H.264/AVC 影像編碼系統在 TI DSP 系統平台上之實現與加速

Acceleration and Implementation of H.264/AVC Based Visual Communication System on TI DSP Platform

研 究 生：陳奕安

指導教授：王聖智 博士

中 華 民 國 九 十 七 年 十 月

H.264/AVC 影像編碼系統在 TI DSP 系統平台上之實現與加速

Acceleration and Implementation of H.264/AVC based Visual Communication System on TI DSP Platform

研　究　生：陳奕安　　　　Student：Yi-An Chen

指導教授：王聖智博士　　　Advisor：Dr. Sheng-Jyh Wang

國　立　交　通　大　學

電子工程學系　電子研究所碩士班

碩　士　論　文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of Master

in

Electronics Engineering

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年十月

H.264/AVC 影像編碼系統在 TI DSP 系統平台上之實現與加速

研究生：陳奕安　　　指導教授：王聖智 博士

國立交通大學

電子工程學系　電子研究所碩士班

摘要

在本論文中，我們針對了 H264/AVC 的影像壓縮規格，實現了一個即時影像傳輸系統。包含影像接收-壓縮-網路傳送端，以及網路接收-解壓縮-播放端，在一端送出經過 H.264/AVC 編碼技術壓縮過的資料，經過網際網路傳輸後可以被另一端收到並進行解碼。我們使用了多線程緒的執行方式，來達成此即時系統。針對 DM642 數位處理晶片，我們提出平行化的方法，並且也對具有多顆數位訊號處理晶片的 MEX 系統做平行化的處理。

Acceleration and Implementation of H.264 based Visual Communication System on TI DSP Platform

Student: Yi-An Chen Advisor: Dr. Sheng-Jyh Wang

Department of Electronics Engineering, Institute of Electronics

National Chiao Tung University

Abstract

In this thesis, we implement an H.264/AVC based real-time video communication system. The two ends of this system include video capturing/encoding/network-transmission and network-reception/ decoding/video-display. The H.264/AVC encoded data are transmitted from one end to the other end. The whole procedure is implemented in multiple threads. To speed up the coding process, both optimization and parallelization of the DSP codes are performed with respect to the DM642 DSP chip and the multi-DSP board, MEX.

誌謝

在這幾年的學習生涯中，我首先要感謝指導教授王聖智老師。從大三開始進實驗室做專題，到碩士班結束共四年時間，老師在其中不僅悉心教導了許多做研究的方法與態度，也展現了待人接物該有的氣度以及處理事情的技巧，使我獲益匪淺。在此向老師致上最高的感謝之意。

同時我也要感謝每次下班還特別撥空與我討論的信嘉學長，不僅耐心的與我討論，也不厭其煩的指出我研究中的缺失。另外也要感謝影像處理實驗室的敬群、禎宇學長，在實驗室時不斷鼓勵我給予我良好的建議。也感謝俊晟、博凱、晴駿同學的幫忙，在實驗室中歡樂的度過美好的時光。當然也感謝瑞男、庭瑋、文中、維辰你們諸多的幫忙。

最後，感謝我的家人與朋友給予我的鼓勵，在此將論文獻給我有有幫助過我的，陪我走過這段求學生涯的所有師友、同學、朋友和家人。

Content

List of Figures

List of Tables

Chapter 1.

INTRODUCTION

1.1. INTRODUCTION

With its higher compression efficiency than all prior video coding standards, the latest video compression standard H.264, which is also known as MPEG-4 part 10 or MPEG-4 AVC, is expected to become the major video standard in the coming years. H.264/AVC provides high coding efficiency through the addition of new features and functionalities. With the H.264/AVC standard, the size of a digital video can be reduced up to 80% than the Motion JPEG format and up to 50% than the MPEG-4 Part-2 standard.

On the other hand, the demand for multimedia services over internet is steadily increasing. With its high coding efficiency, H.264/AVC has become one of the most favorite video compression standards to transmit videos over the internet. However, the high complexity of the H.264/AVC coding process has made the implementation of the H.264/AVC standard very difficult.

The general-purpose Digital Signal Processor (DSP) has been widely used in the implementation of various algorithms. The C64x DSP family, developed and provided by the Texas Instruments (TI), is a popular choice for digital media applications. In this thesis, we implement an H.264/AVC based video communication system based on the multi-DSP board MEX (Multi-Channel Video Platform), which possesses four TMS320DM642 DSP chips. The H.264 based video transmission is implemented in terms of multiple threads. Moreover, to speed up the encoding/decoding process, the optimization and parallelization of the DSP codes are investigated in this thesis.

1.2. OVERVIEW OF THE THESIS

The rest of the thesis is organized as follows. Chapter 2 contains the brief introduction to the H.264/AVC coding standard. In Chapter 3, a brief overview of the DSP platform and the development environment is represented. In Chapter 4, a multi-task multi-thread implementation of the H.264 based video communication system is discussed. Finally, conclusions are given in Chapter 5.

Chapter 2.

CONSPECTUS OF H.264 STANDARD

H.264, also known as MPEG-4 Part 10 or MPEG-4 AVC, is the state-of-the-art video coding standard. It is proposed by the Joint Video Team of both the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Pictures Experts Group (MPEG). The final drafting work on the first version had been completed in May 2003 [1].

The primary goal of H.264/AVC is to develop a video coding standard with high coding efficiency and network-friendly video representation. As shown in Figure 2-1, the H.264 covers a Video Coding Layer (VCL), which efficiently represents the video content, and a Network Abstraction Layer (NAL), whose formats are appropriate for conveyance over particular transport layers or storage media. With the state-of-the-art coding tools, it can achieve lower bit rates than all prior standards, like MPEG-2, H.263, and MEPG-4 Part-2 [2]. Moreover, its packed-based video representation addresses both conversational and non-conversational applications. Outperforming earlier standards, H.264/AVC is becoming the worldwide digital video standard for consumer electronics and video broadcasting. In this chapter, the H.264/AVC standard is briefly introduced. More details about H.264/AVC can be accessed in [3].

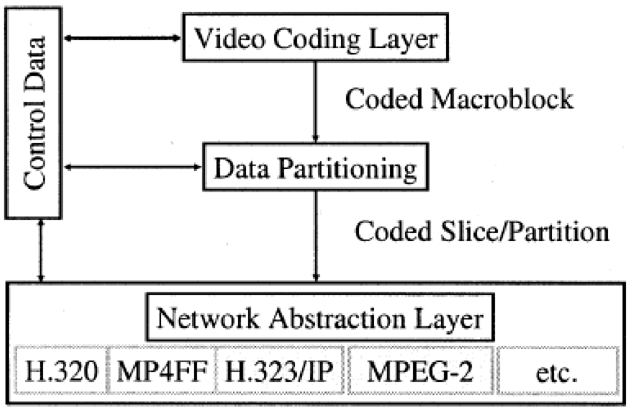


Figure 2-1 Structure of an H.264/AVC video encoder [4]

2.1. OVERVIEW OF H.264/AVC

As shown in Figure 2-2, the scope of H.264/AVC standard includes only the decoder of the typical video coding /decoding chain. The decoder is standardized by prescribing the Bitstream syntax and defining the decoding process. This limitation of the scope of the standard allows the maximal freedom to the encoder for different applications.

Although the encoder /decoder pair is not explicitly defined, encoder and decoder are likely to include the functional elements shown in Figure 2-3 and Figure 2-4 to be complaint to the standard.

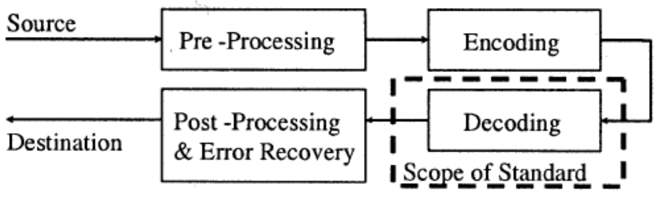


Figure 2-2 Scope of H.264/AVC [4]

2.1.1. THE H.264/AVC ENCODER

A block diagram of a typical H.264/AVC encoder is shown in Figure 2-3. The encoding process is divided into several functionality block diagrams. Except the deblocking filter, most of these functional components (intra/inter prediction, transformation, quantization, entropy encoding) had been presented in these previous standards. However, some important changes in the details of each functional block occur in H.264.

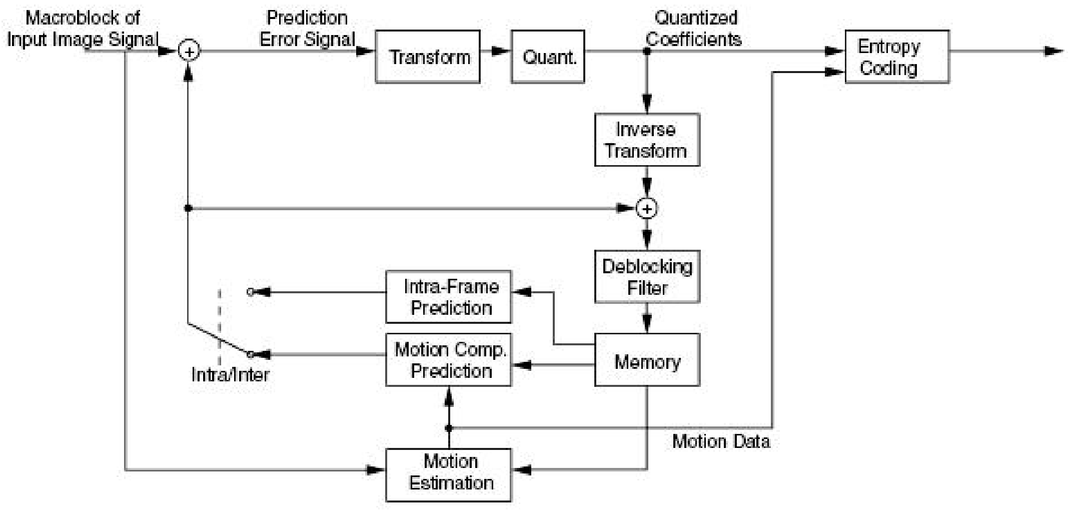


Figure 2-3 H.264/AVC Encoder[5]

The intra prediction and motion estimation/compensation removes spatial redundancy and temporal redundancy respectively. After that, the prediction mode and the residual data are recorded. Then the transformation and quantization are adopted to transform residual data into more suitable data space to drop some details those are less perceptible to human vision. The entropy coding removes the syntax redundancy. In addition, the deblocking is performed to reduce the blocking effect in reconstruction path.

2.1.2. THE H.264/AVC DECODER

Figure 2-3 shows the block diagram of the H.264/AVC decoder. The entropy decoder decodes the quantized coefficients and the motion data, which is used for the motion compensated prediction. As in the encoder, prediction data are obtained by intra or motion estimation, which is added to the inverse transformed coefficients. After deblocking filtering, the macroblock is completely decoded.

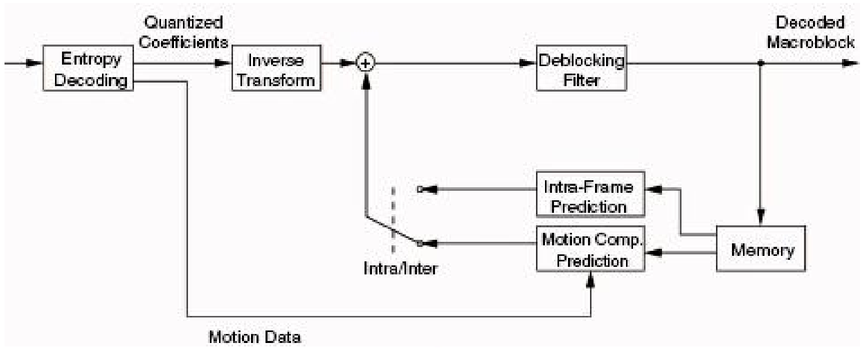


Figure 2-4 H.264/AVC Decoder[5]

2.2. PROFILE AND LEVELS

There are three profiles defined in H.264/AVC standard, these profiles are baseline profile, main profile, and extended profile. The profile is adopted flexibly for different application. The baseline profile, supporting intra coding and inter coding, together with entropy coding with CAVLC is primary for lower-cost application. Designed as the mainstream consumer profile, the main profile supports interlaced video, B-picture, inter coding using weighted prediction and entropy coding using CABAC. With robustness to data losses, the extended profile does not support interlaced video and CABAC, but adds modes to enable switching between Bitstream and to improve error resilience. Table 2-1 lists the coding tools and features of these three profiles.

Table 2-1 Coding tools and features of different profiles [3]

	Baseline	Extended	Main
I and P Slices	Yes	Yes	Yes
B Slices	No	Yes	Yes
SI and SP Slices	No	Yes	No
Multiple Reference Frames	Yes	Yes	Yes
In-Loop Deblocking Filter	Yes	Yes	Yes
CAVLC Entropy Coding	Yes	Yes	Yes
CABAC Entropy Coding	No	No	Yes
Flexible Macroblock Ordering (FMO)	Yes	Yes	No
Arbitrary Slice Ordering (ASO)	Yes	Yes	No
Redundant Slices (RS)	Yes	Yes	No

2.3. INTER PREDICTION

By using the previous encoded video frames or fields, inter prediction can be established from motion estimation and motion compensation. Similar to the prior coding standard, the block-based motion compensation is used. However, variable block size is different from the earlier standards and makes it more efficiency than earlier standards.

In prediction procedure, a predicted block P is searched from the reference picture F_{n-1} by motion estimation. Motion Vector (MV) is the displacement from the current block to the predicted block P. With the encoded information of MVs and residual, motion compensation can reconstruct the current picture from the reference picture F_{n-1}. In this standard, MVs have accuracy of quarter-sample resolution to achieve higher coding efficiency. Next, we will describe these features of H.264 inter prediction

2.3.1. TREE-STRUCTURE MOTION COMPENSATION

In H.264/AVC standard, the luma component of each macroblock can be segmented into one 16x16 partition, two 8x16 partitions, two 16x8 partitions, or four 8x8 partitions, as shown in Figure 2-1. In Figure 2-6, if the 8x8 partitions is chosen, each 8x8 block can be further divide into four different sub-partitions, including 8x8, 8x4, 4x8, and 4x4. In general, the large partitions are appropriate for smooth regions; the smaller partitions have smaller residual, but the number of motion vectors is increased. With the flexibility of variable block-size motion compensation, the coding efficiency can be increased.

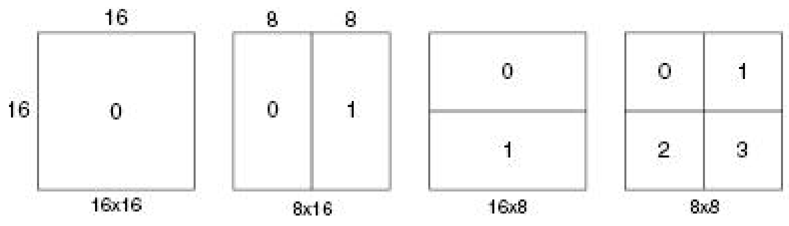


Figure 2-5 Macroblock partitions: 16x16, 16x8, 8x16 and 8x8 [3]

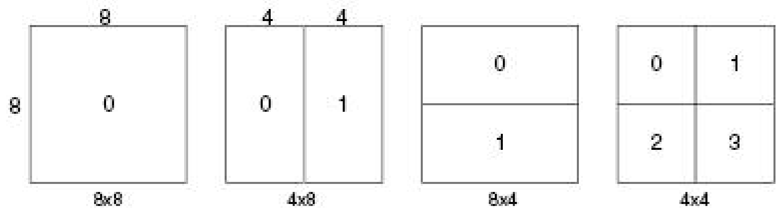


Figure 2-6 Macroblock sub-partitions: 8x8, 8x4, 4x8 and 4x4 [3]

2.3.2. FRACTIONAL PIXEL PRECISION

In order to increase the accuracy of motion compensation, H.264 supports quarter-pixel resolution for luma components and one-eight-pixel resolution for chroma components. If the prediction result of sub pixel is better than that of the integer pixel, the sub pixel will be chosen.

The half-pixel samples are obtained by applying a six tap filter with weights (1/32, -5/32, 20/32, 20/32, -5/32, 1/32). For example, a half pixel b in Figure 2-7 is obtained from the six horizontal integer neighbors E, F, G, H, I, and J with the formulation:

$$b = \text{round}((E - 5F + 20G + 20H - 5I + J)/32)$$

Furthermore, the quarter-pixel samples can be calculated after all the half-pixel macroblock are available. They are produced by linear interpolation between two of their adjacent samples. As shown in Figure 2-8, value of a can be calculate by:

$$a = \text{round}((G+b)/2)$$

In Figure 2-9, the chroma eight-sample component can be acquired by linear interpolation of the neighbor pixels:

$$a=\text{round}([(8-d_x)(8-d_y)A+d_x(8-d_y)B+(8-d_x)d_yC+d_xd_yD]/64)$$

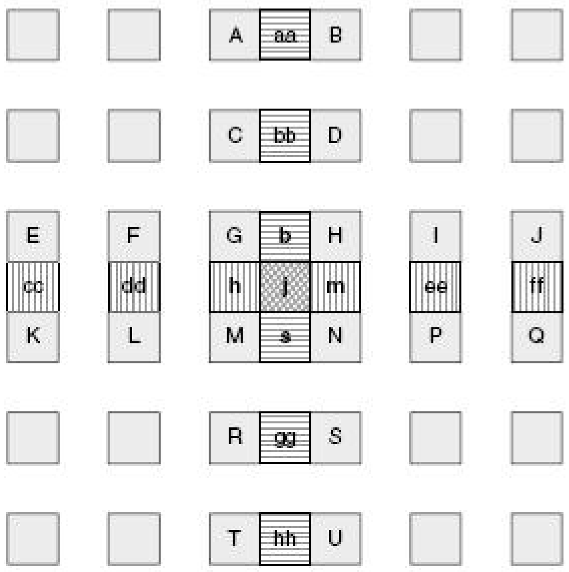


Figure 2-7 Interpolation of luma half-pel positions [3]

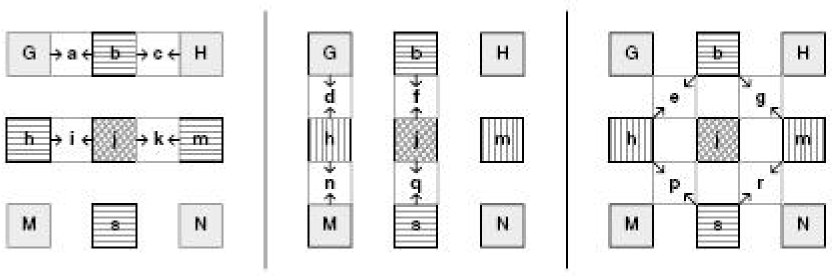


Figure 2-8 Interpolation of luma quarter-pixel positions [3]

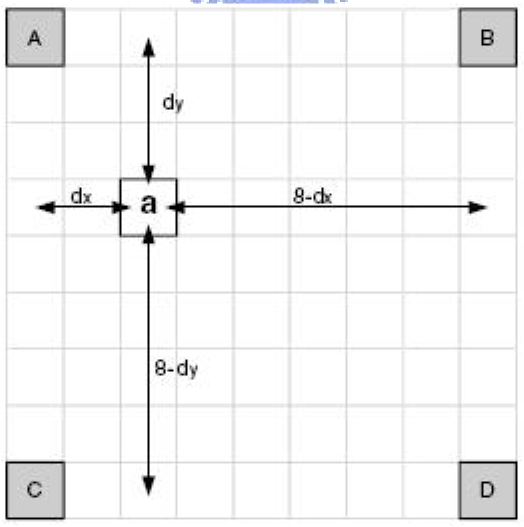


Figure 2-9 Interpolation of chroma samples [3]

2.3.3. MOTION VECTOR PREDICTION

As mentioned in 2.3.1, number of motion vectors increases with the using of variable block partition mechanism. It can cost a significant number of bits to encoding a motion vector for each partition. Since there are high correlations between motion vectors of the neighboring partitions, the motion vector can be predicted by nearby ones. Hence the motion vector prediction (MVp) is generated by the motion vector of the adjacent partitions. The way of forming the prediction MVp depends on the motion compensation partition size and on the availability of nearby vectors. MVp is obtained in a manner of: (see Figure 2-10)

- For 16x8 partitions, the MVp of the upper 16x8 partition is predicted from of B, and the MVp of the lower one is the motion vector of A.
- For 8x16 partitions, the MVp of the left 8x16 partition is predicted from of A, and the MVp of the right one is the motion vector of C.
- The MVp of other partitions is the median of the motion vector of A, B, and C.

The motion vector difference (MVD) is then derived calculate the difference between the MVp and the real motion vector. These MVDs are the final results that should be further encoded. In general cases, fewer bits are needed for encoding the MVDs than encoding real motion vectors.

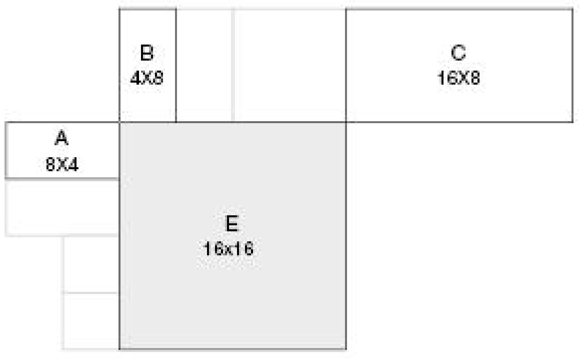


Figure 2-10 Current and neighboring partitions for MVp [3]

2.4. INTRA PREDICTION

The high correlation of neighboring region within a frame implies the high redundancy in spatial domain. As mentioned in 2.1.1, intra predication is imposed to eliminate the spatial redundancy. For the luma samples, intra prediction block is formed for each 4x4 block or 16x16 blocks; for the chroma samples intra prediction block is formed for each 8x8 blocks. The spatial prediction is calculated from the edges pixels of neighboring blocks.

2.4.1. 4X4 LUMA PREDICTION MODES

When intra mode of 4x4 blocks is applied, nine possible modes cab be chosen. As shown in Figure 2-11, the samples above and to the left (labeled A–M) have previously been encoded and reconstructed to form a prediction reference. The prediction block (the gray part) is calculated based in A-M. The arrows in Figure 2-11indicate the direction of prediction in each mode. In mode 0 and mode 1, respectively, the samples of A-D and I-L are extrapolated vertically and horizontally. Mode 2 (DC prediction) is modified depending on the availability of samples A to M. In the rest modes: Mode 3-8, the predicted samples are calculated by a weighted average of the reference samples A-M.

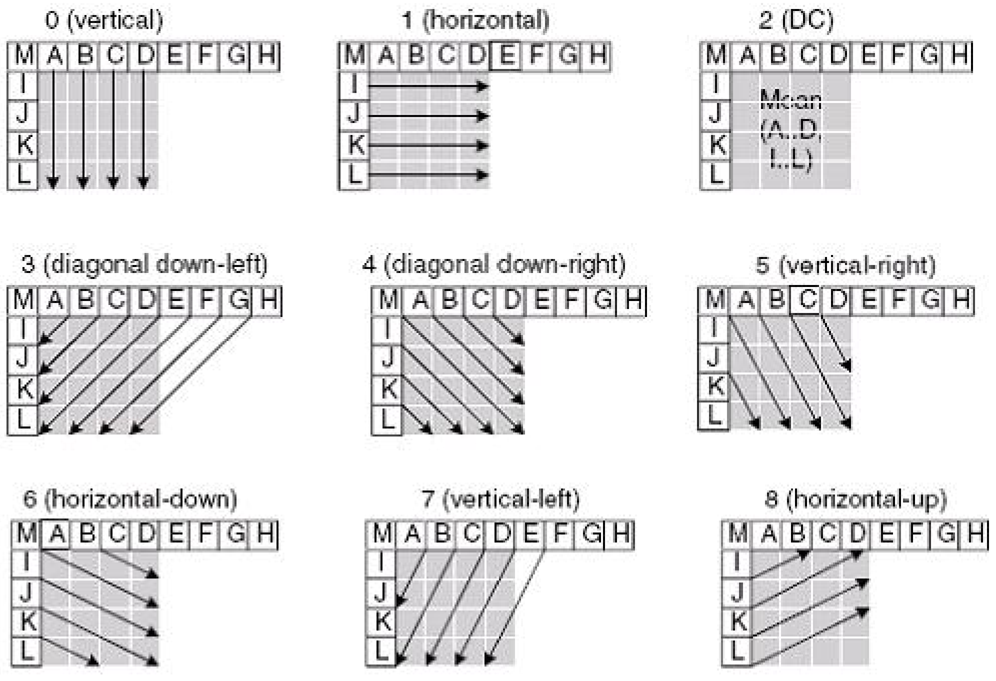


Figure 2-11 4 × 4 luma prediction modes [3]

2.4.2. 16X16 LUMA PREDICTION MODES

In addition to those 4x4 luma modes described in the previous section, there are four modes for 16x16 prediction modes for luma intra prediction. These four luma 16x16 prediction modes are vertical, horizontal, DC, and plane, as shown in Figure 2-12. The requirement of reconstruction of above and left component is similar to the 4x4 luma prediction.

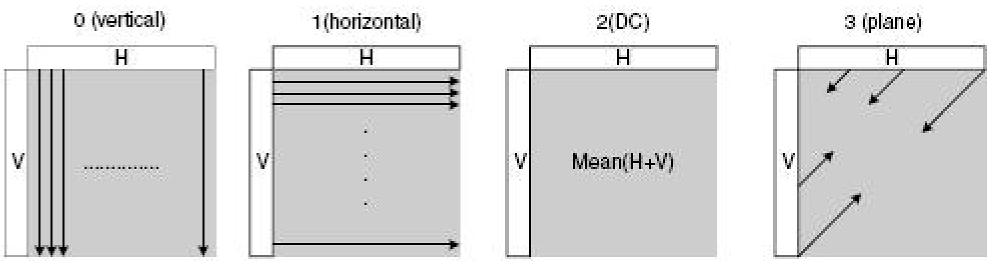


Figure 2-12 Intra 16 × 16 prediction modes [3]

2.4.3. 8X8 CHROMA PREDICTION MODES

Four 8x8 intra prediction modes are provided for the chroma samples. Similar to the 16x16 luma inter prediction in Figure 2-12, the four modes are DC, horizontal, vertical and plane.

2.5. IN-LOOP DE-BLOCKING FILTER

One drawbacks of the block base video compression mentioned above is the visible block boundaries. It is so called blocking effects: the lower bit rate the compression is, the more obvious the edges are. To eliminate the blocking effect, a deblocking filter is applied after the inverse transform in both encoder and decoder. As shown in Figure 2-13, it is applied to vertical or horizontal edges of 4x4 blocks in a macroblock, in the fallowing order: four vertical boundaries (a, b, c, then d) of luma, four horizontal boundaries (e, f, g, then h) of lima, and two vertical boundaries (i, j) horizontal boundaries (k, l).

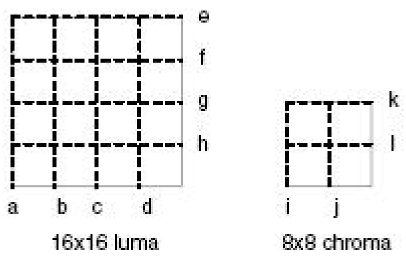


Figure 2-13 Edge filtering order in a macroblock [3]

The filtering is adaptively applied according to the boundary strength and the gradient across the boundaries. The boundary strength depends on the compression mode of a macroblock, the quantization parameter, motion vector, frame or field coding decision, and pixel values.

With this filter, subjective quality is significant improved as shown in Figure 2-14. This filter also reduces the bits rate with ratio of 5%–10% compared with non-filtered video with the same objective quality [4].

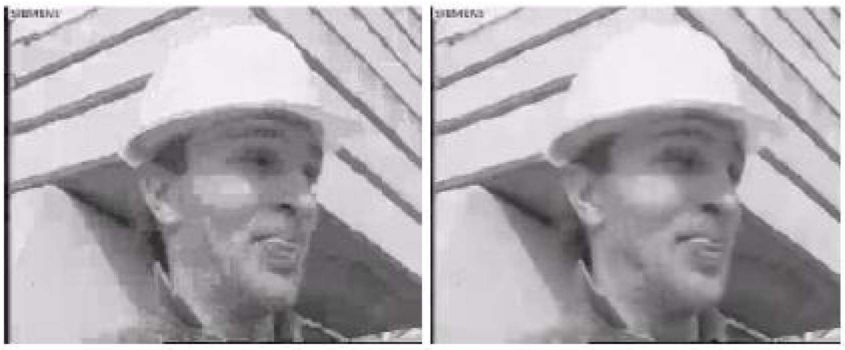

(a) (b)

Figure 2-14 Performance of the deblocking filter for highly compressed pictures (a) without deblocking filter and (b) with deblocking filter [4]

2.6. Transform and Quantization

H.264/AVC, as prior video standard, utilizes the transform coding on the prediction residual. The residual generated in intra or inter prediction is processed the transform for further quantization. One macroblock is divided into 24 4x4 blocks to do the 4x4 integer transform with the transform matrix:

$$H=\begin{bmatrix}1 & 1 & 1 & 1\\ 2 & 1 & -1 & -2\\ 1 & -1 & -1 & 1\\ 1 & -2 & 2 & -1\end{bmatrix}$$

In addition, for each macroblock a 4x4macroblock, a 4x4 Hadamard transform is applied to the DC coefficients of the 16 luma blocks, while a 2x2 Hadamard transform is applied to the DC coefficients of the 4x2 chroma blocks, as shown in Figure 2-15.

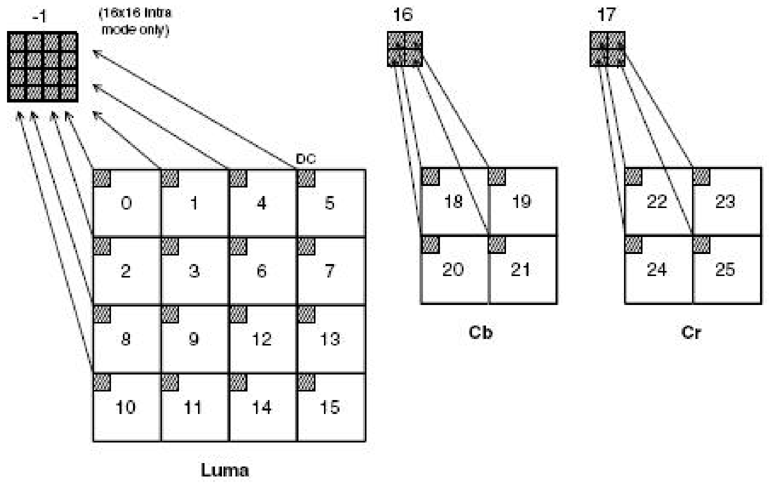


Figure 2-15 Scanning order of residual blocks within a macroblock. [3]

A quantization parameter is used to determine the quantization step for the quantization of transform coefficient. A total of 52 values of quantization step size (Qstep) are supported by this standard, which are indexed by the quantization parameter (QP). Increasing one in the value of QP means an increase of the quantization step size by approximately 12%. An increase of step size by 12% also means a reduction of bit rate by approximately 12% [4].

2.7. ENTROPY CODING

To eliminate the syntax redundancy, the arithmetic coding is applied. The syntax above the slice layer is encoded as fixed- or variable-length codes (VLCs). At the slice layer and below, elements are coded using Content Adaptive Variable Length Coding (CAVLC) or Content Adaptive Binary Arithmetic Coding (CABAC) according to the entropy encoding mode. Parameters that are required to be encoded and transmitted include the following (Table 2-2Table 2-1).

Table 2-2 Examples of parameters to be encoded

Parameters	Description
Syntax elements above slice layer	Headers and parameters
Macroblock type mb type	Prediction method for each coded macroblock
Coded block pattern	Blocks containing coded coefficients within a macroblock
Reference frame index	Identify reference frame(s) for inter prediction
Motion vector	Difference (mvd) from predicted motion vector
Residual data	Coefficient data for each 4 × 4 or 2 × 2 block

2.8. NAL UNIT

By choosing a suitable transporting protocol to represent of video coded content, the coded video is organized as a collection of NAL units. Each NALU is a video picket containing an integer number of bytes. As shown in , the first byte as a header byte of NALU contain NAL unite type (T), the nal_reference_idc (R) that indicates the importance of an NALU for the reconstruction process, and the forbidden_bit (F) which is set to '0' in H.264 encoding.

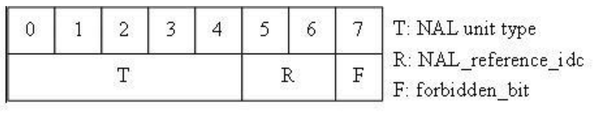


Figure 2-16 NALU header.

2.9. DATA DEPENDENCY OF H.264/AVC

Taking a macroblock as the basic elements In H.264/AVC, the data dependencies cross the macroblocks are illustrated in Figure 2-17 and Figure 2-18. Intra prediction needs the above and the left macroblock to be decoded, further for 4x4 luma block needs the up block, left block, and up right block information. And for deblocking filtering four tap in the upper macroblock and left to the macroblock are needed.

In Figure 2-18, data within the search range of the reference frame is needed to do the interprediciotn.

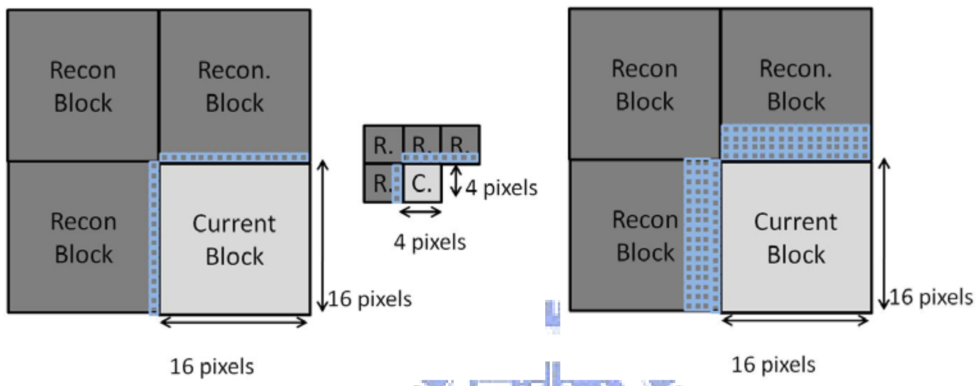


Figure 2-17 Data dependency induced by (Left) intra prediction and (Right)deblocking filter

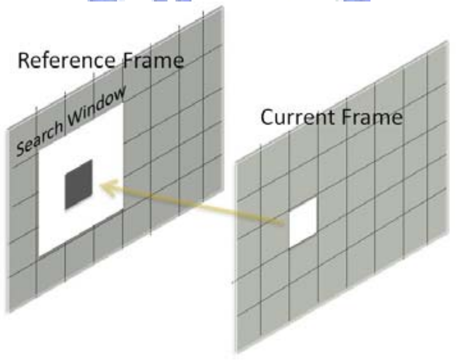


Figure 2-18 Data dependency induced by inter prediction

2.10. Complexity Analysis of H.264/AVC

The H.267/AVC standard only specifies the decoder, and the encoder design remains open. In this paper, we adopted the official H.264/AVC JM as decoder for integrity, and adopted the x264 encoder for the faster encoding speed. Thus we illustrate the complexity of the important functions in Figure 2-19 and Figure 2-20.

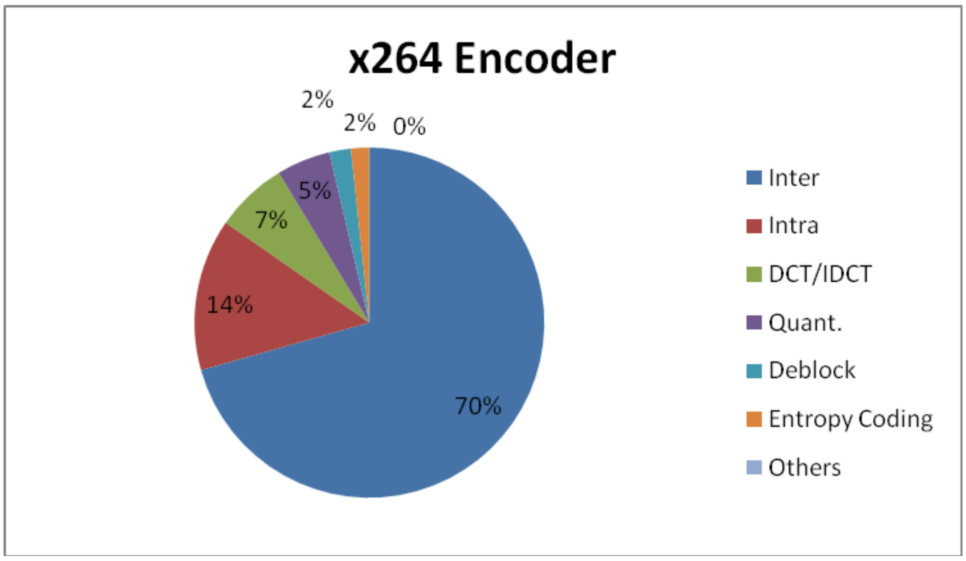


Figure 2-19 Distribution of clock cycle of each function of encoder.

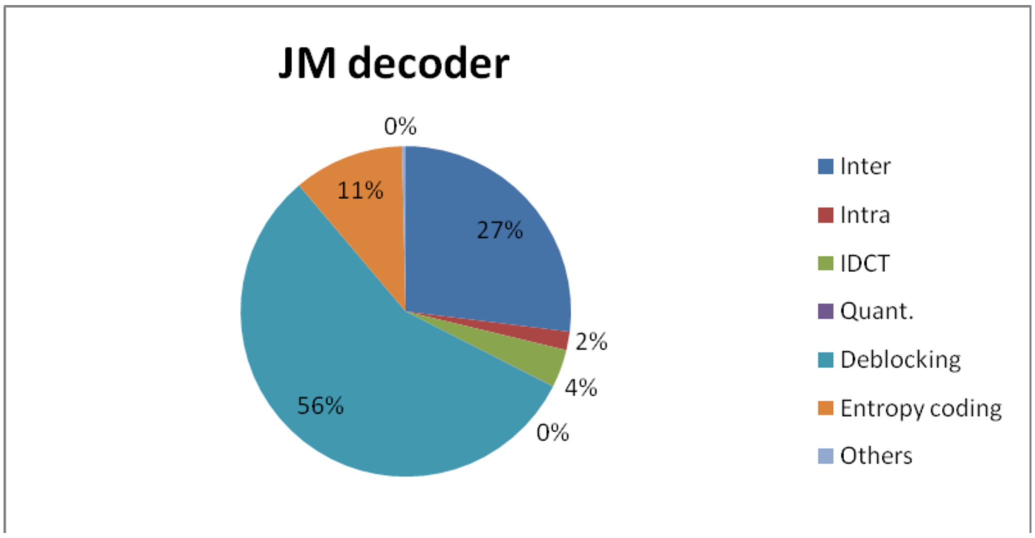


Figure 2-20 Distribution of clock cycle of each function decoder.

Chapter 3.

DSP Implementation Environment

In this chapter, we will briefly introduce the DSP platform environment and some optimization methods. We use the DSP module (MEX) made by Vitec Mult-Media. Four TMS320DM642 DSP chips are housed on this board. Our implementation system includes software system and some peripherals on the board. Thus for the TI DSP, the Code Composer Studio (CCS) and some efficient optimization methods will be introduced. In addition, to facilitate the system and peripherals, Reference Framework 5(RF5) and Network Developer's Kit (NDK) will be bring out as well.

3.1. Introduction of DSP Platform

The DSP board used in our implementation is the MEX (Multi-Channel Video Platform) in Figure 3-1, which is a powerful platform for video application. The architecture of MEX includes four TI DSPs, two FPGA (one as crossbar, the other as PCI interface), eight video decoders, four audio stereo ADCs, and a 100BaseT Ethernet controller, as shown in Figure 3-2.

MEX's key features are listed as below:

- ✓ Four TMS320DM642 DSPs run at up to 600MHz (Fixed point).
- ✓ Each DSP has a private memory of 32MB, which is SDRAM running at 100 MHz with 64 bits.
- ✓ Each DSP has three powerful configurable video ports. By configure the crossbar (implemented in an FPGA), the video architecture are flexible. With proper configuration, the video path way can distribute one vide source on four DSP, four distinct video sources on four DSP, four distinct video sources on one DSP, or so on.
- ✓ DSP-DSP communication or DSP-PCI communication is facilitated by the "Inter-DSP communication & PCI interface" FPGA. Each DSP has a dedicated FIFO inside the FPGA which is mapped in its memory. This FIFO can be written by the DSP and sent to PCI interface and the others DSPs. Those mean PC-DSP communication and DSP-DSP communication respectively.

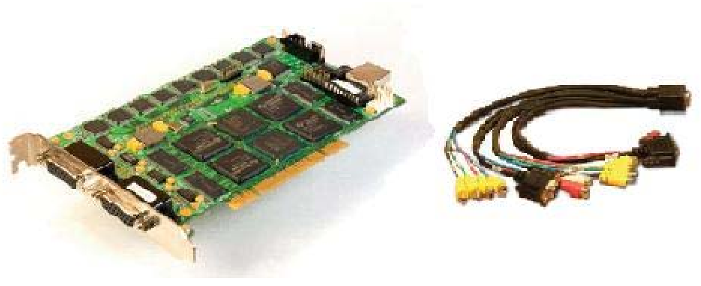


Figure 3-1 MEX (Multi-Channel Video Platform) [6]

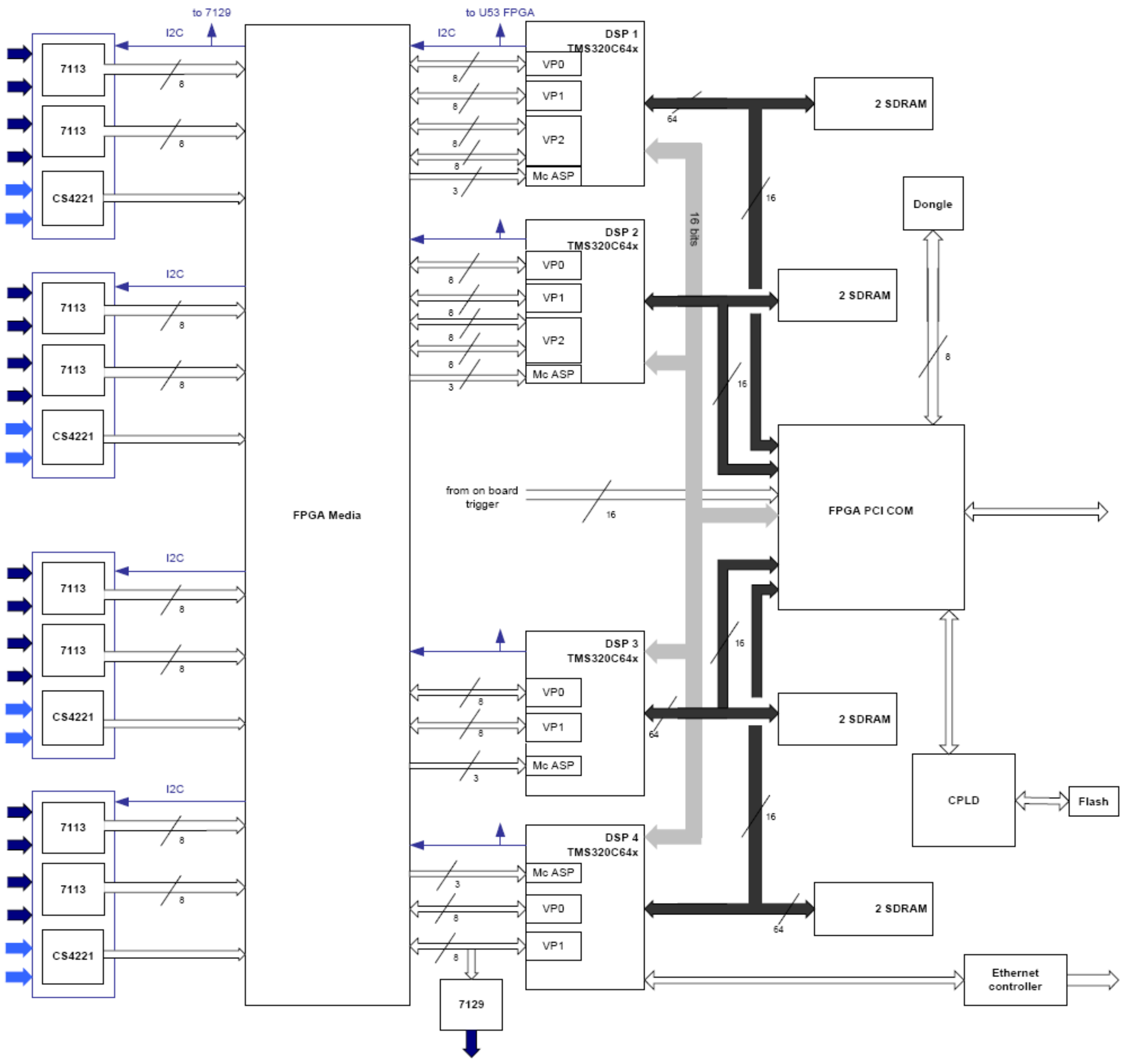
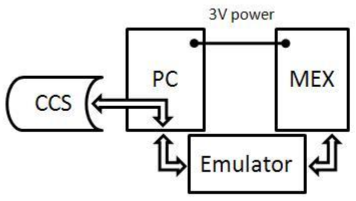
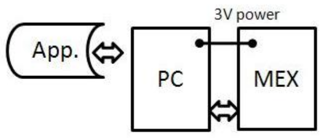


Figure 3-2 Block diagram of the MEX [6]

As shown in Figure 3-2, the flexible architecture include some modules of TMS320DM642 DSP chip: the I2C bus used to configure the Video (7113) / Audio(CS4221) chips, the Video Port set to configure the video acquisition data path, and EMIF that define the address of FPGA seen by DSP. Those DSP modules will further introduced in the following sections.



(a)



(b)

Figure 3-3 Block diagram of (a)emulator system and (b)application system

In the developing phase, a JTAG emulator pod called "USB 560BP" is used to connect the MEX to PC. With the JTAG emulator, the CCS emulation of DSPs on the board is fully supported. We develop our system and debug in this way. After that, the emulator can be removed from this system to expose the stand-alone ability of MEX. The only thing the PC should do is to supply 3V power and load the DSP program to the board. Figure 3-3 are two different block diagram of the system in emulation phase and in application phase.

3.2. DSP CHIP

In our system, the TMS320DM642 DSP chip is the most important part of this system. In this section, we will describe some details of this chip. TMS320DM642, the high-performance fixed-point DSP, is based on the second generation, high performance, advanced VelociTI™ very long instruction word (VLIW) architecture (VelociTI.2™), developed by Texas Instruments (TI). The VelociTI.2 extensions in the eight functional units include new instructions to accelerate the performance in key applications and extend the parallelism of the VelociTI architecture. This VLIW architecture makes the DSP chips an excellent choice for digital media application [8].

The DM642 DSP is a Video/Imaging fixed-point digital signal processor in the TMS320C64x family. It has eight independent functional units running at 600MHZ for peak execution of 4800 MIPS. Some key features of DM642 are listed below.

- ✓ Eight highly independent functional units - two multipliers to generate 32-bit result and six arithmetic logic units (ALUs)
- ✓ The VelociTI.2™ extensions in the eight functional units include new instructions to accelerate the performance in video and imaging manipulations

and to extend the parallelism of the VelociTI™ architecture.

- ✓ Conditional execution reduces cost of branch and increase parallelism.
- ✓ Instruction packing reduces code size, program fetches, and power consumption.
- ✓ 8/16/32/40-bit data support.
- ✓ Saturation and normalization provide support for key arithmetic operations.

Figure 3-4 is the functional block and DSP core diagram of TMS320C64x.

In the following sections, three major components of TMS320C64x DSP, including the central processing unit, memory, and peripherals, will be introduced.

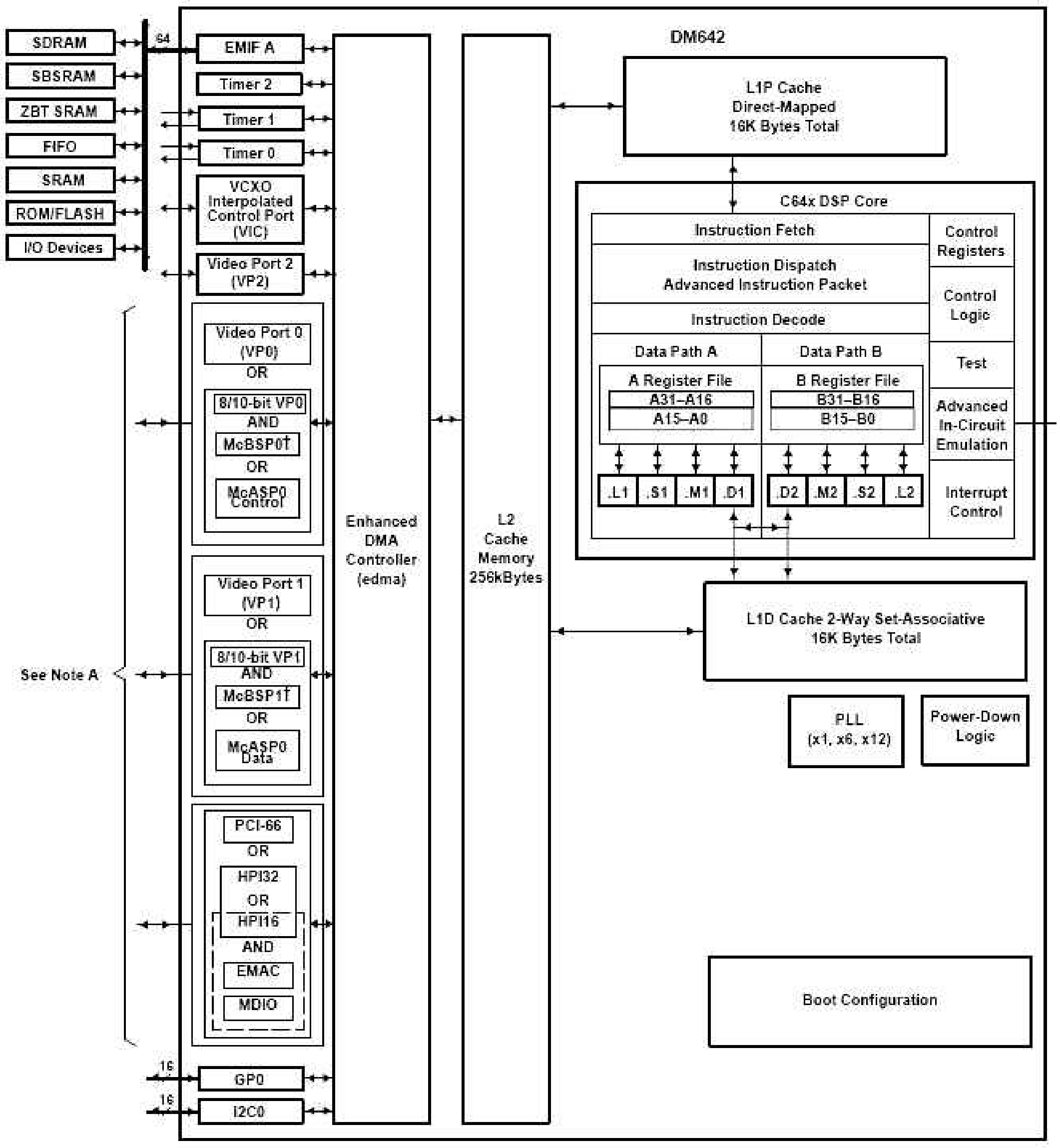


Figure 3-4 Block diagram of the TMSDM642 [9]

3.2.1. CENTRAL PROCESSING UNIT (CPU)

The DSP core of C64 series consists of eight independent fictional units, 64 general purpose registers, program fetch unit, instruction dispatch (attached with advanced instruction packing), instruction decode unit, two data path, test unit, emulation unit, interrupt logic, and etc. The instruction dispatch and decode units could decode and arrange the eight instructions to eight functional units respectively. Thus the program fetch, instruction dispatch, and instruction decode units can deliver up to eight 32-bit instructions to the functional units during every CPU clock cycle. The eight functional units in the C64 architecture could be further divided into two data paths, data path A and data path B as shown in Figure 3-4.

Each data path has 8 functional units for multiplication operations (.M), logical and, arithmetic operations (.L), branch, bit manipulation, and arithmetic operations (.S), and loading/storing and arithmetic operations (.D). Table 3-1 shows these functional units and their operations. Two cross data paths (1x and 2x) allow functional units from one data path to access a 32-bit operand from the register file from the opposite side. Most data lines in the CPU support 32-bit operands, while some support long (40-bit) operands. Each functional unit has its own 32-bit write port to a general-purpose register file and 32-bit read port for source operands src1 and src2 (refer to Figure 3-5). All function units which ends in 1(for example (.L1)) write to register file A, while those function units which end in 2 (for example (.M2)) write to register file B.

Table 3-1 Functional Units and Operations Performed [10]

Functional Unit	**Fixed-Point Operations**	**Floating-Point Operations**
.L unit (.L1, .L2)	32/40-bit arithmetic and compare operations 32-bit logical operations Leftmost 1 or 0 counting for 32 bits Normalization count for 32 and 40 bits **Byte shifts** **Data packing/unpacking** **5-bit constant generation** **Dual 16-bit arithmetic operations** **Quad 8-bit arithmetic operations** **Dual 16-bit min/max operations** **Quad 8-bit min/max operations**	Arithmetic operations DP →SP, INT →DP, INT →SP conversion operations
.S unit (.S1, .S2)	32-bit arithmetic operations 32/40-bit shifts and 32-bit bit-field operations 32-bit logical operations Branches Constant generation Register transfers to/from control register file (.S2 only) **Byte shifts** **Data packing/unpacking** **Dual 16-bit compare operations** **Quad 8-bit compare operations** **Dual 16-bit shift operations** **Dual 16-bit saturated arithmetic operations** **Quad 8-bit saturated arithmetic operations**	cal square-root operations Absolute value operations SP → DP conversion operations
.M unit (.M1, .M2)	16 x 16 multiply operations **16 x 32 multiply operations** **Quad 8 x 8 multiply operations** **Dual 16 x 16 multiply operations** **Dual 16 x 16 multiply with add/subtract operations** **Quad 8 x 8 multiply with add operation** **Bit expansion** **Bit interleaving/de-interleaving** **Variable shift operations** **Rotation** **Galois Field Multiply**	32 X 32-bit fixed-point multiply operations Floating-point multiply operations
.D unit (.D1, .D2)	32-bit add, subtract, linear and circular address calculation Loads and stores with 5-bit constant offset Loads and stores with 15-bit constant offset (.D2 only) **Load and store double words with 5-bit constant** **Load and store non-aligned words and double words** **5-bit constant generation 32-bit logical operations**	Load doubleword with 5-bit constant offset

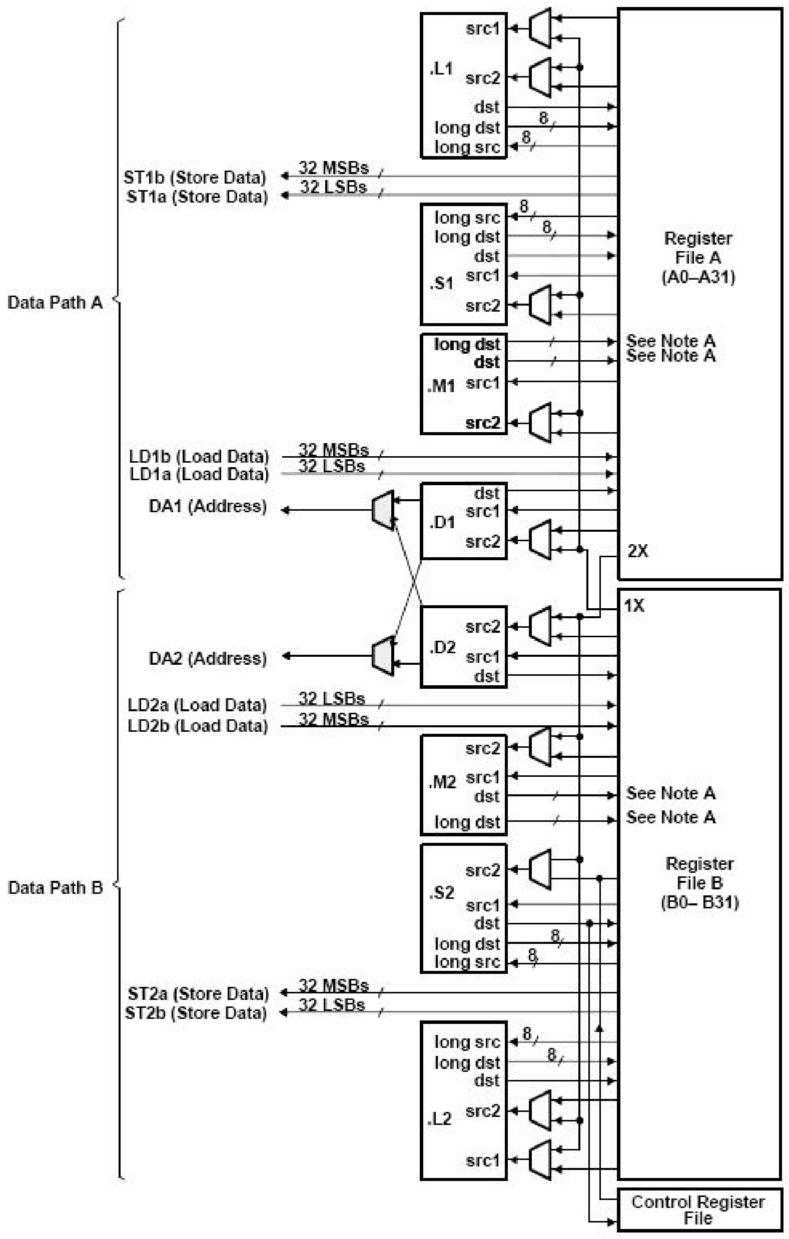


Figure 3-5 TMS320C64x^{TM} CPU (DSP Core)Data Paths [9]

3.2.2. MEMORY ARCHITECTURE

The DM642 uses a two-level cache-based architecture and has a powerful set of peripherals. This memory architecture consists of the following:

- ✓ Internal data/program memory
- ✓ External memory, with external memory interface (EMIF)
- ✓ Enhanced Directed-Memory-Access (EDMA)

Level 1 program cache (L1P) is a 128-Kbit direct mapped cache and the Level 1 data cache (L1D) is a 128-Kbit 2-way set-associative cache. The Level 2 memory/cache (L2) consists of a 2-Mbit memory space that is shared by both program space and data space. The TMS320DM642 internal program memory can be mapped into the CPU address space or operated as a program cache. There is a single port to access internal program memory, with an instruction fetch width of 256 bits. The internal data memory on C64x devices divides the memory into eight 32-bit wide banks. These banks are single-ported, allowing only one access per cycle. This is in contrast to the C621x/C671x devices, which use a single bank of dual-ported memory rather than multiple banks of single-ported memory. There are more details described in [11].

3.2.3. PERIPHERALS

The C64x contains some peripherals such as enhanced direct memory access (EDMA) controller, external memory interface (EMIF), video port peripheral, inter-integrated circuit (I2C) Bus module,10/100 Mb/s Ethernet MAC (EMAC), and etc.

3.2.3.1. EXTERNAL MEMORY INTERFACE (EMIF)

EMIF supports a glueless interface to a variety of external device, including:

- ✓ Pipelined synchronous-burst SDRAM (SBSRAM)
- ✓ Synchronous DRAM (SDRAM)
- ✓ Asynchronous device, including SDRAM, ROM, and FIFOs
- ✓ An external shared-memory device

On MEX board, EMIF serves as the interface between DSP to two SDRAM, memories of 1Meg×32bits×4banks (total 32MB), a synchronous FIFO to write/read data, and various registers via an asynchronous. Thus the external memory map is listed in Table 3-2.

Table 3-2 Memory map using EMIF of each DSP on MEX

Start Address	**End Address**	**Type of memory interface**	**Bus width**
0x80000000	0x81FFFFFF	SDRAM	64 bits
0x90000000		Synchronous FIFO	16 bits
0xB0000000	0xB000000E	Asynchronous interface	16 bits

3.2.3.2. The enhanced direct memory access (EDMA)

The enhanced direct memory access (EDMA) controller handles all data transfers between the Level-two (L2) cache/memory controller and the device peripherals on the C64x DSP. The EDMA controller in the C64x DSP has a different architecture from the previous DMA controller in the C620x/C670x devices. The EDMA includes several enhancements to the DMA, such as 64 channels for the C64x DSP, with programmable priority, and the ability to link and chain data transfers. The EDMA allows movement of data to/from any addressable memory spaces, including internal memory, peripherals, and external memory.

3.2.3.3. Video Port

The DM642 device has three configurable video port peripherals. These video port peripherals provide an interface to common video decoder and encoder devices. The DM642 video port peripherals support multiple resolutions and video standards. These three video port peripherals are configurable and can support video capture and/or video display modes. As shown in Video Port Block Diagram [12], each video port consists of two channels - A and B with a 5120-byte capture/display buffer being splittable between these two channels. The video port peripheral can operate as a video capture port, a video display port, or a transport stream interface (TSI) capture port. For the capture mode, the video port may operate as two 8/10 bits channels of BT.656 or raw video. It may also operate as a single channel of 8/10-bit BT.656, 8/10-bit raw video, 16/20-bit Y/C video, 16/20-bit raw video, or 8-bit TSI. For the display mode, the video port may operate as a single channel of 8/10-bit BT.656, 8/10-bit raw video, 16/20 bit Y/C video, or 16/20-bit raw video. It may also operate in a two-channel 8/10-bit raw mode. There are more details described in [12].

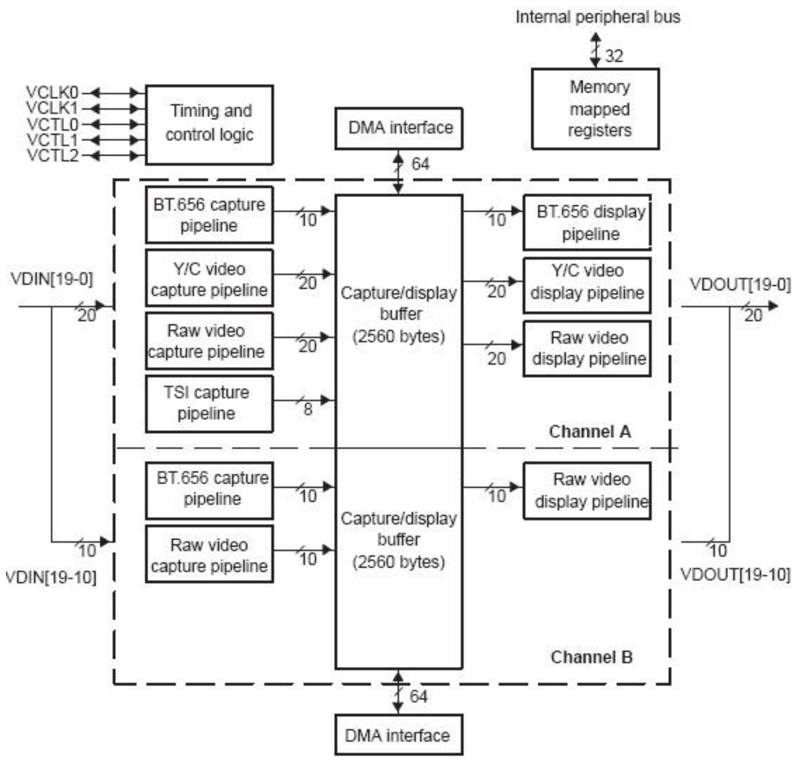


Figure 3-6 Video Port Block Diagram [12]

3.2.3.4. INTER- INTEGRATED CIRCUIT (I2C)

The inter-integrated circuit (I2C) module provides an ideal interface between TMS320C6000 DSP and other devices compliant with Philips Semiconductors Inter-IC bus (I^2C bus) specification. On the MEX board the I2C bus connects the DM642 chip to video (SAA7113) and audio (CS4221) chips, and is used to initial the video/audio chips and configure the video/audio data pathway with the format shown below.

Start	Slave Address	W	ACK	Sub Address	ACK-S	Data	ACK-S	Stop

Write Sequence

Start	Slave Address	W	Sub Address	ACK-S	
Start	Slave Address	R	Data	ACK-S	STOP

Read Sequence

Figure 3-7 I^2C bus format

3.2.3.5. ETHERNET MAC (EMAC)

The Ethernet media access controller (EMAC) provides an efficient interface between the DM642 DSP core processor and the network. It supports both 10Base-T and 100Base-TX, or 10 Mbits/second (Mbps) and 100 Mbps in either half- or full-duplex. The EMAC controls the flow of packet data from the DSP to the physical layer device (PHY). The MDIO module controls PHY configuration and status monitoring. Figure 3-8 Figure 2-1is the EMAC Control Module Block Diagram.

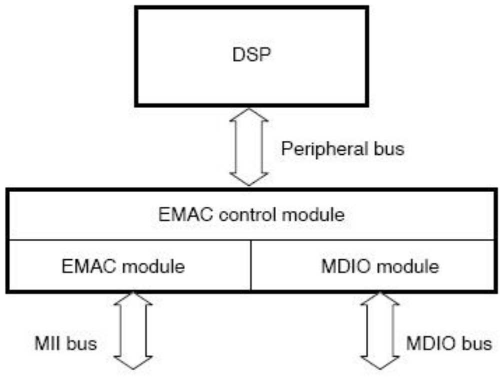


Figure 3-8 EMAC Control Module Block Diagram

3.3. Coding Development Environment

In this section, we will briefly introduce the coding environment of our project. The powerful coding environment tool called Code Composer Studio (CCS) will be described. In CCS, DSP programmers can develop the project, debug the project, and do some optimization. It's necessary for a DSP programmer to be familiar with the coding environment to develop a program efficiently.

3.3.1. Code composer studio

Code composer studio (CCS) extends the basic code generation toll with a set of debugging and real-time analysis capabilities. It speeds and enhances the development process for programmers who create and test real-time, embedded signal processing applications. Every phase of development cycle including conceptual design, coding &building, debugging, and real-time analysis is fully supported. Code Composer Studio includes the following components, which works together as show in:

- ✓ TMS320C6000 code generation tools
- ✓ Code Composer Studio Integrated Development Environment (IDE):
- ✓ DSP/BIOS plug-ins and API
- ✓ RTDX plug-in, host interface, and API

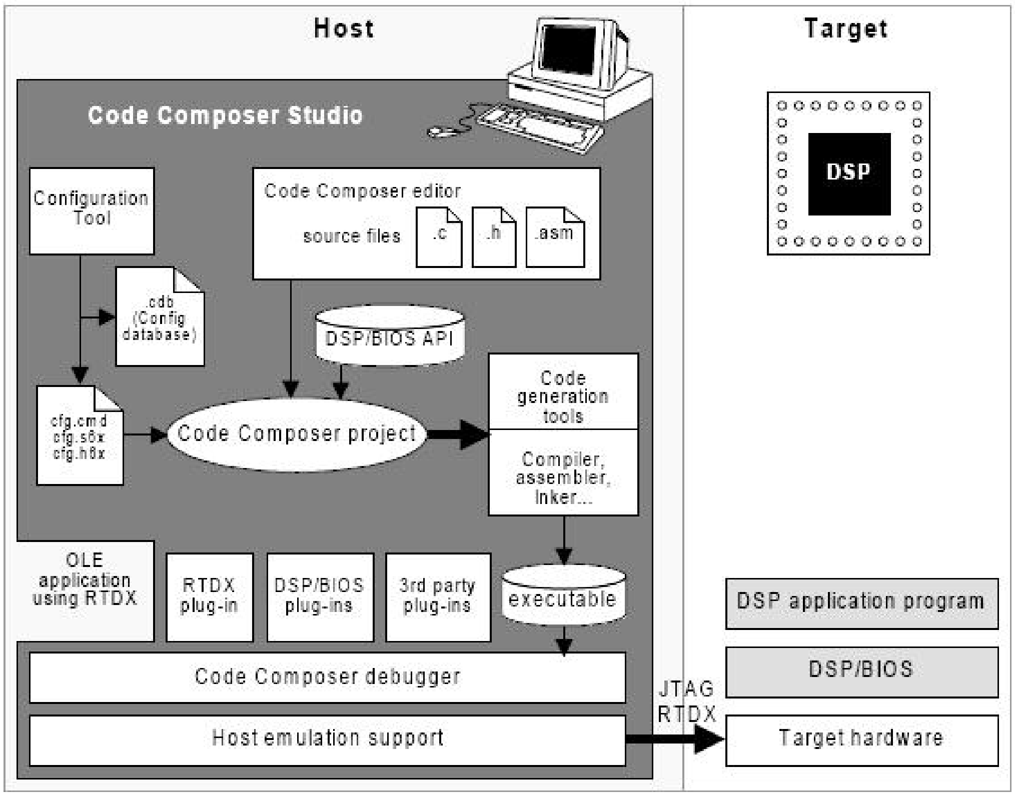


Figure 3-9 Code Composer Studio environment [14]

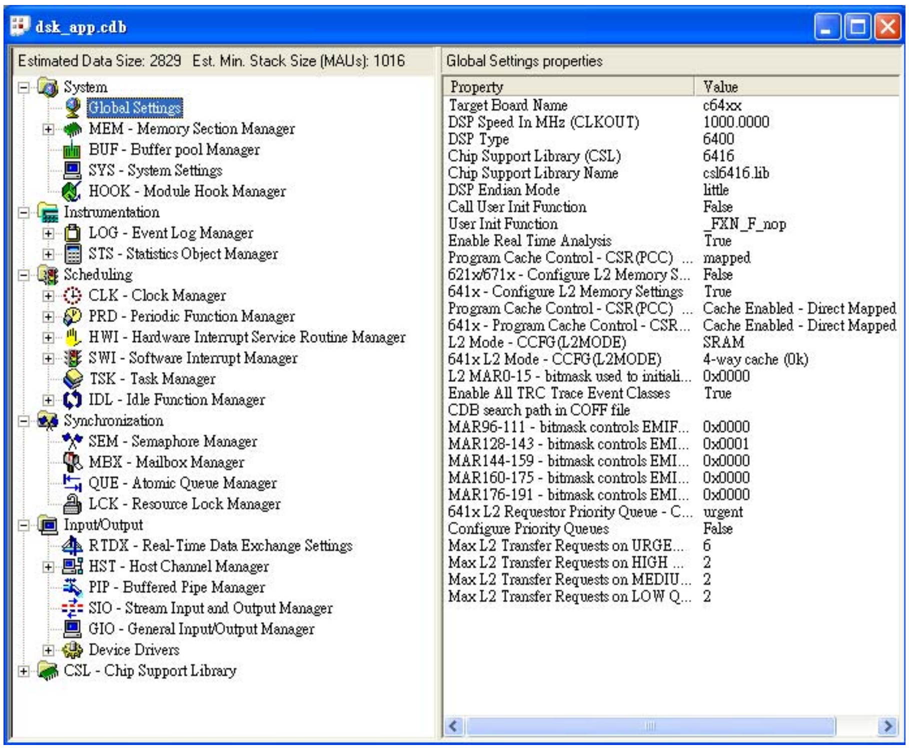
3.3.1.1. CODE GENERATION TOOL AND INTEGRATED DEVELOPMENT ENVIRONMENT

The foundation for the development environment provided by Code Composer Studio is consist of some code generation tools, including C compiler, assembler, assembly optimizer, linker, archives library-build utility, and etc.

The Code Composer Studio Integrated Development Environment (IDE) is designed to allow user to edit, build, and debug DSP target programs. In the coding phase, C source code and the corresponding assembly instructions can be shown and edit. In building phase, different files including C source files, assembly source files, object files, libraries, linker command files, and include files can be added to build the application. In debugging phase, flexibility to setting the breakpoints, accessibility to memory registers, graphical signal, statistics of execution profiling make it easier to debug.

3.3.1.2. DSP/BIOS PLUG-INS

DSP/BIOS gives DSP chips developers the ability to develop and analyze embedded real-time software. DSP/BIOS provides a graphical interface for static system setup, real-time scheduling, real-time analysis (RTA), and real-time data exchange (RTDX). By using the DSP/BIOS Configuration Tool, we can initialize data structures and set various parameters of DSP/BIOS objects. The Configuration Tool provides developers a windows explorer-like interface, as shown in DSP/BIOS Configuration Tool I, to use DSP/BIOS real-time library, DSP/BIOS API, and also CSL.



Estimated Data Size: 2829 Est. Min. Stack Size (MAUs): 1016

- System
  - Global Settings
  - MEM - Memory Section Manager
  - BUF - Buffer pool Manager
  - SYS - System Settings
  - HOOK - Module Hook Manager
- Instrumentation
  - LOG - Event Log Manager
  - STS - Statistics Object Manager
- Scheduling
  - CLK - Clock Manager
  - PRD - Periodic Function Manager
  - HWI - Hardware Interrupt Service Routine Manager
  - SWI - Software Interrupt Manager
  - TSK - Task Manager
  - IDL - Idle Function Manager
- Synchronization
  - SEM - Semaphore Manager
  - MBX - Mailbox Manager
  - QUE - Atomic Queue Manager
  - LCK - Resource Lock Manager
- Input/Output
  - RTDX - Real-Time Data Exchange Settings
  - HST - Host Channel Manager
  - PIP - Buffered Pipe Manager
  - SIO - Stream Input and Output Manager
  - GIO - General Input/Output Manager
  - Device Drivers
- CSL - Chip Support Library

Property	Value
Target Board Name	c64xx
DSP Speed In MHz (CLKOUT)	1000.0000
DSP Type	6400
Chip Support Library (CSL)	6416
Chip Support Library Name	csl6416.lib
DSP Endian Mode	little
Call User Init Function	False
User Init Function	_FXN_F_nop
Enable Real Time Analysis	True
Program Cache Control - CSR(PCC) ...	mapped
621x/671x - Configure L2 Memory S...	False
641x - Configure L2 Memory Settings	True
Program Cache Control - CSR(PCC) ...	Cache Enabled - Direct Mapped
641x - Program Cache Control - CSR...	Cache Enabled - Direct Mapped
L2 Mode - CCFG(L2MODE)	SRAM
641x L2 Mode - CCFG(L2MODE)	4-way cache (0k)
L2 MAR0-15 - bitmask used to initiali...	0x0000
Enable All TRC Trace Event Classes	True
CDB search path in COFF file	
MAR96-111 - bitmask controls EMIF...	0x0000
MAR128-143 - bitmask controls EMI...	0x0001
MAR144-159 - bitmask controls EMI...	0x0000
MAR160-175 - bitmask controls EMI...	0x0000
MAR176-191 - bitmask controls EMI...	0x0000
641x L2 Requestor Priority Queue - C...	urgent
Configure Priority Queues	False
Max L2 Transfer Requests on URGE...	6
Max L2 Transfer Requests on HIGH ...	2
Max L2 Transfer Requests on MEDIU...	2
Max L2 Transfer Requests on LOW Q...	2

Figure 3-10 DSP/BIOS Configuration Tool Interface

For real-time DSP applications, such as our system, it is possible to perform a number of seemingly unrelated functions at the same time. Such functions are called thread. DSP/BIOS enables applications to be structured as a collection of threads, with each of them carrying out a modularized function. Multi-thread programs run on a single processor by allowing higher-priority threads to preempt lower-priority threads, and by allowing various types of interactions among threads, including blocking, communication, and synchronization [15]. The thread types (from highest to lowest priority) provided by DSP/BIOS include: hardware interrupts (HWI), software interrupts (SWI), tasks (TSK), and Background thread (IDL). Programs using multithreads, as opposed to a single centralized polling loop, are easier to design, implement, and maintain.

Since the DSP/BIOS object tasks (TSK) is the major component of our multi-thread system, the way how tasks work is illustrated below. There are 15 level priorities and four states of execution, including running, ready, blocked, and terminated of tasks. Tasks are scheduled for execution according to a priority level assigned to the application. At a time only one task can be running, while other ready tasks are blocked due to their lower priorities. When a task with higher priority is ready, the current running task is blocked until higher-priority task is terminated.
As shown in Figure 3-11, TSK preempts the running task in favor of the higher-priority ready task. During the course of a program, each task's mode of execution can change for a number of reasons. The following figure shows how execution modes change.

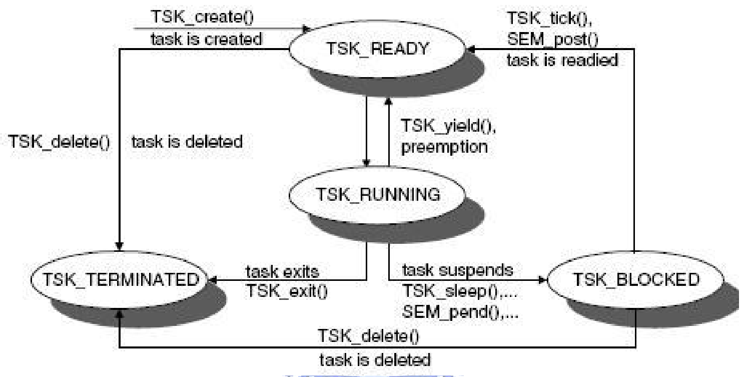


Figure 3-11 TSK module execution flow chart

3.3.1.3. HARDWARE EMULATION AND REAL-TIME DATA EXCHANGE

TI DSPs provide on-chip emulation support that enables Code Composer Studio to control program execution and monitor real-time program activity. An emulator interface, like the TI XDS510, provides the host side of the JTAG connection.

In addition, real-time data exchange (RTDX) capability is exposed through host and DSP APIs, allowing for bi-directional real-time communications between the host and DSP. It provides real-time, continuous visibility into the way DSP applications operate in the real world. As shown in real-time data exchange of DSP, the RTDX between the host and the DSP is achieved via the JTAG emulator.

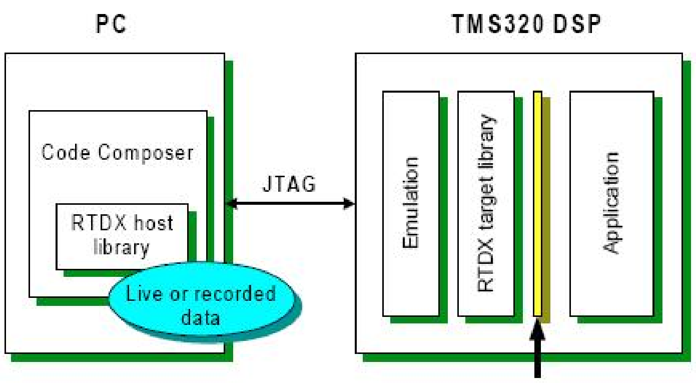


Figure 3-12 real-time data exchange of DSP emulation [14]

3.3.2. DSP PROGRAM DEVELOPMENT FLOW

Tradition development flows in DSP industry have involved validating a C model for correctness on a host PC or UNIX workstation. Programmer will need to take a great effort to port process from C code to hand coded DSP assembly langue. However this is both time consuming and error prone. The recommended code development flow involves utilizing the C6000 code generation tools to aid in optimization than force the programmer to code by hand in assembly. These advantages allow the compiler to do all the exhausting work of instruction parallelizing, pipelining, and register allocation.

The phases of recommended code development flow are illustrated in Figure 3-13.

Figure 3-13 DSP Program Development Flow

In phase one some compiler level optimization can be adopted without any knowledge of the C6000. In the second phase, intrinsic and compiler options are used to improve the code. In the last phase, linear assembly code won't be written unless the software pipeline efficiency is hardly achieved or the unbalanced resource allocation is hardly solved by the compiler.

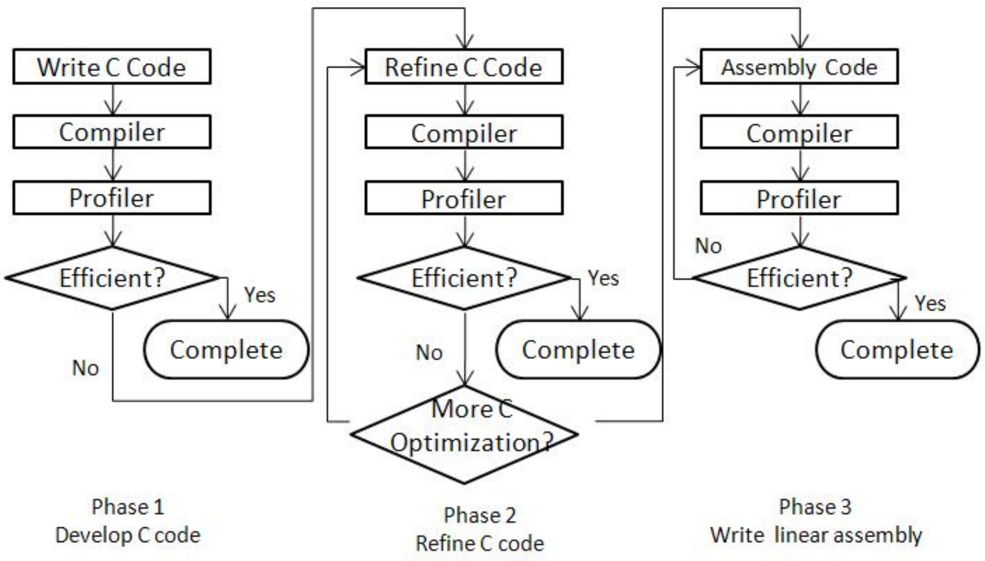


Figure 3-13 DSP Program Development Flow

3.4. OPTIMIZATION ON TI DSP PLATFORM

As shown in Figure 3-13, optimization is adopted to increase the execution performance. In this section some common used optimizations we adopt will be described.

3.4.1. COMPILER LEVEL OPTIMIZATION

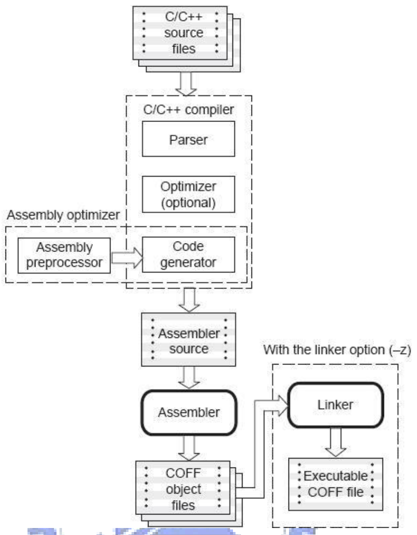


Figure 3-14 Process that translates source program into code [16]

As shown in Figure 2-1, the process that is taken to translate source program into code. Compiler in this process is able to perform various optimizations. High-level optimizations are performed in the optimizer and low-level, target specific optimizations occur in the code generator. The optimizer can reduce code size and improve executing time by using different compiler options. There are four optimization levels –o0, –o1, –o2, and –o3 denoting different type and degree of optimization, naming register level, local level, unction level, file level optimization respectively.

The –o1, register level optimization performs control-flow-graph simplification, allocates variables to registers, performs loop rotation, eliminates unused code, simplifies expressions and statements, expands calls to functions declared inline. Besides the optimization done in –o0, some more optimization will be done in the local level optimization (-o1) includes propagation of local copy/constant, unused assignments removal, and elimination of local common expressions. The function level (-o2) performs all –o1 optimizations, plus software pipelining, loop optimizations, global common sub-expressions and global unused assignments elimination, and loop unrolling. Finally, the highest level file level (-o3) perform all –o2 optimization, and remove never-called functions, simplifies functions with return values that are never used, in-lines calls to small functions, reorders function declarations, propagates arguments into function bodies, and Identifies file-level variable characteristics. In addition to these optimizations, there are some optimizations that are performed regardless of the optimization level. These optimizations cannot be turned off.

3.4.2. PROGRAM LEVEL OPTIMIZATION

Expect the compiler optimization taken by configuring the optimization level of compiler, mentioned in the last section, there are still refinements we can do to speed up the program. There are several optimization methods for the special architecture of TI C64x DSP.

First we can allocate the code sections and the code section into memories. In the two level memory architecture mentioned in 0, there are fast memories with small size such as SRAM or cache and slow memories with large size such as the external SDRAM. By using the pragma CODE_SECTION and DATA_SECTION, we can declare memory sections, and then use the linker commend file to assign these section to the proper memory level. It's intuitive to allocate the frequently used code or data into the fast and higher memory level. The frequency to access the code or the data should be analyzed for better performance. Although, the L2 cache provide such a mechanism to access an external memory efficiently, exploiting the SRAM sometimes reach better performance than using the L2 cache.

Secondly, the software pipeline and loop unrolling done in compiler level optimization mentioned in the previous section can be more efficient with the loop information given in the program. For example with the pragma MUST_ITERATE, the loop iteration information is aid to the compiler in choosing the best software optimization. The UNROLL pragma specifies to the compiler how many times a loop should be unrolled. Sometimes it will help the compiler to reduce code size and sometimes will generate redundant loops. More detailed specification is accessible in [16]. These two pragma are adopted in our project.

Finally, the C6000 compiler provides intrinsic, which are special functions that map directly to in-lined C64x instructions, to optimize C/C++ code efficiently. All these intrinsic functions are optimized codes based on the knowledge and techniques of DSP architecture. A trick of it is that intrinsic use a single instruction multi data. For example, if we can place four 8-bit data or two 16-bite data in a 32-bit register, it can execute one operation instead of four (8-bit) or two (16 bit) operation.

3.5. REFERENCE FRAMEWORK LEVEL 5

To realize the multithread system of video streaming, video processing and transmission the Reference Framework Level 5 (RF5) is used. RF5 that use DSP/BIOS and the TMS320 DSP Algorithm Standard (also known as XDAIS) is intended to enable designers to create extensive applications that use numerous algorithms, multi threads, or multi channels. The four basic elements: tasks, channels, cell, and XDAIS form the data processing of RF5 as shown in Figure 3-15.

At the top level is a DSP/BIOS task. A task is a collection of channels, a channel is a collection of cells, and a cell is a wrapper for an algorithm. The cell provides a standard interface between the algorithm and the outside world, by defining only one processing function. While the channels always perform a fixed operation of executing cells serially. The task is able to execute channels in series, and able to occasionally send control messages to one another task for thread scheduling as described in 3.3.1.2.The tasks those run get-data, execute-channels, send-data form a data processing system.

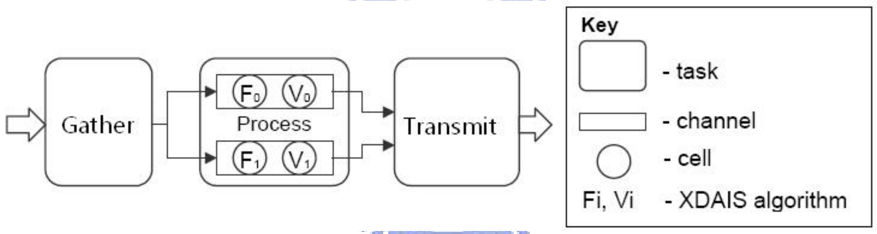


Figure 3-15 Processing elements in RF5

3.5.1. TASK LEVEL DATA COMMUNICATION

For task-level communication, which uses semaphore-based synchronization, we have streaming I/O(SIO) and synchronized communication(SCOM) messages.

SIO interfaces with device drivers and tasks. As shown in Figure 3-16 Communication Between a Task , these standard DSP/BIOS objects element facilitate the typical double buffering. That is to said, each time the task passes empty buffers to the input device driver and collects buffers full of data from the device.

SCOM message are defined by user, and passed among tasks. Tasks allocate memory buffers that other task write data to or read data from. Thus they need to communicate to the other. Each task creates its own receiving SCOM queue (or more than one if necessary), and puts SCOM messages to other tasks' receiving queues. The availability of each task and the data pass to the task is verified by checking if there is any message receiving SCOM queue. Figure 3-18 shows the task communication via

SCOM message.

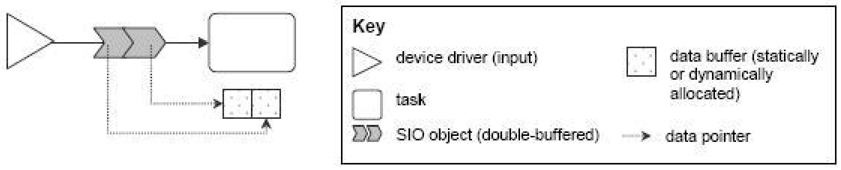

Key
device driver (input)
task
SIO object (double-buffered)
data buffer (statically or dynamically allocated)
data pointer

Figure 3-16 Communication Between a Task and a Device Driver via an SIO Object

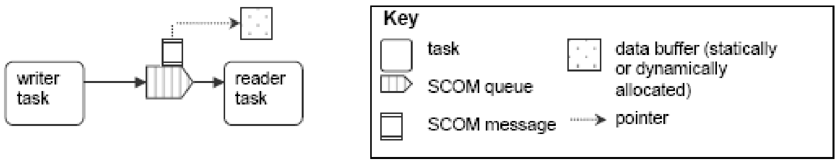


Figure 3-17 Communication Between Two Tasks via SCOM Messages

3.5.2. Cell Level Data Communication

For cell-level communication, we have inter-cell communication (ICC) objects and lists of those objects. The purpose of an ICC object is to describe the buffer from which a cell reads the data, or to which the cell writes the data. For each cell, there are one input list and one output list of those objects. As shown in Figure 3-18, two cells in effect communicate by having the same ICC object in their lists: the cell that writes to a buffer described by an ICC object has the object in its output list, and the cell that reads the buffer has the object in its input list.

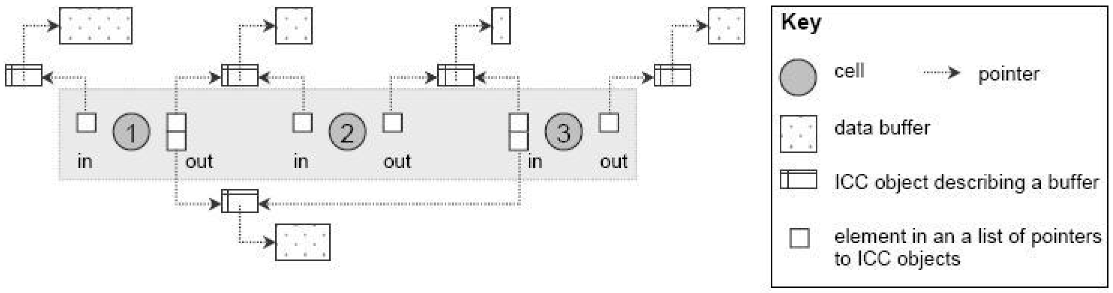

1
2
3
in
out
in
out
in
out
Key
cell
pointer
data buffer
ICC object describing a buffer
element in an a list of pointers to ICC objects

Figure 3-18 Communication between Cells via ICC object

3.6. NETWORK DEVELOPER'S KIT

The Network Developer's Kit provided by TI is designed as a platform for development and demonstration of network enabled application on the DSP. To build the a full TCP/IP functional environment only small memory footprint of around 200K to 250K of program memory and 95K of data memory are required [17]. That make NDK a good choice to implement networking transmit system.

The NDK software package is designed to be a transparent add-on to DSP/BIOS and CCS development tools, as shown in Figure 3-19.

In Figure 3-19, the stack package is organized in terms of function call control flow, including five main libraries: STACK, NETTOOL, OS, HAL, and NETCTRL libraries.

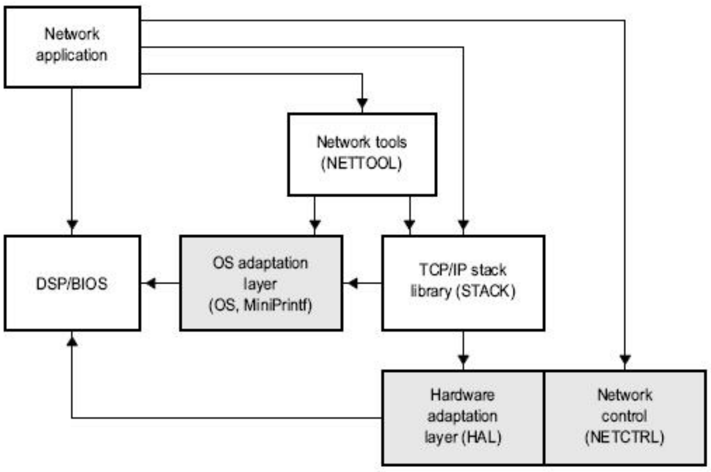

Network application
Network tools (NETTOOL)
DSP/BIOS
OS adaptation layer (OS, MiniPrintf)
TCP/IP stack library (STACK)
Hardware adaptation layer (HAL)
Network control (NETCTRL)

Figure 3-19 Stack Control Flow

Chapter 4. IMPLEMENTATION AND SPEED IMPROVEMENT

4.1. ARCHITECTURE OF H.264/AVC VIDEO COMMUNICATION SYSTEM

In this thesis, the real H.264/AVC based video communication system is implemented. Unlike the other implementations of H.264/AVC codec that only contain the encoder and decoder with file I/O, as shown in Figure 4-1, we construct one more realistic system that describes a real encoding path and decoding path. The encoding path includes video capturing, H.264/AVC encoding, and network transmit, while the decoding path includes network receiving, H.264/AVC decoding and video display on PC, as shown in Figure 4-2.At one end a MEX board is installed on the computer to get the analog video signal from the video device, then do the H.264/AVC video compression to the video content, and finally transmit the coded data to the ethernet. These transmit data was received by another MEX board on different PC. At this end the coded data will be decode back into the video data, and then be displayed on the personal computer.

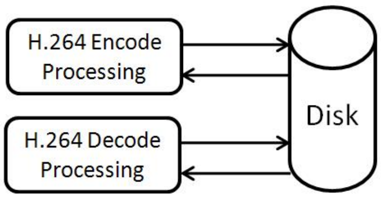


Figure 4-1 Usual implantation of H.264/AVC System

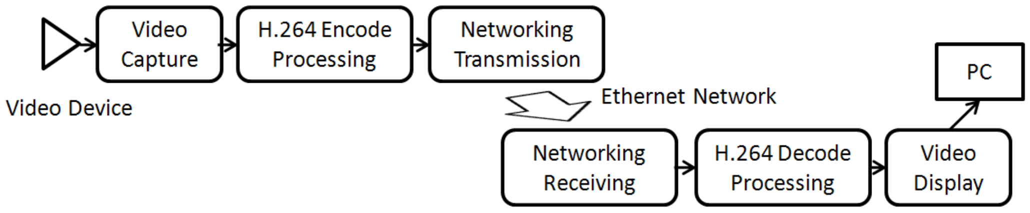


Figure 4-2 H.264 based Visual Communication System

The software in a typical embedded microprocessor system, such as the DM642 we used for development, is composed of two general components, the application software and the system software. In our H.264/AVC based communication system, the video encoding and decoding algorithm and the ethernet communication are the application software, while the video capturing and display are system software. In order to operate correctly in real time, both of application software and the system software should be scheduled well. To realize the mechanism, we implement the real-time system by multi-task with multiple threads.

Refer to the section 3.5, we adopt the RF-5 framework to build our multi thread and multi task system. The entire system including the encoder end and decoder end is decomposed into RF-5 objects as shown in Figure 4-3. There are eight major tasks, three functional tasks including Capture task, Encoding Processing task, Tx networking task for the encoding process, three functional tasks including Display task, Encoding Processing Task, Rx networking task for the decoding process, and one control task for both encoding task and decoding task.

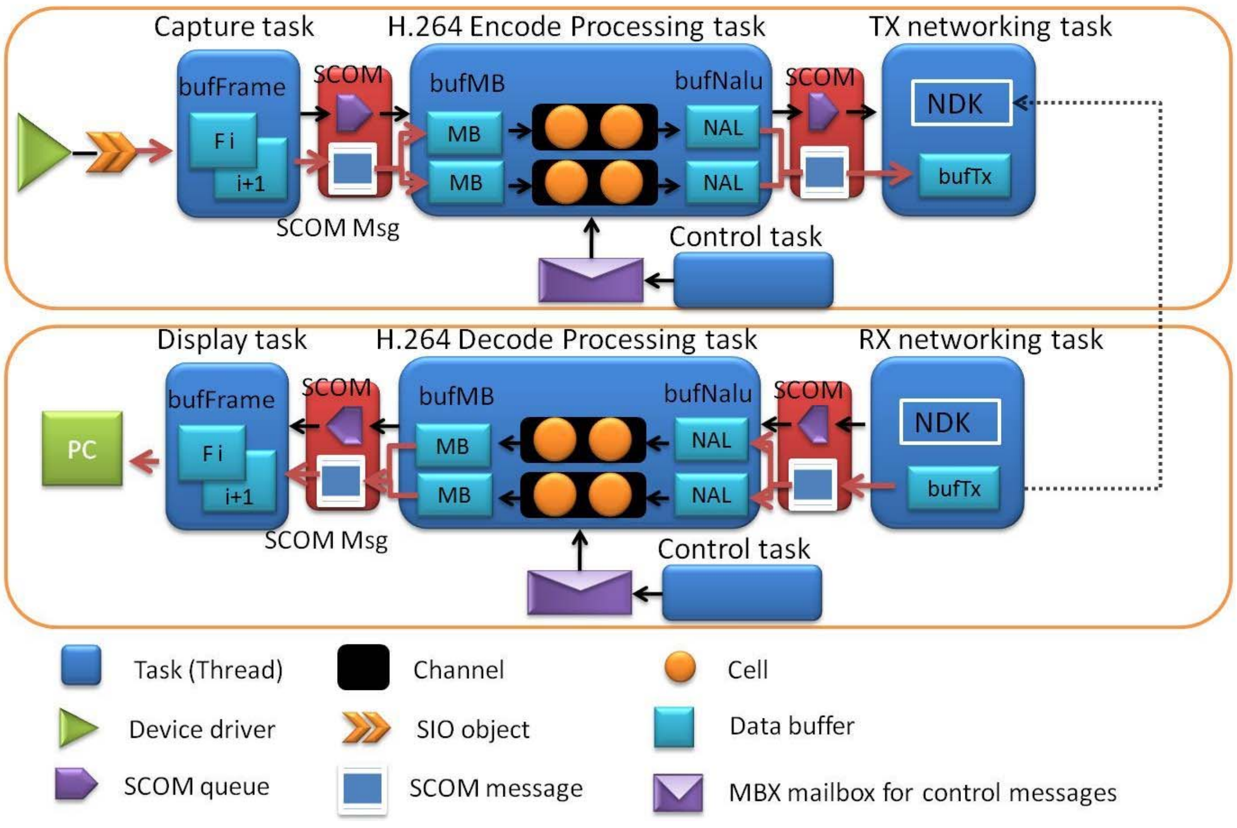


Figure 4-3 System block diagram of Reference Framework level-5

- ✓ Capture task: After initializing the video chip SAA7113 via I2C module in DM642.Vido frame with format of 4:2:2, is captured from the video chip. The EDMA channel synchronized to the video port event (VP0EVTA) is open to get to video data from video port. The frame data is resampled to 4:0:0 format, for to encode task.
- ✓ Display task: In this task, decoded frame with 4:0:0 is resampled to 4:2:2 and sent to the external FIFO which is also accessible to the HOST PC. Then a Win32 API windows function will receive an interrupt from DPS to and then display the decode frame.
- ✓ Tx/Rx networking task: These two was use the NDK module to utilize to transmission and reception of coded data. The encoded data, NAL unit of H.264/AVC is transmitted by the Tx networking task, and are received by the Rx networking task.
- ✓ Encoding Processing task: The H.264/AVC encoder get frame from the Capture task, and send the encode NAL units to the Tx networking task.
- ✓ Decoding Processing task: H.264/AVC decoder get NAL units from the Rx networking task, and send the decode frame to the Display task.
- ✓ Control task: use Mailbox (DSP/BIOS object) to configure the above tasks.

Each task waits until it receives the message from the SCOM queue, and after execution it yield the execution to other tasks by put the SCOM message to the message queue.

4.2. SINGLE-DSP PARALLELIZATION

To facilitate application H.264/AVC based communication described above, we use the multi DSP board, MEX, to implement the system. Firstly the system is carried out and optimized on single DSP. And then further extension to other DSPs is taking into consideration. In this section, single DPS optimization and parallelization is described.

4.2.1. SINGLE-DSP OPTIMIZATION

By adopting the RF-5 framework and doing some modification to the H.264 encoder and decoder source code, the H.264/AVC encoding system and decoding system are implemented on two MEX boards respectively. Thought the optimization of encoder and decoder is taken individually, similar optimization rules as mentioned in section 3.4 are taken to accelerate the encoding and decoding system:

✓ Compiler optimization level:

The highest compiler optimization level (-o3) is taken. Various loop optimizations are performed, such as software pipelining, unrolling, and SIMD. Various file level characteristics are also used to improve performance.

✓ Software pipeline

Adding the "#pragma MUST_ITERATE" pragma in the front of the loop to inform the complier to unroll it and prevent to produce redundant loop when optimizing. However, in a nested loop structure, only the innermost loop will be unrolled, while the inner loop will be ignored. Manual loop unrolling is done to make the software pipelining more efficient.

✓ Allocation of code and data memory sections

There are some frequently accessed data such the entropy coding decoding table, and some frequently executed functions, such as interpolation for fractional motion vector. Since the full 256kB L2 cache is not allowed for a NDK application [17]. We can allocate some memory into the SDRAM such as those frequently used data and code by using. The "#pragma DATA_SECTION" and "#pragma CODE_SECTION" are used to allocate the data memory and code memories respectively.

✓ The table lists intrinsic function we used to replace the original C-operation to accelerate the execution.

Table 4-1Intrinsic functions we used

C Compiler Intrinsic	Assembly Instruction	Description
int _abs2(int src);	**ABS2**	Calculates the absolute value for each 16-bit value

int **_max2** (int src1, int src2);
int **_min2**(int src1, int src2);
uint **_maxu4**(uint src1, uint src2);
uint **_minu4**(uint src1, uint src2);
	MAX2
	MIN2
	MAX4
	MINU4
		Places the larger/smaller of Cvalue. Values can be 16-bit signed or 8-bit unsigned.

_memd8(p)		Unaligned access of double beginning at address p
_memd4(p)		Unaligned access of unsigned int beginning at address
_memd2(p)		Unaligned access of unsigned short beginning at address p

✓ Moreover, the L2 cache is enable with 0, 32, 64, 128 Kbytes.

4.2.2. DOUBLE BUFFERING

The double buffering also known as the ping-pong buffering is a mechanism that allows the CPU activity to be independent of the EDMA activity. In ping-pong buffering, there are multiple (usually two) sets of data buffers for the incoming and outgoing data streams. While the EDMA is transferring data into or out of the ping buffer, the CPU is manipulating data in the pong buffer. When the CPU and EDMA complete their activities, they switch the buffers. The EDMA then writes over the old input data and transfers the new output data. An example of the ping–pong buffering scheme is shown in Figure 4-4. By using double buffering, the data in ping and pong buffers are processed by CPU independently.

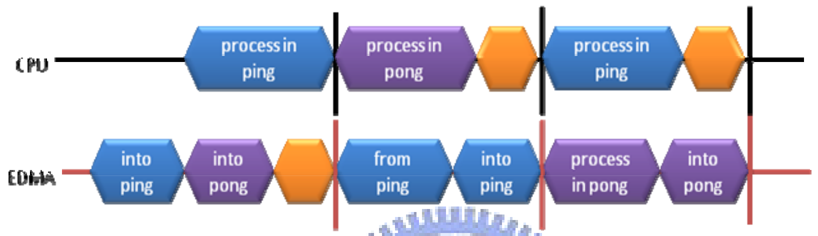


Figure 4-4 Ping-Pong buffering diagram

4.2.3. SINGLE-DSP PARALLELIZATION

As mentioned in the previous section, with the help of EDMA, CPU can serve the ping and pong buffer independently. We produce the pseudo threads of the ping process and pong process respectively. Though the pseudo threads dose not executed at the same time, we can make increase the efficiency by reducing the memory access time rather than the execution time. Here, we exploited the MB-level parallelism in spatial domain. To satisfy the data dependency constraint, describe in section 2.9. The processing of macroblock are in the order as Figure 4-5 shows. Each ping-pong pair is executed at the Time N successively. While the macroblock in ping buffer at Time N-1 is processed, the macroblock in pong buffer is processed right away at Time N-2. The execution condition at time 3, time 4, and time7 is shown below. We can notice that macroblock (2,0) and macroblock (0,1) is a ping-pong pair, so does the macroblock (0,2) and macroblock (4,1). The red pointer indicates the data dependency situation of each macroblock.

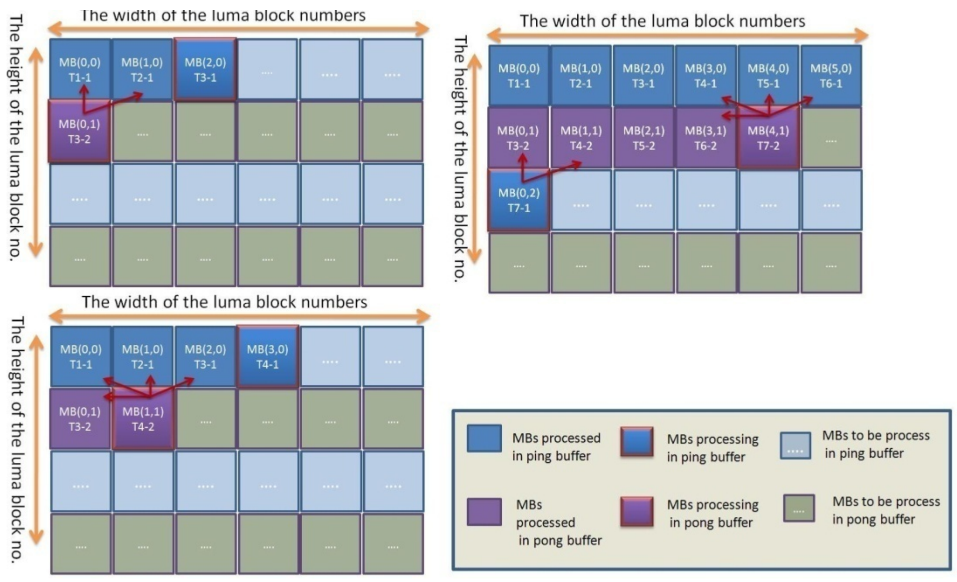

The width of the luma block numbers
The height of the luma block no.
MB(0,0) T1-1
MB(1,0) T2-1
MB(2,0) T3-1
MB(3,0) T4-1
MB(4,0) T5-1
MB(5,0) T6-1
MB(0,1) T3-2
MB(1,1) T4-2
MB(2,1) T5-2
MB(3,1) T6-2
MB(4,1) T7-2
MB(0,2) T7-1
MBs processed in ping buffer
MBs processing in ping buffer
MBs to be process in ping buffer
MBs processed in pong buffer
MBs processing in pong buffer
MBs to be process in pong buffer

Figure 4-5 Single DSP macroblock parallelization

4.3. MULTI-DSP PARALLELIZATION

4.3.1. SYSTEM PROFILE

We make a profiling of the whole system, including Capture TSK, Encode TSK, Tx TSK, Rx TSK, Decode TSK, and Display TSK. The profiling result is listed in Table 4-2. For further speed up, we take the other DSP chips on the MEX board into account. As shown in Table 4-2, the bottleneck of this system is the encoding task. It will be further parallelized, as illustrated in the next section.

Table 4-2 System profile of the major tasks

Encoding	Capture TSK	Encode TSK	Tx TSK
ms	290250	417623219	120706616
%	0.07	77.54	22.4

Decoding	Display TSK	Decode TSK	Rx TSK
ms	107973264	68904482	1382446
%	60.57	38.67	0.77

4.3.2. MULTI-DSP PARALLELIZATION

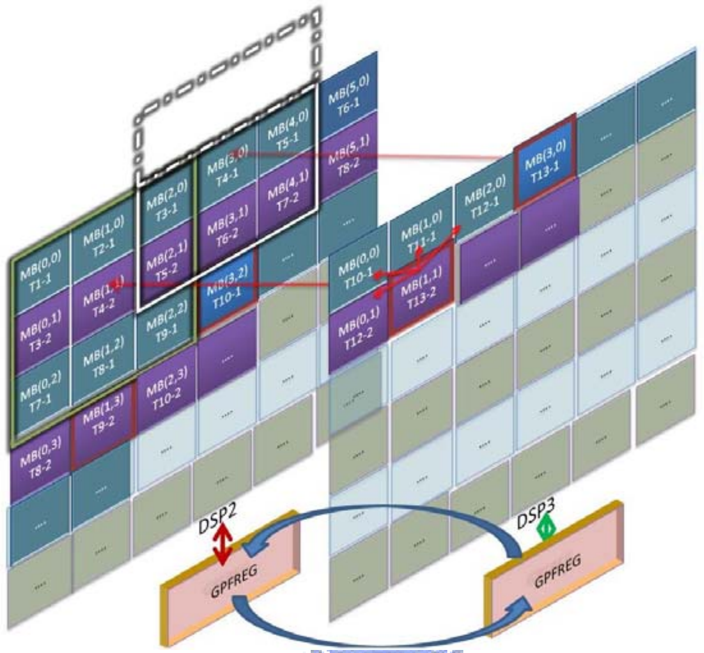


Figure 4-6 Multi DSP macroblock parallelization

To achieve the parallelization by using multi DSPs, a temporal domain parallelization is considered. As shown in Figure 4-6, while in spatial domain parallelization we ease the data dependency induced by intra prediction and deblocking filtering by processing macroblocks in an order as mentioned in the last section. Likely we should ease the dependency in a special order as Multi DSP macroblock parallel shows. In the multi DSP parallelization, frames are distributed into DSPs. In the section 2.9, that the cross frame dependency is induced by the motion estimation; only when search window of the reference frame is reconstructed (either in encoder or decoder), the current macroblock is available to be processed. As shown in Multi DSP macroblock parallel, at the Time10, four macroblocks are processed including macroblock (3, 2), macroblock (1, 3) in frame i, macroblock (3, 0), and macroblock(1,1) in frame i+1.

4.4. EXPERIMENTS & RESULTS

4.4.1. RESULT OF SINGLE DSP OPTIMIZATION

After adopting the optimization methods we introduced before, we use the DM642 Device Cycle Accurate Simulator to profile the functions in encoding and decoding processes. Neither the video input, output task nor the network transmit task is consider in this comparison, because the optimization won't help a lot for those I/O tasks. Though the optimization is done to the project globally, the improvement of speed of each functions in coding process are not the same; it depends on the structure of the function. In the following part, we will illustrate some important functions, compare the execution time of them and calculate the speed up ratio.

As shown in Table 4-3, each function is speeded up by the optimization and some manual modification. Since the inter prediction intra prediction, and DCT/IDCT contain many loop structures in the functions, they are well optimized for the software pipeline e mechanism of the DSP chip. We can notice that most time consuming part is still the inter predictions. If there is any need to write the assembly code as shown in Figure 3-13, it might be the proper choice to write it into the assembly code.

Table 4-3 Average execution cycle of a frame of x264 encoder

	Non optimized		Optimized		
	Cycle count	Percentage	Cycle count	Percentage	Ratio
Inter	385955140	66.80	31345836	58.71	12.3
Intra	76744048	13.28	5435452	10.26	14.1
DCT/IDCT	36449650	6.31	2470440	4.66	14.7
Quantization.	27206566	4.71	2718834	5.13	10.0
Deblocking filter	10887230	1.88	2424248	4.58	4.49
Entropy coding	9221182	1.60	2430716	4.59	3.79
Total	577769679	100	52979974	100	10.91

In Table 4-4, the decoder task is speed up by ratio 5.3. Though the inter prediction, intra prediction, and DCT function are well optimized by the speeded up ratio 1x. The performance of the decoder task is contra trained by the deblocking filter. For the decoder the deblocking filter is the of choice function to written into assembly for further improvement.

Table 4-4 Average execution cycle of a frame of JM10.3 decoder

	Non optimized		Optimized		
	Cycle count	**Percentage**	**Cycle count**	**Percentage**	**Ratio**
Inter	55418473.24	26.96	4163457	10.82	13.3
Intra	3740215.647	1.80	244461	0.64	15.3
IDCT	7638903.176	3.72	1135516	2.95	6.7
Quantization	42497.23529	0.02	14976	0.039	2.8
Deblocking filter	115774629	56.32	17790497	46.25	6.5
Entropy coding	22219077.35	10.81	7875195	20.47	2.8
Total	205535479	100	38467286	100	5.3

4.4.2. Result of Single DSP Parallelization

In the single DSP parallelization, we use ping and pong buffer to allocate the successively executed macroblocks. Most of the program is not changed, but the macroblock processing order. Thus the improvement is due to the overlap of EDMA transmit and CPU processing. In the un-parallelized version, the memory data needed in ping procedure is fetch by CPU before processing; while in parallelized version, the needed data is fetch by EDMA at the last pong execution. We use emulator to observe the effect of real SDRAM accession (external memory accession), and how much is reduced by the proposed method.

Table 4-5 Single DSP parallelization of x264 encoder

	Non- parallelized	Parallelized	Ratio
ms per frame	76.44	63.73	1.199

Table 4-6 Single DSP parallelization of JM10.3 decoder

	Non- parallelized	Parallelized	Ratio
ms per frame	491.26	475.53	1.033

The limitation of the speed up of this method is the ratio of EDMA transmit time and CPU execution time. When the ratio is about 1, or the time EDMA used to transmit is almost equal to the CPU executing time, the speed up ratio might up to two. To achieve this goal, further modification optimization to the code should be adopted to reduce the CPU execution time.

4.4.3. Result of Multi DSP Optimization

In section 4.3.1 and 4.3.2, we proposed a multi DSP parallel mechanism that only applied to the encoder end for the system. To use other DSPs on the MEX board, we retain the network transmit task and the video capture task, but parallelize the encoding task into other DSPs. The parallelization result is shown below.

Table 4-7 Multi DSP parallelization result

	One DSP (original)	Two DSP	Three DSP	Four DSP
ms per frame	475.53	397.55	323.81	290.75
Speed up ratio	1	1.196	1.4685	1.6355

The multi DSP parallelization is restricted to the cross DSP transmit of the MEX board. For more efficient multi DSP parallelization, we should find some board else with better cross DSP communication.

Chapter 5. CONCLUSION AND FUTURE WORK

In this thesis, we construct the H.264/AVC video communication system on MEX. Rather than the disk I/O system, we implement a more realistic system that consists of not only the H.264/AVC codec but also video capturing, displaying and ethernet transmission. Then we propose optimization and parallelization method for the multi DSP board. We conclude our accomplishments as below.

- ✓ We implement of real-time H.264 encoder /decoder system.
- ✓ We establish a multi thread system.
- ✓ We do the optimization of H.264 encoder/decoder for single DSP by using double buffers.
- ✓ We propose a parallelization prototype of H.264 encoder/decoder for multi DSP.

To achieve higher performance, it's important to find an efficient path for cross DSP communication. Since the benefit bought out from the macroblock parallelizing is stock due to the slow cross DSP communication.

REFERENCES

[1] JVT "Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T rec. H.264– ISO/IEC 14496-10 AVC)," March 2003, JVTG050 available on http://ip.hhi.de/imagecom_G1/assets/pdfs/JVT-G050.pdf .

[2] R. Schäfer, T. Wiegand and H. Schwarz, "The Emerging H.264/AVC Standard", EBU Technical Review, Jan. 2003.

[3] I.E.G. Richardson, H.264 and MPEG-4 Video Compression, John Wiley & Sons, 2003.

[4] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra, "Overview of the H.264/AVC Video Coding Standard," IEEE Trans. on Circuits Syst. Video Technol., Vol. 13, No. 7, pp.560 – 576, July 2003.

[5] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockammer and T. Wedi, "Video Coding with H.264/AVC: Tools, Performance, and Complexity", IEEE Circuits and Systems, Vol. 4, No. 1, 2004.

[6] www.vitecmm.com, Preliminary of MEX

[7] www.blackhawk-dsp.com/Usb560bp.aspx, Preliminary of USB 560BP

[8] Texas Instruments, "TMS320C6414T, TMS320C6414T, TMS320C6416T fixed point Digital Signal Processor", Literature Number SPRR226, November 2003.

[9] Texas Instruments, "TMS320DM642 Video/Imaging Fixed-Point Digital Signal Processor: Data manual", Literature Number SPRS200G, July 2002 – Revised August 2004.

[10] Texas Instruments, "TMS320C6000 CPU and Instruction Set Reference Guide", Literature Number SPRU189F, January 2000.

[11] Texas Instruments, "TMS320C64x DSP Two-Level Internal Memory Reference Guide", Literature Number SPRU610C, August 2004.

[12] Texas Instruments, "Video port/VCXO Interpolated Control (VIC) Reference Guide", Literature Number SPRU629F, January 2007.

[13] Texas Instruments, "TMS320C6000 DSP Ethernet Media Access Controller (EMAC)/Management Data Input/ Output (MDIO) Module Reference Guide", Literature Number SPRU628A, March 2004.

[14] Texas Instruments, "TMS320C6000 Code Composer Studio Tutorial" Literature Number SPRU301C, February 2000

[15] Texas Instruments, "TMS320C6000 DSP/BIOS User's Guide", Literature

Number SPRU303B, March 2000

[16] Texas Instruments, "TMS320C6000 Optimizing Compiler User's Guide" Literature Number SPRU187G, March 2000

[17] Texas Instruments, "TMS320C6000 TCP/IP Network Developer's Kit (NDK) Technical Data Quick Reference GuideTMS320C6000 TCP/IP Network Developer's Kit (NDK) Technical Data Quick Reference Guide", Literature Number SPRU568, October 2001