國立交通大學

電機與控制工程學系

碩 士 論 文

高效能雙核心 H.264 編碼器之實現

Implementation of an Efficient Dual-core H.264 Encoder

研究生：林裕傑

指導教授：吳炳飛　教授

中 華 民 國 九 十 五 年 七 月

高效能雙核心 H.264 編碼器之實現

Implementation of an Efficient Dual-core H.264 Encoder

研究生：林裕傑　　Student：Yuh-Jay Lin

指導教授：吳炳飛 教授　　Advisor：Prof. Bing-Fei Wu

國 立 交 通 大 學

電 機 與 控 制 工 程 學 系

碩 士 論 文

A Thesis

Submitted to Department of Electrical and Control Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Electrical and Control Engineering

July 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年七月

高效能雙核心 H.264 編碼器之實現

學生： 林裕傑　　指導教授：吳炳飛　教授

國立交通大學電機與控制工程學系(研究所)碩士班

摘　　要

本論文提出一個具有雙核心特性的 H.264 編碼器。此編碼器在演算法層次上，對於程式執行的流程以及資料的存取，都做了徹底的改善和最佳化。我們在壓縮流程上，減少程式的分支，將壓縮過程中的判斷式減少，使得編譯器能發揮最好的效能去編譯程式。在記憶體使用方面，設計了有效的處理流程，讓 ARM 和 DSP 之間能分工平行處理目前的資料。使用雙核心架構的編碼器，可以讓 DSP 核心全力處理壓縮的運算部份，ARM 核心負責資料流的輸出輸入以及週邊裝置的控制。此種架構除了提昇效能之外，更能最佳化處理器的使用方式。本論文提出的 H.264 編碼器，經由 ARM 讀取儲存裝置或者是週邊裝置裡的原始影像，將資料搬移到 DSP 去進行 H.264 的編碼。處理之後的結果，會由 ARM 端負責包裝成可以播放的影片檔案輸出。在實驗平台 OMAP5912 上執行的表現，對於品質和速度之間取得了平衡。利用平台的特性，此解碼器可應用在網路視訊傳輸或者是本機端錄影機。所提出之解碼器架構和最佳化方法，更能移植到不同的平台上，增加其應用方式。

Implementation of An Efficient Dual-core H.264 Encoder

Student : Yuh-Jay Lin Advisor : Prof. Bing-Fei Wu

Department of Electrical and Control Engineering

National Chiao Tung University

ABSTRACT

In this thesis, a dual-core based and highly optimized H.264 encoder is presented. A major benefit of using a DSP-ARM dual-core processor is that it is possible to integrate system control functionalities with the H.264 encoding on a single chip. However, more benefits of significance can be obtained if DSP and ARM cores are programmed to execute signal processing jobs in parallel. The developed H.264 encoding system consists of a video compression processing part and a system control part. In the video compression processing part, the encoded data is encoded via H.264 encoding algorithm and converted to a playable file such as MKV files. The system control part is in charge of managing the encoded data files and transferring them from the storage to video compression processing part. The proposed encoder achieves balance between quality and speed. With OMAP5912, the implementation is well suited for videophones, and network streaming applications. The DSP core can perform encoding while the other can process network message as well as storing the encoded H.264 bitstream in the local host. The optimization techniques presented can be effectively used for other programmable processors with similar architectures and instruction sets.

Content

List of Figures

List of Tables

1 Introduction

1.1 Overview of this thesis

1.1.1 H.264 Standard

H.264[1] is the latest video coding standard providing for improved compression over existing standards such as H.261[2], H.263[3][4], and MPEG4[5]. With comparable bitrates the increase in visual quality is significant which also means that you can maintain acceptable video quality with up to a 50% reduction in file size.

Already ratified as part of the MPEG-4 standard and the ITU-T's latest video-conferencing standard, H.264 is now mandatory for the HD-DVD[6] and Blu-ray[7] specifications, that are the two formats for high-definition DVDs, and ratified in the latest versions of the Digital Video Broadcasters (DVB)[8] and 3rd Generation Partnership Project (3GPP)[9] standards. Numerous broadcast, cable, videoconferencing and consumer electronics companies consider H.264 the video codec of choice for their new products and services. This adoption by a wide variety of open standards means that any company in the world can create devices — mobile phones, set-top boxes, and DVD players.

1.1.2 Implementation of H.264 encoder on the dual-core platform

In this thesis, a dual-core based and highly optimized H.264 encoder is presented. The features show as follows.

- The dual-core platform

  OMAP5912[10] offers an attractive solution to both DSP[11] and ARM[12] developers, providing the low power real-time signal processing capabilities of a DSP coupled with the command and control functionality of a microprocessor. This low power dual core architecture forms a platform for various 3G wireless multimedia applications.

- Embedded Linux environment

  Embedded Linux [13] provides most everything else, including the system's only scheduler. OMAP Linux drivers such as the CF card, LAN network, USB drivers are available to interface with peripherals, and other Linux drivers can be easily ported. Moreover, the full range of Linux file systems is supported.

- Inter-processor communication with DSP gateway[14]

  DSP Gateway consists of Linux driver and the DSP Firmware. They are responsible for hardware settings, interrupt handlings, and communications in between. From ARM side, DSP can be reached through DSP device file under /dev directory on Linux file system. On DSP side, tasks can easily be synchronized with ARM side by using the API provided by DSP Gateway.

1.1.3 Optimizations of the proposed H.264 encoder

Optimization techniques used in this thesis are grouped into two categories, platform-independent and platform-based.

- Platform-independent optimizations

  To reduce the complexity of H.264 encoder, algorithms are changed in high level language. Apart from implementing optimal algorithms, optimization techniques like loop unrolling, loop distribution and loop interchange are used. Algorithmic optimizations presented below are platform independent and can be used on any

platform.

- Platform-based optimizations

  Due to TMS320C55X DSP architecture, some features of the C language are relevant to compilation on the C55x DSP, performance-enhancing options for the compiler, and C55x-specific code transformations that improve C code performance. The DSP H.264 Encoder is modified by using these features.

1.2 Contribution and Organization

In this thesis, a dual-core highly optimized H.264 encoder is presented. The following chapters describe this encoder in detail.

- Chapter 2 introduces H.264 standard and the algorithm of H.264 in component level.
- Chapter 3 describes the hardware platform, OMAP5912 and the software environment that includes embedded Linux, the DSP programs, and the inter-processor communication library.
- Chapter 4 illustrates implementation of the H.264 encoder and the optimization techniques used in this thesis.
- Chapter 5 shows the experimental results and the comparison. The proposed encoder can achieve an efficient performance.
- Chapter 6 concludes this work and suggests possible applications for this encoder.

2 Overview of H.264 Standard

2.1 H.264 CODEC

H.264 is a new video coding scheme that is becoming the worldwide digital video standard for consumer electronics and personal computers. It has been adopted by the Motion Picture Experts Group (MPEG) to be a key video compression scheme in the MPEG-4 format for digital media exchange.

Similar with former video coding standards, H.264 does not define a CODEC but rather defines the syntax of an encoded video bitstream in addition to the way of decoding this bitstream. In practice, a yielding CODEC is likely to include the functional elements. Excluding the deblocking filter, many basic functional elements, such as prediction, transform, quantization, entropy encoding, are present in previous standards but important changed a lot in H.264 in the details of each functional block.

There are many real applications of H.264. In particular, H.264 has already been selected as a key compression scheme (codec) for the next generation of optical disc formats, HD-DVD and Blu-ray disc (sometimes referred to as BD or BD-ROM). In addition, future delivery of Digital TV signals (both in SD and HD) will use H.264. The 3rd Generation Partnership Project (3GPP) has approved the inclusion of H.264/AVC as an optional feature in release 6 of its mobile multimedia telephony services specifications. Moreover, H.264 will probably be used by various video-on-demand services on the Internet to provide films and television shows directly to computers.

2.2 Profiles and Levels in H.264

H.264 has been developed to address a large range of applications, bit rates, resolutions, qualities, and services; in other words, H.264 intends to be as generically applicable as possible. However, different applications typically have different requirements both in terms of functionalities, e.g., error resilience, compression efficiency and delay, as well as complexity (in this case, mainly decoding complexity since encoding is not standardized). In order to maximize the interoperability while limiting the complexity, targeting the largest deployment of the standard, the H.264 specification defines profiles and levels.

A profile [15] is defined as a subset of the entire bit stream syntax or in other terms as a subset of the coding tools. In order to achieve a subset of the complete syntax, flags, parameters, and other syntax elements are included in the bit stream that signal the presence or absence of syntactic elements that occur later in the bit stream. All decoders compliant to a certain profile must support all the tools in the corresponding profile. However, within the boundaries imposed by the syntax of a given profile, there is still a large variation in terms of the capabilities required of the decoders depending on the values taken by some syntax elements in the bit stream such as the size of the decoded pictures. For many applications, it is currently neither practical nor economic to implement a decoder able to deal with all hypothetical uses of the syntax within a particular profile. To address this problem, a second profiling dimension was created for each profile: the levels.

The level is a specified set of constraints imposed on values of the syntax elements in the bit stream. These constraints may be simple limits on values or alternatively they may take the form of constraints on arithmetic combinations of values. In H.264, the same level definitions are used for all profiles defined. However, if a certain terminal supports more than one profile, there is no obligation that the same level is supported for the various profiles. A profile and level combination specifies the so-called conformance points; this means points of

interoperability for applications with similar functional requirements. Summing up, profiles and levels together specify restrictions on the bit streams and thus minimum bounds on the decoding capabilities, making possible to implement decoders with different limited complexity, targeting different application domains. Encoders are not required to make use of any specific set of tools; they only have to produce bit streams which are compliant to the relevant profile and the level combination.

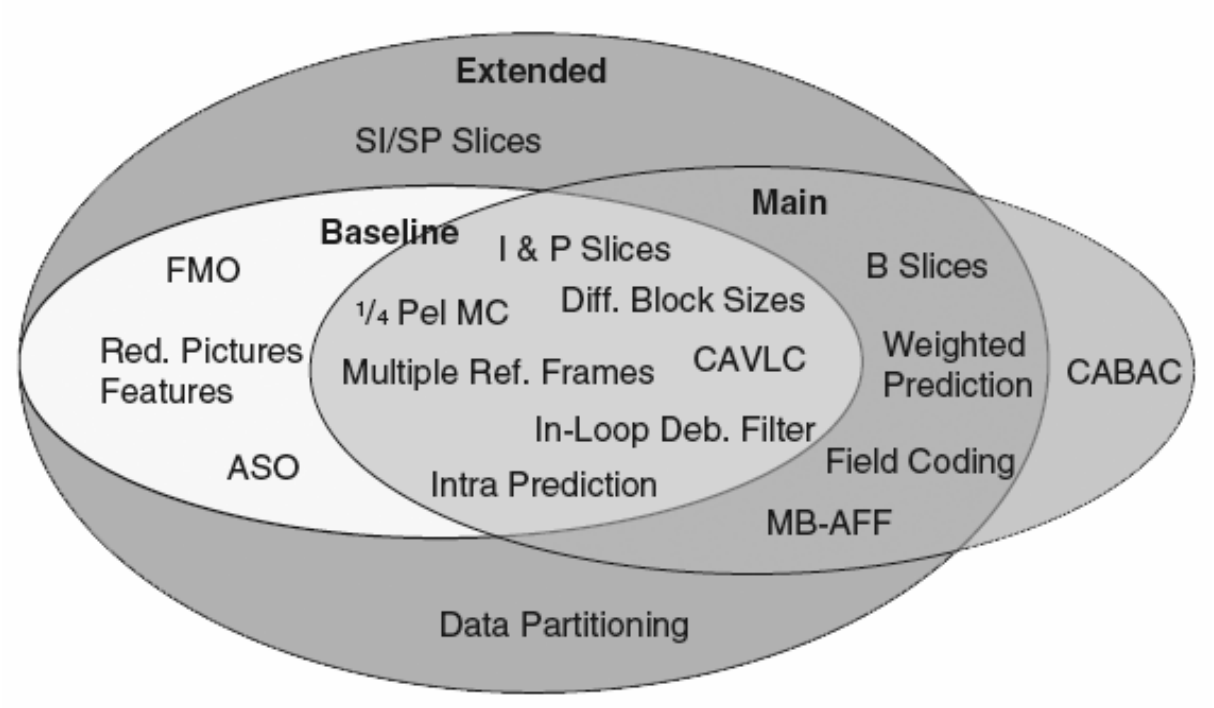


Fig. 1 H.264 Profiles

To address the large range of applications considered by H.264, three profiles have been defined (see Fig. 1):

- **Baseline Profile**

  Typically considered the simplest profile, includes all the H.264 tools with the exception of the following tools: B-slices, weighted prediction, field (interlaced) coding, picture/macroblock adaptive switching between frame and field coding (MB-AFF), Context-based Adaptive Variable Binary Arithmetic Coding (CABAC)[16], SP/SI slices and slice data partitioning. This profile typically targets applications with low complexity and low delay requirements.

- **Main Profile**

  This profile supports together with the Baseline profile a core set of tools (see Fig. 1); however, regarding Baseline, Main does exclude FMO, ASO and redundant pictures features while including B-slices, weighted prediction, field (interlaced) coding, MB-AFF, and CABAC. This profile typically allows the best quality at the cost of higher complexity (essentially due to the B-slices and CABAC) and delay.

- **Extended Profile**

  This profile is a superset of the Baseline profile supporting all tools in the specification with the exception of CABAC. The SP/SI slices and slice data partitioning tools are only included in this profile. From Fig. 1, it is clear that there is a set of tools supported by all profiles but the hierarchical capabilities for this set of profiles are reduced to Extended being a superset of Baseline. This means, for example, that only certain Baseline compliant streams may be decoded by a decoder compliant with the Main profile. Although it is difficult to establish a strong relation between profiles and applications (and clearly nothing is normative in this regard), it is possible to say that conversational services will typically use the Baseline profile, entertainment services the Main profile, and streaming services the Baseline or Extended profiles for wireless or wired environments, respectively. However, a different approach may be adopted and, for sure, may change in time as additional complexity will become more acceptable.

In H.264, 15 levels are specified for each profile. Each level specifies upper bounds for the bit stream or lower bounds for the decoder capabilities, e.g., in terms of picture size (from QCIF to above 4096x2304), decoder processing rate (from 1485 to 983040 macroblocks per second), size of the memory for multi-picture buffers, video bit rate (from 64 kbits/s to 240 Mbits/s), and motion vector range (from [−64, +63.75] to [−512, +511.75]). For more detailed information on the H.264/AVC profiles and levels, refer to Annex A of [1].

2.3 H.264 Encoder Data path

The H.264 encoder scheme shown in Fig. 2 includes two dataflow paths, a forward path and a reverse path. An input frame is processed in units of a macroblock. Each macroblock is encoded in intra or inter mode and, for each block in the macroblock, a prediction is formed based on reconstructed picture samples. In Intra mode, the intra prediction is formed from samples in the current slice that have previously encoded, decoded and reconstructed. In Inter mode, the inter prediction is formed by motion-compensated prediction from one or more reference pictures selected from the set of reference pictures. The prediction reference for each macroblock partition may be chosen from a selection of past or future pictures that have already been encoded, reconstructed and filtered.

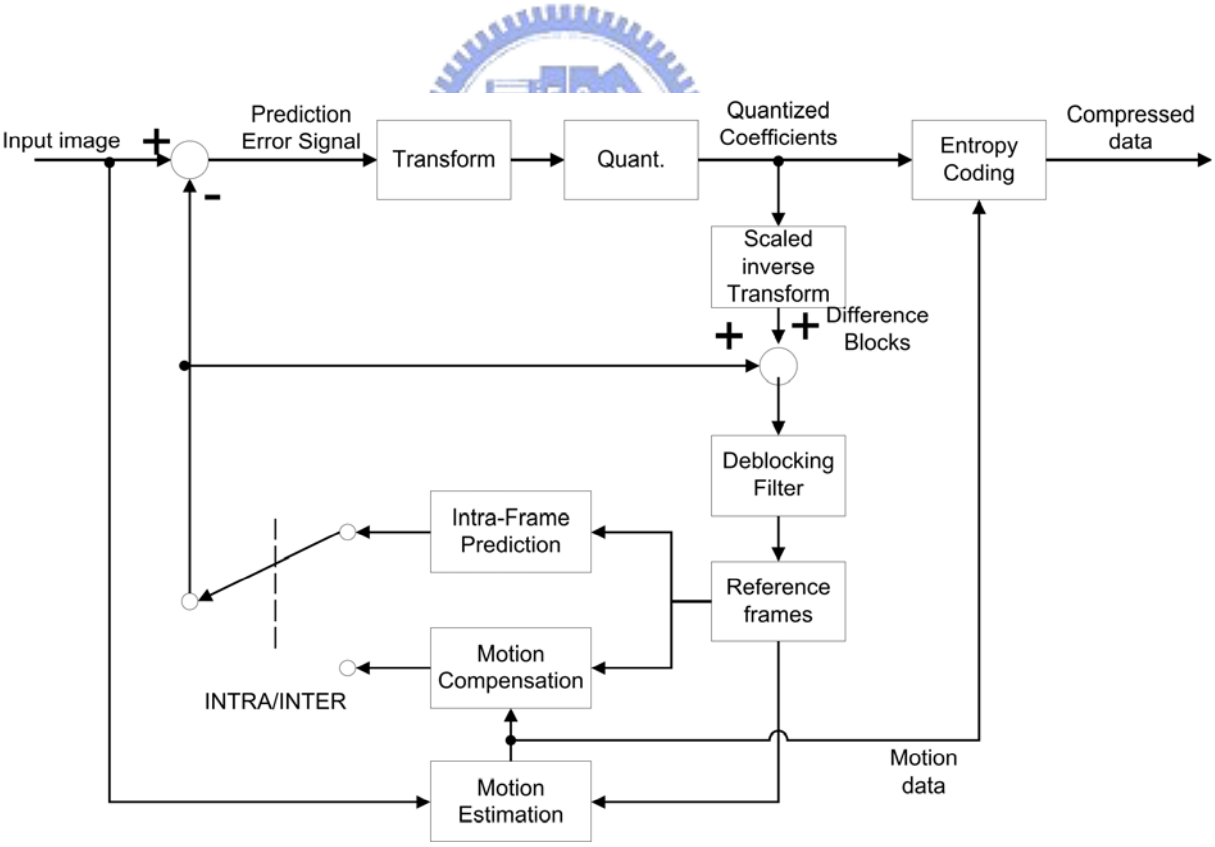


Fig. 2 H.264 encoder scheme

The prediction is subtracted from the current block to produce a residual block Prediction Error Signal (PES) that is transformed and quantized to come with Quantized Coefficients

(QC), a set of quantized transform coefficients which are reordered and entropy coded. The entropy-coded coefficients are combined with the information required to decode each block within the macroblock, such as prediction modes, quantizer parameter, motion vector information, etc. They form the compressed bitstream which is passed to a Network Abstraction Layer (NAL) for later transmission or storage.

As well as encoding and transmitting each block in a macroblock, the encoder decodes (reconstructs) it to provide a reference for further predictions. The coefficients, QC, are scaled and inverse transformed to produce a Difference Block (DB). The prediction block is added to DB to create a reconstructed block. A filter is applied to reduce the effects of blocking distortion and the reconstructed reference picture is created from a series of filtered blocks.

2.4 H.264 Decoder Data path

The dataflow path in the decoder shown in Fig. 3 illustrates the similarities between encoder and decoder.

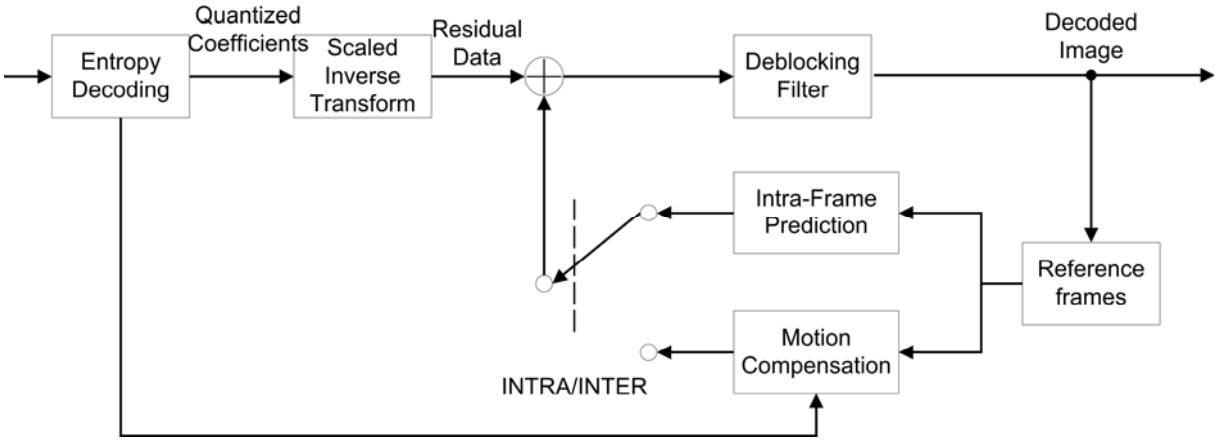


Fig. 3 H.264 decoder scheme

The decoder receives a compressed bitstream from the NAL and entropy decodes the data elements to produce a set of QC. These are scaled and inverse transformed to give residual data. Using the header information decoded from the bitstream, the decoder creates a prediction block, identical to the original prediction formed in the encoder. The prediction is added to residual data to produce which is filtered to create each decoded block.

2.5 Macroblock Prediction

Every coded macroblock in an H.264 slice is predicted from the previously-encoded data. Samples within an intra macroblock are predicted from samples in the current slice that have already been encoded, decoded and reconstructed; samples in an inter macroblock are predicted from the previously-encoded ones.

A prediction for the current macroblock or block (a model that resembles the current macroblock or block as closely as possible) is created from image samples that have already been encoded (either in the same slice or in a previously encoded slice). This prediction is subtracted from the current macroblock or block and the result of the subtraction (residual) is compressed and transmitted to the decoder, together with information required for the decoder to repeat the prediction process (motion vector(s), prediction mode, etc.).

The decoder creates an identical prediction and adds this to the decoded residual or block. The encoder bases its prediction on encoded and decoded image samples (rather than on original video frame samples) in order to ensure that the encoder and decoder predictions are identical.

In addition, we compare macroblock in intra mode and inter mode. The comparison is listed in Table 1.

Table 1 Comparison of intra prediction and inter prediction

	Dependence	Reference	Complexity
Intra prediction	Spatial	Current frame	Medium
Inter prediction	Temporal	Restored frame	High

The following introduces intra prediction and inter prediction in detail.

2.5.1 Intra Prediction

Intra-prediction is based on the observation that adjacent macroblocks tend to have similar properties. Therefore, as a first step in the encoding process for a given macroblock, one may predict the macroblock of interest from the surrounding macroblocks. The difference between the actual macroblock and its prediction is then coded, which results in fewer bits to represent the macroblock of interest. If a block or macroblock is encoded in the intra mode, a prediction block is formed based on the previously encoded and reconstructed blocks in the same frame. This prediction block is subtracted from the current block prior to be encoded. For the luma samples, the block may be formed for each 4×4 sub-block or for a 16×16 macroblock. There are a total of 9 optional prediction modes for each 4×4 luma block as shown in Fig. 4; 4 optional modes for a 16×16 luma block; and one mode that is always applied to each 4×4 chroma block as shown in Fig. 5.

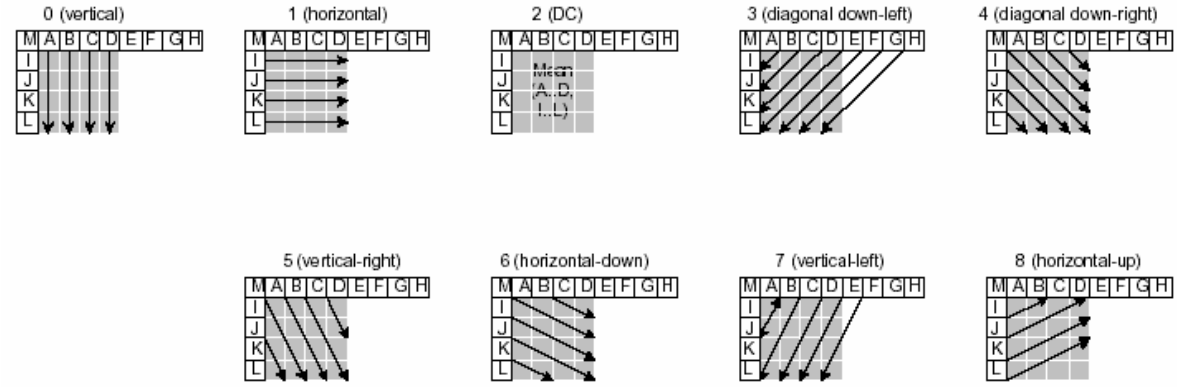


Fig. 4 4×4 Intra Prediction

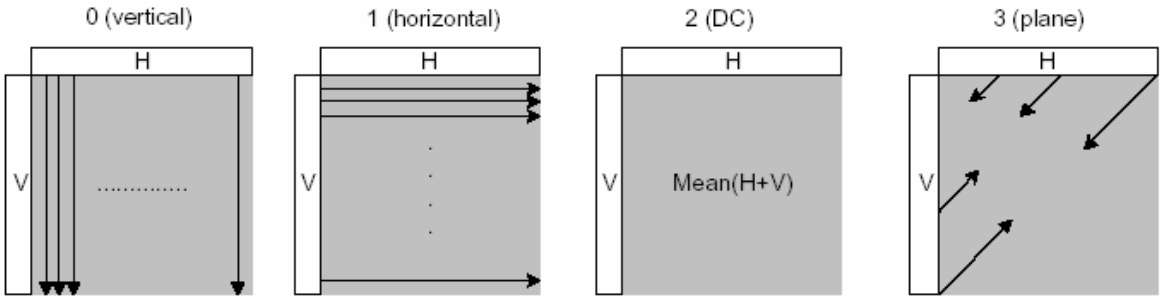


Fig. 5 16×16 Intra Prediction

2.5.2 Inter Prediction

Inter prediction creates a prediction model from one or more previously encoded video frames or fields using block-based motion compensation. Important differences from earlier standards include the support for several of block sizes and fine sub-sample motion vectors. Inter-prediction is based on using motion estimation and compensation to take advantage of the temporal redundancies that exist between successive frames. In H.264, motion estimation supports most of the key features adopted in pervious video standards, but its efficiency is improved by added flexibility and functionality. In addition to supporting P-pictures (with single and multiple reference frames) and B-pictures, H.264 also supports a new inter-stream transitional picture called an SP-picture. The inclusion of SP-pictures in a bit stream enables efficient switching between bit streams with similar content encoded at different bit rates, as well as random access and fast playback modes.

In H.264, Variable block-size motion compensation[18] and Multi-picture motion compensation[19][20][21] are significant new features. Variable block-size motion compensation is the use of block motion compensation with the ability for the encoder to dynamically select the size of the blocks shown in Fig. 6.

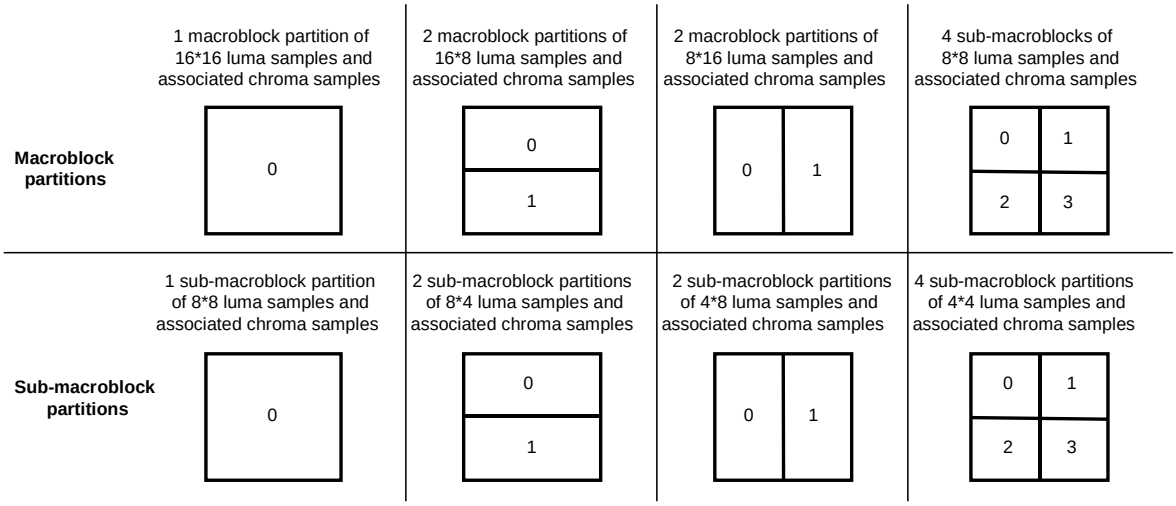


Fig. 6 Block sizes for Variable block-size motion compensation

When coding video, the use of larger blocks can reduce the number of bits needed to represent the motion vectors, while the use of smaller blocks can result in a smaller amount of prediction residual information to encode.

Multi-picture motion compensation uses previously-encoded pictures as references in a much more flexible way than in past standards and allows up to 32 reference pictures to be used in some cases. This particular feature usually allows modest improvements in bit rate and quality in most scenes. But in certain types of scenes, for example scenes with rapid repetitive flashing or back-and-forth scene cuts or uncovered background areas, it allows a very significant reduction in bit rate.

2.6 Transform Coding

H.264 uses three kinds of textual transform[15]. It depends on the type of residual data that is to be coded. See (1). H_1 is Hadamard transform for the 4×4 array of luma DC coefficients in intra macroblocks predicted in 16×16 modes. H_2 is a DCT-based transform for all other 4×4 blocks in the residual data. H_3 is a Hadamard transform for the 2×2 array of chroma DC coefficients in any macroblock.

$$H_1=\begin{bmatrix}1&1&1&1\\2&1&-1&-2\\1&-1&-1&1\\1&-2&2&-1\end{bmatrix}\quad H_2=\begin{bmatrix}1&1&1&1\\1&1&-1&-1\\1&-1&-1&1\\1&-1&1&-1\end{bmatrix}\quad H_3=\begin{bmatrix}1&1\\1&-1\end{bmatrix} \tag{1}$$

For different block sizes, transform matrices are used for post-matrix or pre-matrix as shown in (2), (3), and (4).

$$F=\begin{bmatrix}1&1&1&1\\1&1&-1&-1\\1&-1&-1&1\\1&-1&1&-1\end{bmatrix}\begin{bmatrix}c_{00}&c_{01}&c_{02}&c_{03}\\c_{10}&c_{11}&c_{12}&c_{13}\\c_{20}&c_{21}&c_{22}&c_{23}\\c_{30}&c_{31}&c_{32}&c_{33}\end{bmatrix}\begin{bmatrix}1&1&1&1\\1&1&-1&-1\\1&-1&-1&1\\1&-1&1&-1\end{bmatrix} \tag{2}$$

$$F=\begin{bmatrix}1&1&1&1\\1&1&-1&-1\\1&-1&-1&1\\1&-1&1&-1\end{bmatrix}\begin{bmatrix}c_{00}&c_{01}\\c_{10}&c_{11}\\c_{20}&c_{21}\\c_{30}&c_{31}\end{bmatrix}\begin{bmatrix}1&1\\1&-1\end{bmatrix}$$

(3)

$$F=\begin{bmatrix}1&1\\1&-1\end{bmatrix}\begin{bmatrix}c_{00}&c_{01}\\c_{10}&c_{11}\end{bmatrix}\begin{bmatrix}1&1\\1&-1\end{bmatrix}$$

(4)

Data within a macroblock are transmitted in the order. If the macroblock is coded in 16×16 intra mode, then the block is labeled "−1", containing the transformed DC coefficient of each 4×4 luma block, is transmitted first. Next, the luma residual blocks 0-15 are transmitted in the order shown in Fig. 7 (the DC coefficients in a macroblock coded in 16×16 Intra mode are not sent). Blocks 16 and 17 are sent, containing a 2×2 array of DC coefficients from the Cb and Cr chroma components respectively and finally, chroma residual blocks 18-25 (without DC coefficients) are sent.

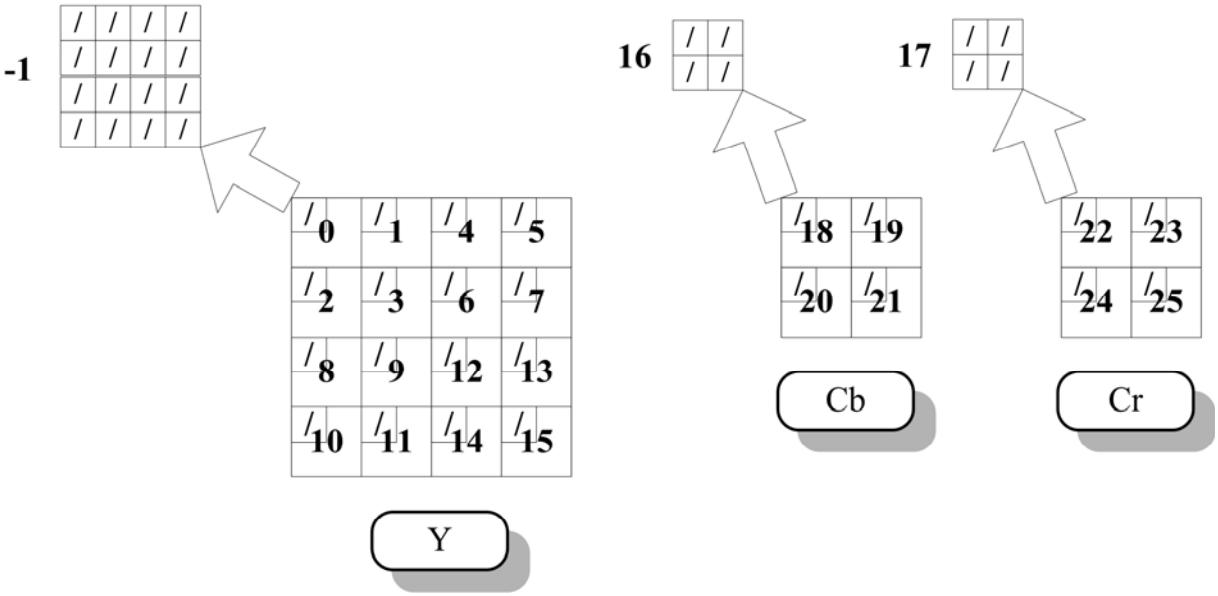


Fig. 7 Transmission order of all coefficients of a macroblock

2.7 Context-adaptive Entropy Coding

CAVLC is the baseline entropy coding method of H.264. Its basic coding tool consists of a single VLC of structured Exp-Golomb codes, which by means of individually customized mappings is applied to all syntax elements except those related to quantized transform coefficients. A given block of transform coefficients is first mapped on a 1-D array according to a predefined scanning pattern.

Typically, after quantization a block contains only a few significant, i.e., nonzero coefficients, where, in addition, a predominant occurrence of coefficient levels with magnitude equal to 1, so-called trailing 1's (T1), is observed at the end of the scan. Therefore, as a preamble, first the number of nonzero coefficients and the number of T1s are transmitted using a combined codeword, where one out of four VLC tables is used based on the number of significant levels of neighboring blocks. Then, in the second step, sign and level value of significant coefficients are encoded by scanning the list of coefficients in reverse order. By doing so, the VLC for coding each individual level value is adapted on the base of the previously encoded level by choosing among six VLC tables.

Finally, the zero quantized coefficients are signaled by transmitting the total number of zeros before the last nonzero level for each block, and additionally, for each significant level the corresponding run, i.e., the number of consecutive preceding zeros. By monitoring the maximum possible number of zeros at each coding stage, a suitable VLC is chosen for the coding of each run value. A total number of 32 different VLCs are used in CAVLC entropy coding mode, where, however, the structure of some of these VLCs enables simple on-line calculation of any code word without recourse to the storage of code tables. For typical coding conditions and test material, bit rate reductions of 2-7% are obtained by CAVLC relative to a conventional run-length scheme based on a single Exp- Golomb code.

2.8 In-loop Deblocking Filter

The block-based structure of the H.264/AVC architecture containing 4×4 transforms and block-based motion compensation can be the source of severe blocking artifacts. Filtering the block edges has been shown to be a powerful tool to reduce the visibility of these artifacts. Deblocking[22] can in principle be carried out as post-filtering, influencing only the pictures to be displayed. Higher visual quality can be achieved though, when the filtering process is carried out in the coding loop, because then all involved past reference frames used for motion compensation will be the filtered versions of the reconstructed frames. Another reason to make deblocking a mandatory in-loop tool in H.264 is to enforce a decoder to approximately deliver a quality to the customer, who was intended by the producer and not leaving this basic picture enhancement tool to the optional good will of the decoder manufacturers.

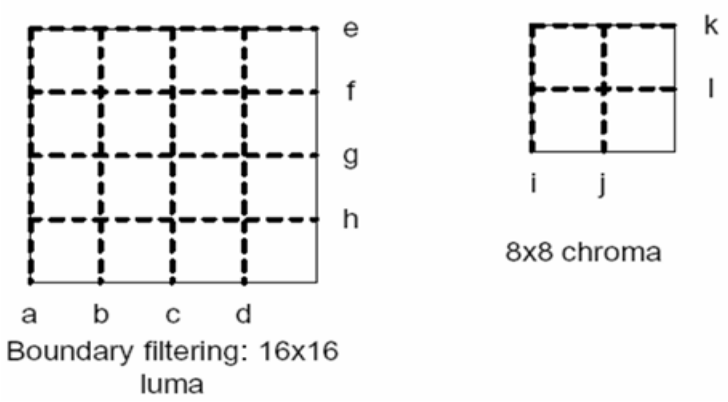


Fig. 8 Boundary Filtering

The filter described in H.264 standard is highly adaptive. Several parameters and thresholds are the local characteristics of the picture itself control the strength of the filtering process. On block edge level, the filtering strength is made dependent on inter and intra prediction decision, motion differences, and the presence of coded residuals in the two participating blocks. From these variables a filtering-strength parameter is calculated, which

can take values from 0 to 4 causing modes from no filtering to very strong filtering of the involved block edge as shown in Fig. 8

On sample level, it is crucially important to be able to distinguish between true edges in the image and those created by the quantization of the transform-coefficients. True edges should be left unfiltered as much as possible. In order to separate the two cases, the sample values across every edge are analyzed. For an explanation, it denotes the sample values inside two neighboring 4×4 blocks as p3, p2, p1, p0 | q0, q1, q2, q3 with actual boundary between p0 and q0 as shown in Fig. 9 Filtering of the two pixels p0 and q0 only takes place, if their absolute difference falls below a certain threshold α. At the same time, absolute pixel differences on each side of the edge (|p1 − p0| and |q1 − q0|) have to fall below another threshold β, which is considerably smaller than α. To enable filtering of p1(q1), additionally the absolute difference between p0 and p2 (q0 and q2) has to be smaller than β. The dependency of α and β on the quantizer, links the strength of filtering to the general quality of the reconstructed picture prior to filtering. For small quantizer values the thresholds both become zero, and filtering is effectively turned off altogether.

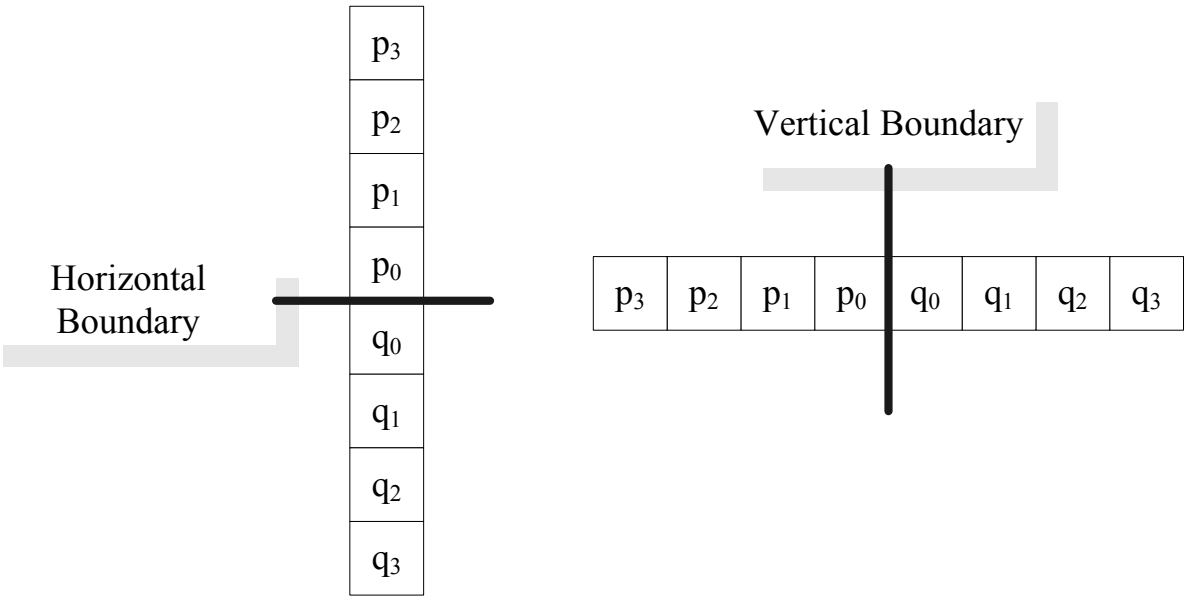
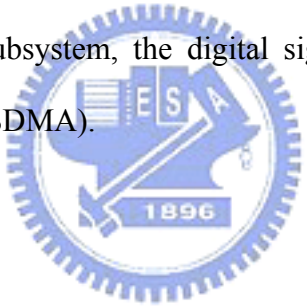


Fig. 9 Boundary of samples

3 Hardware and Software Development Environment

3.1 Target platform “OMAP5912”

OMAP5912 is the next generation of OMAP processors and succeeds the Texas Instruments OMAP5910 processor. It is built with TI 130-nm process technology and has dual input/output voltage (1.8V−3.3V) capability. The OMAP5912 includes the microprocessor unit (ARM) subsystem, the digital signal processor (DSP) subsystem, the system direct memory access (SDMA).



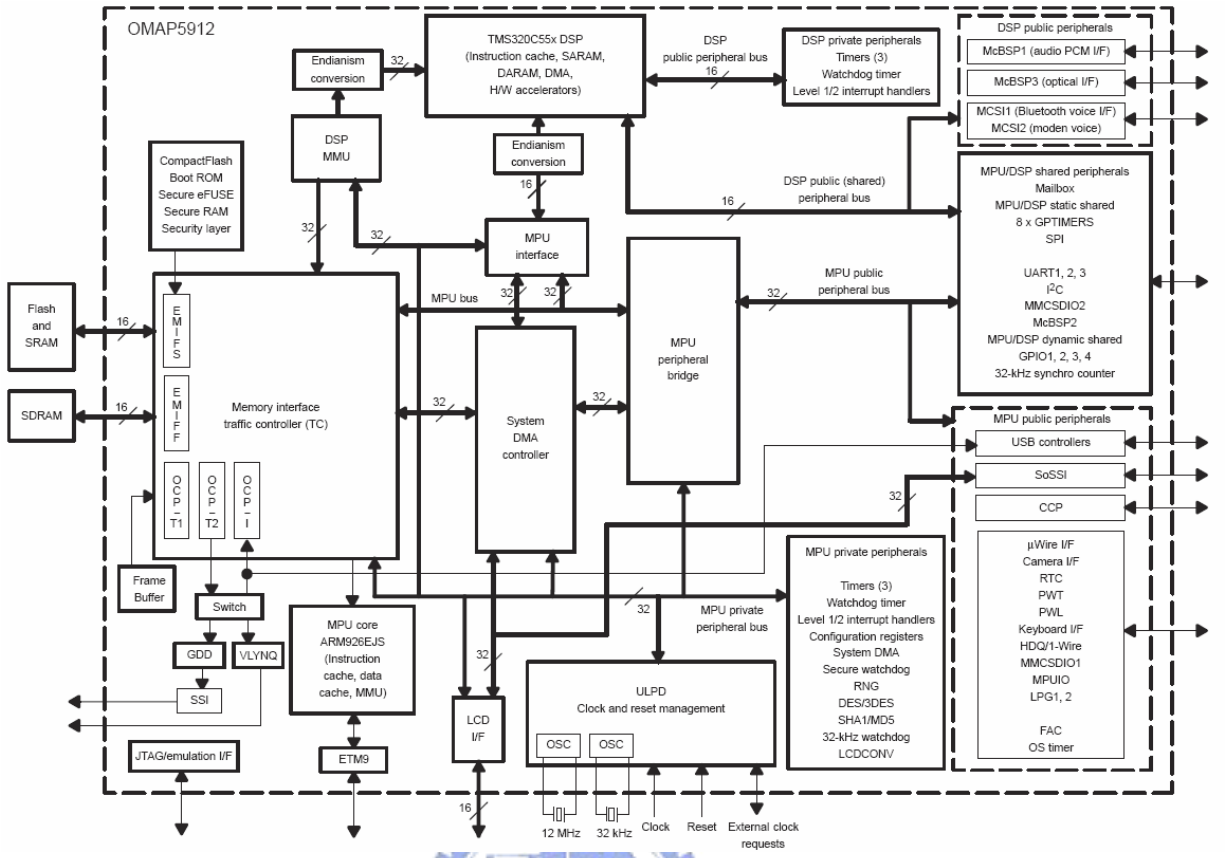


Fig. 10 The OMAP Architecture

This unique dual core architecture offers an attractive solution to both DSP and ARM developers, providing the low power real-time signal processing capabilities of a DSP coupled with the command and control functionality of a microprocessor. This low power dual core architecture forms a platform for various 3G wireless multimedia applications. The system structure of the dual core platform is shown in Fig. 10.

The ARM core has an instruction cache, a data cache and a write buffer. The DSP subsystem includes a DSP core, a software configurable instruction cache, embedded SRAM and a program ROM. It also has an embedded internal DMA controller for simultaneous data transfers and hardware accelerators for video processing, pixel interpolation and motion estimation.

The ARM is always the master in the system and is responsible for setting up and bringing the DSP out of reset. Once the DSP is out of reset, it can start executing the DSP code. The DSP applications can be either in its own local memory or can be in the shared

system memory. The DSP tasks and resources are controlled dynamically through the DSP-MMU by the ARM.

In order for effective communication to exist between the ARM and the DSP subsystems, a handshaking protocol is used. The handshaking between the ARM and DSP cores is performed through a set of inter-processor mailboxes. In a typical application, the ARM will send a message to the DSP to start performing a certain task by writing to the ARM2DSP mailbox after the DSP-MMU is configured. Once the DSP completes the task, it writes to the DSP2ARM mailbox to indicate that it is done with the current task. This kind of handshaking maintains the system coherency.

To support memory accesses, an optimized traffic controller was designed that allows dynamically configurable data throughput and asynchronous or synchronous scalable operations between the system interface, the DSP and the ARM subsystem. The programmability of the traffic controller allows for efficient performance utilization of the cores, as well as maintaining low system power at the same time. The traffic controller also supports various arbitration algorithms that can be used to control and utilize the available bandwidth efficiently.

The design provides various external memory interfaces to allow glueless hookup to standard memories such as Flash, SRAM, ROM and low power mobile SDRAM/DDR. Also, a rich set of peripherals was designed to support other multimedia functions.

To boost the system performance, the design implemented a 16 channel programmable DMA controller to enable 2-D graphic processing and simultaneous transfers of data between the host and the memories, the host and the peripherals, the internal and the external memories, and from peripherals to other peripherals.

Since the platform architecture is based on multiple cores, it is important to provide an efficient and relative-easy means to trace back an error to the source of the problem. In this platform, various emulation and debugging facilities were implemented such as a Catscan and

a window tracer at the system level besides the core specific emulators and tracers in the DSP and ARM subsystems. The Catscan facility allows the users to set a breakpoint, execute emulation, stop the execution at the breakpoint and dump out internal data and states through scan chains. This feature allows developers to isolate a problem fast and easy. The window tracer enables the users to define an address window to monitor bus transactions of all shared target interfaces. The information of the bus transaction is captured when the address of the access address falls in the defined window. The captured data is sent out through a serial tracer interface for software and hardware debugging. These target tracers provide visibility of the order of program execution on a per target basis in the complex system.

Since this application platform is targeted towards wireless devices, longer battery life is a necessity. Power reduction techniques were implemented at the Process, Architecture, Logic Design and Physical Design phases of the development. The dual core architecture enables assigning a task to one of the processors that is best suited for the task. This dynamic task allocation leads to a significant reduction in the number of processor cycles required to perform a task, which leads to a significant decrease in power consumption.

The OMAP architecture ensures that the software and the operating system (OS) have full control on all the clocking and idle modes of platform. If an application does not need a particular resource, the software can put the resource in an idle mode and even can turn off the power of the resource. This feature enables the application developer to write the application code such that the power consumption is minimized. An aggressive strategy of local clock gating coupled with the reduction of unnecessary signal/bus toggling and an optimal floor-plan has been used to further reduce power consumption.

3.2 OMAP Core Features

The OMAP3.2 includes the following features:

- ARM926EJS megacell including:
  - ARM926EJS, running at a maximum frequency of 192 MHz
  - MMU with translation lookaside buffer (TLBx)
  - L1 16K-byte, four-way, set-associative instruction cache
  - L1 8K-byte, four-way, set-associative data cache with write buffer
- ARM interrupt handler level 1
- Embedded trace macrocell module, ETM version 2.a in a 13-bit mode configuration or in a 17-bit demultiplexed mode configuration
- DSP megacell rev 2.0a+ including:
  - Embedded ICE emulator interface through JTAG port
  - TMS320C55x (C55x) DSP rev 2.1, running at a maximum of 192 MHz

- L1 cache (24K bytes)
- 16K-byte, two-way, set-associative instruction cache (on the OMAP5912 prototype, one wait state is introduced in case of discontinuity)
- 2 x 4K-byte RAM set for instruction
- DARAM 64K-byte, zero wait state, 32-bit organization
- SARAM 96K-byte, zero wait state, 32-bit organization
- PDROM (32K bytes)
- DMA controller: six physical channels, five ports
- DSP trace module
- Hardware accelerators motion estimation (ME), discrete/inverse discrete cosine transform (DCT/IDCT), and pixel interpolation (PI)
- DSP interrupt handler level 1 in the C55x DSP core

■ DSP MMU

■ DSP level 2 interrupt handler, which enables connection to 16 additional interrupt lines outside OMAP. The priority of each interrupt line is controlled by software.

■ DSP interrupt interface, which enables connection to the interrupt lines coming out of the level 2 interrupt handler and the interrupt lines requiring more priority. The outcome interrupt of this module is then connected to the DSP megacell to be processed by the DSP. This module mainly ensures that all interrupts going to the DSP megacell are level-sensitive.

■ DSP peripherals:

- 3 x 16-bit DSP private timers
- 1 x 16-bit DSP private watchdog

■ Mailboxes:

- Four mailboxes are implemented
  - ◆ Two read/write accessible by ARM, read-only by the DSP

- Two read/write accessible by the DSP, read-only by the ARM

- Each mailbox is implemented with 2 x 16-bit registers. When a write is done into a register by one processor, it generates an interrupt, released by the read access of the other processor.

- ARM peripherals
  - 3 x 32-bit private timers; their clock is either the OMAP3.2 reference input clock or the divided ARM clock.
  - 1 x 16-bit private watchdog; can be conFig.d as a 16-bit general purpose timer by software. Its clock is the OMAP3.2 reference input clock divided by 14.
  - External LCD controller support in addition to the OMAP LCD controller
  - LCD controller with its own tearing-effect logic

- Memory traffic controller
  - External Memory Interface Fast (EMIFF) is a memory interface that enables 16-bit data SDRAM memory access at 96-MHz maximum frequency. It supports connection to a 128M-byte maximum of SDRAM. The address width is 16 bits, and two bank selection bits are also provided. The OMAP5912 chip requires interfacing with a maximum of four banks of 64M x 16-bit SDRAM memory with DDR capability.
  - External Memory Interface Slow (EMIFS) connects external device memories (such as common flash and SRAM memories) at 80-MHz maximum frequency. This interface is also used internally to connect the boot ROM, the secure RAM, and their secure eFuse components accessible in secure mode. This interface enables 16-bit data accesses and provides four chip-selects. Each chip-select is able to support up to 64M bytes of address space through a 25-bit address bus.

- Emulator interface through JTAG port

- System DMA running at 96 MHz. It consists of:
  - Seventeen logical channels
  - Seven physical ports + one for configuration
  - Four physical channels
  - The ports are connected to the L3 OCP targets, the external memory, the TIPB bridge, the MPUI, and one dedicated port connected to an LCD controller. The system DMA controller can be controlled via the ARM private TIPB or by an external host via the OCP-I port. The system DMA controller is designed for low-power operation. It is partitioned into several clock domains where each clock domain is enabled only when it is used. All clocks are disabled when no DMA transfers are active (synchronous to the ARM TIPB, this feature is totally under hardware control; no specific programming is needed).

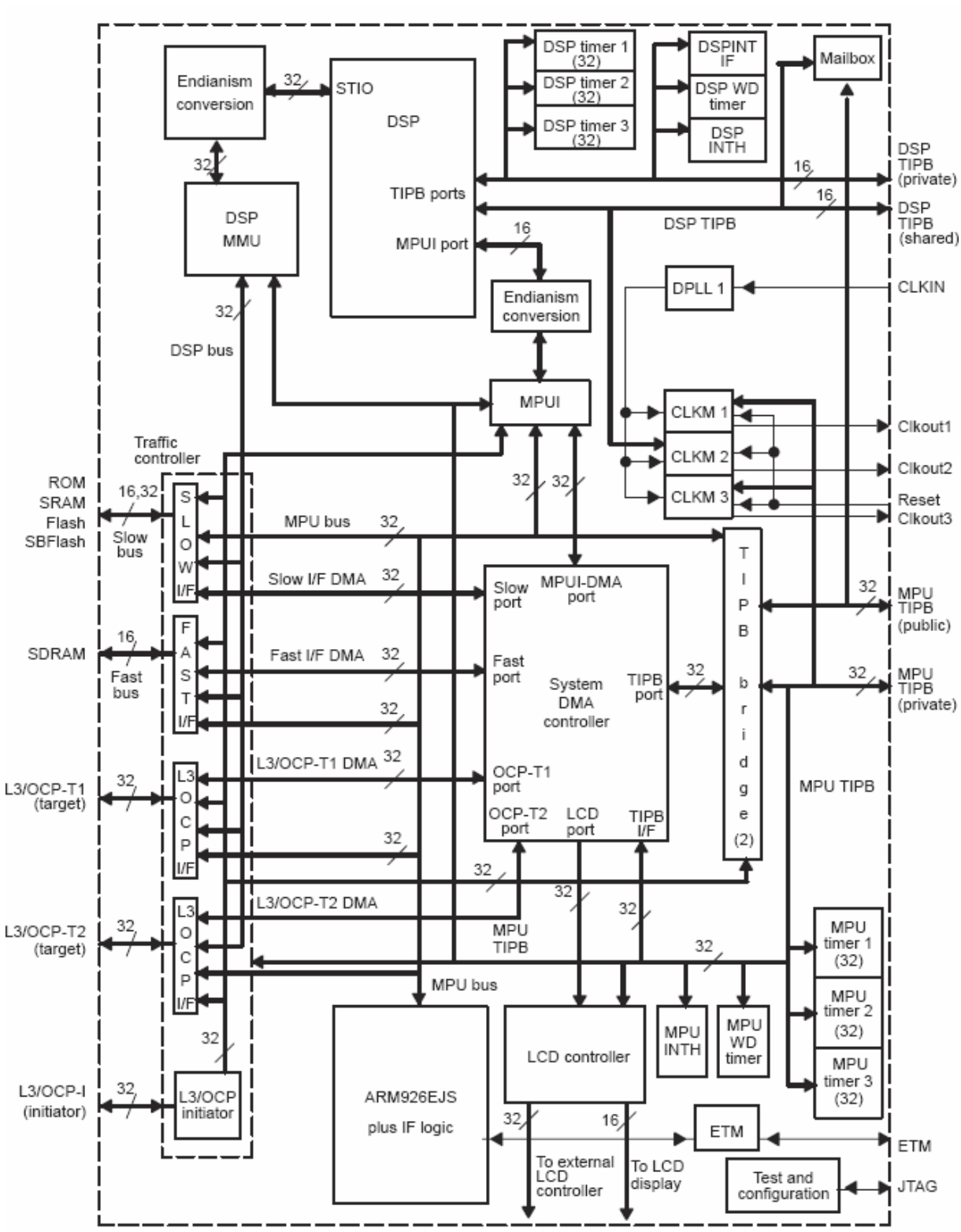


Fig. 11 OMAP Gigacell Core

3.3 ARM Linux for OMAP

ARM Linux [23] is a port of the successful Linux kernel to ARM processor based machines, lead mainly by Russell King, with contributions from many others. ARM Linux is under almost constant development by various people and organizations around the world. The ARM Linux kernel is being ported, or has been ported to more than 500 different machine variations, including complete computers, network computers, hand held devices and evaluation boards.

Linux for the OMAP software development platform can be built either by obtaining an ARM tool chain and kernel source, the preview kit from MontaVista, or by installing an open source tool chain and building an open source Linux kernel.

The preview kit is designed to use the same network-based development environment as MontaVista Linux Professional Edition, including useful components such as DHCP, NFS and tftp. With the preview kit, developers receive a full MontaVista Linux kernel based on Linux version 2.4 running on a range of popular reference platforms including eleven processors from six architecture families.

The preview kit lets developers explore the building and debugging process and invites them to evaluate MontaVista Linux networking and real-time performance capabilities through a variety of tools and software components. It also offers a hands-on preview of the MontaVista developer environment, designed for cross-platform development of embedded applications.

3.4 Open source H.264 encoders

- JM reference software model[24]

Reference software is useful in providing users a video coding standard to establish and test conformance and interoperability, and to educate users and demonstrate the capabilities of the standard. For these purposes, the accompanying software is provided as an aid for the study and implementation of ITU-T Rec. H.264 | ISO/IEC 14496-10 advanced video coding.

- **x264[25]**

x264 is a free library for encoding H.264/MPEG-4 AVC video streams. The code is written by Loren Merritt, Laurent Aimar, Eric Petit, Min Chen, Justin Clay, Måns Rullgård, Radek Czyz, Alex Izvorski, Alex Wright, and Christian Heine from scratch. It is released under the terms of the GNU General Public License.

In Table 3, Comparing x264 to the H.264 reference software model JM [24], the performance of x264 is much better. To achieve better encoding speed, x264 is chosen to be the base of our work. Besides, x264 was written in clear and modular style. If we want to work efficiently and replace some functional parts with our optimized code, it will be easier.

Table 2 Comparison of x264 and JM

	x264	JM
Encode Speed	Fast	Much Slower
Quality	Good	Good
Programming language	ISO C99 and nasm	Plain C
Supported profile	Baseline, Main, High	All
Output file format	Mp4 or Mkv	NAL bitstream

3.5 Use of DSP Compiler

The overall compiling flow is shown in Fig. 12. When programmers has coded the source code, compiling is very important procedure. The TMS320C55X C/C++ compiler accepts C/C++ source code and produces C55x assembly language source code. An optimizer is part of the compiler. The optimizer modifies code to improve the efficiency of C/C++ programs. The assembler translates assembly language source files into machine language object files. The TMS320C55x tools include two assemblers. The mnemonic assembler accepts C54x and C55x mnemonic assembly source files. The algebraic assembler accepts C55x algebraic assembly source files. The machine language is based on common object file format (COFF). The linker combines object files into a single executable object module. As it creates the executable module, it performs relocation and resolves external references. The linker accepts replaceable COFF object files and object libraries as input.

Standard runtime-support library functions are provided as source code in rts.src. The runtime-support libraries contain the ISO standard runtime-support functions, compiler-utility functions, floating-point arithmetic functions, and C I/O functions that are supported by the C55x compiler. The C55x debugger accepts executable COFF files as input, but most EPROM programmers do not. The hex conversion utility converts a COFF object file into TI-Tagged, ASCII-hex, Intel, Motorola-S, or Tektronix object format. The converted file can be downloaded to an EPROM programmer.

The compiler uses a sophisticated optimization pass that employs several advanced techniques for generating efficient, compact code from C/C++ source. General optimizations can be applied to any C/C++ code, and C55x specific optimizations take advantage of the features specific to the C55x architecture.

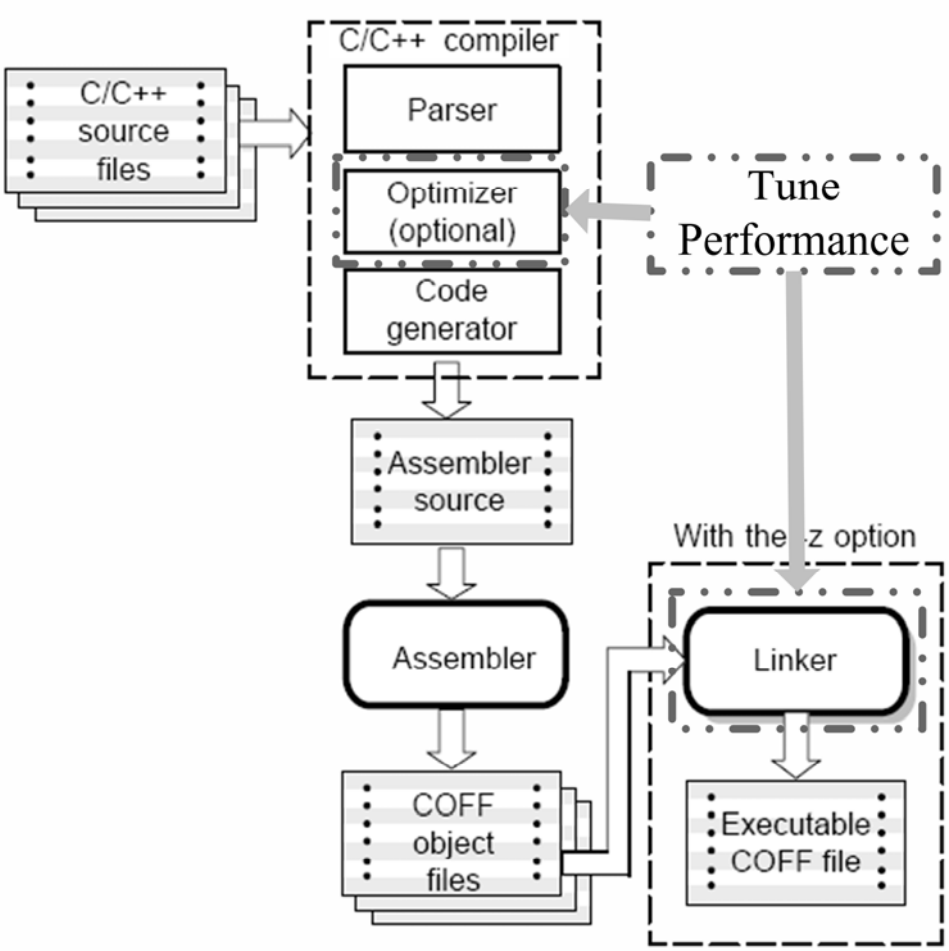


Fig. 12 Overflow Compiler Working Flow

The performance is adjusted in the optimizer. The C/C++ compiler is able to perform various optimizations. High-level optimizations are performed in the optimizer, which must be used to achieve optimal code. The easiest way to invoke the optimizer is to specify the –On option on the cl55 command line. The n denotes the level of optimization (O0, O1, O2, and O3), which controls the type and degree of optimization:

- **-O0**
  - 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

- **-O1**
  - Performs all -O0 optimizations, plus:
  - Performs local copy/constant propagation
  - Removes unused assignments
  - Eliminates local common expressions
- **-O2**
  - Performs all -O1 optimizations, plus:
  - Performs loop optimizations
  - Eliminates global common sub-expressions
  - Eliminates global unused assignments
  - Performs loop unrolling
  - The optimizer uses -O2 as the default if you use -O without an optimization level.
- **-O3**
  - Performs all -O2 optimizations, plus:
  - Removes all functions that are never called
  - Simplifies functions with return values that are never used
  - Inlines calls to small functions
  - Reorders function declarations so that the attributes of called functions are known when the caller is optimized
  - Identifies file-level variable characteristics

The levels of optimization described above are performed by the stand-alone optimization pass. The code generator performs several additional optimizations, particularly processor-specific optimizations; it does so regardless of whether you invoke the optimizer. These optimizations are always enabled although they are much more effective when the optimizer is used.

3.6 DSP Gateway

3.5.1 Overview

ARM Linux has become a popular platform for embedded systems and thousands of pre-compiled binary applications for ARM Linux can be found on the Internet. However, when it comes to the DSP utilization from ARM Linux, there are only a few movements. This library supports one OMAP Linux model that is making the best of OMAP processors.

DSP Gateway is a mechanism that would help programmers to easily use ARM and DSP at the same time. DSP Gateway consists of Linux drivers and the DSP firmware. They are responsible for hardware settings, interrupt handlings, and communications in between. With help from DSP Gateway, DSP enhanced applications for Linux can be developed without low-level hardware knowledge. From ARM side, DSP can be reached through DSP device file under /dev directory on the Linux file system. On the DSP side, tasks can easily be synchronized with ARM side by using the API provided by DSP Gateway. Of course, we could prepare a library sitting between ARM Applications" and the "Linux DSP Driver" that hides DSP from programmers, so that DSP enhanced applications can be easily developed without even being aware of DSP upon using this library.

3.5.2 Inter-processor communication

The ARM-DSP inter-processor communication is implemented as ‘mailbox’ build in OMAP system illustrated in Fig. 13 . There are four sets of mailbox registers: two for ARM to send messages and enable an interrupt to DSP, the other for DSP to send messages and issue an interrupt to ARM. Every set of mailbox registers consists of two 16-bit registers and a 1-bit flag register. The interrupting processor can use one 16-bit register to pass a data word to the interrupted processor and the other 16-bit register to pass a command word.

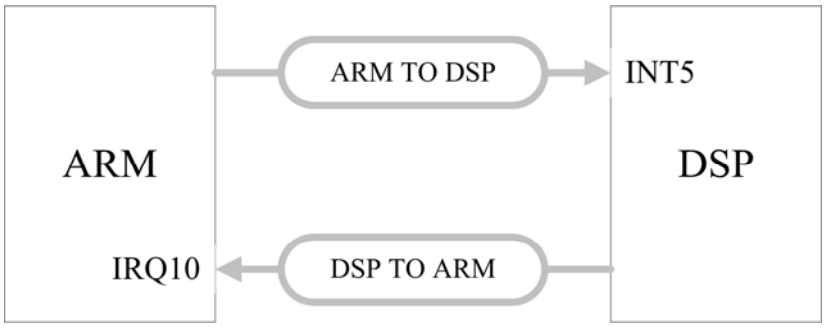


Fig. 13 Mailboxes of ARM and DSP

3.5.3 Mailbox Command Protocol

Tasks in the DSP are identified with TID (Task ID), and most commands are sent with the TID. IPBUFs are used for the block data transfer between ARM and DSP.

An IPBUF is owned by ARM and DSP. Only one core which has the its ownership can access to it . When a data transfer with an IPBUF is performed, the ownership of the IPBUF moves to the recipient. 7-bit mailbox commands are defined as in Table 2-2. These commands are sent on CMD_H field (bits [14:8]) in the command register, along with supplementary information on CMD_L bits and data register.

Table 3 Mailbox Command Definition

Command	CMD_H	CMD_L	data register	Short Description
WDSND	0x10	TID	word data to send	Word Send
WDREQ	0x11	TID	-	Word Request
BKSND	0x20	TID	BID	Block Send
BKREQ	0x21	TID	requesting count	Block Request
BKYLD	0x23	-	BID	Block Yield
BKSNDP	0x24	TID	-	Block Send Private
BKREQP	0x25	TID	-	Block Request Private
TCTL	0x30	TID	ctlcmd / data	Task Control
WDT	0xd0	-	notification value	Watchdog Timer Notification
TCFG	0xe0	TID	configuration data	Task Conriguration
TADD	0xe2	TID	BID	Task Add
TDEL	0xe3	TID	-	Task Delete
TSTOP	0xe5	-	-	Task Stop
DSPCFG	0xf0	CFGTYP	configuration data	System Configuration
REGRW	0xf2	TYPE	address / data	Register Read / Write
GETVAR	0xf4	VARID	data	Get Variable
SETVAR	0xf5	VARID	data	Set Variable
ERR	0xf8	EID	command caused the error	Error Information

3.7 DSP Gateway Linux APIs

Fig. 14 shows Linux APIs for the DSP Gateway driver. There are five device interfaces regarding the DSP Gateway - DSP task devices, DSP task watch device, DSP control device, DSP error detection device and DSP memory device. In general, user applications which want to utilize DSP task (like codec) need to access only to the DSP task devices. Other device I/Fs are used by special utility programs, i.e. DSP application Dynamic Loader Daemon and dspctl utility.

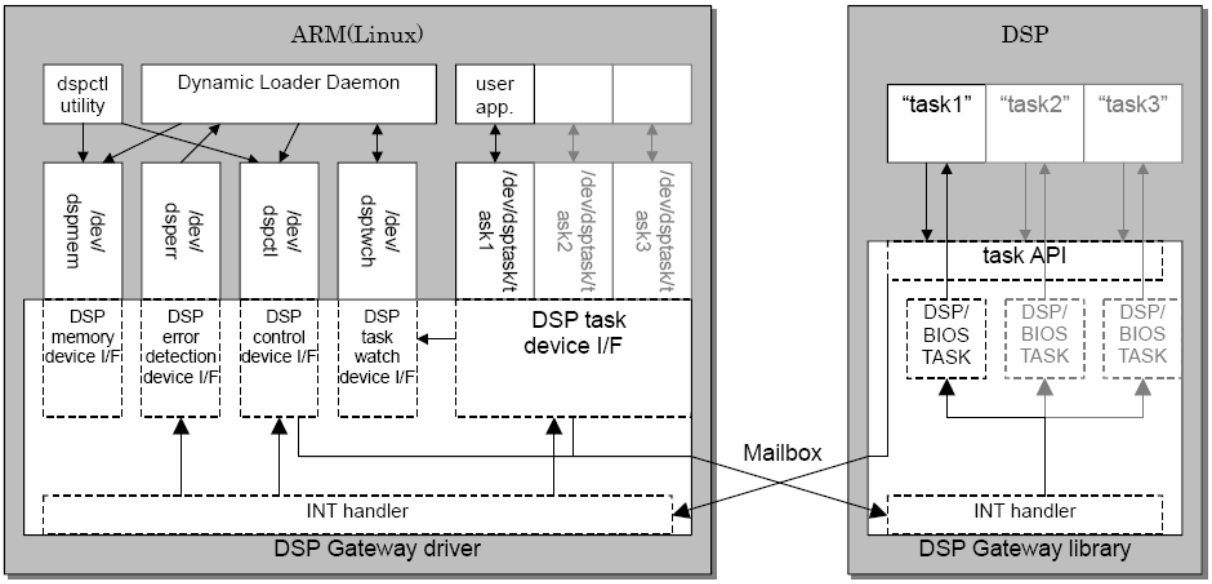


Fig. 14 DSP Gateway Linux APIs

- DSP Task Devices

The DSP task devices provide interfaces to the DSP tasks for Linux applications. Those device files are created automatically or explicitely at /dev/dsptask directory like below.

```
crw-r--r-- 1 root root 97, 0 Jan 1 0:01 /dev/dsptask/task0
crw-r--r-- 1 root root 97, 1 Jan 1 0:01 /dev/dsptask/task1
```

Reading from and writing to those devices mean receiving and sending data from/to the DSP tasks. There are two types of the DSP tasks, static task and on-demand task. The

static tasks are linked with the tokliBIOS kernel statically, and are loaded to DSP memory together. They become available by the DSP configuration (i.e. DSPCFG ioctl command to the DSP control device), and stay alive until DSP unconfiguration (i.e. DSPUNCFG ioctl command to the DSP control device). Unlike the static tasks, the on-demand tasks are not linked with the tokliBIOS kernel statically but loaded dynamically. To use the on-demand tasks, Linux side needs to create the task device file with the MKDEV ioctl command to the DSP task watch device (/dev/dsptwch), then load the task program object to the DSP memory, and issue the TADD ioctl command to the DSP task watch device.

- DSP Control Device

The DSP control device provides DSP control API for Linux. Linux applications can execute DSP reset, read DSP configuration, and perform other various control commands. The device file is at /dev/dspctl.

```
crw-r--r-- 1 root root 96, 0 Jan 1 0:00 /dev/dspctl
```

- DSP Memory Device

The DSP memory device provides the access to the DSP memory space for the DSP program loader in the Linux side. The device file is at /dev/dspmem.

```
crw-r--r-- 1 root root 101, 0 Jan 1 0:00 /dev/dspmem
```

The DSP program loader loads the DSP binary image to the DSP internal memories (i.e. DARAM and SARAM) through this device. Moreover, the user can extend the usable range of the DSP memory by mapping the external SDRAM block to the DSP memory space using the ioctl commands as in Fig. 15. After mapping, the DSP application loader can access to the extended memory as well.

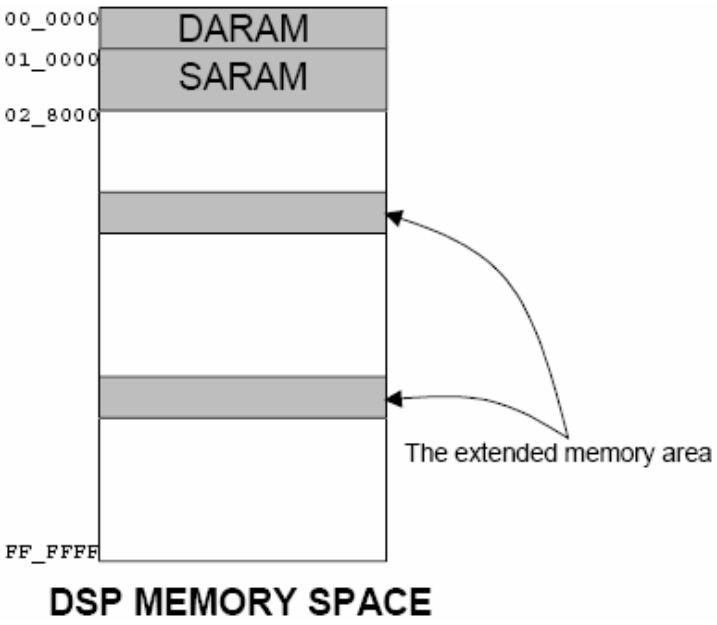


Fig. 15 External Memory Mapping for DSP Space

- DSP Task Watch Device

The DSP Task Watch device provides functionalities needed for the DSP Dynamic Loader Daemon.

```
crw-r--r-- 1 root root 96, 1 Jan 1 0:00 /dev/dsptwch
```

The daemon can obtain the information such as which task is in use, not in use or requesting to be loaded and started. It adds and deletes the on-demand the DSP tasks, creates and removes DSP task device files for those tasks through the “ioctl” commands for this device.

- DSP Error Detection Device

The DSP Error Detection device is used to detect error from DSP, such as watchdog timer expiration, DSP MMU error interrupt and etc.

```
crw-r--r-- 1 root root 96, 2 Jan 1 0:00 /dev/dsperr
```

3.8 DSP programming with DSP Gateway

Fig. 16 shows the DSP software block chart. When a user application accesses to the DSP task device, /dev/dsptask/task1 for example, the driver generates a “Mailbox” command to DSP. In the DSP side, the system kernel called tokliBIOS receives the “Mailbox” command and registers it into the queue of the corresponding DSP/BIOS TSK. DSP/BIOS TSK processes the commands in the queue by calling corresponding task function, which is a member of the task1 in this case. This task function is the program which DSP application programmer should implement. The task functions can send back Mailbox commands to ARM by calling task API functions in the tokliBIOS.

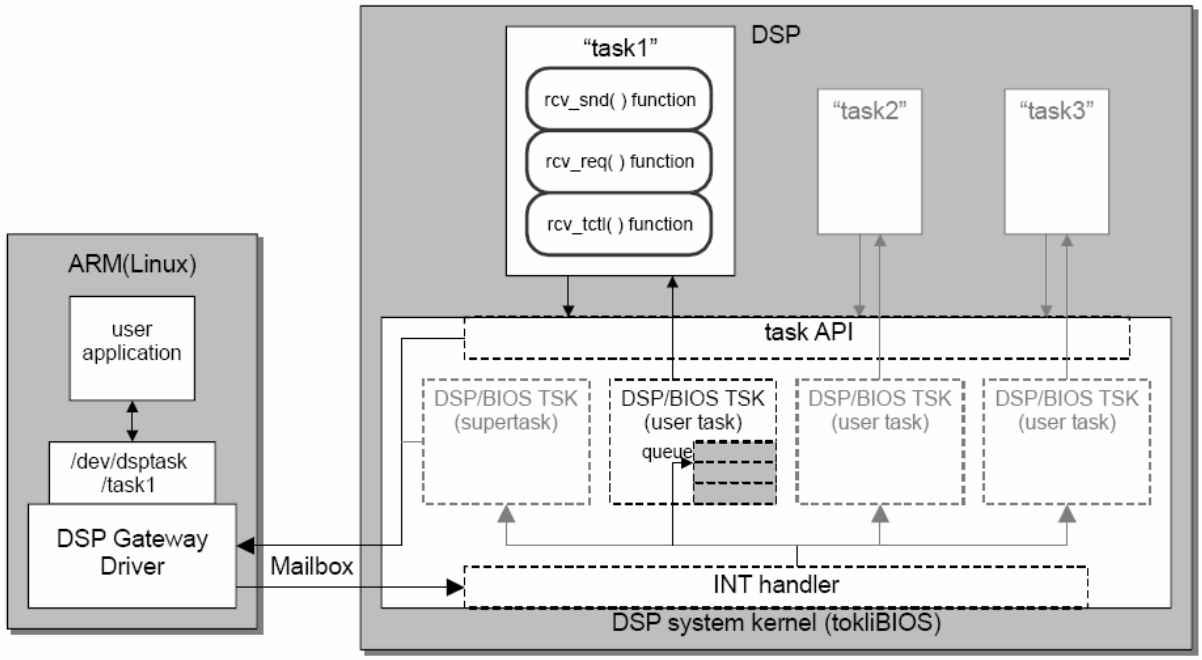


Fig. 16 DSP Software Block Chart

DSP programmer should implement the following functions and the data defined in the dsptask structure for each task.

```
struct dsptask {
Uns tid;
String name;
Uns ttyp;
Uns (*rcv_snd)();
// (*rcv_wdsnd)(struct dsptask *task, Uns data);
// (*rcv_bksnd)(struct dsptask *task, Uns bid, Uns cnt);
// (*rcv_bksndp)(struct dsptask *task, Uns cnt);
Uns (*rcv_req)();
// (*rcv_wdreq)(struct dsptask *task);
// (*rcv_bkreq)(struct dsptask *task, Uns cnt);
Uns (*rcv_tctl)(struct dsptask *task, Uns ctlcmd);
struct TSK_Attrs *tsk_attrs;
};
```

The maximum number of the DSP task is 126 (TID `0x00-0xfd`), leaving `0xfe` and `0xff` for the special purposes for the IPBUF owner information.

Moreover, DSP C55x compiler defines its own size for each C data type (signed and unsigned):

`char`	16 bits
`short`	16 bits
`int`	16 bits
`long`	32 bits
`long long`	40 bits
`float`	32 bits
`double`	64 bits

Floating point values are in the IEEE format.

3.9 Memory access of OMAP C55X DSP subsystem

The C55X DSP architecture uses a unified program and data memory space composed of memory internal and external to the DSP subsystem. The internal memory is made up of tightly coupled memory blocks, whereas DSP external memory is mapped to the OMAP system memory. The C55X DSP architecture provides access to a maximum of 8M words (16M bytes) of program/data memory space. The C55X DSP internal memory consists of four types of tightly coupled memories which provide the DSP core with maximum efficiency.

- Dual-access RAM (DARAM)

  The DARAM memory consists of 8 blocks of 8K bytes each. The DARAM (64K bytes) can support up to two memory accesses into each RAM block in one DSP core clock cycle. Accesses can be made from any internal data, program, or DMA bus.

- Single-access RAM (SARAM)

  The SARAM memory consists of 12 blocks of 8K bytes each. The SARAM (96K bytes) can support one memory access into each RAM block in one DSP core clock cycle. This access can be a 32-bit value. Accesses can be made from any internal data, program, or DMA bus.

- Programmable dynamic ROM (PDROM)

  The PDROM memory consists of 1 block of 32K bytes. The programmable dynamic ROM (32K bytes) can support one memory read in one DSP core clock cycle. This access can be a 32-bit value. Accesses can be made from any internal data read or program bus.

- Configurable I-Cache structure (optional)

  The DSP instruction cache (I-Cache) module is a special-purpose, tightly coupled,

RAM-based program memory. The module is designed to significantly improve DSP core performance by buffering the instructions most recently fetched from DSP external memory. The entire external program memory space is cacheable.

The DSP core and the DMA controller use the external memory interface (EMIF) to access the DSP external memory. The external memory for the DSP subsystem ranges from byte address 0x02 8000 to 0xFF 8000 if the internal PDROM is enabled, or to 0xFF FFFF if it is not enabled. All DSP external memory access requests are passed through the DSP memory management unit (MMU). If this unit is enabled and configured by the ARM core, it translates the DSP external memory access request address, also called a virtual address, into a system memory address, also called a physical address, that is then passed to the traffic controller. The traffic controller completes the memory access through one of the three system memory interfaces: internal memory (IMIF), slow external memory (EMIFS), or fast external memory (EMIFF).

Four major steps are taken when the DSP subsystem accesses the DSP external memory.

1. The DSP core or the DSP DMA requests an access to DSP external memory.
2. The DSP EMIF receives that request and forwards it to the DSP MMU.
3. The MMU checks its translation look-aside buffer for a match on the virtual address tag.

   If there is a TLB hit and the correct access permissions for the type of access (read or write) are found, the MMU translates the virtual address from the EMIF into a physical address and forwards the request to the traffic controller with the appropriate endianess conversion. If the virtual address tag is not found, the MMU uses its table walking logic to fetch the translation from translation tables and updates the TLB. If correct access permissions are found, the MMU carries out the virtual-to-physical address translation and forwards the request to the traffic controller. If the correct access permissions are not found, MMU generates an interrupt to the ARM core and stalls the DSP EMIF until the error is cleared. When the ARM core clears this error, these DSP

MMU repeats this entire steps.

4. The traffic controller accesses the actual OMAP resource.

Fig. 17 shows the major blocks involved during an access to DSP external memory by the DSP subsystem.

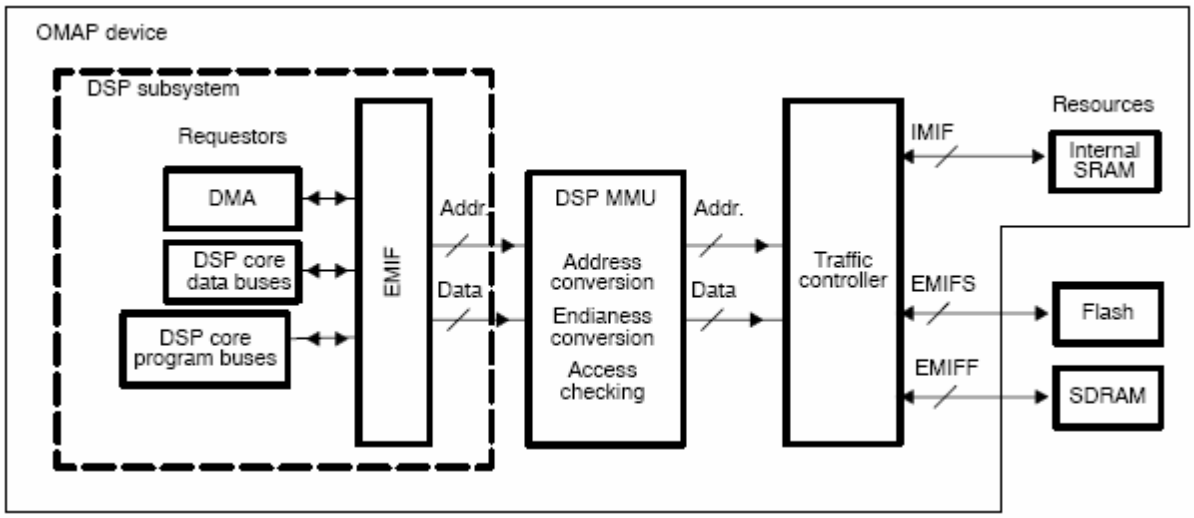


Fig. 17 DSP Subsystem External Memory Connections

3.10 Controlling DSP MMU

The DSP core and The DSP DMA accesses to DSP external memory are handled by the EMIF in conjunction with the DSP MMU. The DSP MMU maps external memory requests to the OMAP physical address space. The MMU also provides fault and permission checking, and performs endianess conversion. It is configured by the ARM core.

The use of an MMU offers two major benefits:

- Memory defragmentation: Fragmented memory can be translated into continuous virtual memory without moving any data.
- Task protection: Illegal, non-allowed accesses to memory locations can be detected and prevented.

There are two ways to use the MMU:

- The contents of the TLB can be written manually by the ARM core.

  Using this approach does not require any translation tables. However, the ARM core has to update the TLB when no valid address translation is found (TLB miss).
- The MMU table walking logic can be enabled to automatically update the TLB by reading a structure of translation tables.

  The translation table structure has to be set up by the ARM core before the MMU is enabled. However, no action from the ARM core is required on a TLB miss.

Whenever an address translation is requested (that is, for every memory access with the DSP MMU enabled), the DSP MMU checks first to see whether TLB contains the requested translation. TLB acts like a cache, storing recent translations.

If the translation is contained in the TLB and the access permissions are correct, the corresponding physical address is calculated and the memory request is forwarded to the traffic controller. If the memory request lacks the correct access permissions, the MMU generates a fault interrupt to the ARM core.

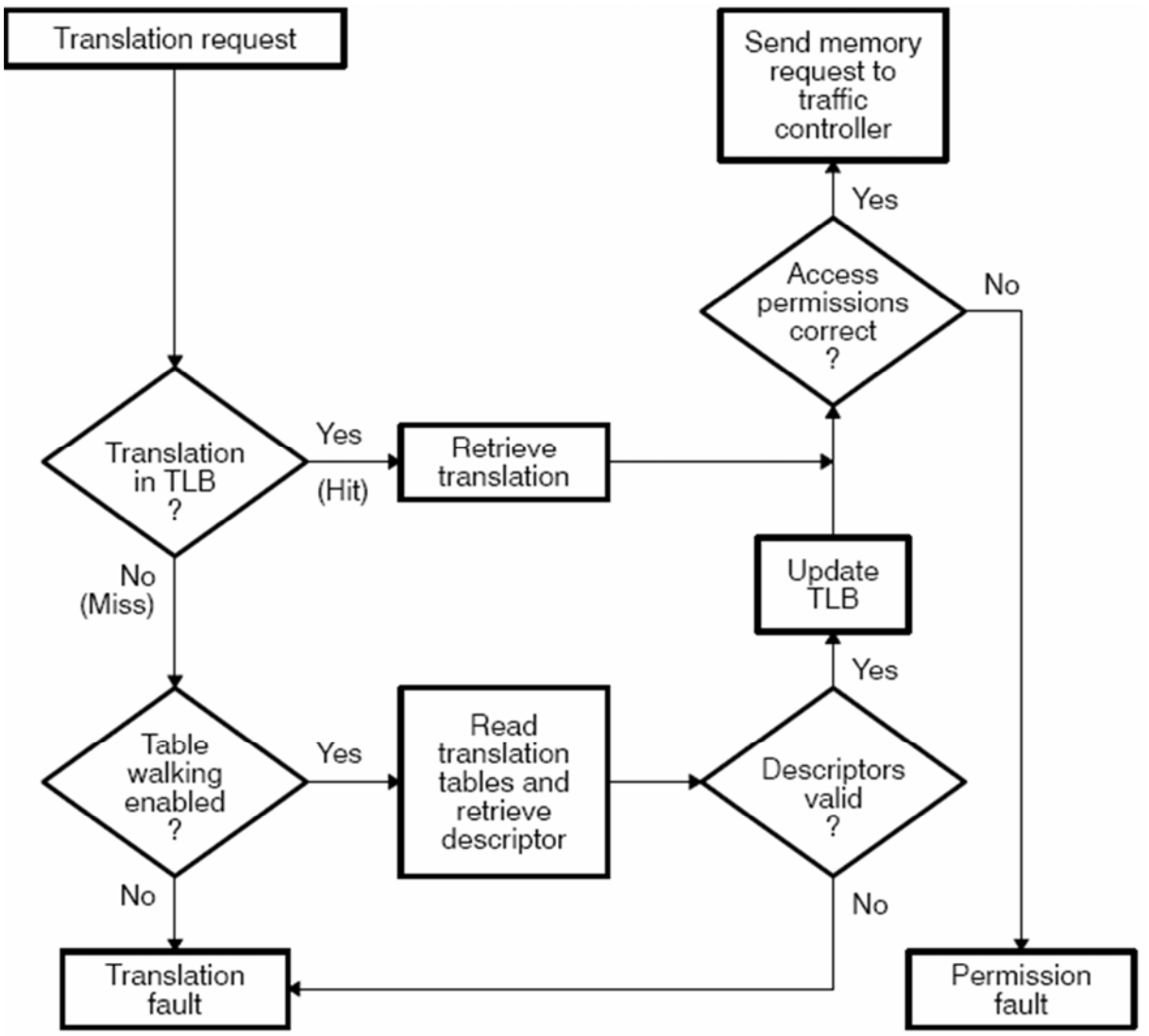


Fig. 18 MMU Translation Process

When the requested translation is not in the TLB, the table walking logic (if enabled) retrieves the translation by reading a set of translation tables. If the table walking logic is disabled, the MMU generates a fault interrupt to the ARM core. When the table walking logic finds a valid translation, it updates the TLB and, if the access permissions are correct, the corresponding physical address is calculated and the memory request is sent to the traffic controller. If the request does not have the correct permissions, or if no valid translation is found in the translation tables, then the MMU generates a fault interrupt to the ARM core. Fig. 18 summarizes the entire DSP MMU translation process.

4 Implementation and Optimization

4.1 Whole System implementation

The proposed system is based on a processor employing a dual-core (DSP and ARM) architecture as in Fig. 20. The H.264 encoder is implemented by the DSP core with high optimization, and the ARM core performs bitstream buffering and system control. The system with this configuration can support an efficient parallel processing between the DSP and the ARM cores, so that it is possible to minimize the computational overhead. The implemented system employs NFS or CF card for the storage or uploads the encoded bitstream during network applications. It supports Baseline profile in H.264 standards. The H.264 encoder developed in this work is suitable for portable communication devices.

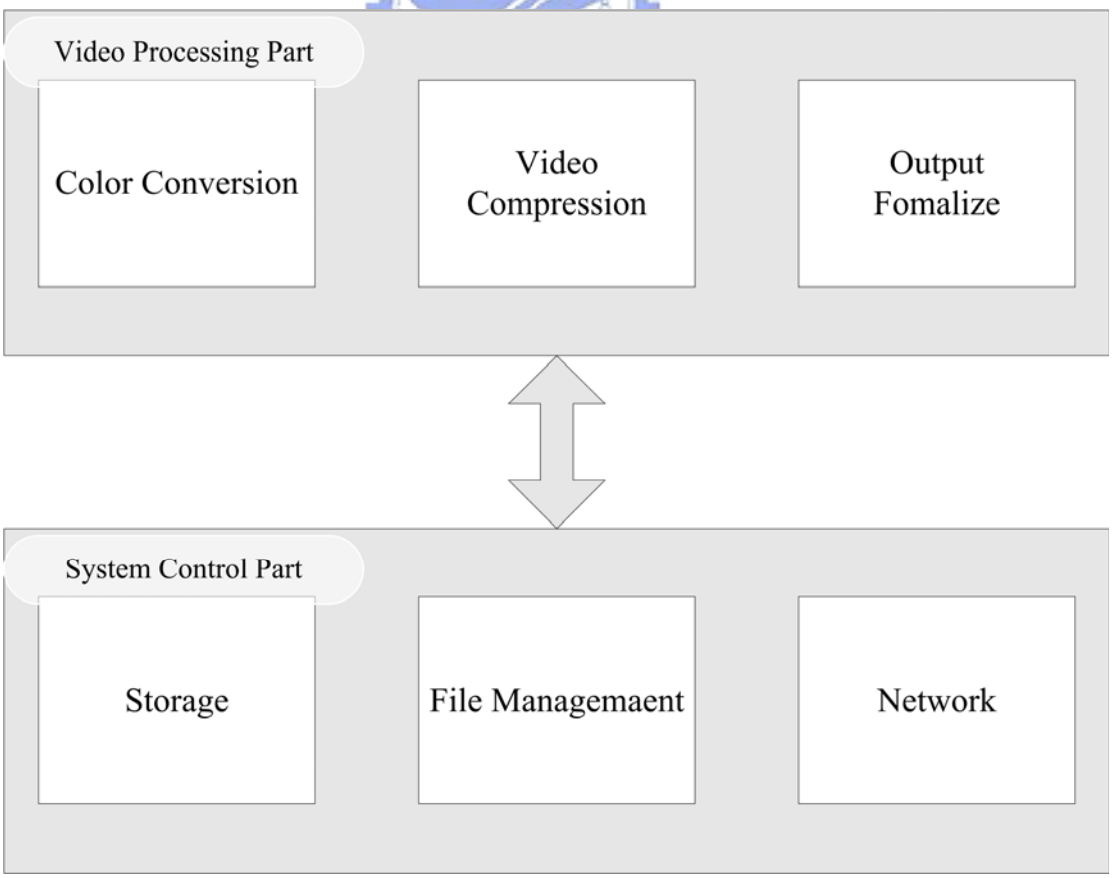


Fig. 19 Architecture of H.264 Encoder

A major benefit of using a DSP-ARM dual-core processor is that it is possible to integrate system control functionalities with the H.264 encoding on a single processor. However, more benefits of significance can be obtained if DSP and ARM cores are programmed to execute signal processing jobs in parallel. A single programmable DSP may be sufficient for implementing core routines of the H.264 encoding algorithm. However, when a complete H.264 encoding system is considered, we may need an additional ARM for the bitstream fetch and system I/O control. Issues in such case are focused on how to integrate multiple processors on a singe system with minimum overhead in power consumption and system cost. The alternative to the use of DSP and ARM combination is to use DSP with ARM extension, ARM with DSP extension, or only ARM. Especially, a dual-core processor comprising a DSP and an ARM cores is beneficial for the implementation of H.264 encoding system with low-power.

The developed H.264 encoding system consists of a video compression processing part and a system control part, as shown in Fig. 20. In the video compression processing part, the encoded data is encoded via the H.264 encoding algorithm and converted to a playable file such as MKV file. The system control part is in charge of managing the encoded data files and transferring the files from the storage to video compression processing part. In video processing part, main concerns are about resource usage such as less computational complexity and lower memory requirement.

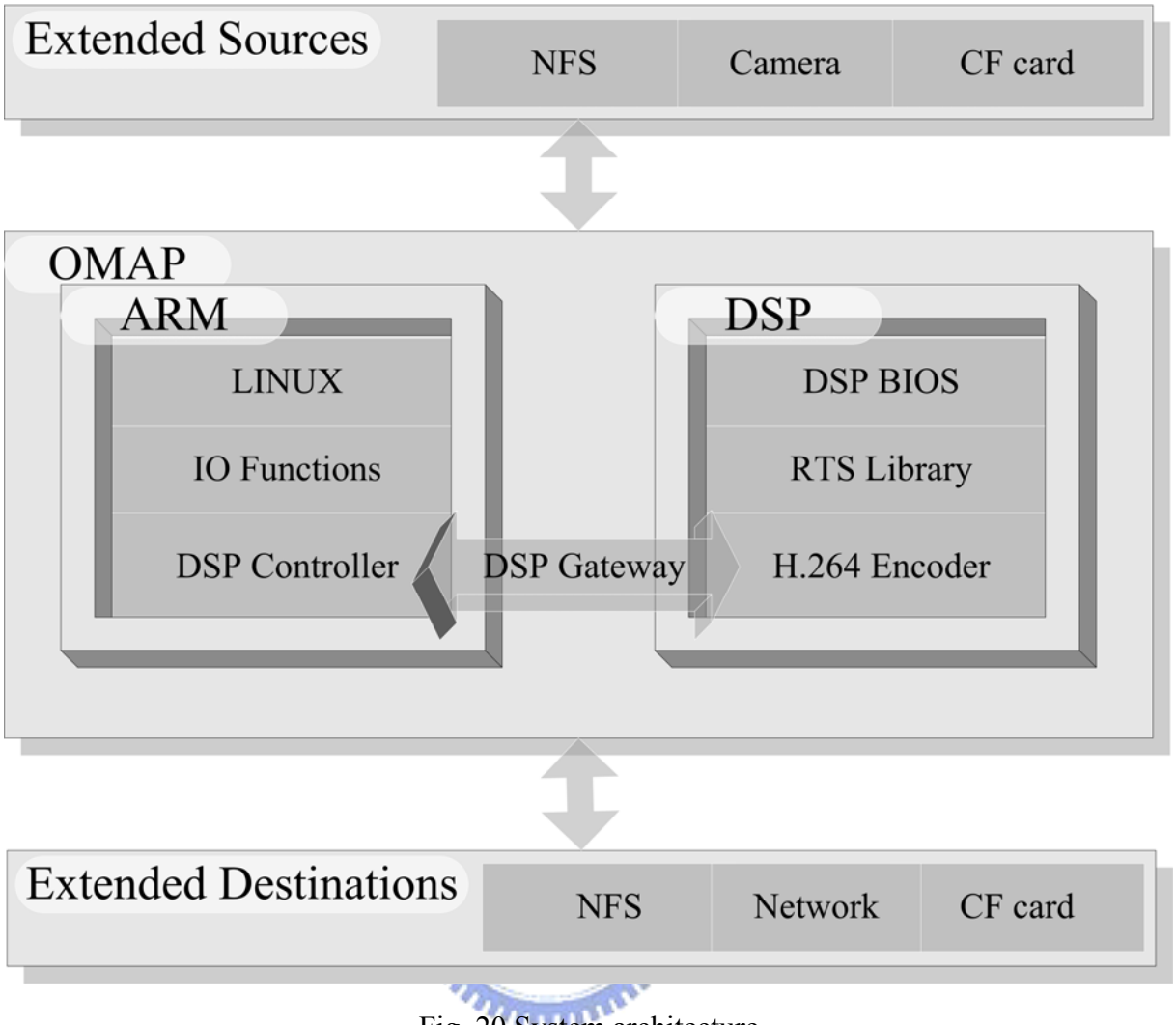


Fig. 20 System architecture

In Fig. 20, this system enables the parallel processing by properly allocating functional jobs for video processing and the system control into DSP and ARM.

The H.264 encoding algorithm is implemented by DSP with its excellent arithmetic features, and H.264 frame buffering, file management, and System I/O are performed with the ARM core.

4.2 The Proposed H.264 encoder

The proposed encoder is developed based on the open source x264 encoder. The reason, why we choose it as the basis, is its excellent performance. Although, this is not a good encoder for embedded devices, the computational complexity and the memory usage are huge, and its target platform and program flow is designed for X86. Therefore, a whole program optimization should be done, and the device feature must be involved to the design.

Optimization techniques employed to reduce the computational complexity of modules are grouped into three categories: platform-independent algorithmic optimizations, platform-based DSP-enhanced optimizations and platform-based memory optimizations.

- Algorithmic optimizations

  To reduce the complexity of H.264 encoder, algorithms are modified in high level language. Apart from implementing optimal algorithms, optimization techniques like loop unrolling, loop distribution and loop interchange are used. Algorithmic optimizations presented below are platform independent and can be used on any platform.

- DSP enhanced optimizations

  OMAP takes advantage in processing video signal. With the DSP core, some applications are enhanced to perform better. A lot of platform-based optimizations, such as hardware loops, intrinsics, and the dual MAC, are performed for implementation of H.264 encoder.

- Memory optimizations

  As the memory requirement of encoder is significant, many cycles are spent by DSP core while waiting for that ARM moves data from external memory to DSP memory. In this encoder, the data is moved in an efficient way. Due to the dual-core feature, much time is saved in parallel processing.

4.3 Reduce DSP Computational Complexity

4.3.1 Rearrange x264 encoding control flow

The original x264 control flow is straight and greed. It follows a well-concerned strategy that is trying all possible block modes to produce better quality of the encoded stream. The overall encode control flow is shown in Fig. 21. When a macroblock is ready to be coded, selection of INTER or INTRA mode is depends on the frame type of this macroblock. In INTRA mode, it will be tested in different block size (4 x 4, 8 x 8, and 16 x 16) to find the least SAD of it. For each block size, every block will be subtracted from the reference block in different directions determined by tits location to produce the SAD. For instance, if a macroblock is predicted in intra mode, it will be separated to sixteen 4 x 4 blocks. The intra prediction repeats up to 4 times. In each intra prediction, a reference block in one direction is read from the encoded frame and subtracted from the current block. Each 4 x 4 block has 1 to 4 SADs, and every SAD is stored in the arrays. Repeat the above steps, 8 x 8 and 16 x 16 block sizes are used for intra prediction again an d again. Finally a specified block size with least SAD in the corresponding direction is determined. In INTER mode, it will be participated in different size (4 x 4, 4 x 8, 8 x 4, 8 x 8, 16 x 8, 8 x 16, and 16 x 16) for motion-compensated estimation, and all SAD in responding to each partition will be found out. Like intra prediction, inter prediction is executed several times, but only one SAD is found for one block size. Although in general speaking, the block size selection is choosing the block with the least SAD, and then the block is encoded in the selected size, it consumes much time in calculating different SADs. Most of the computed SAD is useless for the next block or the next frame.

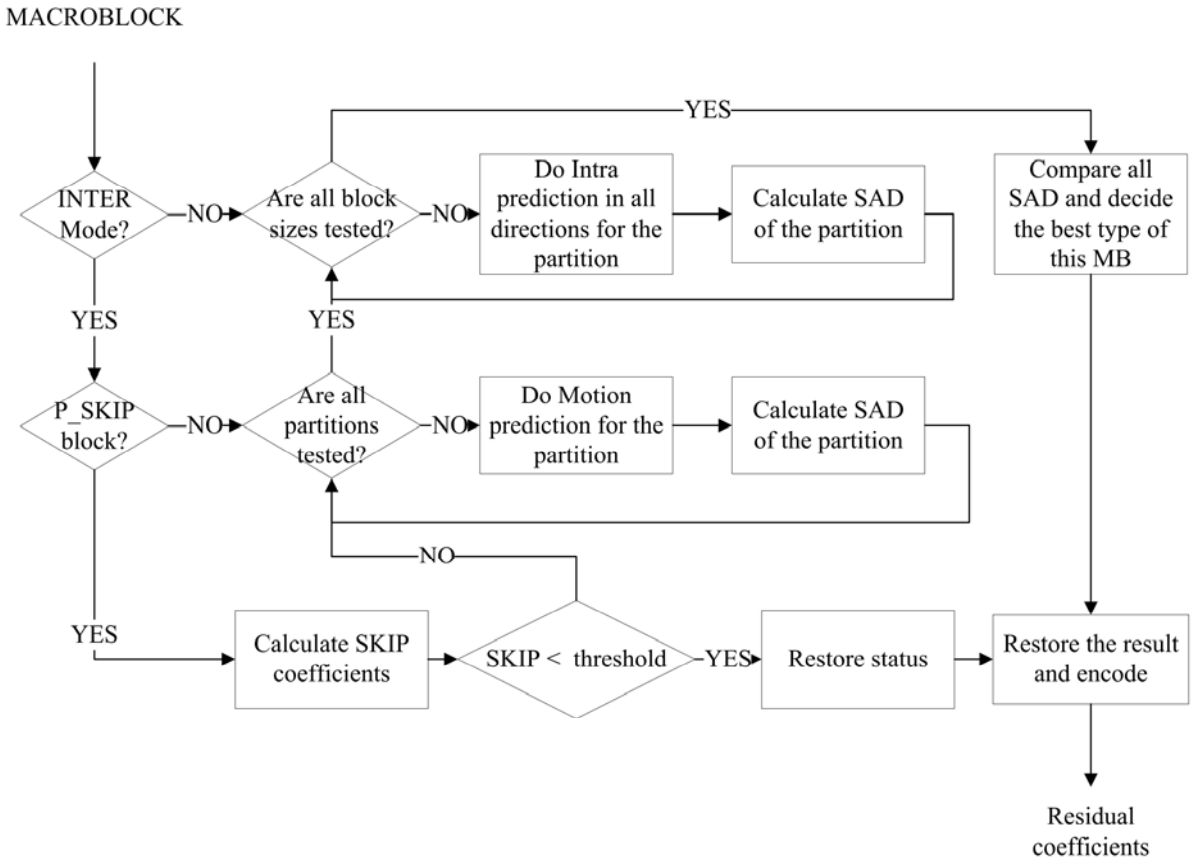


Fig. 21 The original x264 encode control flow

For purpose of accelerating encode speed, processing less modes will decrease more used time. By the way, in the intra predictions loop and the inter predictions loop, the threshold is set for each loop to terminate the loop earlier. The flow is shown in Fig. 22. In theoretical speaking, it looks like a good method to reduce computational complexity. The intra predictions loop and inter predictions loop will be terminated if the calculated SAD is less then the dynamic threshold, but in real cases the SAD differs a lot in different sources. The I_threshold and P_threshold corresponding to various sources are difficult to be decided. More overheads are needed to calculate the thresholds. In spite of determining the threshold, block size decision, a faster method can be used as illustrated in Fig. 23.

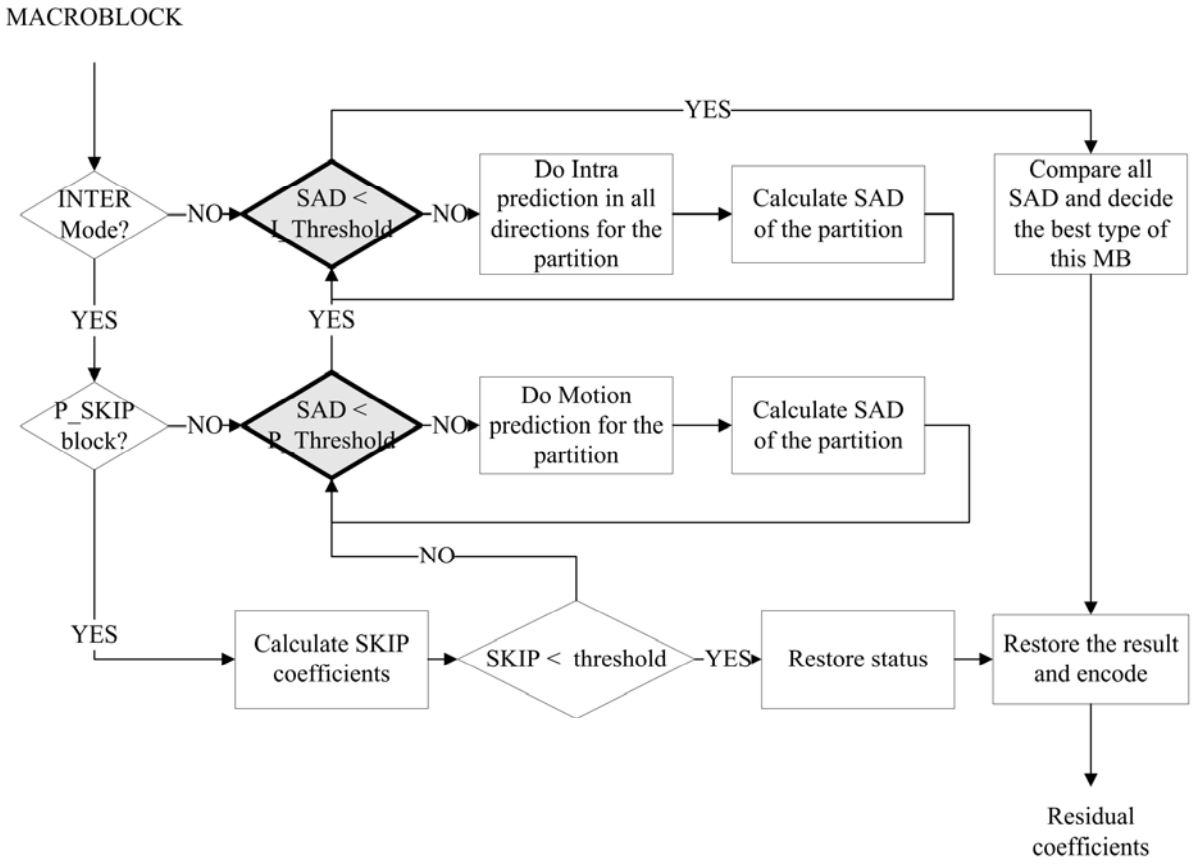


Fig. 22 The proposed x264 encode control flow with the threshold

Moreover, the DSP core has a useful feature, zero-overhead loop. If the loop codes are kept small enough to enable the compiler to make use of the native local repeat instruction, the compiler will generate local repeat for small loops that do not contain any control flow structures other than forward conditionals. Local repeat loops consume less power than other looping constructs. In addition, to generate a hardware loop, the compiler would need to emit code that would determine the number of loop iterations at run time. This code would require an integer division. Since this is computationally expensive, the compiler will not generate such code and will not generate a hardware loop. By the way, the block size is fixed and the intra predictions loop and the inter predictions loop are canceled, because the block decision loop can compiled to hardware loop, and more overhead will consumed here. The optimized control flow is summarizes in Fig. 23.

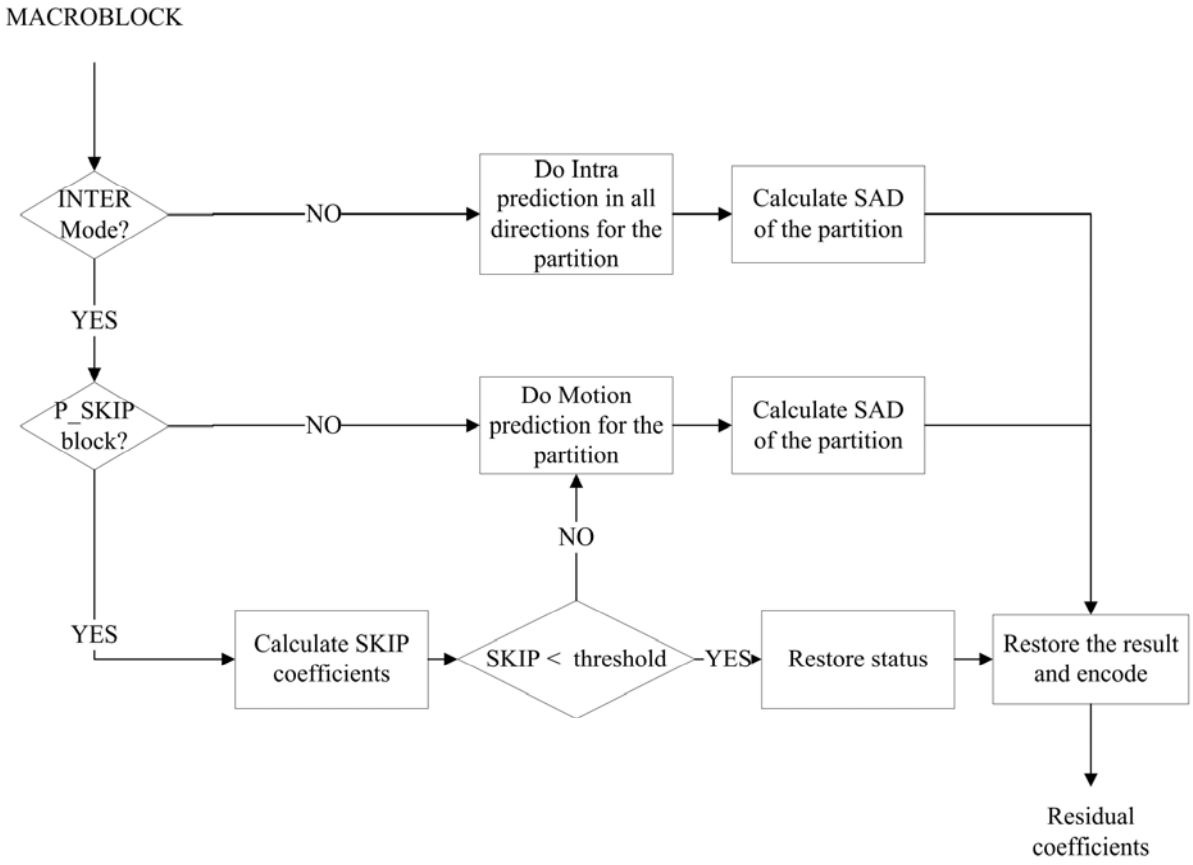


Fig. 23 The proposed x264 encode faster control flow

The proposed faster x264 control flow disables block decision in order to reduce computational complexity. However, this method will decrease PSNR or CR gain. If the RD strategy focuses on bitrate, PSNR will decrease for the specified bitrate. In other words, CR gain will decrease for better quality. Skip 4 x 4 block predictions can save 80% computing power.

4.3.2 Maximize DSP Compiler's Performance

Due to the TMS320C55X DSP architecture, some features of the C language are relevant to compilation on the C55x DSP core, performance-enhancing options for the compiler, and C55x-specific code transformations that improve C code performance. The DSP-enhanced H.264 encoder is modified by using these features. The following features are used in this encoder.

- Create loops that efficiently use C55x hardware loops.

  Whenever possible avoid using function calls within loops, because repeat labels and counts would have to be preserved across calls, and the compiler will decide never to generate hardware loops that contain function calls. This leads to inefficient loop code.

- Use intrinsics to replace complicated C/C++ code.

  The C55x compiler provides intrinsics, special functions that map directly to inlined C55x instructions, to optimize your C code quickly. Intrinsics are specified with a leading underscore "_" and are accessed by calling them as a function call.

- Use long accesses to reference 16-bit data in memory.

  The primary use of treating 16-bit data as `long` is to transfer data quickly from one memory location to another. Since 32-bit accesses also can occur in a single cycle, this could reduce the data-movement time by half. The only limitation is that the data must be aligned on a double word boundary (that is, an even word boundary). The code is even simpler if the number of items transferred is a multiple of 2.

- Write efficient control code.

Control code typically tests a number of conditions to determine the appropriate action to take. The compiler generates similar constructs when implementing nested if-then else and switch/case constructs when the number of case labels is fewer than eight. Because the first true condition is executed with the least amount of branching, it is best to allocate the most often executed conditional first. When the number of case labels exceeds eight, the compiler generates a .switch label section. In this case, it is still optimal to place the most often executed code at the first case label.

In the case of single conditionals, it is best to test against zero. For example, consider the following piece of C code:

```
if (a!=1) /* Test against 1 */
    <inst1>
else
    <inst2>
```

If the programmer knows that "a" is always 0 or 1, the following more efficient C code can be used:

```
if (a==0) /* Test against 0 */
    <inst1>
else
    <inst2>
```

In most cases, this test against zero will result in more efficient compiled code.

When the code is refined in an efficient style, the compiler options are enabled for optimally compiling. For each C source file, the correct options are specified so that the better performance is achieved.

4.3.3 Optimize x264 encoding data path

This H.264 encoder is developed in a dual-core platform. Fig. 24 shows a simple co-work flow. The original flow works in a sequential way. A frame is read in memory and sent to DSP by ARM. When DSP processes this frame, ARM is waiting for the signal for DSP and doing nothing.

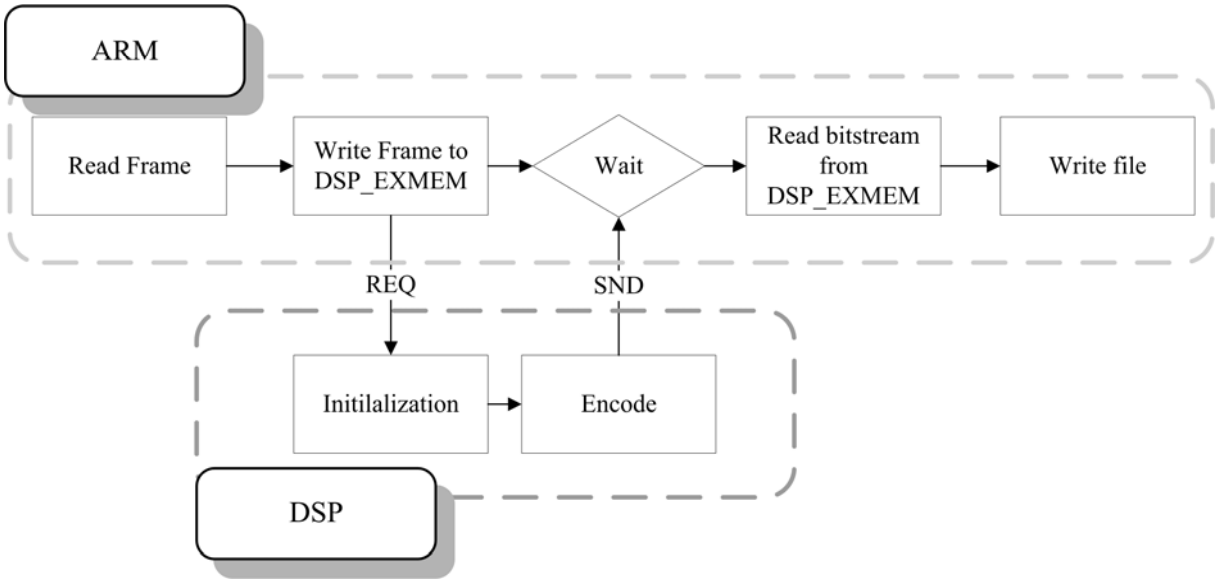


Fig. 24 The ported original x264 encoding data path

Fig. 25 describes the efficient cross works between the ARM and the DSP cores. As can be seen from the figure, two cores operate in parallel when the H.264 encoding function is enabled. At first, when the function is called, the ARM core initializes the system, reads frames in the storage devices and then writes into the DSP exmem. When the source is ready, the ARM core activates the H.264 encoding routine of the DSP core, and then the DSP core inquires the ARM core of next frame buffering whenever the side information of fame is processed. At the same time, ARM core performs frame buffering in parallel to the DSP. After encoding of the current proceeding frame, the encoding routine restores the necessary information and transited to the next frame. In this configuration, ARM core performs a part

of the H.264 encoding process in addition to system control. In should be noted that ARM generally has better architecture for the frame buffering than DSP. In the system view, ARM is the master core, and DSP is the slave component. Thus, it is possible to increase the efficiency of the overall system.

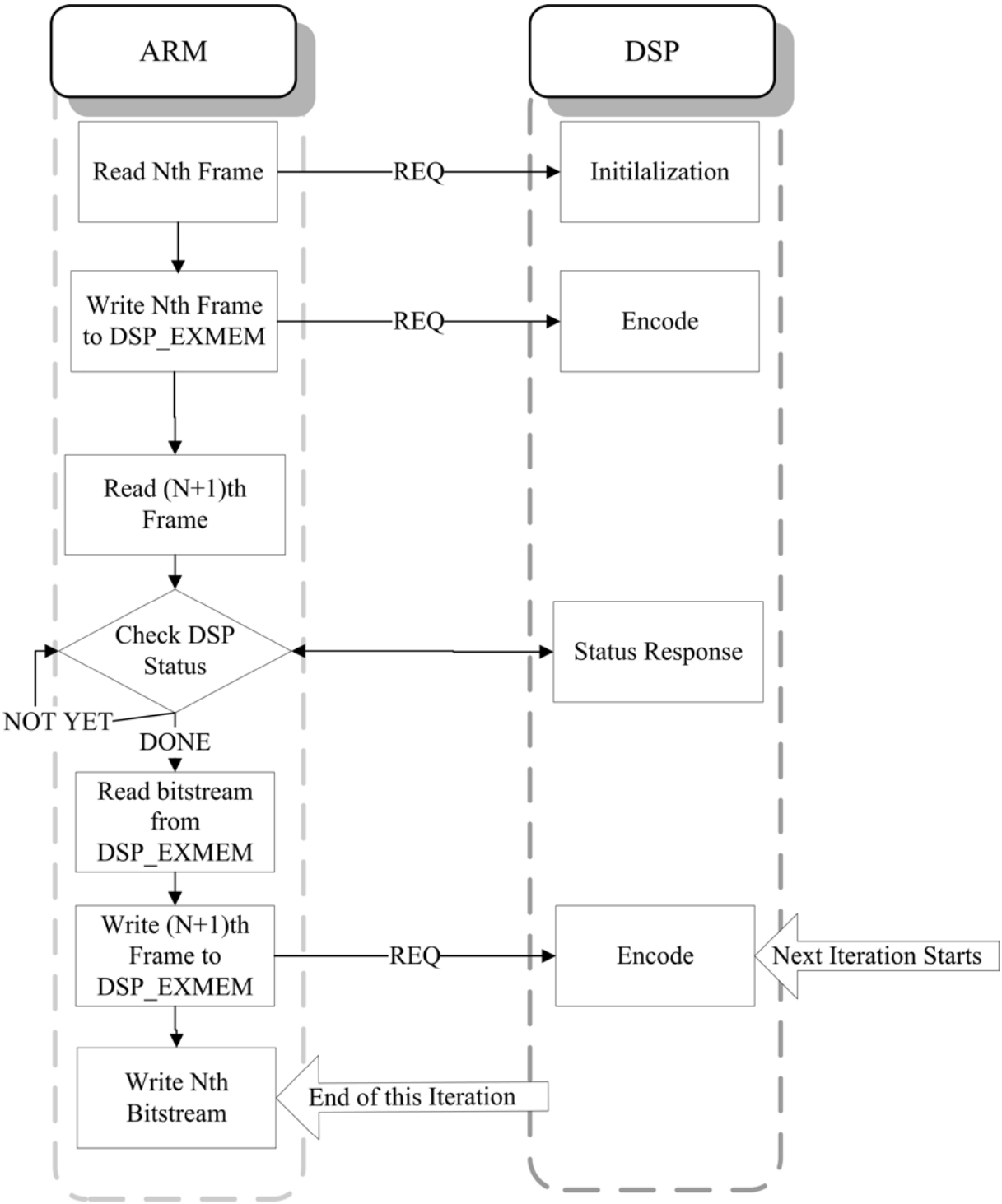


Fig. 25 The proposed x264 encoding data path

4.4 DSP subsystem Memory Management

4.4.1 Issues

In DSP programming, memory is limited and expensive. When the designers construct programs on PC, memory requirement is not a big problem. However, in an embedded system, memory allocation may consumes the most time in programming. A lot of restrictions will be concerned in C55X DSP programming. A video coder requires much space for storing the reference frames and other temporary use, so the internal memory is not enough for video coding. By the way, the following four steps are needed for construct the H.264 encoder.

1. Verify the memory usage in the whole encoder and allocate global buffer for temporary use.
2. Put the frequently accessed part to the internal ram and the others to the external ram.
3. Map OMAP SDRAM to DSP memory space for being DSP external memory by configuring DSP MMU.
4. Allocate share memory for ARM and DSP communication.

4.4.2 Memory Usage

While a video encoder runs, many parameters are stored for current or next frames. They may be previous block SADs, memory address of the frame buffer, RD coefficients, or etc. A global encoder handler is needed to manage these parameters. In x264 source code, it defines several structure to save these parameters. Despite the memory can be allocated and released in different places during running, embedded processor can not offer too much memory overhead. In spite of allocating memory dynamically, pre-allocating all necessary memory is

a better strategy.

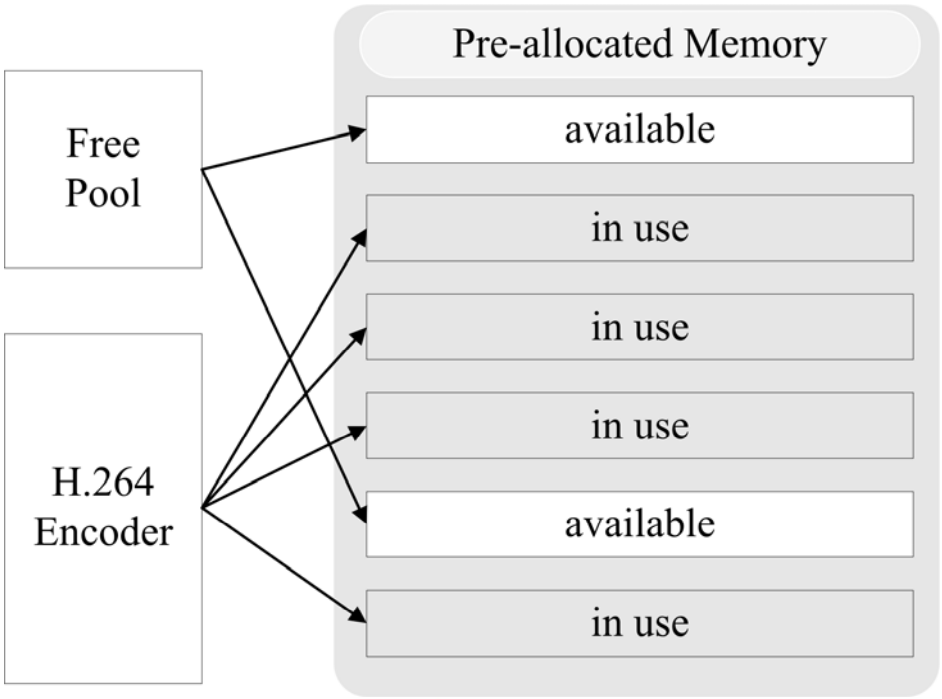


Fig. 26 The memory pool system

Memory pool is a technique that uses dynamic memory allocation comparable to malloc, but dose not run-time allocates memory. As the implementation, malloc suffers from fragmentation because of variable structure sizes; it is used in a real time system due to performance. A more efficient solution is pre-allocating a number of memory blocks called the memory pool with the same size. The application can allocate, access and free blocks represented by handles at runtime.

Memory pool allows memory allocation with constant execution time (no fragmentation). The memory can release thousands of objects in a pool in one operation, not one by one if one uses malloc to allocate memory for each object. The memory pools can be grouped in hierarchical tree structures, which is suitable for special programming structures like loops and recursions. On the other hand, they need to be tuned for the application which deploys them. Therefore, a structure named encoder_handler is build that includes encoding information and memory pointers which point to memory pre-allocated blocks as shown in Fig. 26.

4.4.3 Memory-mapped share buffer

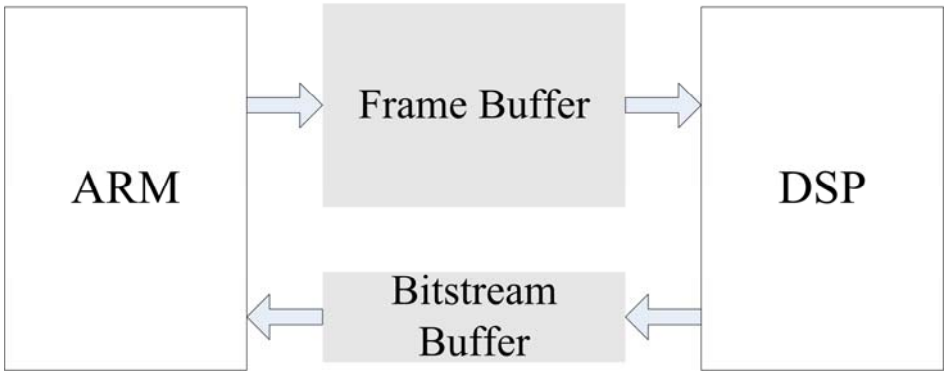


Fig. 27 The shared memory

As mentioned in inter-processor data path, it is needed to design a global buffer between ARM and DSP by the "ioctl" function, exmap. (See Fig. 27) ARM core writes frames to the frame buffer, DSP core reads and processes them. After compression, DSP core writes H.264 bitstream to the bitstream buffer.

Except mapping memory space in Linux, the external memory buffer should be declared with the DATA_SECTION pragma in the C source code and assigned to the specified external address in the Linker command file.

In the C code:

```
#pragma DATA_SECTION(frame_buffer, "frame_sect")
char frame_buffer[W*H];
```

In Linker command file

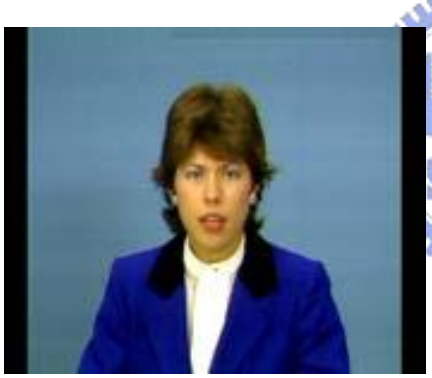
```
MEMORY {
   E_BUF:     origin = EXT_ADDR,      len = SIZE
}
SECTIONS {
   frame_buffer() > E_BUF
}
```

5 Experimental Results

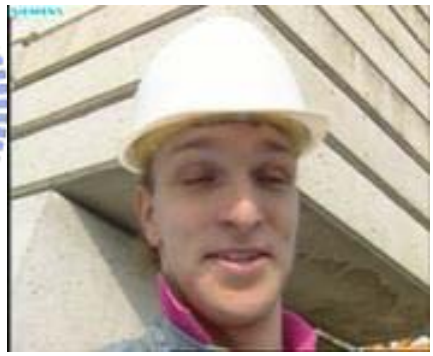
5.1 Experiment Conditions

- Test Environment

The optimized H.264 encoder is tested for different sequences representing typical video conferencing content. The sequences are encoded at QCIF resolution, 15 frames per second. The objective quality of the encoded sequences are measured in terms of peak signal to noise ratio (PSNR) and the processing power required is measured in DSP cycles



Claire



Foreman

Akiyo

Suzie

5.2 Experimental results of Intra frame coding

The following experiments show the simulated cycle counts and system processing performance of intra frame encoding.

In Table 4, the Quantization part and the CAVLC part costs the most execution time, the reason is their complex algorithm. As mentioned in Chapter 2, the Quantization part is different from MPEG4, H.263, or the former standards; it's more complicated because the DCT part is simplified to a kind of integer transform. Therefore, the quantization step is involved in the context and uses more time to execution.

Table 4 Execution Cycles/s of encoding intra frame

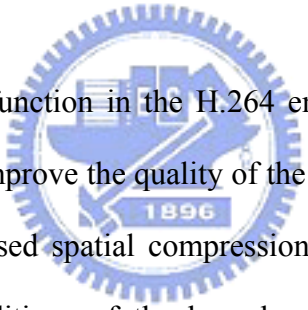
Test Sequences	Claire	Foreman	Akiyo	Suzie
Intra Prediction	1,602,046	1,602,046	1,602,046	1,602,046
Textual Transform	2,901,799	2,901,799	2,901,799	2,901,799
Quantization	5,861,333	5,853,513	5,854,843	5,855,563
Inverse Quantization	2,820,718	2,820,718	2,820,718	2,820,718
Inverse Transform	3,089,572	3,089,572	3,089,572	3,089,572
Deblocking Filter	3,280,098	2,705,551	2,793,040	2,845,548
CAVLC	4,407,618	6,937,364	7,041,757	5,027,914
Packing NAL	127,663	200,173	207,270	137,919
Total	24,090,847	26,110,735	26,311,045	24,281,079

The CAVLC module, discussed in Chapter 2, spends the most time in encoding intra frames. The CAVLC results 4 different VLC tables and a total of about 450 codewords and accounts for about 20% as in Table 5 of the computation complexity of the encoder. For example, it takes more than 30 times to read and match codewords for encoding Coeff_token information when the length of the codewords exceeds 10. Because the CAVLC consists of a kind of bit-level operations, general processor (like RISC or MIPS) and DSP incorporating

multiple parallel arithmetic units (like SIMD or VLIW) are ineffective to encode it. Therefore, the CAVLC is the bottleneck of encoding intra frames.

Table 5 Percentage of execution time of encoding intra frame

Test Sequences	Claire	Foreman	Akiyo	Suzie
Intra Prediction	6.65	6.14	6.09	6.59
Textual Transform	12.04	11.11	11.03	11.95
Quantization	24.33	22.42	22.25	24.12
Inverse Quantization	11.71	10.80	10.72	11.62
Inverse Transform	12.82	11.83	11.74	12.72
Deblocking Filter	13.62	10.36	10.62	11.72
CAVLC	18.30	26.57	26.76	20.70
Packing NAL	0.53	0.77	0.79	0.56
Total	100	100	100	100



The in-loop, deblocking function in the H.264 encoder is a filtering process of a 4x4 reconstructed image block to improve the quality of the reconstructed picture by removing the blocky artifacts from block-based spatial compression. The choice of filtering process and outcome depends on the conditions of the boundary strength and the gradient of image samples across the 4x4 block boundary. There are three types (3, 4, 5 taps) of filters for both vertical and horizontal directions and the filter decision is based on the combinations of conditions including boundary strengths, image gradients and thresholds that are affected by the average quantization parameters across the block boundary. These conditional processes do not fit well into the DSP architecture resulting in very inefficient implementation of the deblocking filtering.

5.3 Experimental results of Inter frame coding

The following experiments show the simulated cycle counts and system processing performance of inter frame encoding. In general speaking, encoding inter frames is much slower (more 50% cycles) than encoding intra ones because motion estimation takes a lot of time.

Table 6 Execution cycles of encoding inter frame

Test Sequences	Claire	Foreman	Akiyo	Suzie
Motion Estimation	5,913,882	6,715,035	5,151,828	6,700,669
Motion Compensation	894,530	851,409	894,133	1,198,985
Textual Transform	1,451,358	2,708,291	1,422,097	2,007,322
Quantization	2,932,435	5,465,830	2,870,132	4,052,336
Inverse Quantization	1,417,945	2,645,941	1,389,357	1,961,109
Inverse Transform	1,012,036	2,624,673	1,370,521	1,801,887
Deblocking Filter	2,263,972	2,403,123	2,309,712	2,523,592
CAVLC	615,406	2,065,991	1,458,069	934,545
Packing NAL	15,488	51,388	37,924	22,940
Total	16,517,052	25,531,682	16,903,773	21,203,385

In Table 6 and Table 7, because of the optimizations discussed in Chapter 4, the used time of encoding inter frames is less than intra ones. This is a very important key to the whole system performance, because, in real cases, the frame period is lager than 150; in other words, every 150 inter frames there is one intra frame. With this feature, the proposed encoder can compress frames faster.

Table 7 Percentage of execution time of encoding inter frame

Test Sequences	Claire	Foreman	Akiyo	Suzie
Motion Estimation	35.80	26.30	30.48	31.60
Motion Compensation	5.41	3.33	5.29	5.65
Textual Transform	8.79	10.61	8.41	9.47
Quantization	17.75	21.41	16.98	19.11
Inverse Quantization	8.58	10.36	8.22	9.25
Inverse Transform	6.12	10.28	8.10	8.49
Deblocking Filter	13.71	9.41	13.66	11.90
CAVLC	3.73	8.09	8.63	4.41
Packing NAL	0.09	0.20	0.22	0.11
Total	100	100	100	100

Motion estimation constitutes one of the most critical parts for several reasons: First, estimating the motion vectors for each image block requires a high computational load which usually sums up to half of the overall complexity of the encoding process. Second, the reliability of motion estimation has a big influence on the encoder performance in terms of picture quality at a given bit rate. And finally, video coding standards do not dictate the way in which motion vectors are extracted from the video material. Thus, there is considerable freedom in optimizing the video quality by selecting a suitable motion estimation scheme. Instead of using a full search scheme for motion estimation, the much less computationally diamond search method is used. This simplified search scheme performs better than traditional methods.

5.4 System performance

The system performance including data read and memory movement is listed in Table 8 and around 50% power is wasted in system I/O and task switching. However, it is necessary, because the ARM core is running under Linux OS, and each process must be handled by the scheduler. There are a lot of exceptions, such as CF card I/O, TCP/IP interrupt, and etc., such that the DSP program is affected in these cases.

Table 8 System performance of the proposed encoder

Test Sequences	Claire	Foreman	News	Suzie
Simulated Cycles	16,567,209	25,535,516	16,966,072	21,223,767
FPS	5.5	4.9	5.4	5.2

In Table 9 , the performance of PSNR is better than the architecture executed on the ADI BF561, but the needed cycles are more than it. The Blackfin is the world's first family of DSPs to integrate the high-performance Micro Signal Architecture (MSA) jointly developed by Intel Corporation and Analog Devices. The Blackfin core features a dual datapath modified Harvard architecture and is optimized for both DSP and micro-controller functions. By the report from BDTI (listed in [27]) the C55 DSP is a power efficient DSP, its speed performance is lower than the ADI Blackfin BF5xx series.

Table 9 Comparison of PSNR measured in dB

	Test Sequences	Claire	Foreman	News	Suzie
Proposed	96 Kbps	44.39	33.90	44.89	38.83
	128 Kbps	45.76	35.10	46.91	39.43
Kant[26]	96 Kbps	43.95	33.21	42.00	38.87
	128 Kbps	44.83	34.23	43.64	39.99

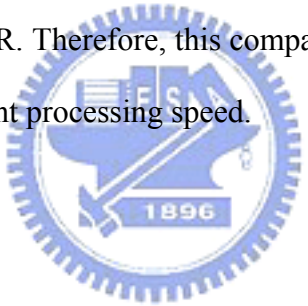
Table 10 Comparison of computation time measured in cycles

Test Sequences	Claire	Foreman	News	Suzie
Proposed	16,567,209	25,535,516	16,966,072	21,223,767
Kant[26]	2,550,000	3,600,000	2,580,000	3,170,000

Table 11 Comparison of adjusted computation time measured in cycles

Test Sequences	Claire	Foreman	News	Suzie
Proposed	3,084,110	4,753,628	3,158,361	3,950,963
Kant[26]	2,550,000	3,600,000	2,580,000	3,170,000

From the BDTI's report, the C55x DSP gets scores in range of 780 to 1460, and the ADI Blackfin gets scores in range of 1680 to 4190. If the Kant's performance is scaled with the BDTI's report as illustrated in Table 11, the proposed performance is close to theirs but gets better quality measured in PSNR. Therefore, this comparison shows the proposed encoder has good picture quality and efficient processing speed.



6 Conclusion

In this thesis, optimization techniques employed to reduce the computational complexity of modules are grouped into two categories: platform-independent algorithmic optimizations, platform-based DSP-enhanced optimizations and platform-based memory optimizations. Algorithms are modified in high level language. Apart from implementing optimal algorithms, optimization techniques like loop unrolling, loop distribution and loop interchange are used. Algorithmic optimizations are platform independent and can be used on any platform. In system view, a lot of platform-based optimizations are performed for implementation of H.264 encoder. The data is moved in an efficient way. Due to the dual-core feature, much time is saved in parallel processing.

H.264 achieves the best-ever compression efficiency[28] for a broad range of applications, such as broadcast, DVD, video conferencing, video-on-demand, streaming and multimedia messaging. And true to its advanced design, H.264 delivers excellent quality across a wide operating range, from 3G to HD and everything in between. Whether you need high-quality video for your mobile phone, iChat, internet streaming, broadcast or satellite delivery, H.264 provides exceptional performance at impressively low data rates. This thesis summarizes dual-core implementation of H.264 encoder on OMAP5912. The implementation is well suited for videophone, and network streaming applications. The DSP core can perform encoding while other can process network message as well as store the encoded H.264 stream in local host. The optimization techniques presented in the present thesis can be effectively used for other programmable processors with similar architecture and instruction set.

7 Bibliography

[1] ISO/IEC 14496–10:2003, “Coding of Audiovisual Objects—Part 10: Advanced Video Coding,” 2003, also ITU-T Recommendation H.264“Advanced video coding for generic audiovisual services.”

[2] ISO/IEC 11172: “Information technology—coding of moving picture sand associated audio for digital storage media at up to about 1.5 Mbit/s,” Geneva, 1993.

[3] ISO/IEC 13818–2: “Generic coding of moving pictures and associated audio information—Part 2: Video,” 1994, also ITU-T Recommendation H.262.

[4] ITU-T Recommendation H.263, “Video Coding for Low bit rate Communication” version 1, Nov. 1995; version 2, Jan. 1998; version 3, Nov. 2000.

[5] ISO/IEC 14496–2: “Information technology—coding of audiovisualobjects—part 2: visual,” Geneva, 2000.

[6] http://www.hddvdprg.com/

[7] http://www.blu-raydisc.com/

[8] http://www.dvb.org/

[9] http://www.3gpp.org/

[10] http://focus.ti.com/docs/prod/folders/print/omap5912.html

[11] http://focus.ti.com/docs/prod/folders/print/tms320vc5510a.html

[12] http://www.arm.com/

[13] http://www.linux.org/

[14] http://dspgateway.sourceforge.net/pub/index.php

[15] ISO/IEC 14496–10:2003, Annex A of “Coding of Audiovisual Objects—Part 10: Advanced Video Coding,” 2003, also ITU-T Recommendation H.264“Advanced video coding for generic audiovisual services.”

[16] D. Marpe, H. Schwarz, and T. Wiegand, “Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 620–636, July 2003.

[17] H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, “Low-Complexity transform and quantization in H.264/AVC,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 598–603, July 2003.

[18] T. Wedi and H.G. Musmann, “Motion- and aliasing-compensated prediction for hybrid video coding,” *IEEE Transactions on Circuits and Systemsfor Video Technology*, vol. 13, pp. 577–587, July 2003.

[19] T. Wiegand, X. Zhang, and B. Girod, “Long-term memory motion-compensated prediction for video coding,” *IEEE Trans. Circuits Syst. VideoTechnol.*, vol. 9, pp. 70–84,

Feb. 1999.

[20] T. Wiegand, H. Schwarz, A. Joch, and F. Kossentini, “Rate-constraine dcoder control and comparison of video coding standards,” *IEEETransactions on Circuits and Systems for Video Technology*, vol. 13, pp.688–703, July 2003.

[21] B. Girod, “Efficiency analysis of multihypothesis motion-compensated prediction for video coding,” *IEEE Trans. Image Processing*, vol. 9, Feb. 1999.

[22] P. List, A. Joch, J. Lainema, and G. Bjontegaard, “Adaptive deblocking filter” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 614–619, July 2003.

[23] http://www.arm.linux.org.uk/

[24] http://iphome.hhi.de/suehring/tml/download

[25] http://en.wikipedia.org/wiki/X264

[26] Kant, S. Mithun and U. Gupta, “Real time H.264 video encoder implementation on a programmable DSP processor for videophone applications”**,** ICCE '06 Digest of Technical Papers, pp. 93 – 94, Jan. 2006

[27] http://www.bdti.com/bdtimark/BDTImark2000.htm

[28] Ralf Schäfer, Thomas Wiegand, and Heiko Schwarz, “The emerging H.264/AVC standard; EBU Technical Review”, *Special Issue on “Best of 2003"*, January 2003.