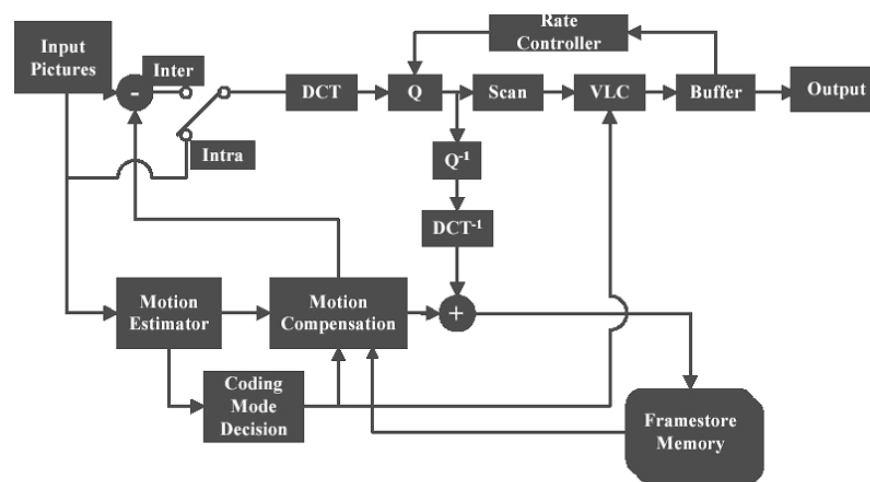


Fig 4 MPEG4 encoding scheme

The “DCT” block stands for “Discrete Cosine Transform”, which is a function that transforms image data in YCbCr format into frequency domain representation. After the transform, the same information is presented in a form that is more suitable for compression. In effect, the image is represented in a range of frequency components, where higher frequency components denote 'sharper' edges and changes in the image and lower frequency components denote more gradual changes in the image.

The “Q” block stands for “Quantization”, which is a process whereby values after the DCT block are divided by the quantization factor. The higher the factor, the higher the compression ratio (and lower the image quality). The quantization factor is either selected by the user or by the automatic rate control algorithm that is responsible for

ensuring that the amount of data generated by the encoder is within the bounds set by the transmission channel or storage device. The general aim is to provide the best image quality that is permissible by the transmission channel or by the video quality selected by the user on a storage device. The coefficients divided by the quantization factor are then rounded according to certain rules, and as a result the higher the quantization factor, the more often the result of the division is zero. This phase in the process causes data rates to drop dramatically because the higher frequency components (depending on the quantization factor and the image content) end up being rounded to zero. The benefit of carrying out the quantization in the frequency domain, as opposed to in the XY spatial domain, is that discarding the higher frequency components compromises the sharp edges and contrast of the image (more or less depending on quantization), which, when done within reason, is not easily perceived by the human eye.

Motion estimation is an important part of the codec. Whereas DCT and quantization serve to compress the spatial aspects of one frame, motion estimation is used to compress temporal redundancy, i.e., in the time domain over two consecutive frames. Take a typical scene, from a feature film for example. 25-30 consecutive frames are displayed every second by the TV or film projector. However, chances are that within an arbitrary one second sequence most aspects of the image remain the same. The background may not change at all, the characters remain the same and therefore the consecutive frames are very similar.

The motion estimation performs a delta analysis between two consecutive frames and determines whether areas of the image have changed or moved between the frames. In many cases an area stays exactly as it was in the previous frame and therefore it is sufficient for the encoder to inform the decoder to display this area as it was in the previous frame. If the area moves in a certain direction, the motion estimation algorithm directs the decoder to use the same piece of image as in the previous frame, but to move it a certain amount in a defined direction. In practice this will be accomplished by sending motion vectors within the MPEG4 bit stream. These vectors will guide the decoder in choosing the appropriate portions of the previously decoded frame to be used in the reconstruction of the current frame.

It should be clear that this vastly increases the compression rates. In fact, some types of content can be compressed to an enormous extent due to the lack of movement in the image. One example is the 'talking head' type of content, such as a newscaster, which results in a very compact MPEG4 stream. As one may expect, the motion estimation is a very computationally intensive function. Searching through an image for all the possible areas that could change place for every potential location requires a lot

of calculations. If we are concerning of real time application, there exists a very restricted time limit to achieve real time constraint, only 1/15th to 1/30th of a second to do this before the next frame arrives for processing.

Motion compensation performs the reconstruction of the frame based on the received motion vectors, received delta frame data (different data between two consecutive frames, see motion estimation) and the previously decoded image. So, if delta data is received, the current frame is reconstructed by adding the delta frame data to the data from the previously decoded frame in the specific locations indicated by the received motion vectors.

As illustrated previously, the combination of motion estimation, cosine transform and quantization alone can dramatically reduce the bit rate needed for digital video. However, a few more steps are required before full MPEG4 compressed video stream is at hand. As stated, MPEG4 compression works in a block level, i.e., 8 x 8 pixel blocks (or matrices) are compressed at the same time. The output coming from the quantization function is further processed by a Zigzag scan coder. This coder forms a 64 (8 x 8 pixel) element long vector out of the 8 x 8 matrix so that the low frequency components will be placed at the front of the vector. The reason for this action becomes useful in the later stages of processing.

Next, the 64 element long vector is analyzed by the run-length encoder module. The run-length encoder calculates the number of consecutive zeros in the vector and forms Run-Length Code (RLC) words based on the calculation. As noted before, after the quantization there is a high probability of a significant amount of zeros in the 8 x 8 matrix (likely to be in the high frequency components) and there is no need to transmit or store such information. So, one RLC word represents the number of zeros between the consecutive non-zero elements in the vector. Also, the value of the last non-zero element after the zeros is represented in one RLC word as well as the information as to whether this value was the very last component in the vector. Thus, each RLC word consists of three components.

The RLC words are then Huffman-coded using Variable Length Codes (VLC). Basically this means that certain RLC words are given a specific bit pattern. The most common RLC words are given the shortest VLC bit pattern. The VLC patterns are specified in the MPEG4 standard and were generated based on the vast amount of video test material.

Chapter 2 Previous work

The multimedia processing ability and the implementation of a MPEG-4 SP codec on the OMAP platform is described in [3]. However, the system described in [3] utilizes only the DSP core for the codec. Even though the performance numbers published in [3] are higher than the numbers we have achieved, it is probably due to extensive use of assembly codes and a more efficient C model to begin with. With similar optimization, it should be possible for the proposed method to achieve even better performance. Another codec framework for dual-core architecture is described in [4], where an MP3 decoding system is implemented. In this paper, MP3 decoding algorithm runs on DSP core, and RISC core acts as the master of the system. The RISC core receives commands such as: play, stop, previous, and next from the user interface and send a corresponding instruction to the DSP core, and the RISC core will fall into an idle status. Upon reception of an instruction, the DSP core will execute a function to complete the requested task. Finally, when DSP core finished the task, it will assert an interrupt to the RISC core to report current status. Fig. 5 shows the software architecture of this work.

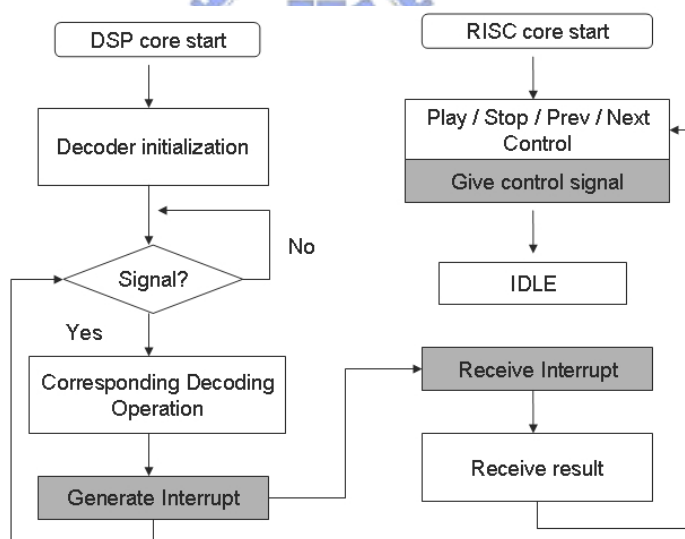


Fig 5 MP3 decoding system on dual-core architecture

As described in [4], the RISC core in this design play the operating system role, its main duty is to communicate with peripherals, including I/O devices and storage devices. The DSP core handle the complete MP3 decoding algorithm alone. This dual-core architecture is a typical architecture generally adopted on most applications.

At close examination, the characteristic of audio processing and video processing tasks is a little bit different. It is clear that audio processing are composed of mainly

signal processing tasks which need lots operation of multiplication and accumulate (MAC). But in video processing, in addition to traditional signal processing tasks, it also contains many sophisticated control operations such as the address generation unit of motion estimation, interpolation and motion compensation. With new generations of RISC cores for MCU, the cost/performance ratio for executing these tasks may not be that far away behind the DSP cores. Therefore, in this thesis, we investigate the efficiency of executing the codec tasks in parallel by both the RISC and the DSP cores.

In [5], an implementation of the H.263 visual codec is implemented on a TMS320C6201 DSP. In this implementation, DCT, Quantization, Dequantization, and IDCT are combined into a single module for efficient memory utilization. Table 1 shows the concept. This technique has been adopted in our implementation as well.

```

For(every macroblock in INTRA frame)
{
    DCT_and_Quantize_Macroblock(); //DCT-transform
    VLC_Code_INTRA_Macroblock_to_Stream(); //Coding
    IDCT_and_InverseQuantize_Macroblock(); //Reconstructing
}

```

Table 1 Combination of transform coding

In [5], the motion estimation search method is Parallel hierarchical one dimensional search (PHODS). This method has low computation complexity since the x-component and y-component of motion vector are computed in one dimensional space separately and later assembled into a complete motion vector. In our implementation, in order to take advantage of the built-in hardware motion estimation extension of TI C55x DSP, the four steps hierarchical search is used.

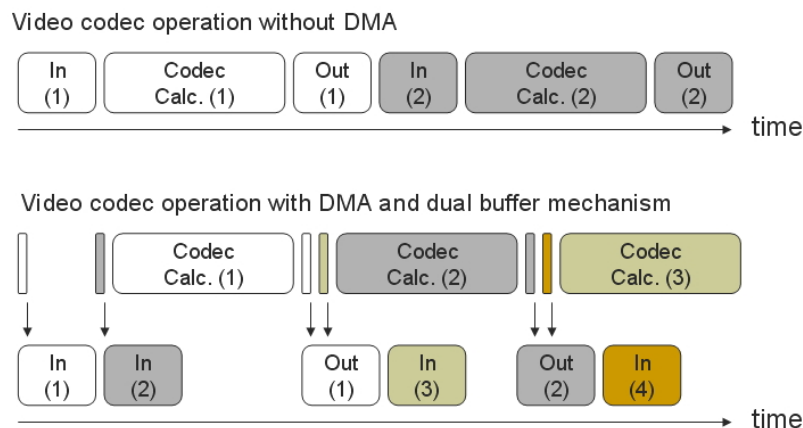


Fig 6 DMA and dual buffer mechanism on DSP

In [6], it presents an implementation of MPEG-4 video codec on a general DSP core (NEC uPD77210). The paper analyzes the effects of different motion search algorithms and discusses why DMA module can improve the transfer efficiency. The paper also presents a concept of dual buffer mechanism to avoiding waiting transfer on DSP core. The execution flow of its DMA module and dual buffer concept is shown in Fig. 6.

In [8], the same concept of DMA module and dual buffer mechanism on DSP core are mentioned again. Whenever there are dual buffers on the DSP core, the DSP core should execute a task using the data in one of the buffer while a DMA is preparing in parallel the data in the other buffer for the next task. As soon as the first job is completed, the DSP core can move on to the next job using the other buffer without waiting for the data.

A multi-core architecture is presented in [14]. The paper discusses how a multi-core architecture is useful for multimedia processing applications. Multi-core architecture is an attractive architecture for multimedia processing as multimedia tasks in general can be partitioned into stream oriented, block oriented, and DSP oriented functions, which can all be processed in parallel on different cores. Each core can be adapted towards a specific class of algorithms, and individual tasks can be mapped efficiently to the most suitable core. In general, parallelism can be employed at instruction level (e.g., very long instruction word, VLIW), data level (e.g., single instruction multiple data, SIMD), or task level (e.g., simultaneous multithreading). Another technique to accelerate multimedia processing is to adapt programmable processors to specific algorithms by introducing specialized instructions for frequent operations of higher complexity.

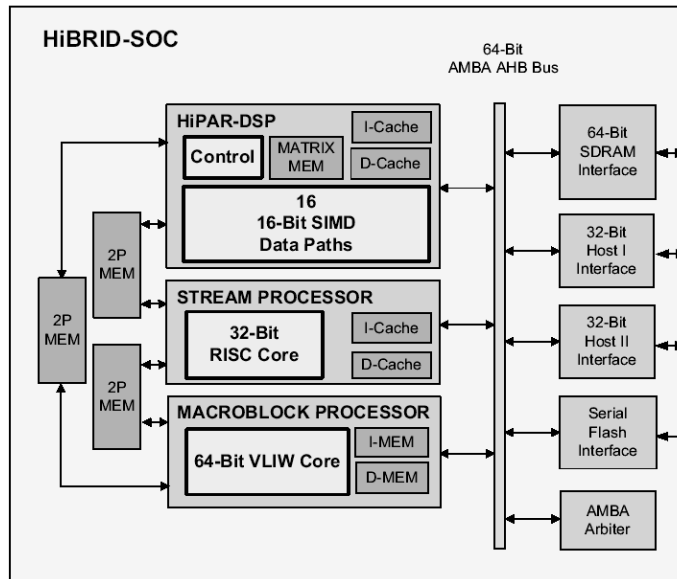


Fig 7 Multi-core architecture

The multi-core architecture proposed in [14] is shown in Fig. 7, comprises three programmable cores that have each been specifically optimized towards a particular class of algorithms by employing different architectural strategies. The HiPAR-DSP is a 16-datapath SIMD processor core controlled by a four-issue VLIW and is particularly optimized towards high-throughput two-dimensional DSP-style processing, such as FFT-intensive applications or filtering. The second core, the Stream Processor (SP), consists of a scalar 32-Bit RISC architecture that is more optimized towards control-dominated tasks such as bit stream processing or global system control with a particular focus on high-level language programmability. The Macroblock Processor (MP), finally, has been designed specifically for the efficient processing of data blocks or macroblocks that are typical for many video coding schemes. It has a heterogeneous data path structure consisting of a scalar and a vector unit controlled by a dual-issue VLIW, offers flexible sub word parallelism, and contains instruction set extensions for typical video processing computation steps. This architecture provides multi-core to let each processor to do suitable task for best utilization.

Chapter 3 The Proposed Framework

In this thesis, MPEG-4 Simple Profile (SP) encoder is used as the experimental target. MPEG-4 SP codec is a typical block-based motion compensated hybrid transform codec composed of several modules including temporal predictive coding, transform coding, quantization, and entropy coding. Some systems also implement pre-processing and post-processing. The follow figure is the system architecture of MPEG-4 Simple Profile visual encoder.

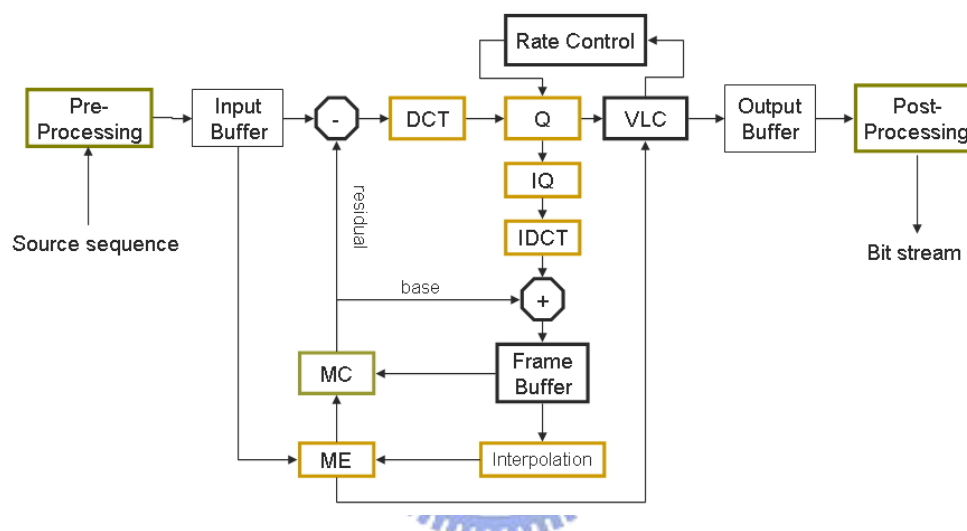


Fig 8 Our MPEG4 encoding scheme

The code for OMAP is ported from an implementation for Intel IA32 architecture. The original data structure of our codec processes frame data in 8-bit width pixels. However, the DSP core, TMS320C55x, of the OMAP platform accesses data in 16-bit width. Therefore, the data structure has to be modified accordingly to improve the performance. To be more specific, each time DSP fetches pixel data from memory for computation, it can't just fetch one pixel alone if data is arranged in 8 bits per pixel. In fact, it will also fetch the adjacent pixel. As a result, packing and unpacking between 8-bit pixel per pixel and 16-bit per pixel data formats has to be done whenever the ARM core is exchanging data with the DSP core. Fig. 9 represents the difference fetch behavior on ARM core and DSP core when the pixel of p2 needs to be processed.

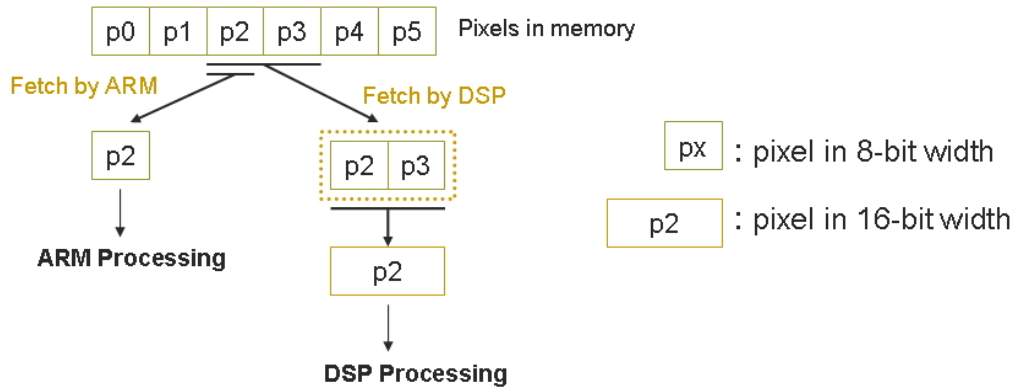


Fig 9 Fetch data on ARM and DSP

Another approach to solve the data access unit problem on OMAP is to use the hardware extension of the TI C55x DSP. In C55x, there is a built-in hardware extension module with the ability to process pixels in 8-bit width. However, if this strategy adopted, our codec may spending lots time on developing corresponding function module which uses the built-in hardware extension, and it will become harder to maintain and modify because of its readability. In spite of these concerns, it is no doubt that it will increase the efficiency. So, this is a tradeoff between developing time, memory utilization, maintain concern and efficiency.

3.1 Dual-core processing architecture

There are two computation cores, ARM and DSP, on the OMAP Innovator board. Before using DSP core to improve our codec for better performance, it is essential to design the dual-core control architecture first. The control architecture contains the communication mechanisms between ARM and DSP. In our proposed architecture, each type of subtasks in encoding a frame will be dispatched to ARM and DSP simultaneously and executed in parallel. There is no need to pre-arrange the execution order and the portion of tasks assigned to each core because the processing cores will work asynchronously. So that ARM core can execute its job continuously without waiting, and just need to handle the begin section and end section while corresponding DSP event happens in the encoding loop.

Although the detail architecture of each module may be different, the general architecture can be summarized in Fig. 10..

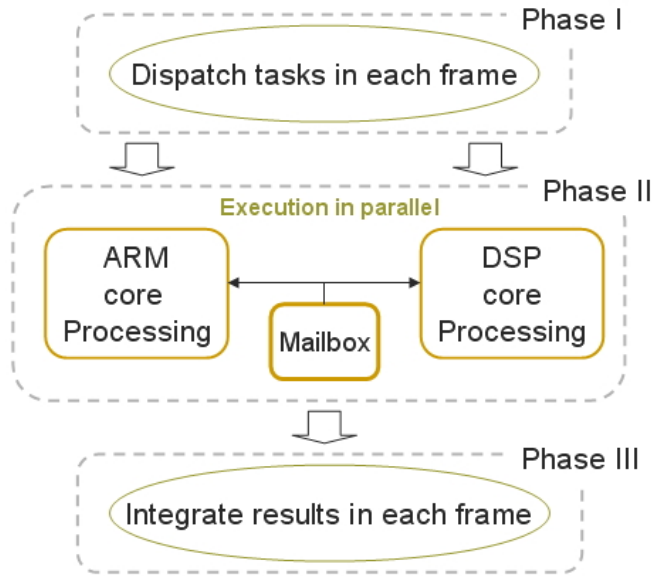


Fig 10 Main phases of dual-core execution flow

Generally speaking, each codec module can be divided into three phases when implemented on a dual-core platform. In phase I, task units in each frame will be defined; basically, a task unit can be defined as a block or a macroblock in each frame. After task unit has been defined, the operations of this task will be specified, such as motion estimation, interpolation, and sometimes a series processing which contains DCT, Quantization, de-Quantization, and IDCT.

In phase II of the figure, since jobs to ARM or DSP have been defined and assigned in phase I, then it will fetch data for computations. On the ARM side, source input data for computation can be fetched directly. But on the DSP side, it should transfer all input data from ARM side into DSP core's memory for computations. There are two transfer methods have been implemented on the Innovator board, one is transfer by CPU directly, and the other one is transfer by DMA support. Obviously, transfer data by DMA support will get better performance than transfer data by CPU. After each core has received its input for computation, they start execute their jobs in parallel. ARM core and DSP core can be communicated with each other by the mailbox to hold the progress of executions.

In phase III of this figure, after ARM and DSP have produced the computation results, then the dual-core module start to receive and integrate these results from each core at this stage. On the ARM side, the results can be integrated directly. On the DSP side, when DSP core has completed its job, it will asserts interrupt to ARM, and thus

control center can know the execution status of DSP core. An ARM interrupt routine will be invoked to receive computation results from DSP for integration. Again, using DMA module will improve the efficiency of transfer.

These three phases generally summarize the execution flow of our proposed dual-core architecture. How to perform this dual-core architecture to improve each computation intensive modules in our codec will be discussed detail in follow sections.

The main procedures for implementing these dual-core computation modules are described as follows. First, adjust data structure and execution flow of each module to fit dual-core architecture. It is because that if data structure doesn't be defined properly, it may just be able to execute one task at a time from the limit of control parameter, or any other resource conflictions. And about the execution flow, if each task's execution flow with deep dependency from other task unit's computation result, it may lead tasks execute sequentially, not in parallel.

Second, since operations of each function module are different, specific corresponding signals and interrupt handlers for each function module and its relational operations must define well. After define these control signals and handler routine, such information will be added into ARM core's interrupt architecture and DSP core's DSP interface which manage receiving ARM core's signal and assign job to corresponding function module.

Third, if DMA module is used to improve memory transfer in this dual-core processing framework, then one must also define corresponding interrupt handler, signals, and operations.

Following these procedure, almost all computation modules in the MPEG4 Simple Profile encoder can be constructed to execute efficiently on dual-core architecture.

3.2 Pre-processing and Post-processing

This section describes the pre-processing and post-processing in our codec, and the follow figure shows the detail behavior of these two processing. At this moment, our codec implement without operation system support, so that loading program and test sequences into the Innovator board is completely through the JTAG device. Loading complete input sequence from HOST (PC) to memory on Innovator board will be completed before encoding starts. This is because loading data through JTAG device

transfers slowly, if reading each input frame from HOST in encoding loop, and then it will spend most time on waiting transfer of frame. This condition should be resolved in the future by using different input channel such as USB or SD/MMC card. After completion of loading program and input sequence, adjust input sequence into proper format for our codec input is performed and besides, the bit width concern between ARM and DSP which mentioned before will also adjust here.

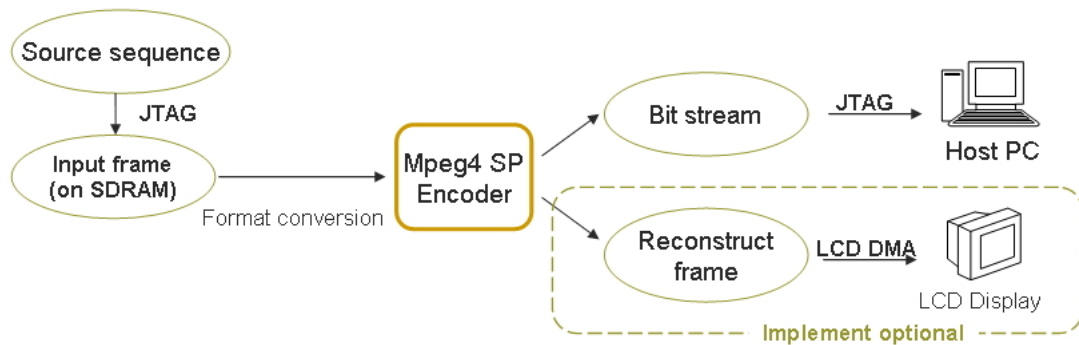


Fig 11 Pre-processing and Post-processing

About the post-processing block, after encoder has completed each frame in encoding loops, it can be optionally shows the reconstructed frame for present purpose by sending frame data into the LCD display frame buffer and then the LCD dedicated DMA will automatically transfer it to LCD display module and show the image on the screen. Since the output reconstructed frame was based on YUV format, and LCD display module was based on RGB module. As a result, in order to display, it should do conversions of our reconstructed frame from YUV color space into RGB color space.

Until after all of the encoding have finished, writing out the whole encoded bit stream from Innovator board to HOST through JTAG starts. This is also because the data transfer through JTAG is very slow, so this is a temporal method to avoid waiting.

3.3 Intra-frame processing

In the intra-frame processing, it encodes the input frame itself without any other reference frames. According to the experiment result which the encoder compresses sequential 150 frames in intra mode; the follow table shows the weightings between each module in the intra-frame processing. Clearly, the intra macroblock encoding (FDCT, Quantization, Dequantization, and IDCT) occupies most computation time of all. So, it is no doubt to improve this module with DSP support for better performance in the Intra-frame processing.

Table 2 Intra frame encoding result

Qcif,150 I frames	Execution time (ms)	Percentage
Initialization	236	0.735
Coding	4111	12.793
Sequence conversion	1684	5.241
Prediction	2631	8.190
DCT/Q/Q⁻¹/IDCT	22297	69.396
Total	30963	100
Encoding frame rate =4.7		

3.3.1 Intra macroblock encoding

In order to let ARM and DSP execute its own tasks parallel and smoothly, it is essential to define a task unit properly. As the Mpeg4 Simple Profile standard mentioned, each frame was divided into macroblocks for encoding. As a result, with adjust corresponding data structure and execution flow, and then, the encoder can dispatch tasks on macroblock level in scan line order in a frame.

After observe memory utilization from these four computation modules: FDCT, Quantization, Dequantization and IDCT, respectively. Since their memory utilization are relative between each other, it isn't reasonable to just divide them to run in DSP core individually, because it may cause memory transfer redundancy and wasting of time while transfer the source macroblock data to DSP and just do one of these four computation modules. As a result, our design is to combine these four computation modules into one function module called intra macroblock encoding for best memory utilization and thus reduce the transfer load.

The follow figure shows the execution flow and memory utilization in intra macroblock encoding.

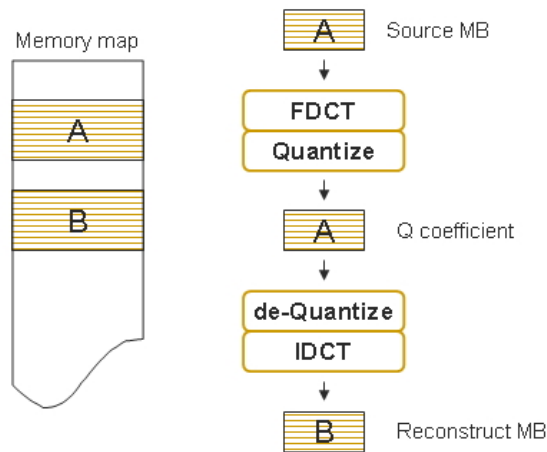


Fig 12 Intra macroblock encoding

Since task unit has been defined in intra macroblock encoding, and thus this module can start dispatch jobs and execute them. The follow figure is our proposed dual-core intra macroblock encoding architecture. And the follow two tables give some descriptions of the control flow and the specification which DSP core treated in this figure. At first, it divides input source frame into macroblocks as task units, and then dispatch the first task and macroblock data of this frame into DSP. And then ARM core will do next job and fall into an encoding loop until all tasks are completed. Since the first job has been dispatched to DSP, DSP core will start work according to the signal which ARM sent through mailbox. Each time while DSP core has completed its work, it will assert an interrupt to ARM core. And thus, ARM core's original job will be paused and fall into the interrupt handler to receive the computation result from DSP. After receiving and integrating the results from DSP, the control module will decide whether to send next job to DSP or not. And then ARM will resume its original job in encoding loop.

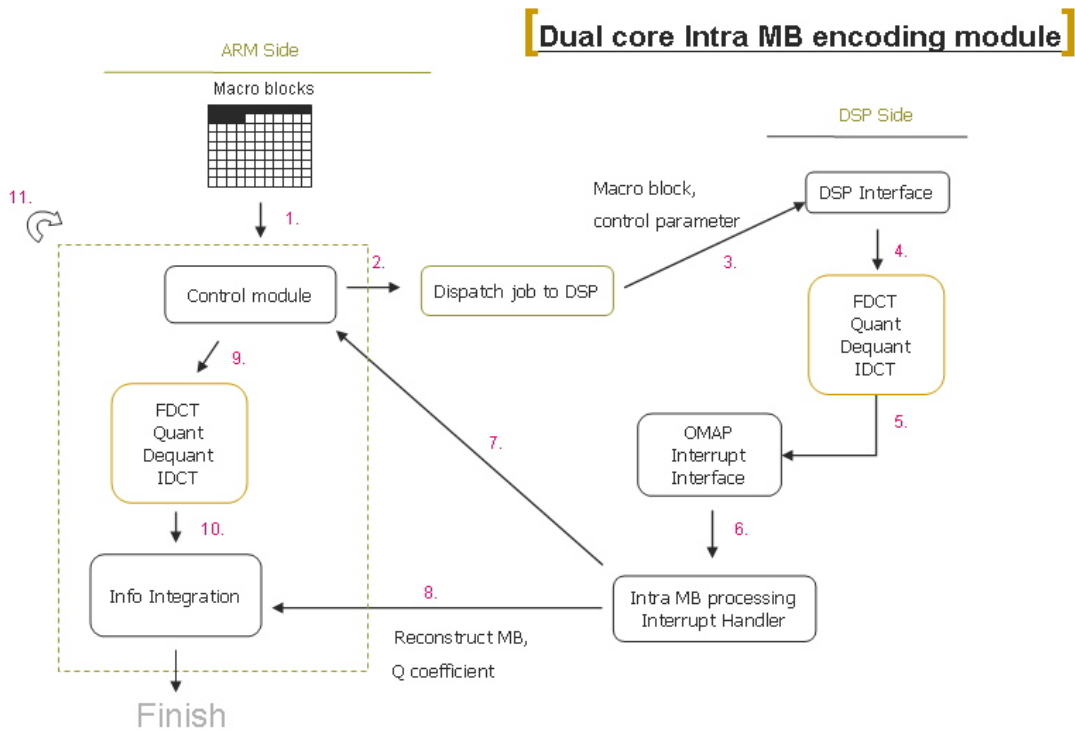


Fig 13 Dual-core intra macroblock encoding scheme

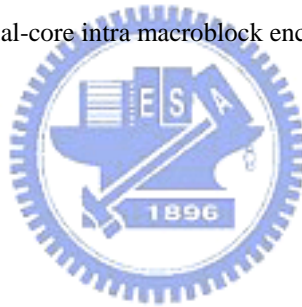


Table 3 Description of dual core I MB encoding module

Step	Description
1	Process the frame in scanline order
2	Control module decide to dispatch job to DSP
3	Transfer MB data and control parameter to DSP
4	DSP enters I MB encoding
5	After I-MB coding, DSP asserts an interrupt
6	MCU executes handler routine
7	Control module decides next step
8	Integrate result from DSP.
9	Control module decides to let MCU do I MB encoding
10	Integrate the result from ARM
11	The encoding loop repeats, until all jobs have been completed

Table 4 Specification of DSP I macroblock encoding

Specification of DSP I macroblock encoding	
DSP Input	<ol style="list-style-type: none"> 1. One source macroblock data <ul style="list-style-type: none"> ■ Four Y data blocks (8*8*4*2 bytes) ■ One U data blocks (8*8*2 bytes) ■ One V data blocks (8*8*2 bytes) 2. QP parameter for DSP quantization <ul style="list-style-type: none"> ■ 2 bytes
DSP output	<ol style="list-style-type: none"> 1. One Q coefficient macroblock data <ul style="list-style-type: none"> ■ Four Y data blocks (8*8*4*2 bytes) ■ One U data blocks (8*8*2 bytes) ■ One V data blocks (8*8*2 bytes) 2. One reconstructed macroblock data <ul style="list-style-type: none"> ■ Four Y data blocks (8*8*4*2 bytes) ■ One U data blocks (8*8*2 bytes) ■ One V data blocks (8*8*2 bytes)

The control module maintained here contains some important synchronous parameters such as global status, finish flag, ARM core status and DSP core status. Actually, the control module can be treated as a synchronous control table. Each time before job dispatched to DSP or ARM, the control module must be accessed first. In the control module, it checks the global progress status with the finish condition to decide whether to dispatch job to DSP or ARM. If it decides to dispatch job, then we will update DSP core or ARM core's status with corresponding parameter such as MB axis, CPU status, or even the profile information. By this way, after either ARM or DSP has completed its job, integration module could thus know how to integrate the computation result according to the status set before. The control module contains a finish flag, if the global status matches the condition of the finish condition, it won't dispatch jobs, and it will set the finish flag to announce ARM core, and then intra macroblock encoding finishes. With the support of control module, ARM and DSP can avoid to do the same jobs in the execution flow, and thus hold the execution progress exactly without out of control. The follow figure is the data structure and control flow of control module.

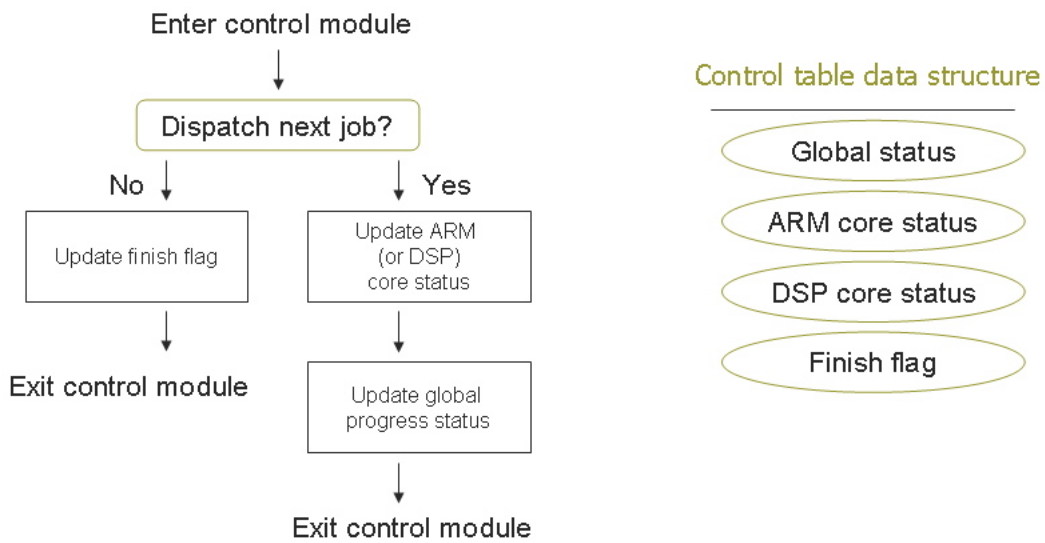


Fig 14 Control module

Control module plays an important role on this dual-core architecture; as a result, there should be a protection mechanism to promise that every time when either ARM core or DSP core accessing the control module, it can't be corrupted by any interrupt events, or that it will lead bad results. This is because interrupt may assert any time, if interrupt is asserted while control module are modifying the control parameter, it will cause that global status can't be updated successfully, and thus the same task may be dispatched more than one time.

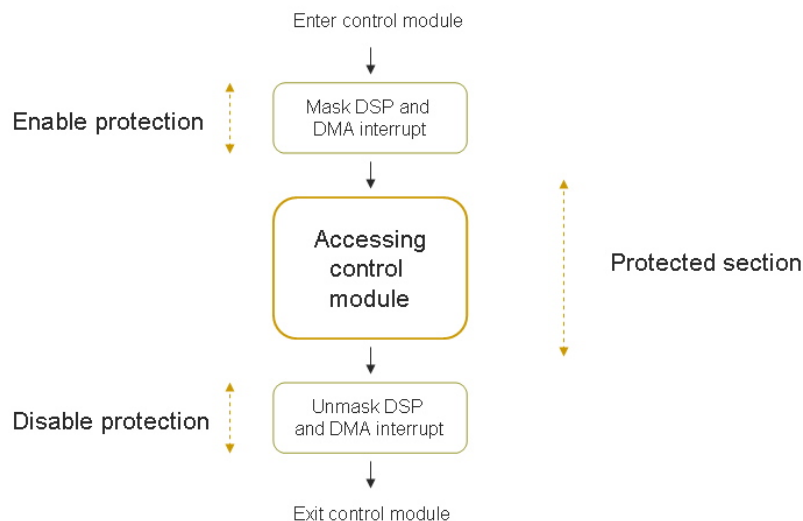


Fig 15 Protection of control module

The above figure shows protection scheme of the control module. Each time when control module will be accessed, it will mask corresponding interrupt first, this will

promise that control module can execute and update the global status without any bother. Until finishing accessing control module, corresponding interrupts will be unmask and work again.

Since that DSP has to do different functions for each need, construct an interface as a bridge between ARM and DSP is essential. This DSP interface is implemented like a shell; it starts when DSP core have enabled, it checks mailbox in repeat until ARM send signal through mailbox. The detail execution flow of the DSP interface is shown in the follow figure, which presents the relation between ARM core, DSP core, and DSP interface.

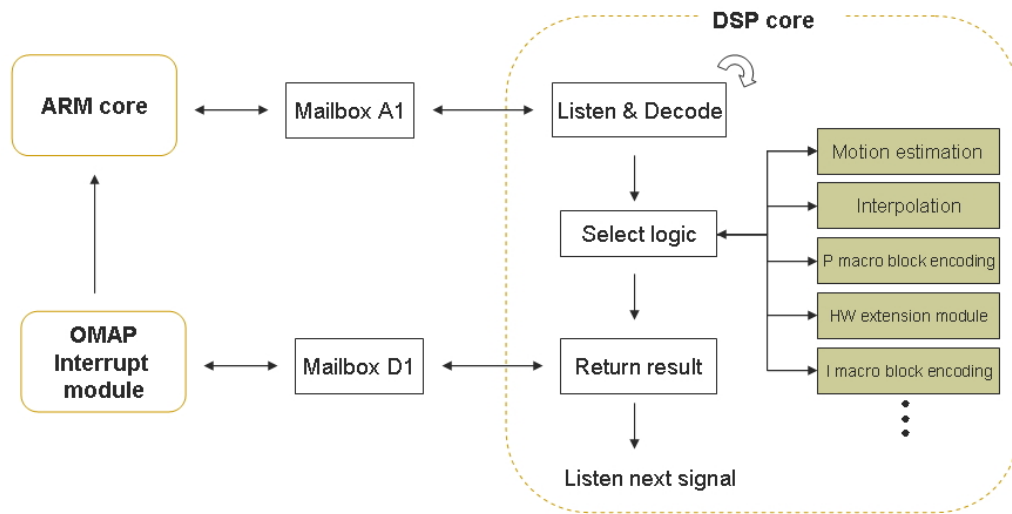


Fig 16 DSP interface architecture

Each time when ARM dispatches task for DSP, it will write signals into the mailbox which contains command for DSP. And then DSP interface will receive this signal, and it will decode the command in the mailbox immediately. After DSP realizes the content of the command, DSP will transfer its control into corresponding function module to handle tasks. After function module has completed its jobs, it returns the control to DSP interface. Then DSP interface write signal to the mailbox, and this will also assert an interrupt to ARM. Then, ARM will check the information in the mailbox for executing corresponding handler routine to receive computation result from DSP.

When ARM and DSP have completed its task in a frame, then it will integrate computation result. Since each time when job dispatched to ARM or DSP, the control module will record the corresponding the axis of this macroblock to control module, and thus integration module can use this to construct our integration module. The integration module is implemented by maintaining two buffers with identical size of source frame, one is for storing quantized coefficient, and the other one is for storing reconstructed frame. When either ARM or DSP complete its current job, it puts

computation result into corresponding position into these two buffers according to the axis in control module which we set before. By this way, ARM and DSP can do their jobs in parallel without corrupt each other.

For reduce the transfer load, DMA module has been used to improve transfer efficiency between ARM and DSP on our proposed architecture. Based on our original design, transfer data to DSP and retrieve computation result from DSP was implemented by CPU support. This implementation will cause CPU spending lots time on handling memory movement and thus decrease ARM core’s processing ability.

The follow figure shows how DMA module improves the performance on dual-core architecture. Instead of moving data by CPU directly, ARM will just need to setup a DMA module for moving data. After ARM has setup the DMA control module properly, DMA module transfers data immediately. When DMA is the transferring, ARM can do its original job in parallel. After DMA module has finished its transfer, DMA will assert an interrupt to ARM, so that ARM can hold the transfer status, and decide what next to do.

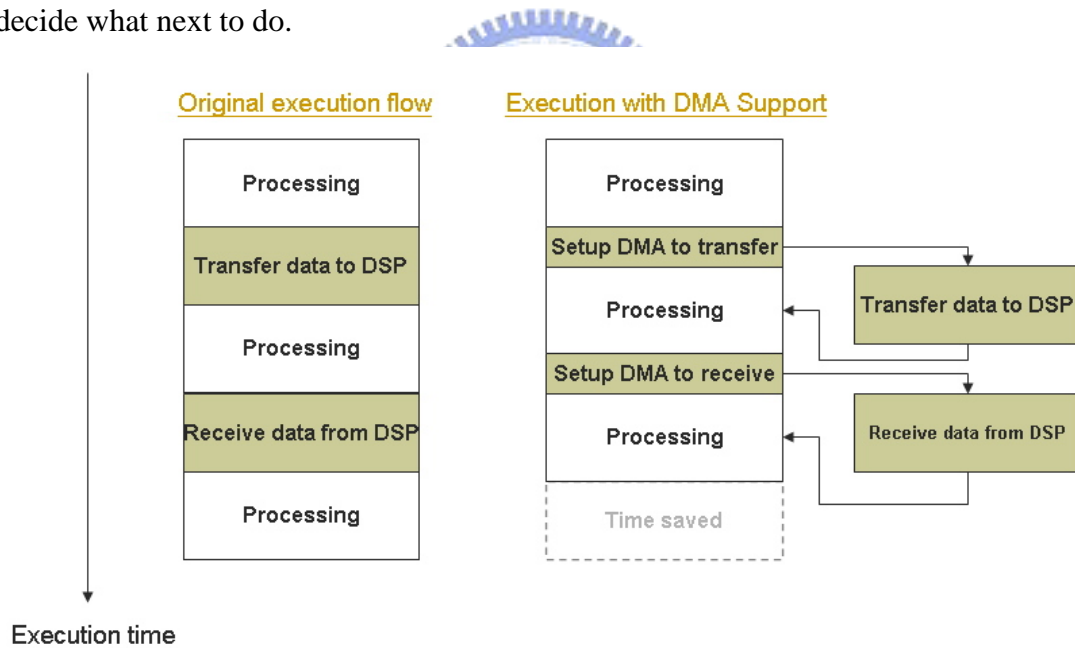


Fig 17 Improvement from DMA support

Our DMA architecture uses one system DMA channel. As a result, when receiving two macroblocks under one channel, since two macroblocks may exist in two different memory addresses and the DMA module’s control parameter seems not powerful enough, so that our DMA control module will be accessed twice to get these two MB results. The follow figure shows the transfer behavior of how DMA receives results from DSP.

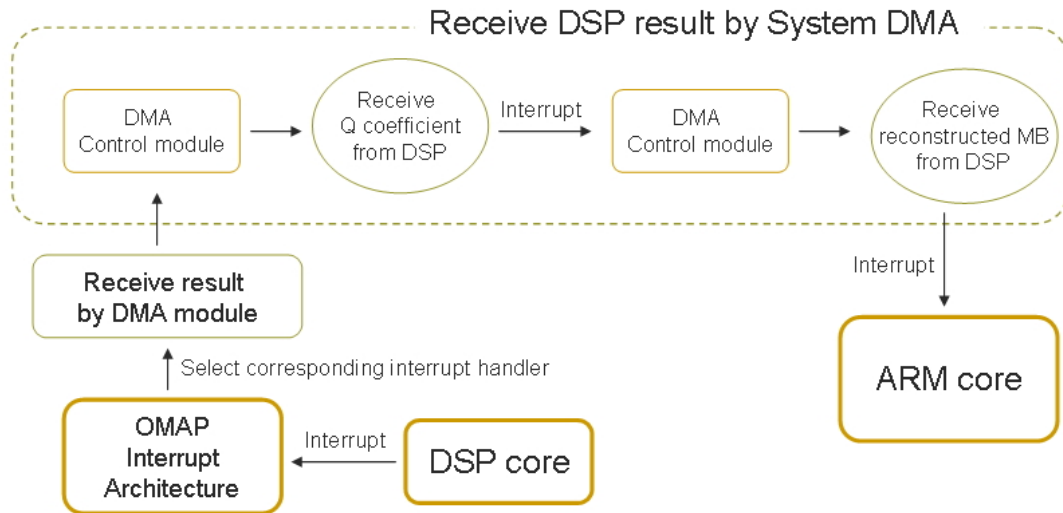


Fig 18 DMA architecture for dual-core intra macroblock encoding

And there is one concern that should be notified that whether if the time interval of setup DMA module will greater than the time interval of moving data by ARM directly. If yes, this design may decrease the original performance. It is because that every time when using DMA module to improve performance, it will also need to handle some overheads. There are two factors in the DMA overheads; one is that before DMA module really starts its transfer, DMA control parameter must set first. And the other one is that after DMA module has complete its transfer, it will assert interrupt to tell ARM about the fact. This will cause ARM pause its original job to handle DMA interrupt routine and this will be the main overhead of using DMA module. Fortunately, such worse case doesn't happen in our architecture even in our minimum transfer unit.

In our implementation of using the system DMA module, we still have some problems now. The problem is that when we set some reasonable control parameter to drive DMA module, it will leads some strange conditions such as interrupt disable, or makes DSP hardware extension set out of control. Although assign other control parameter which is equal transfer result but different transfer behavior to resolve this problem is possible. But it decreases some DMA performance; we will still pay attention to this problem.

3.4 Inter-frame processing

In inter-frame processing, it removes the temporal redundancy in current frame and last frame. Although this processing can get great coding efficiency, but it also costs more processing energy than intra-frame processing. Before we start improve this inter-frame processing, we can see the experiment result in the follow table first. The follow table shows the weightings between each module in the inter-frame processing.

Table 5 Inter-frame processing result

Qcif, 150 frames (IPPP...)	Execution time (ms)	Percentage
Initialization	236	0.140
Coding	767	0.456
Set edge	1027	0.610
Sequence conversion	1686	1.002
Prediction	70	0.042
Rate control	3464	2.058
Motion compensation	4375	2.600
I MB encoding	60	0.035
Interpolation	8693	5.166
Motion estimation	132649	78.831
DCT/Q/Q ⁻¹ /IDCT	13328	7.921
Total	168270	100
Encoding frame rate =0.9		

According to the weighting from table, it shows that interpolation module, motion estimation module, and P macroblock encoding takes most percentage of all in the inter-frame processing. As a result, these modules will be improved with DSP support for better execution performance. Detail design of each module will be described in the follow sections.

3.4.1 Interpolation processing

In the MPEG4 standard, it not only computes motion vector for compressing on pixel level, but also takes consider of the half-pixel level to find out better motion vector for better coding efficiency. Since there are two modules: motion estimation, motion complementation will reference the half pixel frame. As a result, at this moment, this interpolation module will be implemented individually instead of on-the-fly architecture. The follow figure and table shows out our proposed dual-core interpolation module.

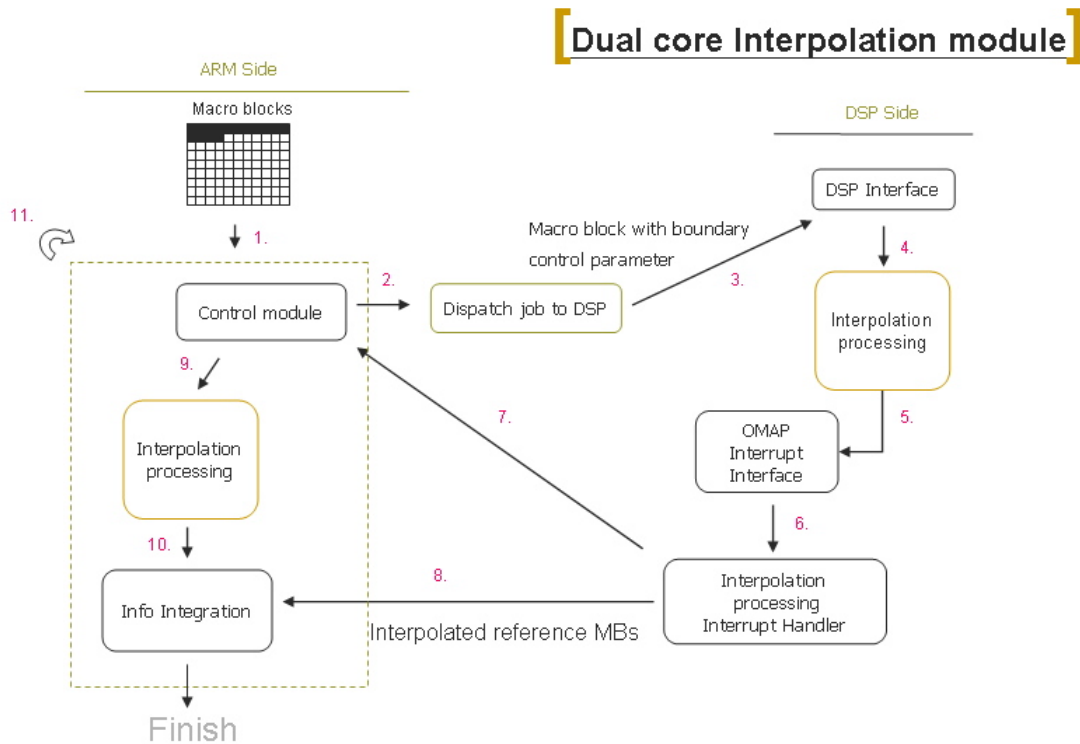


Fig 19 Dual-core interpolation processing scheme

Table 6 Description of DSP interpolation processing module

Step	Description
1	Process the frame in scanline order
2	Control module decides to dispatch next job to DSP
3	Transfer MB data and control parameter to DSP
4	DSP enters interpolation process
5	After interpolation, DSP asserts an interrupt
6	MCU executes handler routine
7	Control module decides next step
8	Integrate results from DSP
9	Control module decides to dispatch next job to MCU
10	Integrate results from MCU
11	The encoding loop repeats, until all jobs have been completed

Table 7 Specification of DSP interpolation processing

Specification of DSP interpolation module	
DSP Input	<ol style="list-style-type: none"> 1. One extension source macroblock data <ul style="list-style-type: none"> ■ One Y data macroblock (16*16*2 bytes) 2. Rounding parameter for DSP interpolation <ul style="list-style-type: none"> ■ 2 bytes
DSP output	<ol style="list-style-type: none"> 1. Interpolated macroblock in horizontal <ul style="list-style-type: none"> ■ One Y data macroblock (16*16*2 bytes) 2. Interpolated macroblock in vertical <ul style="list-style-type: none"> ■ One Y data macroblock (16*16*2 bytes) 3. Interpolated macroblock in horizontal and vertical <ul style="list-style-type: none"> ■ One Y data macroblock (16*16*2 bytes)

Our proposed architecture of dual-core interpolation module is similar to the architecture of dual-core I macroblock encoding which mentioned before. This is because these modules were constructed based on the same communication architecture which adopts the DSP interface module between ARM and DSP. As a result, only the main execution flow and main features of this dual-core interpolation architecture will be discussed in the follow.

Before this module start works, the task unit in each reference frame must be defined first. It is similar as before that macroblocks in the reference frame will be treated as task units. And there is one thing need to notify that since DSP core can't see the whole frame, so that ARM needs to prepare all relational data to DSP. Thus, since our goal is to receive the computation results in macroblock level from DSP, transfer content will includes not only the data of reference macroblock, but also the boundary of this macroblock for computation need. The follow figure shows an example to realize this condition. If results of 2x2 size blocks are expected, it needs to transfer a 3x3 size block to DSP for interpolation processing.

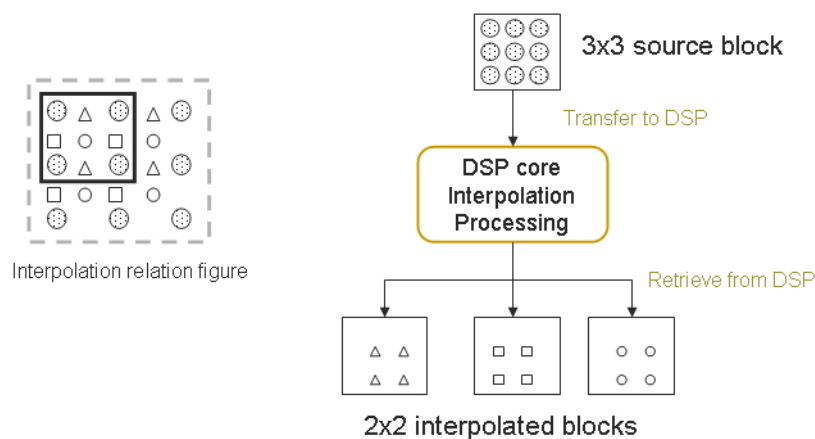


Fig 20 Example of interpolation processing on DSP

After task unit has been defined in this dual-core interpolation processing, then this module can start to dispatch the first job to DSP; and then ARM will execute next job and fall into a processing loop. Each time when DSP complete its job, DSP will assert an interrupt to ARM. When ARM receives the interrupt event, it starts to receive three interpolated reference macroblock computation results from DSP. After receiving computation results, the control module will decide whether if dispatch next job to DSP. Until all tasks in the reference frame have completed, then the encoder will exit this dual-core interpolation module. By the way, the control module in this architecture is similar to the control module in the dual-core I macroblock encoding module. Each time when ARM and DSP want to clan a task to execute, control module will update the global progress status and corresponding control parameters, and thus hold the execution flow running exactly.

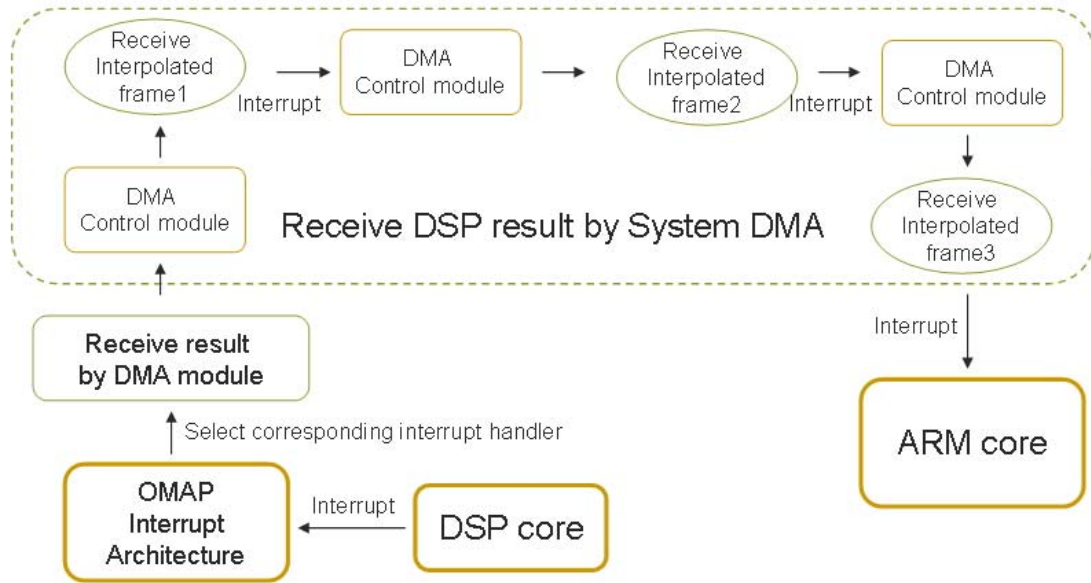


Fig 21 DMA architecture for dual-core interpolation processing

Besides improving performance by DSP, DMA module can also be added to this dual-core architecture to improve transfer. Since three interpolated reference macroblocks need to be received from DSP, some modifications are needed to let our DMA control architecture fit the need. The above figure represents that how to use system DMA module to receive three interpolated macroblocks from DSP with one system DMA channel.

3.4.2 Motion estimation

In the video compression, motion estimation plays an important role to reduce the temporal redundancy that may exist within a video sequence, while spatial redundancy is reduced by other techniques such as DCT. Generally speaking, it is the technique to provide the sum of absolute difference (SAD) and the corresponding location (motion vector) between a 16x16 reference block and some blocks in a reference frame for video compression. Therefore, for an efficient video coding, a robust motion estimation technique is required. The motion estimation usually takes a very long processing time and thus, we will improve this module by dual-core processing. The following figure shows the dual-core motion estimation architecture.

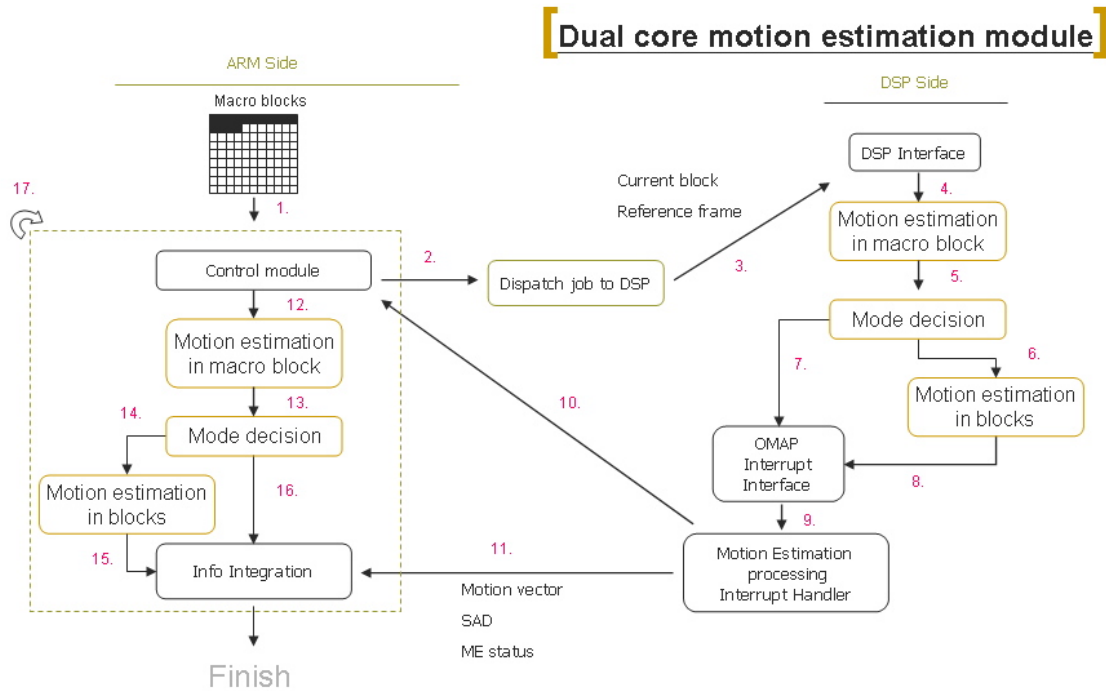


Fig 22 Dual-core motion estimation scheme

Table 8 Description of dual core motion estimation processing module

Step	Description
1	Process the frame in scanline order.
2	Control module decides to dispatch next job to DSP.
3	Transfer MB data and control parameter to DSP.
4	DSP executes Motion Estimation.
5	Do mode decision.
6	Continue to do motion estimation in block level.
7	Mode decision module set as intra mode, and asserts interrupt.
8	After motion estimation, DSP asserts an interrupt.
9	MCU executes handler routine.
10	Control module decides next step.
11	Integrate results from DSP.
12	Control module decide let ARM do motion estimation.
13	Do mode decision.
14	Continue to do motion estimation in block level.
15,16	Integrate motion estimation computation result from ARM.
17	The encoding loop repeats, until all jobs have completed.

Table 9 Specification of DSP motion estimation processing

Specification of DSP Motion Estimation module	
DSP Input	<ol style="list-style-type: none"> 1. One current macroblock <ul style="list-style-type: none"> ■ One Y data macroblock (16*16*2 bytes) 2. Four reference frames <ul style="list-style-type: none"> ■ Reference frame (48*48*2 bytes) ■ Reference frame in horizontal (48*48*2 bytes) ■ Reference frame in vertical (48*48*2 bytes) ■ Reference frame in horizontal and vertical (48*48*2 bytes)
DSP output	<ol style="list-style-type: none"> 1. One set of motion vector <ul style="list-style-type: none"> ■ 4*2 bytes 2. SAD result <ul style="list-style-type: none"> ■ 4 bytes 3. Motion Estimation execution status <ul style="list-style-type: none"> ■ 2 bytes

The task units of each frame in this dual-core motion estimation module are still defined in macroblocks of the frame. Basically, this motion estimation can be divided into two continuous sub-modules. One is search in macroblock level, and the other is search in block level, and there is a mode decision module in the middle of these two motion search sub-modules. The functionality of this mode decision module is to avoid the condition that if motion vector we have found isn't good enough for compressing, it will take more bits on compressing this macroblock than encoding it by intra-coding method directly. Generally speaking, in order to avoid wasting time on motion estimation while current macroblock is more suitable for intra macroblock encoding such as scene change condition, mode decision must works as early as possible.

And there will be a threshold for mode decision module to decide whether continue to search in block level right after search in macroblock level or not. In the current status, this threshold is decided by two factors, one is the deviation of current macroblock, and the other is the SAD result calculated from search in macroblock level. Each time while motion search in MB level has completed, it will fall into the mode decision module, and then compare the SAD and the deviation value of current macroblock to decide what mode it suits. The follow figure describes how mode decision module works.

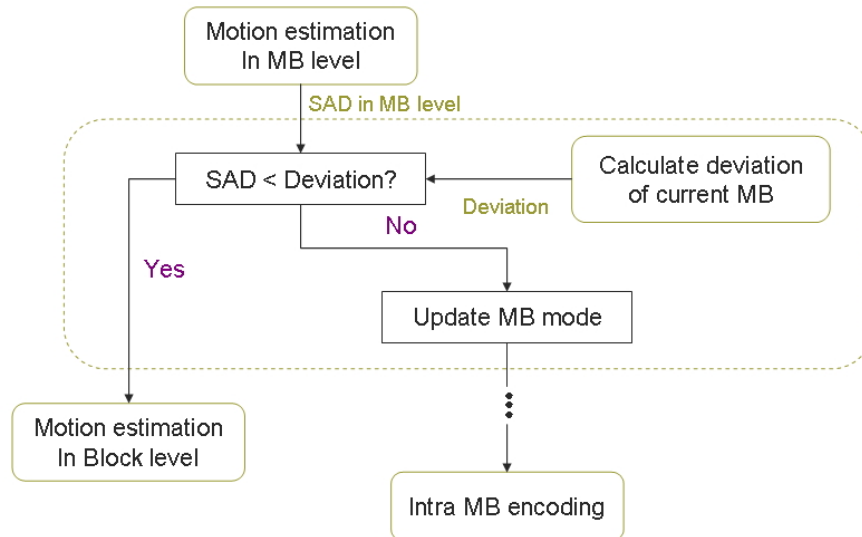


Fig 23 Mode decision module

After brief description about the execution flow of motion estimation, then the dual-core execution flow will be discussed. At first, control module dispatch the first job to DSP including of current macroblock, reference frames and some control parameters. And ARM will do next jobs and fall into an encoding loop. As same as architectures mentioned before, ARM and DSP will execute their jobs which dispatched by control module in parallel. And after motion search has completed, it will return the motion vectors, corresponding SADs, and ME execution status if it executed in DSP. The ME execution status records the result of mode decision module in the side of DSP core. Through implement on proposed dual-core method, this most computation intensive module takes full advantage of dual-core platform and thus become more efficient than before. After integrating all computation results in this frame, it will then encode the motion vector mode of these task units, and then the dual-core motion estimation module finished.

About our motion estimation algorithm, in order to get balance between efficiency, accuracy, and TI DSP C55X hardware extension image library, we adopt the four step hierarchy search algorithm. This motion search algorithm are adopted whether search in macroblock level or search in block level. And the reference frame size as 48*48 pixels and search range is from -16 to +16. The search pattern of our motion estimation algorithm is shown in the follow figure and the detail algorithm is as follows.

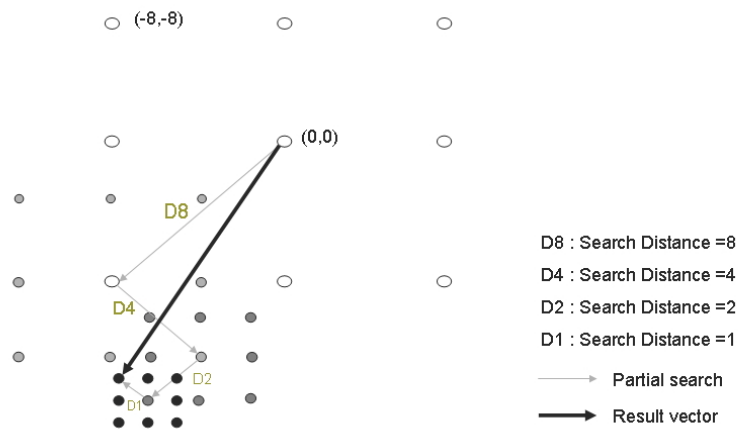


Fig 24 Four step hierarchy search algorithm

Step one: At first, search start at the position of $(0, 0)$ motion vector. And then searches nine search positions with distance eight, and decide the best match point according to which SAD is minimal between these search points as the starting point for next step search.

Step two: According to the best match point decided by step one; it still searches nine points corresponding to the starting point with distance four, and decide the best match point as the starting point for next step search.

Step three: Same, according to the best match point decided by step two; it still searches nine points corresponding to the starting point with distance two and then decide the best starting point for next step search.

Step four: According to the best match point decided by step three; it still searches nine points corresponding to the starting point with distance one. Then, we will choose the best match point as the starting point for next step search – half pixel refine.

Step five: According to the best match point decided by step four; it will do half pixel refine. It is like the search pattern in step four, but the reference frames will be changed according to the corresponding search point when it performs search.

Final step: After these search, it returns the best motion vector and corresponding SAD result, and then finish this motion search algorithm.

DMA module also can be used to reduce transfer time in this architecture. Since that our motion estimation algorithm searches in full pixel level and half pixel level. ARM need to transfer full pixel reference frame and half pixel reference frames to DSP for motion estimation processing. The follow figure shows the DMA transfer module for dual-core motion estimation processing with one system channel.

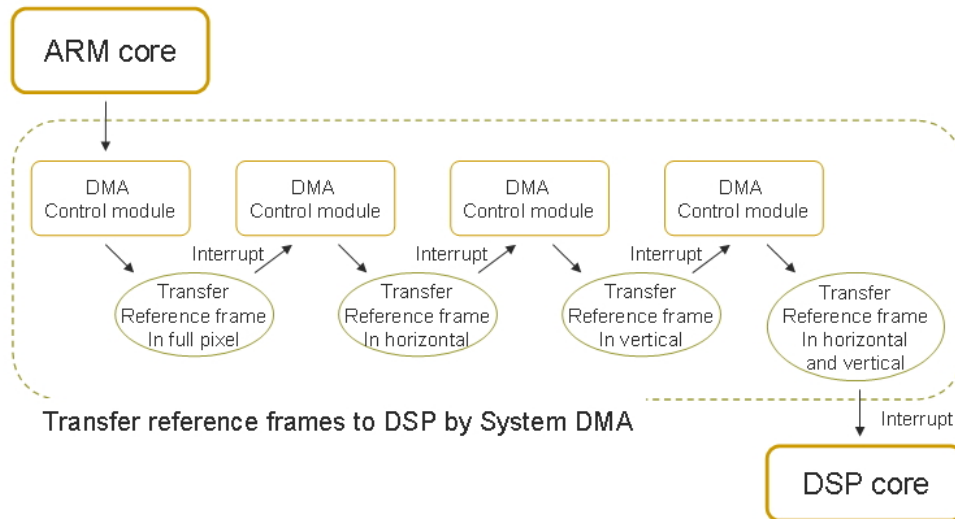


Fig 25 DMA architecture for motion estimation processing

Besides above architecture, we also implement other method to improve performance. We observe that each time while job dispatched to DSP, DSP is in idle status before transferring of all reference frames including of frames in half pixel to DSP has completed. Since DSP has great computation performance, so that we use DSP to computes reference frames in half pixel from reference frame in full pixel instead of just stays in idle status. And from the experiment, this way will be faster than waiting for receiving all reference frames in half pixel from ARM. The follow figure describes this method.

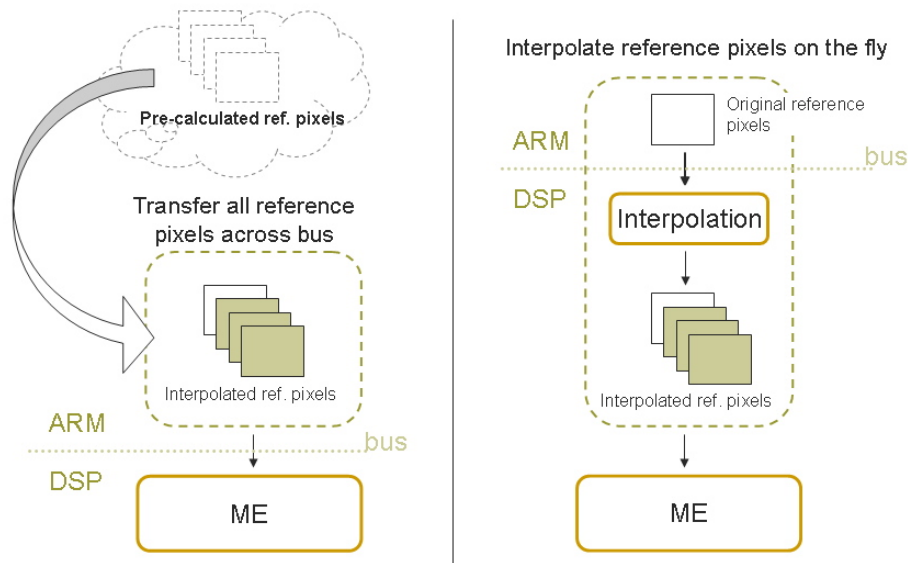


Fig 26 Motion estimation with on-the-fly architecture on DSP

Through this method, each time when dispatching jobs to DSP, it just needs to transfer one reference frame in full pixel to DSP, and after DSP receives the reference frame in full pixel, it interpolate this reference frame and thus gets three reference frames in half pixel for motion estimation. The follow table defines the inputs and outputs of DSP module while implement this method. And there are something need to notify, there is a difference of the reference frame size we transfer to DSP has became 49 pixels * 49 pixels. This is for interpolation concern which mentioned in DSP interpolation module before.

Table 10 Specification of DSP ME processing with interpolation on the fly

Specification of DSP Motion Estimation module	
DSP Input	<ol style="list-style-type: none"> 1. One current macroblock <ul style="list-style-type: none"> ■ One Y data macroblock (16*16*2 bytes) 2. One reference frame <ul style="list-style-type: none"> ■ Reference frame (49*49*2 bytes) 3. Rounding parameter for interpolation in ME <ul style="list-style-type: none"> ■ 2 bytes
DSP output	<ol style="list-style-type: none"> 1. One set of motion vector <ul style="list-style-type: none"> ■ 4*2 bytes 2. SAD result <ul style="list-style-type: none"> ■ 4 bytes 3. Motion Estimation execution status <ul style="list-style-type: none"> ■ 2 bytes

Through this method, if DMA module is adopted to improve the transfer of reference frame from ARM to DSP, its control architecture will become easier than before. Since its need is just to transfer the reference frame in full pixel only, the numbers of setting DMA control module will be decrease to just one time only.

3.4.3 Inter macroblock encoding

As same as I macroblock encoding in intra frame processing, there is also a transform coding module in inter frame processing. And there are some differences between these two encodings. First, in this P macroblock encoding, our input changes as the residuals between current frame and last frame. Second, the control flow of P macroblock encoding is more complex than I macroblock encoding. There are I macroblock encodings and P macroblock encodings in the inter frame processing according to result of the mode decision module in the motion estimation processing. In order to handle these encodings in parallel and reasonable for transform coding of inter frame, our design is to combine these two encodings in this architecture. As a result, before we dispatch jobs to ARM or DSP, we will check this macroblock's feature for executing proper function module. It also means that we define the P macroblock encoding as a series of processing of FDCT, Quantization, Dequantization, and IDCT for the same reason of I macroblock encoding. The follow figure and tables shows and describes the dual-core inter macroblock encoding architecture.

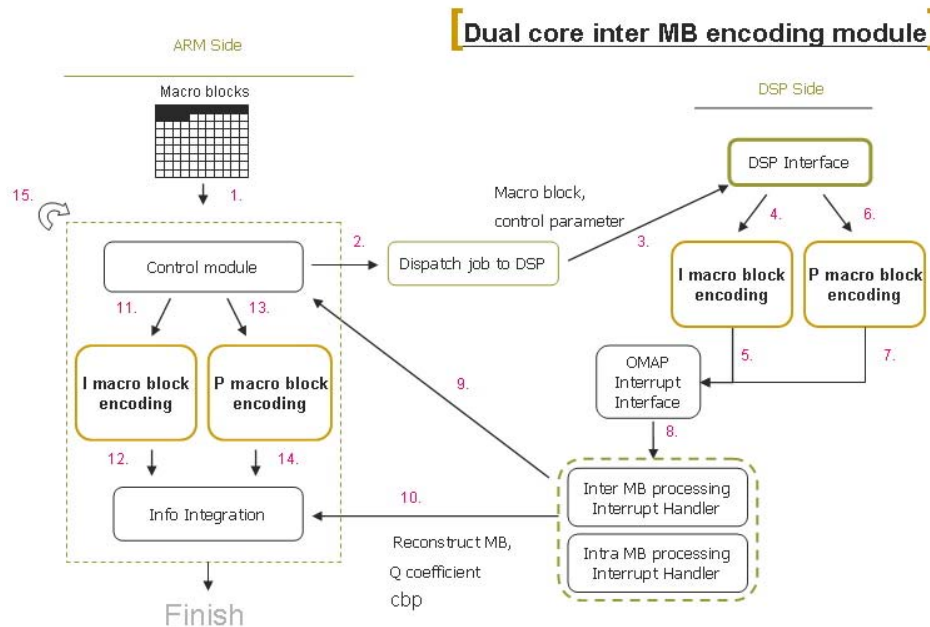


Fig 27 Inter macroblock encoding scheme

Table 11 Description of dual-core P MB encoding module

Step	Description
1	Process the frame in scanline order
2	Control module decide to dispatch next job to DSP
3	Transfer MB data and control parameter to DSP
4	DSP interface decide to do I MB encoding
5	After DSP completing I MB encoding, it asserts interrupt.
6	DSP interface decide to do P MB encoding
7	After DSP completing P MB encoding, it asserts interrupt.
8	MCU executes handler routine.
9	Control module decides next step.
10	Integrate results from DSP.
11	Control module decides to let ARM do I MB encoding.
12	Integrate the computation result from ARM.
13	Control module decides to let ARM do P MB encoding.
14	Integrate the computation result from ARM.
15	The encoding loop repeats, until all jobs have completed

Table 12 Specification of DSP P MB encoding

Specification of DSP P macroblock encoding	
DSP Input	<ol style="list-style-type: none"> 1. One source macroblock data <ul style="list-style-type: none"> ■ Four Y data blocks (8*8*4*2 bytes) ■ One U data blocks (8*8*2 bytes) ■ One V data blocks (8*8*2 bytes) 2. QP parameter for DSP quantization <ul style="list-style-type: none"> ■ 2 bytes
DSP output	<ol style="list-style-type: none"> 1. One Q coefficient macroblock data <ul style="list-style-type: none"> ■ Four Y data blocks (8*8*4*2 bytes) ■ One U data blocks (8*8*2 bytes) ■ One V data blocks (8*8*2 bytes) 2. One reconstructed macroblock data (transfer according to execution status) <ul style="list-style-type: none"> ■ Four Y data blocks (8*8*4*2 bytes) ■ One U data blocks (8*8*2 bytes) ■ One V data blocks (8*8*2 bytes) 3. P MB encoding execution status <ul style="list-style-type: none"> ■ 2 bytes

From the above execution flow figure, this module's dual architecture is similar to the dual-core intra I MB encoding with extra P MB encoding module. The main feature of this P MB encoding is that it may decide whether to do Dequantization and IDCT or not according to the status of quantization result. In the follow figure, it shows the detail execution flow of these two transform encoding component in this dual-core module. In the middle of the figure is the control module which decides either I MB encoding or P MB encoding need to be performed. And the modules in left side and right side show the detail execution flow of each module. And if P MB encoding is performed, the condition of getting different amounts of DSP computation result will occur, and thus, it will have less transfer of computation result from DSP than I MB encoding.

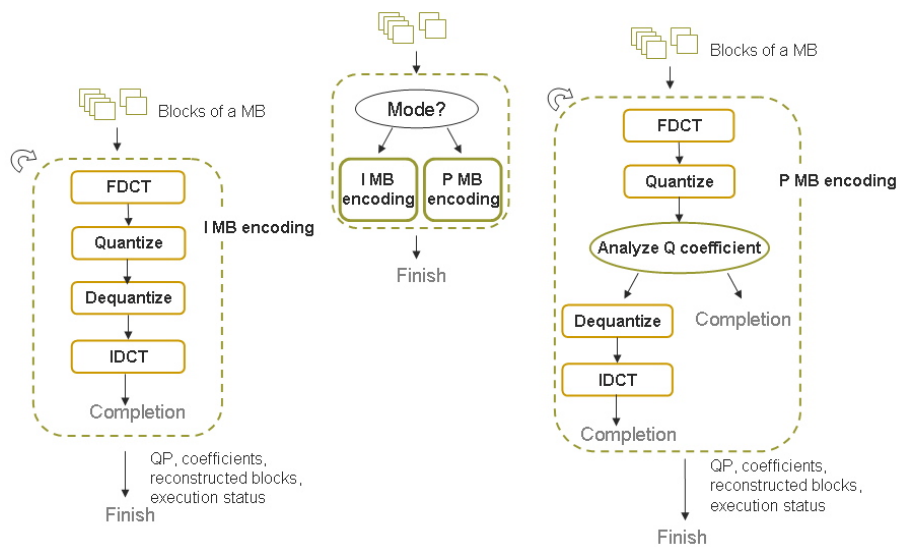


Fig 28 Execution flow in Inter macroblock encoding scheme

DMA module is still used to improve this architecture, there are two implementation method for this module, one is as same as I macroblock encoding DMA module which may have redundant transfer but has lower complexity on DMA handler, and the other one is shown in the follow figure. There is an issue between these two methods, since P macroblock encoding sometimes doesn't reconstruct blocks from Q coefficients, we may let DMA module ignores these transfers for efficiency. But if we make our DMA module support the detection of whether transfer each block or not, our DMA control module will become more complex and thus may decrease DMA performance. This is because we should concern with the overhead of DMA module and interrupt overhead. So that before we finalize our design, we reference the implement results of these two methods, and find that the DMA module support detection has better performance.

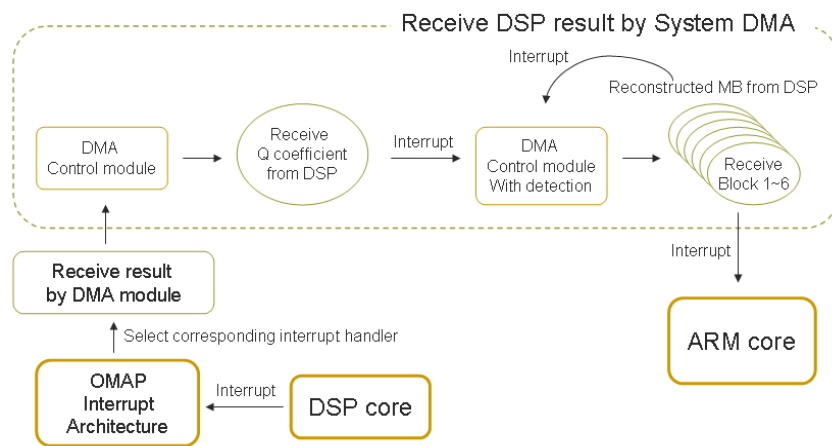


Fig 29 DMA architecture for dual-core P MB encoding



Chapter 4 Implementation using DSP Hardware

Extension for Video Coding

The TMS320C55x DSP core was created with an open architecture that allows the addition of application-specific hardware to boost performance on specific algorithms. And the TI C55x IMGLIB is an optimized image/video processing functions library for C programmers using TMS320C55x devices. It includes many C-callable, assembly-optimized, general-purpose image/video processing routines. This library is implemented by using the TI C55x hardware extension set, so that through this IMGLIB library, we can utilize the max power of TI C55x easily. And these routines are typically used in computationally intensive real-time applications where optimal execution speed is critical. By using these routines, it will help us to achieve execution speeds considerably faster than equivalent code written in standard ANSI C language. In addition, by providing ready-to-use DSP functions, TI IMGLIB may shorten our image/video processing application development time. The TI C55x IMGLIB contains commonly used image/video processing routines. And it also provides source code for us to modify functions to match our specific needs. There are many application fields it provides, and since our focus is on the compressing application at this moment, so that we will describe how to use this motion estimation, interpolation, FDCT, and IDCT algorithms in TI IMGLIB library which implement in TI DSP hardware extension set.

4.1 FDCT module and IDCT module

In the TI C55x IMGLIB library, it provides DCT and IDCT algorithms which is implemented in TI DSP C55x hardware extension set. We can use these two modules to improve our design in I macroblock encoding, and P macroblock encoding. We can instead our own FDCT and IDCT modules by TI IMGLIB's DCT and IDCT modules. The follow two tables show the specification of FDCT and IDCT modules.

Table 13 Specification of FDCT with HW extensions

FDCT for an 8x8 Image using built-in hardware extensions	
Syntax	<code>void IMG_fdct_8x8(short *fdct_data, short *inter_buffer);</code>
Arguments	Inputs:
	<ul style="list-style-type: none"> ■ <code>fdct_data</code>: Points to a short format array [0...63] containing an 8x8 macroblocks row by row. Data format is Q16.0. ■ <code>inter_buffer</code>: Points to a short format array [0...71] used as a temporary buffer that contains intermediate results in the transform.
	Outputs:
	<ul style="list-style-type: none"> ■ <code>fdct_data</code>: Points to a short format array [0...63] containing the results of 2-D DCT for the macro-block. Data format is Q16.0.
Description	The routine <code>IMG_fdct_8x8</code> implements the Forward Discrete Cosine Transform (FDCT) using built-in hardware extensions for an 8x8 image block. Input terms are expected to be signed Q16.0 values, producing signed Q16.0 results.

Table 14 Specification of IDCT with HW extensions

IDCT for an 8x8 image block using built-in hardware extensions	
Syntax	<code>void IMG_idct_8x8(short *idct_data, short *inter_buffer);</code>
Arguments	Inputs:
	<ul style="list-style-type: none"> ■ <code>idct_data</code>: Points to a short format array [0...63] containing an 8x8 macro-block row by row. Data format is Q13.3. ■ <code>inter_buffer</code>: Points to a short format array [0...71] used as a temporary buffer that contains intermediate results in the transform.
	Outputs:
	<ul style="list-style-type: none"> ■ <code>idct_data</code>: Points to a short format array [0..63] containing the results of 2-D IDCT for the input block. Data format is Q16.0.
Description	The routine <code>IMG_idct_8x8</code> implements the Inverse Discrete Cosine Transform (IDCT) using built-in hardware extensions for an 8x8 image block. Input terms are expected to be signed Q13.3 values,

producing signed Q16.0 results.

After realizing these specifications, we can find that their input and output are similar to our original design. So, in IMGLIB's FDCT module, there should be a source block data and one temp buffer for it, and then take it instead of our FDCT module directly. But in IMGLIB's IDCT module, there is one thing need to be notified that its input data format is Q13.3, so that we must adjust our input of Q coefficients from Q16 format into Q13.3 format to fit the specification. The follow figure shows how to perform format conversion from Q16 format to Q13.3 format by shifting.

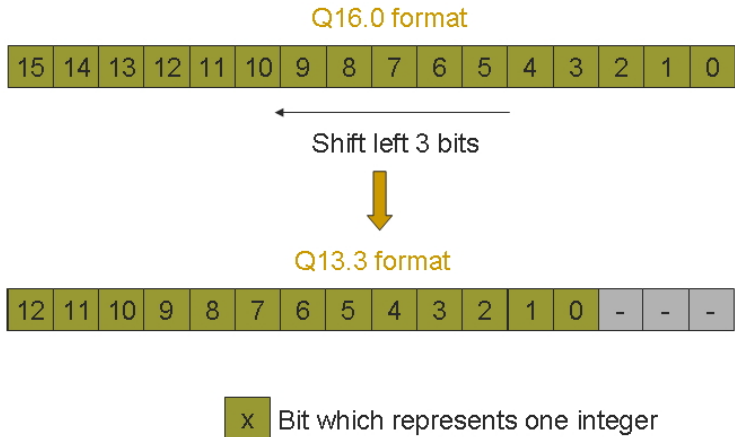
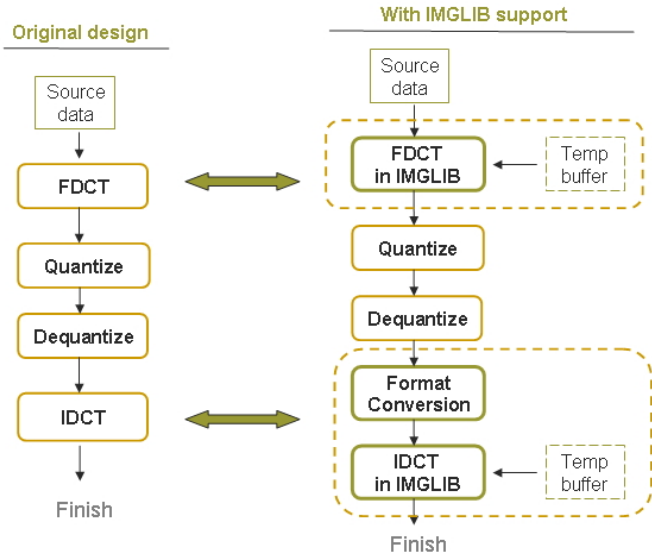


Fig 30 Format conversion

After realizing corresponding background, the follow figure shows how to add FDCT and IDCT hardware extension set modules into our I MB encoding, and P MB encoding.



4.2 Interpolation module

There is an interpolation module in IMGLIB, it implements pixel interpolation for a 16x16 source block located in reference window using built-in hardware extensions. As a result, this module can be used to instead our original interpolation module for accelerating computations.

Before we use this module, there is an issue need us to think. The design in our visual encoder processes pixels in 16-bit width for the concern of processing in ARM and DSP which we mentioned before. But this interpolation module in IMGLIB processes two pixels data in 16-bit width. As a result, before we use this module, some format conversion to fit its specification is needed, and this operations decrease performance. But in the other hand, if we process pixels data in 8-bit width, this will increase the complexity of function modules in DSP side which don't have corresponding hardware extension set support. So, this exist a tradeoff. By the way, FDCT and IDCT modules don't have such issue; this is because they process source data in 16-bit width in theory. And then, we can know its specification of this module from the follow table.

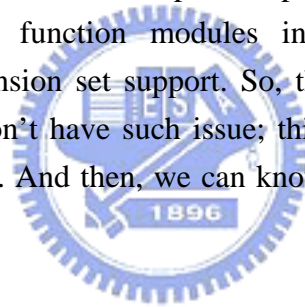


Table 15 Specification of Interpolation with HW extensions

Pixel Interpolation for 16x16 Image block using built-in hardware extensions	
Syntax	IMG_pix_inter_16x16(short *reference_window, short *pixel_inter_block, int offset, short *align_variable);
Arguments	Inputs:
	<ul style="list-style-type: none"> ■ reference_window: Points to a packed integer format buffer [0...1152] that contains a 48x48 image block row by row. Must be doubleword aligned. Every four pixels are packed into one 32-bit doubleword. Data format Q16.0. ■ offset: Specifies the top-left corner index of the 18x18 MBE (MBE=16x16 macroblock + extension) in reference_window. Offset is even because of the doubleword alignment. ■ align_variable: Configures four alignment cases of the MBE in the reference_window.
	Outputs:
	<ul style="list-style-type: none"> ■ pixel_inter_block: Points to a packed integer format buffer [0...612] that contains the 36x34 interpolated result. Only the lower 33x33 part that corresponds to the whole 36x34 interpolated zone is usually used. Every four pixels are packed into one 32-bit doubleword.
Description	<p>The routine IMG_pix_inter_16x16 implements pixel interpolation for a 16x16 source block located in reference_window using built-in hardware extensions and it is useful in video compression. To implement full interpolation for the 16x16 source block, the 18x18 MBE (MBE=16x16 macroblock + extension) is needed. The full interpolated zone is composed of 36x34 pixels, but only the lower 33x33 part corresponding to the full interpolated zone is usually interested.</p>

In this specification, we can see that it supports some align modes for us to use. We can choose the align mode which is most fit for our architecture to implement. In the format of input reference frame, it announces large space to put source data, and just use the size of macroblock to interpolate. The purpose of this design is that, TI's

IMGLIB want to corresponding modules to help each other. So, this design will help the motion estimation module to do half refine, since they have the identical size of reference frame. But in our own design, we design the interpolation module as an individual module, so there are some modifications need to be performed before using this interpolation in IMGLIB.

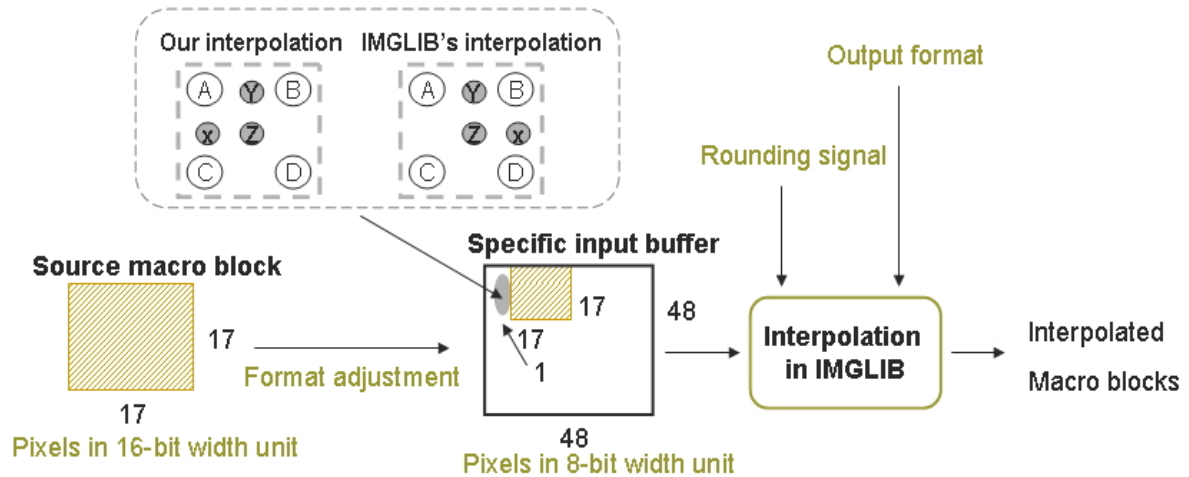


Fig 32 Interpolation processing with built-in hardware extension support

We can see the above figure to realize the execution flow of using IMGLIB's interpolation module. At first, we need to adjust our input source macroblock from one pixel in 16-bit width to two pixels in 16-bit width. And then put it into corresponding position in the specific input buffer as the input of IMGLIB's interpolation module. Because there is a little difference of the computation behavior between these two modules, so that we will shift right 1 pixel unit for exactness. And there are two important control parameters which we must set by ourselves. The first one is the rounding signal; it decides whether do rounding in this interpolation, the default value in IMGLIB is enabling. The second one is the output format; it decides the arrangement of the interpolation module's output.

4.3 Motion estimation module

Motion estimation is the most time-consuming part in video compression algorithms such as MPEG4 and H263. So that, it is no doubt that there will be a motion estimation module in IMGIB. The follow table shows the specification of this motion estimation module. And its motion estimation algorithm is as same as our motion estimation algorithm: four step hierarchy search algorithm.

Table 16 Specification of ME in HW extensions

Motion Estimation by 4-step search using built-in hardware extensions	
Syntax	IMG_mad_16x16_4step(short *src_data, short *search_window, unsigned int *match);
Arguments	Inputs:
	<ul style="list-style-type: none"> ■ src_data: Points to a packed integer format buffer [0...128] that contains 16x16 source data row by row. Data format is Q16.0. Every two pixels are packed into one 16-bit integer. ■ search_window: Points to a packed integer format buffer [0...1152] that contains the 48x48 search-window row by row. Data format is Q16.0. Every two pixels are packed into one 16-bit integer.
	Outputs:
	<ul style="list-style-type: none"> ■ match [2]: The location of the best match block is packed in match[0]. The upper halfword contains the horizontal pixel position, and the lower halfword contains the vertical pixel position of the best matching 16x16 block in the search window. The minimum absolute difference value at the best match location is packed in match [1].
Description	The routine IMG_mad_16x16_4step implements the motion estimation by 4-step (distance=8, 4, 2, 1) search using built-in hardware extensions. The 4-step search is a popular fast searching technique. Input terms are packed in 16-bit integers and doubleword aligned. Input and output data format is Q16.0.

Before using this motion estimation module to improve our codec, there are something need to pay attentions. Generally, one often calculate the motion vectors by comparing with the center point of the reference frame. But, in this built-in hardware extension motion estimation module, it calculates the motion vectors by comparing with the left top point of the reference frame. So some compensation to its motion vector to fit our codec is needed. The follow figure shows the execution flow of our

motion estimation module with built-in hardware extension module.

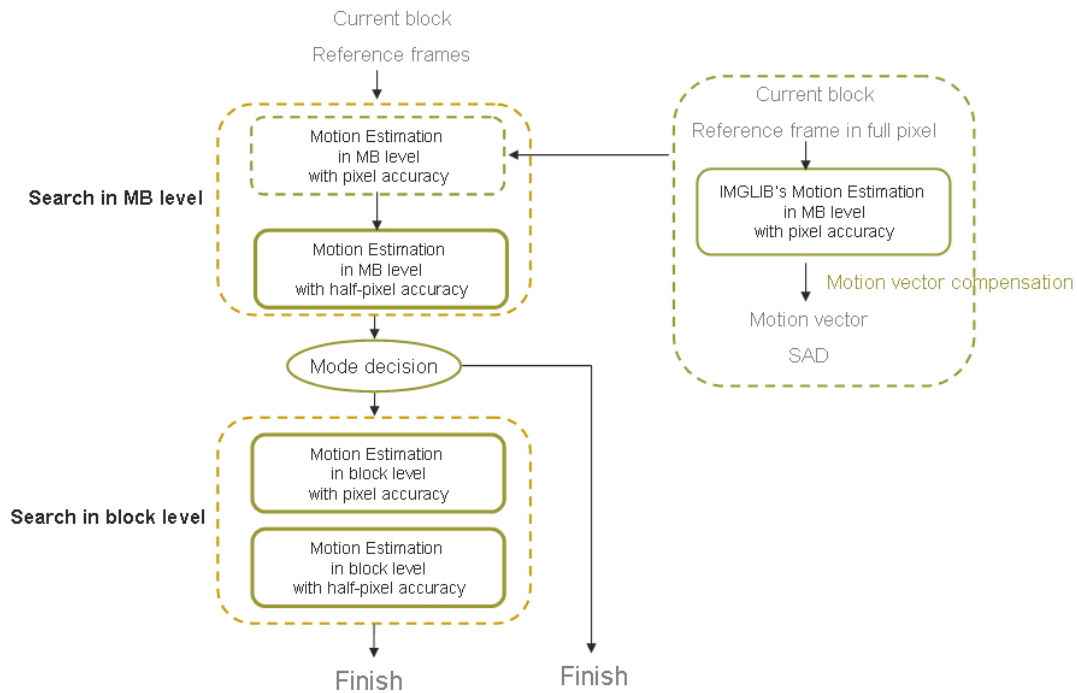


Fig 33 Motion estimation with built-in hardware extension support

From the above figure, it shows how to add IMGLIB's motion estimation module to our architecture. Our own motion estimation module is replaced by IMGLIB's motion estimation module with some adjust of inputs and outputs. And thus, it gets better performance from the support of built-in HW extensions.

In fact, we just complete partial of the motion estimation module with built in hardware extension motion estimation module. Because we face the conditions of implementation time and little information about the instruction set of the hardware extension set now. As a result, we just use the IMGLIB provided by TI, and follow the rules provided by IMGLIB. So that since we haven't know the detail specifications and algorithms of remain motion search modules provided by IMGLIB, we can' add them into our motion estimation architecture. We will improve this condition in the future.

Chapter 5 Experimental results

Some experimental results are shown in this section. The QCIF version of the Stefan sequence is used for the experiments. The first 150 frames of this sequence is encoded and the target bit rate is set at 96 kbps. The test environment are configured similarly to the general test environment which often used by TI on OMAP platforms. The follow table shows the main features of the test environment in this experiment. On the ARM side, the main program is stored in SDRAM, and the SRAM is used as the frame buffer for the LCD controller. On the DSP side, main program sections are put in the SARAM, and data sections are put on the DARAM. And the MPUI mode is set as shared mode for ARM core to access DSP core's memory.

Table 17 Setup of experiment environment

Experiment environment	
ARM core	150 MHz
DSP core	150 MHz
Traffic controller	75MHz
System DMA	No burst, 16-bit width

5.1 Experiment of Intra frame processing

5.1.1 Overall result

In this section, the main goal is to experiment with the I MB encoding module, and the encoding mode of all frames are set as intra frame mode for intra frame processing experiment. The implementation result and improvement will be shown step by step here.

Execution with pure ARM core

At first, we see the experiment result of Intra frame processing. The follow table shows the implementation result of execution on ARM core alone. Thus, we can know the original performance of our codec which ported from PC on intra frame processing.

Table 18 Experiment result of pure ARM core

Qcif,150 I frames	Execution time (ms)	Percentage
-------------------	---------------------	------------

Initialization	236	0.735
Coding	4111	12.793
Sequence conversion	1684	5.241
Prediction	2631	8.190
DCT/Q/Q⁻¹/IDCT	22297	69.396
Total	30963	100
Encoding frame rate =4.7		

Execution with pure DSP core

The follow table shows the implementation result of execution on DSP core alone. This illustrates the computation ability of the DSP core.

Table 19 Experiment result of pure DSP core

Qcif,150 I frames	Execution time (ms)	Percentage
Initialization	236	0.885
Coding	4123	15.455
Sequence conversion	1683	6.308
Prediction	2637	9.886
DCT/Q/Q⁻¹/IDCT	16811	63.011
Total	26680	100
Encoding frame rate =5.6		

Execution with pure DSP core, FIQ

The follow table shows the implementation result of execution using only the DSP core with interrupt mode - FIQ. Through this experiment, it shows that the interrupt mode improve the performance of our codec minor.

Table 20 Experiment result of pure DSP core, FIQ

Qcif,150 I frames	Execution time (ms)	Percentage
Initialization	236	0.886
Coding	4123	15.470
Sequence conversion	1683	6.314
Prediction	2637	9.895
DCT/Q/Q⁻¹/IDCT	16785	62.974
Total	26654	100

Encoding frame rate = 5.6

Execution with pure DSP core, FIQ, HW extensions

The follow table shows the implementation result of execution using only the DSP core with interrupt mode – FIQ. And the built-in hardware extension module of DCT and IDCT are used for improving I MB encoding. Through this experiment, it shows the outstanding performance from the support of hardware extension.

Table 21 Experiment result of pure DSP core, FIQ, HW extensions

Qcif,150 I frames	Execution time (ms)	Percentage
Initialization	236	1.158
Coding	4121	20.214
Sequence conversion	1683	8.255
Prediction	2638	12.941
DCT/Q/Q⁻¹/IDCT	10519	51.598
Total	20388	100
Encoding frame rate = 7.4		

Execution with dual-core

The follow table shows the implementation result of the proposed dual-core architecture with interrupt mode – FIQ. And we also use the built-in hardware extension module of DCT and IDCT for improving I MB encoding. It shows that this architecture will increase efficiency if ARM core take a part to share the computation load from DSP core. And the content of A/D in the follow table shows the ratio of tasks executed on ARM core and DSP core.

Table 22 Experiment result of dual-core

Qcif,150 I frames	Execution time (ms)	A/D	Percentage
Initialization	236		1.212
Coding	4133		21.227
Sequence conversion	1684		8.647
Prediction	2652		13.622
DCT/Q/Q⁻¹/IDCT	9572	1:6.07	49.159

Total	19471		100
Encoding frame rate = 7.7			

Execution with dual-core, DMA

The follow table shows the implementation result of the proposed dual-core architecture with interrupt mode – FIQ. DMA module is used to improve the transfer between ARM core and DSP core. And we use the built-in hardware extension module of DCT and IDCT for improving I MB encoding. Through this design, ARM can reduce the transfer overhead from transfer data to DSP by ARM core support. Instead of transfer completely by ARM core, we just let ARM core to set the DMA module for transfer and then ARM core can continue its original job. And thus, since ARM have more ability to handle its original jobs; we can see the condition that ARM executes more jobs of all from the content of A/D.

Table 23 Experiment result of dual-core with DMA

Qcif,150 I frames	Execution time (ms)	A/D	Percentage
Initialization	236		1.266
Coding	4128		22.147
Sequence conversion	1685		9.042
Prediction	2620		14.059
DCT/Q/Q⁻¹/IDCT	8765	1:3.71	47.031
Total	18637		100
Encoding frame rate =8.0			

Performance comparison

In the follow table, it lists main experiment result of Intra frame processing and show the final improvement ratio on our dual-core architecture.

Table 24 Performance comparison

Module name	ARM core	DSP core	Dual-core	Dual-core with DMA	Improvement ratio
DCT/Q/Q⁻¹/IDCT	22297	10519	9572	8765	2.54

From the above table, it shows that execution on DSP alone has higher efficiency

than execution on ARM core alone. But even though, dual-core architecture will still have the ability to improve the performance. And from the experimental result, one can also see that if the system does transfers complete by ARM core's support, the performance increase is minor. But if DMA modules are added to this dual-core architecture, the system will have great improvement.

5.1.2 Profile for dual-core I macroblock encoding module

After seeing the general results of each module on intra frame processing, we will further measure the detail execution status of each module for realizing each module's behavior and its bottlenecks for future improvements.

In the follow figure, it shows the main computation components of dual-core Intra macroblock encoding module. The execution time of each module will be measured in the follow table.

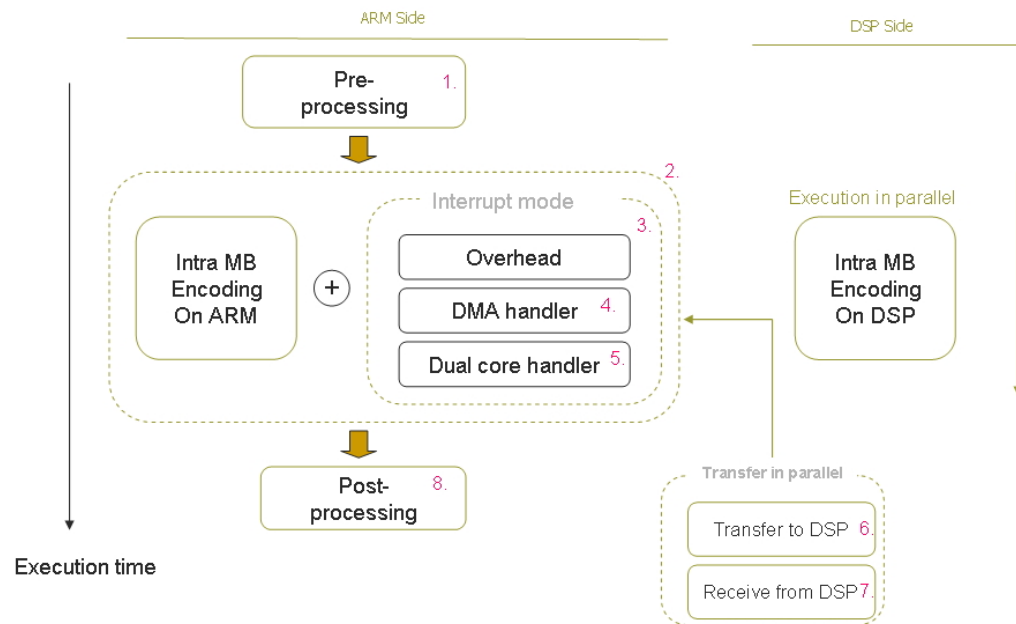


Fig 34 Profile for dual-core I macroblock encoding module

Table 25 Experiment result of profile I MB encoding

Number	Execution time (ms)	Description
1	1253	Time in pre-processing
2	5286	Time in dual-core processing
3	682	Time in interrupt mode
4	343	Time in handling DMA handler
5	122	Time in handling Dual-core handler

6	761	Time in transfer data to DSP by DMA
7	1520	Time in receive data from DSP by DMA
8	2191	Time in post-processing
A/D	1:4.18	
Total time	8730	

In the above figure, we can generally classify the sequentially execution time into three sections. The first one is pre-processing section; in this section it perform the operations of adjust the data format of this frame into proper format for intra MB encoding.

And the second section is the main dual-core execution section in this I macroblock encoding module, it lets ARM core, DSP core and DMA module executes in parallel. And there is an interrupt mode component in the figure; this module is accessed while either DMA or DSP asserts an interrupt and then the control of ARM will fall into corresponding interrupt handler. Besides executing these handlers, there are some overheads on handling the interrupt architecture, so that we can see the overhead component in this interrupt mode. And by these experiment result, ARM core execution time and DSP core execution time can thus be calculated from the follow two equations.

$$\text{ARM core execution time} = \text{Execution time of ((2) - (3))}$$

$$\text{DSP core execution time} = \text{Execution time of ((2) - ((3) + (6) + (7)))}$$

In the third section, we do format adjustment from the dual-core execution section to proper format for later use.

As a result, we can compute these two execution times in the follow table.

Table 26 Calculated result of profile P MB encoding

Description	Execution time (ms) or Percentage
ARM core execution time	4604
DSP core execution time	2323
Pre-processing section percentage in this module	14%
Dual-core execution section percentage in this module	60%
Post-processing section percentage in this module	25%

5.2 Experiment of Inter frame processing

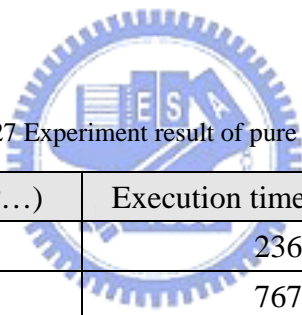
5.2.1 Overall results

In this section, the experiments for inter frame processing is presented. The encoder prediction pattern used here is IPPP.... The main focus here is to test the interpolation module, the motion estimation module, and the transform/quantization module. Since dual-core I MB encoding module has been tested before, it will set to the fastest mode in this experiment.

Execution with pure ARM core

The follow table shows the implementation result of execution on the ARM core alone. Thus, the original performance of this codec will be presented.

Table 27 Experiment result of pure ARM core



Qcif, 150 frames (IPPP...)	Execution time (ms)	Percentage
Initialization	236	0.140
Coding	767	0.456
Set edge	1027	0.610
Sequence conversion	1686	1.002
Prediction	70	0.042
Rate control	3464	2.058
Motion compensation	4375	2.600
I MB encoding	60	0.035
Interpolation	8693	5.166
Motion estimation	132649	78.831
DCT/Q/Q ⁻¹ /IDCT	13328	7.921
Total	168270	100
Encoding frame rate =0.9		

Execution with pure DSP core

The follow table shows the implementation result of execution on theDSP core

alone.

Table 28 Experiment result of pure DSP core

Qcif, 150 frames (IPPP...)	Execution time (ms)	Percentage
Initialization	236	0.277
Coding	771	0.903
Set edge	1027	1.203
Sequence conversion	1686	1.975
Prediction	75	0.088
Rate control	3463	4.057
Motion compensation	4370	5.120
I MB encoding	60	0.070
Interpolation	6917	8.104
Motion estimation	54712	64.096
DCT/Q/Q⁻¹/IDCT	10124	11.861
Total	83559	100
Encoding frame rate =1.8		

Execution with pure DSP core, FIQ

The follow table shows the implementation result of execution using only the DSP core with fast interrupt mode.

Table 29 Experiment result of pure DSP core, FIQ

Qcif, 150 frames (IPPP...)	Execution time (ms)	Percentage
Initialization	236	0.277
Coding	779	0.913
Set edge	1027	1.203
Sequence conversion	1686	1.975
Prediction	74	0.086
Rate control	3463	4.058
Motion compensation	4370	5.120
I MB encoding	58	0.068
Interpolation	6908	8.093
Motion estimation	54703	64.092
DCT/Q/Q⁻¹/IDCT	10118	11.854
Total	85351	100

Encoding frame rate =1.8

Execution with pure DSP core, FIQ, HW extensions

The follow table shows the implementation result of execution using only the DSP core with fast interrupt mode and built-in hardware extensions.

Table 30 Experiment result of pure DSP core, FIQ, HW extensions

Qcif, 150 frames (IPPP...)	Execution time (ms)	Percentage
Initialization	236	0.309
Coding	783	1.024
Set edge	1028	1.344
Sequence conversion	1686	2.205
Prediction	73	0.095
Rate control	3464	4.530
Motion compensation	4373	5.719
I MB encoding	58	0.076
Interpolation	6695	8.755
Motion estimation	50745	66.362
P MB encoding	5409	7.074
Total	76466	100
Encoding frame rate =2.0		

Execution with dual-core

The follow table shows the implementation result of execution on the proposed dual-core architecture with fast interrupt mode and built-in hardware extensions.

Table 31 Experiment result of dual-core core

Qcif, 150 frames (IPPP...)	Execution time (ms)	A/D	Percentage
Initialization	236		0.324
Coding	768		1.053
Set edge	1029		1.411
Sequence conversion	1686		2.311
Prediction	72		0.098
Rate control	3464		4.749

Motion compensation	4375		5.999
I MB encoding	58		0.080
Interpolation	6505	1:18.8	8.919
Motion estimation	48013	1:15.299	65.828
P MB encoding	4814	1:6.35	6.600
Total	72937		
Encoding frame rate =2.0			

Execution with dual-core, DMA

The follow table shows the implementation result of execution on the proposed dual-core architecture with fast interrupt mode and built-in hardware extensions. And DMA module is used to help the transfer between ARM and DSP.

Table 32 Experiment result of dual-core with DMA

Qcif, 150 frames (IPPP...)	Execution time (ms)	A/D	Percentage
Initialization	236		0.461
Coding	787		1.537
Set edge	1027		2.006
Sequence conversion	1686		3.292
Prediction	74		0.145
Rate control	3464		6.764
Motion compensation	4370		8.534
I MB encoding	58		0.114
Interpolation	5026	1:2.96	9.816
Motion estimation	28209	1:4.31	55.088
DCT/Q/Q⁻¹/IDCT	4343	1:3.85	8.481
Total	51207		100
Encoding frame rate =2.9			

Execution with dual-core, DMA, ME with interpolate on the fly

The follow table shows the implementation result of execution on our proposed dual-core architecture with fast interrupt mode and built-in hardware extensions. And DMA module is used to help the transfer between ARM and DSP. Besides, motion estimation module which interpolates half-pixel reference frames on-the-fly is adopted to reduce the load of transfer.

Table 33 Experiment result of dual-core, DMA, enhanced ME

Qcif, 150 frames (IPPP...)	Execution time (ms)	A/D	Percentage
Initialization	236		0.556
Coding	803		1.891
Set edge	1025		2.414
Sequence conversion	1686		3.969
Prediction	70		0.164
Rate control	3464		8.156
Motion compensation	4375		10.302
I MB encoding	59		0.139
Interpolation	5185	1:3.12	12.209
Motion estimation	19212	1:6.59	45.238
DCT/Q/Q⁻¹/IDCT	4430	1:3.82	10.430
Total	42468		100
Encoding frame rate =3.5			

Performance comparison

And in the follow table, here lists some experiment results of Inter frame processing to show the improvement ratio of our dual-core architecture.

Table 34 Performance comparison

Module name	ARM core	DSP core	Dual-core	Dual-core with DMA	Improvement ratio
Interpolation	8693	6695	6505	5026	1.67
Motion estimation	132649	50745	48013	28209 19212(OTF)	6.90
DCT/Q/Q⁻¹/IDCT	13328	5409	4814	4343	3.00

From the above table, it shows that the dual-core architecture perform great performance on inter frame processing. And after the use of DMA module, the dual-core architecture becomes more efficient.

5.2.2 Profile for dual-core Interpolation module

The follow figure shows the main computation components of dual-core Interpolation module with the fastest mode experimented before. And the follow table

shows the execution time of each component in this module.

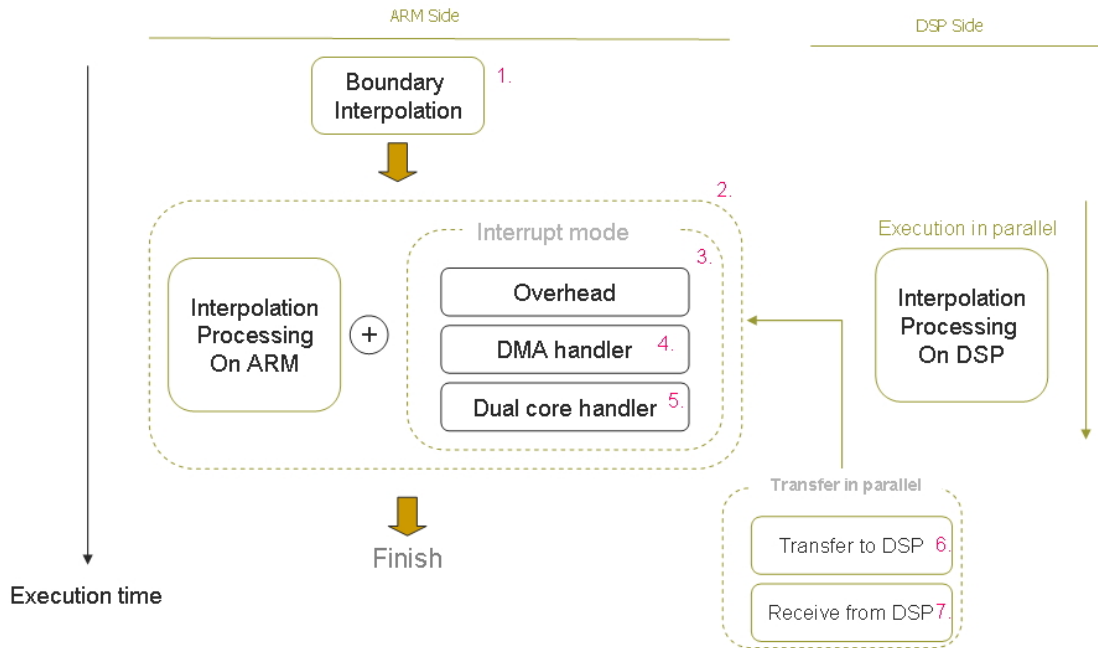


Fig 35 profile for dual-core Interpolation module

Table 35 Experiment result of profile interpolation module

Number	Execution time (ms)	Description
1	2106	Time in boundary interpolation
2	2919	Time in dual-core processing
3	808	Time in interrupt mode
4	444	Time in handling DMA handler
5	114	Time in handling Dual-core handler
6	541	Time in transfer data to DSP by DMA
7	1458	Time in receive data from DSP by DMA
A/D		1:2.95
Total time		5025

The operation of boundary interpolation is to interpolate the margin of each frame, since its computation complexity is lower, so that in our implementation it is not included in the dual-core execution modules. Besides, the transfer overhead may be greater than the performance gain from execution with DSP support. The results are shown in Table 36.

Table 36 Calculated result of profile interpolation module

Description	Execution time (ms) or Percentage
ARM core execution time	2111
DSP core execution time	112
Boundary interpolation section percentage	41%
Dual-core execution section percentage	58%

From the above tables, we can find that the transfer time of DMA is greater than DSP core execution time. In fact, while we implement the dual-core interpolation module in the early stage, we sometimes face the condition that the performance decreases than execution on pure ARM core. Because the transfer overhead is very large, so that we can take the computation complexity of this module as a frame of reference before we implement other dual-core modules.

5.2.3 Profile for dual-core Motion Estimation module

The follow figure shows the main computation components of dual-core Motion estimation module with the fastest mode we experimented before. We have measured the motion estimation and the motion estimation with interpolation on the fly modules.

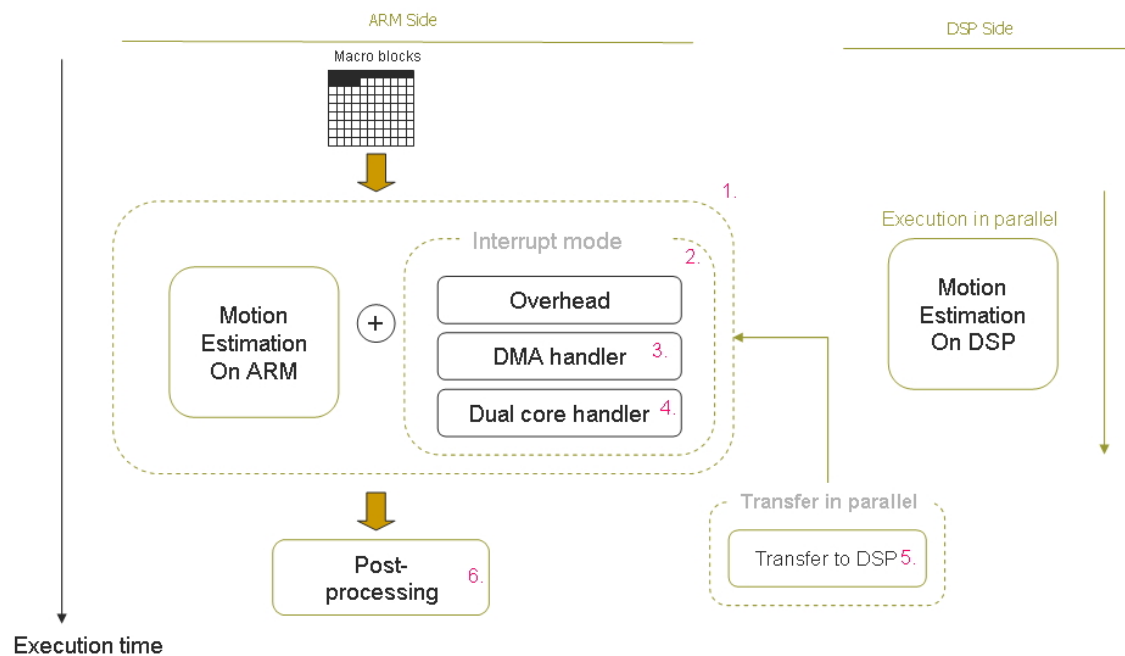


Fig 36 Profile for dual-core Motion Estimation module

The operations of post processing here are deciding the motion vector mode of each macroblock after motion estimation. At first, it shows the result of motion estimation without interpolation on the fly on the DSP side.

Table 37 Experiment result of profile Motion estimation module

Number	Execution time (ms)	Description
1	28303	Time in dual-core processing
2	1406	Time in interrupt mode
3	726	Time in handling DMA handler
4	292	Time in handling Dual-core handler
5	19269	Time in transfer data to DSP by DMA
6	499	Time in post processing
A/D	4.19	
Total time	28803	

Table 38 Calculated result of profile Motion estimation module

Description	Execution time (ms) or Percentage
ARM core execution time	26897
DSP core execution time	7628
Dual-core execution section percentage	98%
Post processing section percentage	1%

The results in the above table were calculated from the above table. It shows that the transfer time is greater than the DSP core execution time. Obviously, this implementation does not take full advantage of the DSP core. Therefore, we have implemented another one motion estimation method.

The follow tables are the results of motion estimation with interpolation on the fly on the DSP side.

Table 39 Experiment result of profile Motion estimation module

Number	Execution time (ms)	Description
1	18599	Time in dual-core processing
2	683	Time in interrupt mode
3	217	Time in handling DMA handler
4	292	Time in handling Dual-core handler
5	5810	Time in transfer data to DSP by DMA
6	489	Time in post processing
A/D	6.58	
Total time	19088	

Table 40 Calculated result of profile Motion estimation module

Description	Execution time (ms) or Percentage
ARM core execution time	17916
DSP core execution time	12106
Dual-core execution section percentage	97%
Post processing section percentage	2%

From the above tables, it illustrates that the DSP core execution time increases and the transfer time decreases. It also shows the DSP core's computation ability is much greater than the transfer ability of system DMA in our experiment environment. As a result, this method has been adopted as our final implementation.

5.2.4 Profile for dual-core P macroblock encoding module

The follow figure shows the main computation components of dual-core P macroblock processing module with the fastest mode we mentioned before.

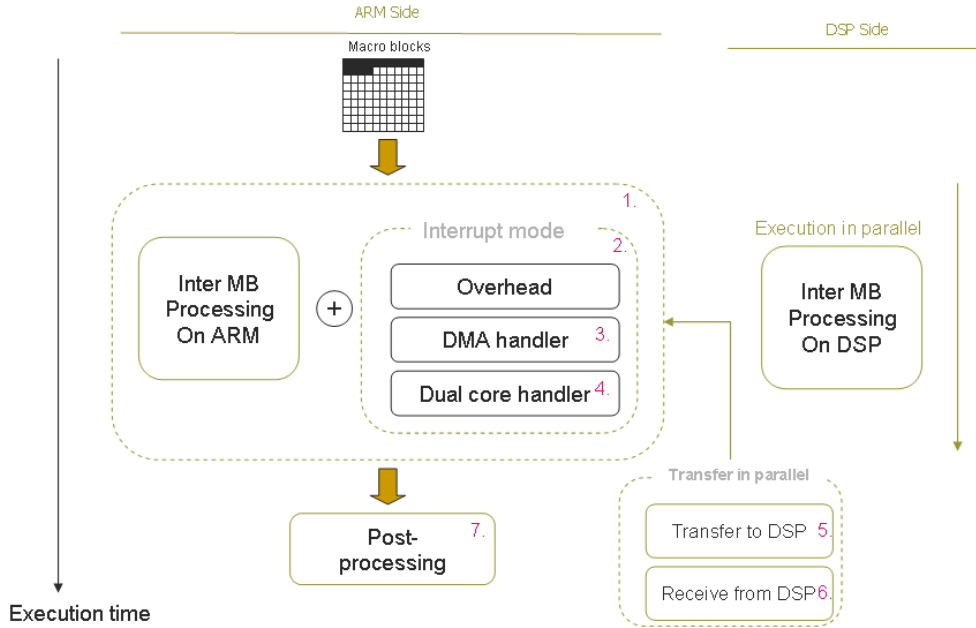


Fig 37 Profile for dual-core P macroblock encoding module

We experimented with two different implementations of DMA control modules in this thesis. The first one transfers all memory section of the reconstructed frames from DSP to ARM. This implementation has lower complexity of DMA handler, but increases the amount of redundant transfer. The second implementation transfers memory sections of the reconstructed frames from DSP to ARM according to its encoding status. Although this approach makes the DMA handler more complex, it removes redundant data transfer.

The follow two tables show the result of transfer with redundant transfer.

Table 41 Experiment result of profile P MB encoding

Number	Execution time (ms)	Description
1	4261	Time in dual-core processing
2	689	Time in interrupt mode
3	357	Time in handling DMA handler
4	126	Time in handling Dual-core handler
5	727	Time in transfer data to DSP by DMA
6	1452	Time in receive data from DSP by DMA
7	523	Time in post-processing
A/D		3.11
Total time		4785

Table 42 Calculated result of profile P MB encoding

Description	Execution time (ms) or Percentage
ARM core execution time	3572
DSP core execution time	1393
Dual-core execution section percentage	89%
Post processing section percentage	10%

And the follow two tables show the result of transfer without redundant transfer.

Table 43 Experiment result of profile P MB encoding

Number	Execution time (ms)	Description
1	3813	Time in dual-core processing
2	742	Time in interrupt mode
3	397	Time in handling DMA handler
4	136	Time in handling Dual-core handler
5	756	Time in transfer data to DSP by DMA
6	895	Time in receive data from DSP by DMA
7	526	Time in post-processing
A/D		3.74
Total time		4341

Table 44 Calculated result of profile P MB encoding

Description	Execution time (ms) or Percentage
ARM core execution time	3071
DSP core execution time	1420
Dual-core execution section percentage	87%
Post processing section percentage	12%

Obviously, it shows that the implementation which transfers data without redundancy is more efficient, although the overhead of the DMA handler becomes more complex than the other method, but its overall performance gain is better than the implementation redundant transfer.

Chapter 6 Conclusions and Future Works

From these experiments, one can see that the proposed dual-core codec partitioning framework achieves better performance than using the DSP core alone. The thesis also shows that data transfer overhead between the RISC core and the DSP core is crucial to the performance of the system. Efficient use of DMA module for data transfer also plays an important role in this framework. For future improvements, instead of executing jobs at frame-level, data structures and execution flows of our codec should be modified for execution at slice-level or macroblock-level. This allows the combination of multiple function modules into one single module and reduces large data transfer overhead. Fig. 38 shows this concept, the left-hand side of the figure shows current execution flow, and the right-hand side shows the improved architecture for future work.

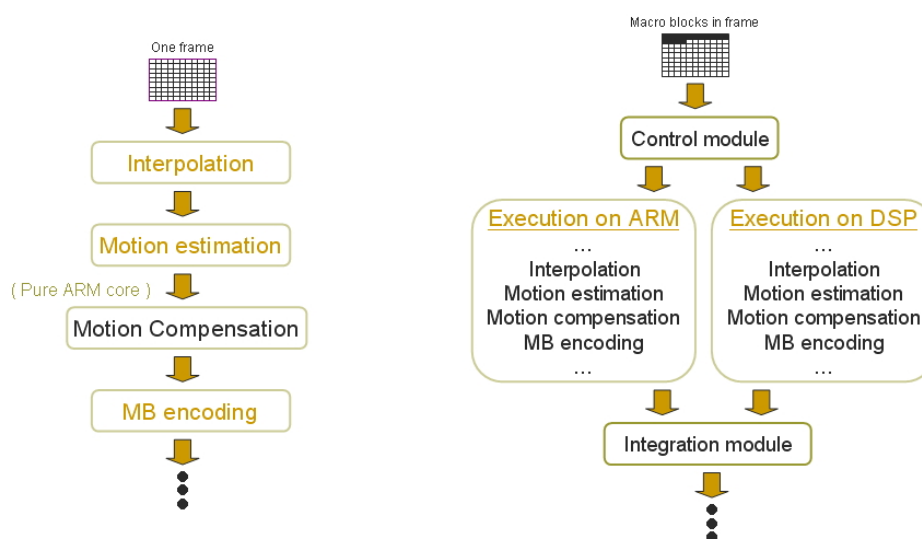


Fig 38 Architecture of future work

In our current design, motion compensation module is running on the RISC core alone. With the above-mentioned modification, the motion estimation/compensation subtasks can be completely hosted on the same core (either RISC or DSP) without extra data transfer overhead.

In addition, as demonstrated by many researches, employing dual buffer mechanism on the DSP core can increase memory bandwidth greatly. This is a key technique to improve system performance. It can be expected that we can also use similar design to increase performance of our design, since one of the major bottleneck of the proposed dual-core framework is from the limited memory bandwidth between the RISC core and the DSP core.

Finally, the research conducted in this thesis is a prelude to the design of a dynamic scheduling kernel for asymmetric multiple processors (AMP) platforms. Based on the experiments conducted in this thesis and the simple shell-like DSP command processor developed for this work, one can design an AMP kernel that dispatch tasks to different processor cores on the fly.

Chapter 7 References

- [1] S. De-Gregorio, M. Budagavi, and C. Chaoui, *Bringing Streaming Video to Wireless Handheld Devices*, Texas Instrument Technical White Paper SWPY005, May 2002
- [2] ISO/IEC JTC 1/SC 29/WG11, *Information technology -- Coding of Audio-visual objects - Part II: Visual*, ISO/IEC 14496-2:2003, Apr. 2003.
- [3] Jamil Chaoui, Ken Cyr, Sebastien de Gregorio, Jean-Pierre Giacalone, Jennifer Webb, Yves Masse, *Open multimedia application platform: enabling multimedia applications in third generation wireless terminals through a combined RISC/DSP architecture*, Proceeding of ICASSP2001, Pages:1009 - 1012 vol.2, May 2001
- [4] Kyu Ha Lee, Keun-Sup Lee, Tae-Hoon Hwang, Young-Cheol Park and Dae Hee Youn, *An architecture and implementation of MPEG audio layerIII decoder using dual-core DSP*, IEEE Transactions on Consumer Electronics, Vol 47, No4, NOVEMBER 2001
- [5] Olli Lehtoranta, Timo Hamalainen and Jukka Saarinen, *Real-time H.263 encoding of QCIF-images on TMS320C6201 fixed point DSP*, ISCAS 2000 - IEEE International Symposium on Circuits and Systems, May6 28-31, 2000, Geneva, Switzerland
- [6] Atsushi Hatabu, Takashi Miyazaki, and Ichiro Kuroda, *QVGA/CIF resolution MPEG-4 video codec based on a low-power and general-purpose DSP*, IEEE 2002
- [7] James Song, Thomas Shepherd, Minh Chau, Ayesha Huq, Ikram Syed, Somdipta Roy, Achuta Thippiana, Kaijian Shi, Uming Ko. *A low power open multimedia application platform for 3G wireless*, IEEE 2003
- [8] Byeong-Doo Choi, Kang-Sun Choi, Sung-Jea Ko, Senior Member, IEEE, and Aldo W. Morales, Senior Member, IEEE, *Efficient real-time implementation of MPEG-4 audiovisual decoder using DSP and RISC chips*. IEEE 2003
- [9] *TMS320 C55x User guide* (TI publication)
- [10] *OMAP5910 Dual-Core Processor Technical Reference Manual*, SPRU602B, January 2003 (TI publication)
- [11] Rishi Bhattacharya, *System Initialization for the OMAP5910 Device*, SPRA828A, August 2002 (TI publication)
- [12] *TMS320C55x Hardware Extensions for Image/Video Applications Programmer's Reference*, SPRU098, February 2002 (TI publication)
- [13] *DSP/BIOS Bridge Programming Guide (WinCE/BIOS) Version 1.10*, November 22, 2002 (TI publication)
- [14] Hans-Joachim Stolberg, Mladen Berekovic, Lars Friebe, Sören Moch, Sebastian Flügge, Xun Mao, Mark B. Kulaczewski, Heiko Klußmann, and Peter Pirsch, *HiBRID-SoC: A Multi-Core System-on-Chip Architecture for Multimedia Signal Processing Applications*, Proceedings of the Design, Automation and Test in

- Europe Conference and Exhibition (DATE'03), IEEE 2003
- [15] Thanh Tran, Ph.D. *OMAP5910 Video Encoding and Decoding*, SPRA985 December 2003
(TI publication)
- [16] Bill Winderweede, *OMAP System DMA Throughput Analysis*, SPRA883 – December 2002
(TI publication)

