İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

IMPLEMENTATION AND OPTIMIZATON OF REAL-TIME H.264 BASELINE ENCODER ON TMS320DM642 DSP

M.Sc. Thesis by

Ender MERİÇ, B.Sc.

Department : Computer Engineering

Programme: Computer Engineering

JUNE 2007

__İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY__

IMPLEMENTATION AND OPTIMIZATON OF REAL-TIME H.264 BASELINE ENCODER ON TMS320DM642 DSP

M.Sc. Thesis by

Ender MERİÇ, B.Sc.

(504041511)

Date of submission : 7 May 2007

Date of defence examination: 13 June 2007

Supervisor (Chairman): Prof. Dr. A. Coşkun SÖNMEZ

Members of the Examining Committee Asst. Prof. Dr. Berk ÜSTÜNDAĞ

Asst. Prof. Dr. Hasan F. ATEŞ (IÜ.)

JUNE 2007

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

GERÇEK ZAMANLI H.264 TEMEL KODLAYICININ TMS320DM642 DSP ÜZERİNDE UYGULAMASI VE ENİYİLEMESİ

YÜKSEK LİSANS TEZİ

Müh. Ender MERİÇ

(504041511)

Tezin Enstitüye Verildiği Tarih : 7 Mayıs 2007
Tezin Savunulduğu Tarih : 13 Haziran 2007

Tez Danışmanı :	**Prof. Dr. A. Coşkun SÖNMEZ**
Diğer Jüri Üyeleri	**Yrd. Doç. Dr. Berk ÜSTÜNDAĞ**
	Yrd. Doç. Dr. Hasan F. ATEŞ (IÜ.)

HAZİRAN 2007

FOREWORD

A Real-time H.264 baseline encoder realization and optimization on TMS320DM642 digital signal processor is aimed in this master thesis. I would like to thank my supervisor Prof. Dr. Coşkun SÖNMEZ who gave me a chance for the objective.

Next, I would like to thank to my family, İskender ERBAY, Mehmet GÜNEY, especially Vestek R&D Corp. for the moral and material supports, and also thanks to Asst. Prof. Dr. Hasan F. Ateş for his guidance and support in my M.Sc. thesis.

June 2007 Ender Meriç

CONTENTS

ABBREVIATOINS

AVC	: Advanced Video Coding
MPEG	: Motion Picture Experts Group
ISO/IEC	: International Organization of Standardization, International Electrotechnical Commission
ITU-T	: International Telecommunication Union, Telecommunications
JVT	: Joint Video Team
VCEG	: Video Coding Experts Group
VCL	: Video Coding Layer
NAL	: Network Adaptation Layer
NALU	: Network Adaptation Layer Unit
VLC	: Variable Length Coding
CAVLC	: Context-Adaptive Variable Length Coding
FMO	: Flexible Macroblock Ordering
ASO	: Arbitrary Slice Ordering
DSP	: Digital Signal Processor
RBSP	: Raw Byte Sequence Payload
SPS	: Sequence Parameter Set
PPS	: Picture Parameter Set
IDR	: Instantaneous Decoding Refresh
CPU	: Central Processing Unit
DSP	: Digital Signal Processor
VLIW	: Very Long Instruction Word
EDMA	: Enhanced Direct Memory Access
RAM	: Random Access Memory
LSB	: Least Significant Bit
MSB	: Most Significant Bit
TI	: Texas Instruments
CCS	: Code Composer Studio
IDE	: Integrated Development Environment
SAD	: Sum of Absolute Differences
SAE	: Sum of Absolute Energy
MB	: Macro Block
CIF	: Common Intermediate Format
QCIF	: Quarter Common Intermediate Format
HD	: High Definition

TABLE LIST

FIGURE LIST

IMPLEMENTATION AND OPTIMIZATION OF REAL-TIME H.264 BASELINE ENCODER ON TMS320DM642 DSP

SUMMARY

Recently, digital video coding is mandatory in many applications such as digital surveillance systems, video conferencing, mobile applications as well as video broadcasts. The H.264/MPEG-4 Part 10, an international video compression standard, is developed for improving the coding efficiency compared to previous standards. However, the coding improvement comes with an increase in coding complexity. In this thesis, an H.264 baseline profile encoder is implemented on Texas Instruments TMS320DM642 digital signal processor.

The real-time implementation of the H.264/AVC encoder on DM642 DSP core offers most of the standard H.264/AVC baseline profile coding tools except error resiliency tools and quarter-pel motion estimation. Instead of quarter-pel motion compensation, integer and half pixel position motion estimation and compensation for all luminance and chrominance components are implemented.

The target platform, DM64 DSP core, is designed as a high-performance digital media processor with two-level memory/cache hierarchy and very long instruction word (VLIW) architecture. The subject of the thesis is H.264 baseline encoder system realization and optimization on the target platform. Moreover, the study of optimization phases covering algorithmic, architectural and memory strategies are clarified in details.

The H.264/AVC encoder system is verified both to execute on the development workstation and DM642 EVM (Evaluation Module) hardware platform. Briefly, the uncompressed input of a YUV video sequence with CIF resolution to the encoder system is compressed to H.264 Annex-B file format and displayed on screen. Additionally, the encoder output is verified with H.264 reference software and the compliancy is proven.

GERÇEK ZAMANLI H.264 TEMEL KODLAYICININ TMS320DM642 DSP ÜZERİNDE UYGULAMASI VE ENİYİLEMESİ

ÖZET

Günümüzde sayısal video kodlama sayısal gözetim sistemleri, video konferans, mobil uygulamalar ve video yayını gibi bir çok uygulamada zorunlu hale gelmiştir. Uluslararası bir video sıkıştırma standardı olan H.264/MPEG-4 bölüm 10, daha önceki standartlara göre kodlama verimini iyileştirmek amacıyla geliştirilmiştir. Fakat, bu kodlama geliştirmesi beraberinde kodlama karmaşıklığının da artmasına yol açmaktadır. Bu tez çalışmasında Texas Instruments TMS320DM642 sayısal sinyal işleyici üzerinde H.264 temel profil kodlayıcı gerçeklenmiştir.

DM642 DSP çekirdeği üzerindeki gerçek zamanlı H.264/AVC kodlayıcı uygulaması hata esnekliği araçları ve çeyrek piksel hareket dengeleme dışında standart tüm H.264/AVC temel profil kodlama araçlarını sunmaktadır. Çeyrek piksel hareket dengelem yerine, tüm parlaklılık ve renklik bileşenleri için tam sayı ve yarım piksel pozisyonlarında hareket kestirim ve dengeleme gerçeklenmiştir.

Kullanılan DM642 DSP çekirdeği platformu, 2-seviyeli bellek/önbellek aşama düzenine sahip ve çok uzun komut kelimesi (VLIW) içeren yüksek performanslı sayısal işlemci olarak tasarlanmıştır. Sunulan H.264 temel kodlayıcı sistemin gerçeklenmesi ve eniyilemesi bu tezin konusudur. Üstelik, algoritma bazlı, mimari ve bellek stratejilerini içeren eniyileme çalışma fazları detaylarıyla açıklanmaktadır.

H.264/AVC video kodlayıcının hem geliştirme ortamında hem de DM642 EVM donanım ortamında çalışması doğrulanmıştır. Kısaca, kodlayıcı sisteme giriş olan CIF çözünürlükte sıkıştırılmamış YUV video dizisi H.264 Annex-B dosya biçiminde ve de ekrana video çıktı verilerek sıkıştırılmaktadır. Ek olarak, kodlayıcı çıktısı H.264 referans yazılımla doğruluğu kontrol edilmiş ve uyumluluğu kanıtlanmıştır.

1. INTRODUCTION

An uncompressed digital video or image is unreasonable for transmission and storage due to the need of mass space. Because of the bandwidth and storage space limitations, the usage of compression is very important and necessary in video based applications. Nowadays, digital video compression plays an important role in many applications such as digital surveillance systems, video conferencing, mobile applications as well as digital TV. The video compression becomes inevitable in such applications, so the coding standards are developed and deployed.

Video compression systems are one of the most attractive fields of consumer electronics. H.264/MPEG-4 Part 10 standard is formed for the purpose of improving the efficiency and performance of existing standards and providing the applicability of video compression in new implementations. H.264, the next generation video coding standard, employs variable block size motion compensation with multiple reference frames at quarter-pixel motion vector accuracy [1] that enables 50% reduction in bit rate while still achieving similar visual quality compared to previous video coding standards [2].

1.1 Implementing the H.264/AVC Standard

As of now, real-time computing of video applications remains a challenge. The high coding efficiency of the H.264 standard comes at the expense of increased algorithmic complexity, which affects the performance of the H.264 encoders in many real-time applications. Although hardware's evolution is accelerated, high complexity of H.264 hardware and software implementation is still a concern for application industry. Therefore, it is high time to optimize the existing H.264 coding approach to practical techniques so that it can satisfy the requirements of those emergent storage and streaming applications.

Software solutions based on general purpose processor or DSP (Digital Signal Processor) can provide flexibility and adaptability, but hardware solutions need dedicated ICs [3]. Developing dedicated hardware is a time consuming and thus an expensive task. Moreover, once the hardware platform is implemented, it is difficult to make changes. But industrial experience in this market shows that hardware adaptation to consumers needs is required in almost every design. Also, the H.264 video codec can be implemented on different DSPs, using the signal processor's high computational efficiency according to real-time constraints. TI TMS320DM64x series DSP with its enhancements is appropriate for a real-time H.264 software encoder that provides high performance and flexibility.

The proposed H.264 encoder implementation in this thesis, which is verified with the JM reference software [4], achieves real-time encoding for CIF resolution format on a DM642 DSP core. Therefore, this implementation can be used in a real world implementation, especially in video conferencing and mobile applications. Also, the realization and optimization of the encoder on TI's TMS320DM642 DSP core is represented based on the target platform features. Moreover, the flexibility and programmability of DSP implementation enables easy adaptation for higher performance solutions as a future work.

1.2 Literature Survey

In literature, there are several examples of real-time implementation of video coding and H.264 standard. In [5], realization and optimization of H.264 baseline encoder on TI TMS320C6416 DSP is presented. The study represented in that paper focuses on the optimization issues using JM81a reference software. The study of optimization of encoder includes key module optimization using SIMD (Single instruction multiple data) and linear assembly, especially data scheduling and storage allocation strategy. It is demonstrated that the access of the significant amount of data stored in off-chip memory is a bottleneck for real-time implementation. The realization and optimization results of encoder in the case study shows that 23~26 frames per second coding rate for QCIF (176x144) resolution can be reached to meet the real-time application requirements. However, the QCIF resolution can not be sufficient for many implementations and the encoder should be optimized up to CIF (352x288) format as well.

Another study in [6] is a H.264 encoder implementation on Analog Device's BF561 programmable DSP. The implementation is designed for videophone application at CIF resolution, 30 fps and bit rate of 384 and 512 kbps. In the proposed study, the algorithmic, architectural and memory optimizations are covered. Due to the high complexity of H.264, the algorithmic optimizations are focused on motion estimation (fast motion search, quarter pixel ME) and intra prediction. The architectural optimization covers the full utilization of the core for higher performance with reducing overhead and using SIMD operation. As in the previous study, the memory optimization is essential because of the significant memory requirements of encoder. That is to say, the memory optimizations have to be issued in the proposed encoder for real-time implementation.

In [2], the analysis and optimization of UB Video's H.264 baseline encoder on TI TMS320DM642 DSP is presented. The study proposes DM642-speicific methods to increase the encoding speed of the UB Video's encoder, enabling interactive real-time video applications for CIF resolution on a single DSP core. The analysis and optimization study especially on data transfer reduces the data transfer complexity by 50 % and overall encoder speed is increased by about 15%.

Finally, application report in [7] summarizes the video encoding optimization techniques on TMS320DM64x/C64x processors. These techniques include algorithmic/system optimization, memory structures, EDMA (Enhanced Direct Memory Access) usage for background (hidden) data transfer and cache utilization. The video encoder system/algorithm optimization covers macro-block level loop/module separation for decreasing cache miss penalties, and also using fast motion estimation algorithms in an encoder. Memory buffering scheme of a video encoder and data transfer with EDMA controller phases are emphasized for reducing memory result latencies and cache utilization. Lastly, the cache optimization and cache tuning is represented for improving cache performance.

The articles and studies show that a video coding application includes parallelism, so the parallel processing should be increased in the target platform as well. Realization and optimization of a real-time encoder becomes possible with the optimization techniques such as algorithmic, architectural and memory optimizations. Most of the encoder studies on an embedded platform agree that the high number of memory access is a video encoder bottleneck and memory access scheme should be well organized (i.e. EDMA usage). In this thesis, the memory organizations and optimizations are implemented and represented. The algorithmic and architectural optimizations are software oriented tasks in a real-time encoder to utilize the core for higher performance. Therefore, these tasks are represented with the proposed encoder optimization steps in thesis.

2. OVERVIEW OF H.264/AVC

In early 1998, the Video Coding Experts Group (VCEG) ITU-T SG16 Q.6 issued a call for proposals on a project called H.26L, with the target to double the coding efficiency (which means halving the bit rate necessary for a given level of fidelity) in comparison to any other existing video coding standards for a broad variety of applications. The first draft design for that new standard was adopted in October of 1999. In December of 2001, VCEG and the Moving Picture Experts Group (MPEG) ISO/IEC JTC 1/SC 29/WG 11 formed a Joint Video Team (JVT), with the charter to finalize the draft new video coding standard for formal approval submission as H.264/AVC [8] in March 2003 [1].

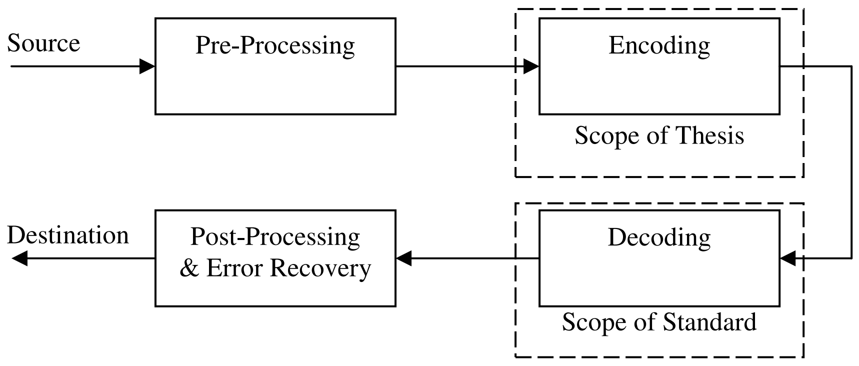


Figure 2.1: Scope of video coding standardization

The scope of the standardization is illustrated in Figure 2.1, which shows the typical video encoding/ decoding chain (excluding the transport or storage of the video signal) [1]. In all ITU-T and ISO/IEC video standards, by restricting bitstream and syntax, and defining the decoding process of the syntax elements, only the video decoder is standardized. Every decoder, according to the restrictions and definitions in the standard, has to produce similar output for the given encoded bitstream that conforms to the standard.

The H.264 standard is designed in two layers: a video coding layer (VCL), that is designed to represent the video content, and a network adaptation layer (NAL), which provides header information and VCL representation of the video for transfer and storage, as shown in Figure 2.2.

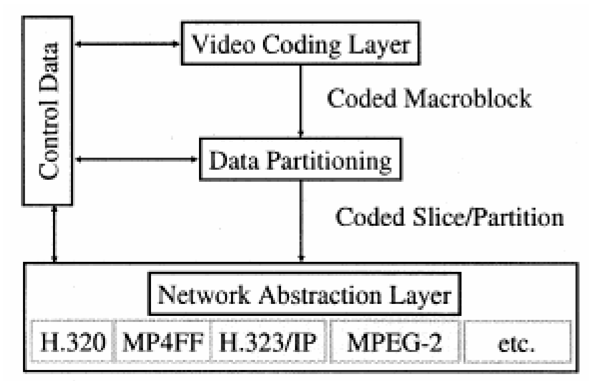


Figure 2.2: Structure of H.264/AVC video encoder [1]

As in H.263 and MPEG-2, H.264 VLC uses translational block-based motion compensation and transform based residual coding. However, there are significant differences in details, such as scalar quantization with adjustable quantization step, content adaptive run-length VLC of the quantized transform coefficients [9].

2.1 Profiles and Levels of H.264/AVC

Until now, 4 basic profiles of H.264/AVC standard have been issued as baseline, main, extended and high profiles. These H.264 profiles can be used in many different application areas such in video conferencing to digital cinema according to target usage as stated in Table 2.1.

Table 2.1: H.264/MPEG-4 Part 10 profiles and their major application areas [10,11]

Profile	**Typical Applications**
Baseline	Video Conferencing & Telephony and Mobile Applications
Main	Video Storage & Playback, Broadcast Video
Extended	Streaming Media
High Profile (Fidelity Range Extension)	Studio Editing, Post Processing, Digital Cinema

Table 2.2: H.264/AVC profiles with the coding feature

	Baseline	**Main**	**Extended**	**High**	**Proposed Baseline Encoder**
I Slices	**X**	**X**	**X**	**X**	**X**
P Slices	**X**	**X**	**X**	**X**	**X**
Deblocking Filter	**X**	**X**	**X**	**X**	**X**
Variable Block Size	**X**	**X**	**X**	**X**	**X** (16x16 to 8x8)
CAVLC	**X**	**X**	**X**	**X**	**X**
¼ Pel Motion Compensation	**X**	**X**	**X**	**X**	**X** (Half-pel MC only)
Error Resilience Tools (FMO, ASO, Redundant Slices)	**X**		**X**		
B Slices		**X**	**X**	**X**	
SI/SP Slices			**X**	**X**	
CABAC		**X**		**X**	
Weighted Prediction		**X**	**X**		
Data Partitioning			**X**		
Interlaced Coding		**X**	**X**	**X**	

In Table 2.2, the included coding tools of the H.264 profiles are summarized respectively, and also the proposed baseline encoder is given. In the proposed baseline encoder “Error Resilience Tools” is not covered and only the half pixel motion compensation is applied for the purpose of the real-time constraints on a digital signal processor.

The profile independent levels defined in the H.264/AVC standard are about the picture resolution and frame rate of the video codec that should be supported. In Table 2.3, the levels and performance of the H.264 standard are listed related to resolution and frame rate.

Table 2.3: Performances of H.264/AVC levels

Level	Performance
1.0	QCIF @ 15 fps
1.1	QCIF @ 30 fps
1.2	CIF@ 15 fps
2.0	CIF@ 30 fps
2.1	HHR @ 15 or 30 fps
2.2	SDTV @ 15 fps
3.0	SDTV: 720x480x20i., 720x576x25i 10Mbps (max)
3.1	1280x720x30p., SVGA (800x600) 50+p.
3.2	1208x720x60p.
4.0	HDTV: 1920x10870x30i., 1280x720x60p., 2k1kx30p. 20Mbps (max)
4.1	HDTV: 1920x10870x30i., 1280x720x60p., 2k1kx30p. 50Mbps (max)
5.0	SHDTV / D-Cinema: 1920x108x60p., 2,5kx2k.
5.1	SHDTV / D-Cinema: 4kx2k.

2.2 Coding Features of the H.264

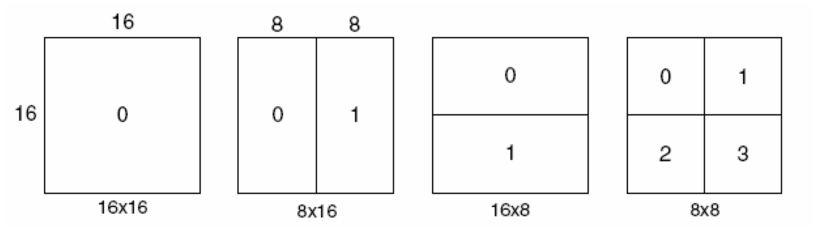
The common coding features of the H.264 profiles, Table 2.2, are described in this section according to their application in the proposed encoder scheme.

I Slices (Intra-coded Slices): Intra-coded slices can only contain intra-coded macroblock, which can only be predicted with in the same slice (spatial correlation). An intra-coded MB does not contain references from the previous or successive slices (temporal correlation); therefore, I Slices have no temporal information.

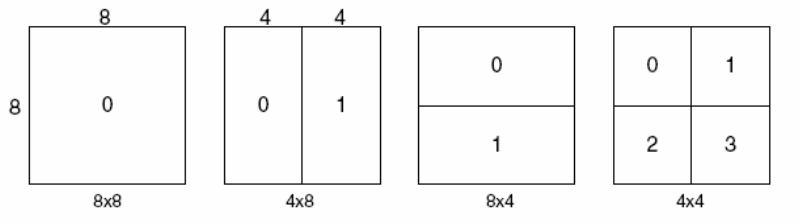
P Slices (Predictive-coded Slices): Predictive-coded slices can contain both intra-coded and inter-coded, which is predicted from previously coded pictures for P slices, macroblocks. An inter macroblock is derived with at most one motion vector and reference index for its prediction.

In-the-loop Deblocking Filter: Block-based video coding produces artifacts known as blocking artifacts that can originate from both the prediction and residual difference coding stages of the decoding process. Application of an adaptive deblocking filter is a well-known method of improving the resulting video quality, and when designed well, this can improve both objective and subjective video quality. Building further on a concept from an optional feature of H.263+, the deblocking filter in the H.264/AVC design is brought within the motion-compensated prediction loop, so that this improvement in quality can be used in inter-picture prediction to improve the ability to predict other pictures as well [1].

Variable Block Size: The supports more flexibility in the selection of motion compensation block sizes and shapes than any previous standard, with a minimum luminance motion compensation block size as small as 4x4 [1]. The H.264/AVC standard defines seven possible Macroblock sizes. The Figure 2.3.a are the large Macroblock partition size and Figure 2.3.b are the sub-macroblock partitions of the 8x8 block size. Large partition types can be used for homogeneous areas of the picture, while small partition size may be beneficial for detailed areas.



(a)



(b)

Figure 2.3: (a) Macroblock partitions: 16x16, 8x16, 16x8, 8x8, (b) Sub-macroblock partitions: 8x8, 4x8, 8x4, 4x4

CAVLC (Context-Adaptive Variable Length Coding) Entropy Coding: This feature includes VLC (Variable Length Coding) tables (Huffman Table in MPEG standards) for various syntax elements that are used for the entropy coding of the transformed and quantized residual data. Additionally, Exp-Golomb code words [12] with a regular construction are used for variable length coding as well.

¼ Pixel (Sub-pel) Motion Compensation: In addition to the integer pixel accurate motion compensation, H.264 standard allows using half-pixel and quarter-pixel accurate motion compensation to improve the coding standard, while adding computational complexity to the interpolation process. Interpolation process details, especially the half-pel interpolation process, will be described in realization and optimization steps of thesis.

Error Resilience Tools:

FMO (Flexible Macroblock Order), in which macroblocks may be coded out of raster sequence order. FMO makes it possible to map the sequence of MBs to multiple slice groups in a flexible way.

ASO (Arbitrary Slice Order), in which slices may be coded out of raster sequence order. ASO is defined to be in use if the first macroblock in any slice in a decoded frame has a smaller macroblock address than the first macroblock in a previously decoded slice in the same picture [13].

RS (Redundant Slice) belongs to the redundant coded data obtained by same or different coding rate, in comparison with previous coded data of same slice.

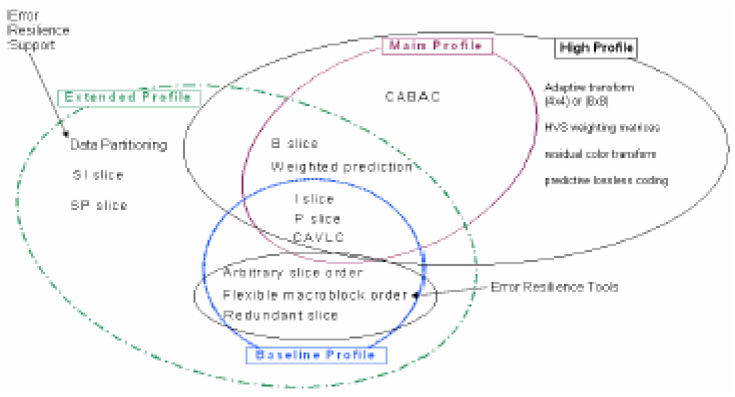


Figure 2.4: Specific coding parts of the H.264 profiles [14]

The overall described coding parts are the baseline profile elements as shown in Figure 2.4. According to the proposed encoder, only the error resilience tools are excluded for implementation.

The Baseline profile has the lowest complexity compared to the other profiles in H.264, so it is the simplest profile to implement and use in different applications. The main profile allows additional reduction in bandwidth over the Baseline profile through mainly Bi-directional (B Slices), Context Adaptive Binary Arithmetic Coding (CABAC) and Weighted Prediction.

B Slices obtains a compression advantage as compared to P Slices by allowing past and future reference pictures (similar to B-picture prediction in earlier MPEG video standards) with a variety of prediction modes for each macroblock partition. Each macroblock in an inter-coded macroblock in a B slice may be predicted from one or two reference pictures, before or after the current picture in temporal order.

A weighted average of the pixel values in the reference pictures is then used as the predictor for each sample in B slices' macroblock. B Slices also have a special mode "Direct mode", in which the motion vectors for a macroblock are not explicitly sent. The encoder can specify in the slice header either for the decoder to derive the motion vectors by scaling the motion vector of the co-located macroblock in another reference picture or to derive it by inferring motion from spatially-neighboring regions [15]. By allowing variety of prediction modes, the prediction accuracy is improved, often reducing the bit-rate by 5-10%.

Context Adaptive Binary Coding (CABAC) provides selection of probability for each syntax element according to the element's context in both encoder and decoder side. The compression performance is increased through adapting probability estimations using local statistics and using arithmetic coding rather than variable length coding for the syntax elements.

Context-based adaptive binary arithmetic coding is used as the standard as a way of gaining additional performance relative to CAVLC coding, at the cost of additional complexity. The CABAC mode has been shown to increase compression efficiency by roughly 10% relative to the CAVLC mode, although CABAC is significantly more computationally complex [15].

Weighted prediction allows modifying (scaling) the samples of the motion compensated prediction data in an inter macroblock using global multiplier and a global offset. Explicit or implicit derived multiplier and offset specified by the encoder are used to weighting and offsetting the prediction. The weighted prediction can improve coding efficiency for sequences with fades, lighting change, and can be used flexibly for other purposes as well.

SP and SI slices are specially-coded slices that enable (among other things) efficient switching between video streams and efficient random access for video decoders. A common requirement in a streaming application is for a video decoder to switch between one of several encoded streams. For example, the same video material is coded at multiple bit rates for transmission across the Internet and a decoder attempts to decode the highest bit rate stream it can receive but may require switching automatically to a lower bit rate stream if the data throughput drops [13].

Interlaced Coding enables coding of the picture frames at field level; fields can be treated as frames. Rather than progressive video coding, interlaced frames are used in TV signals (CRT TV technology) especially in digital television systems.

Data Partitioning enables an encoder to reorganize the coded data within a video packet to reduce the impact of transmission errors. As in Mpeg-4, but with further differences in data partitions, the packets are split into partitions for a reasonable reconstruction for damages and transmission errors.

2.3 Comparison of the Emerging H.264 Video Coding With Other Standards

As described in [1], the H.264 VCL uses translational block-based motion compensation and transform based residual coding as in other coding standards. However, H.264 features accommodate significant differences in details such as integer transform (4x4 block size), multiple reference and ⅛ accuracy motion vectors.

As drawn in Figure 2.1, the separation of the architecture into two layers with a video coding layer (VCL) for efficient compression and a network abstraction layer (NAL) for packing that coded data for transmission and network is the concept of the standard. The layered structure of the H.264 enables superior error resilience due to enhancement layer and error resiliency tools [11].

The video coding layer of H.264/AVC is similar in spirit to other standards such as MPEG-2 Video. It consists of a hybrid of temporal and spatial prediction, in conjunction with transform coding [16]. Although, the standard consists of transform, quantization, motion compensation and entropy coding blocks, there are major differences in that block according to previous standards. Variable block size 16x16 to 4x4 (Figure 2.3.a and 2.3.b), increased precision motion vector with quarter pixel resolution, Context-Adaptive Variable Length Coding (CAVLC) and Context-Adaptive Binary Arithmetic Coding (CABAC) for entropy coding can be indicated for superiority of H.264/AVC standard. Moreover, in-loop deblocking filter is added to the standard for reducing the blocking artifacts.

According to test results in [9], the H.264 standard achieves 50% average coding gain over MPEG-2, 47% average coding gain over H.263 baseline, and 24% average coding gain over H.263 high profile encoders. These results show us that H.264 provides efficient coding not only in bit rate compression but also in quality as well.

2.3.1 H.264 Encoder Complexity

The coding tools of H.264/AVC when used in an optimized mode allow for bit savings of about 50% compared to previous video coding standards like MPEG-4 and MPEG-2 for a wide range of bit rates and resolutions. However, these savings come at the price of an increased complexity. The decoder is about 2 times as complex as an MPEG-4 Visual decoder for the Simple profile, and the encoder is about 10 times as complex as a corresponding MPEG-4 Visual encoder for the Simple profile. The H.264/AVC main profile decoder suitable for entertainment applications is about four times more complex than MPEG-2. The encoder complexity depends largely on the algorithms for motion estimation as well as for the rate-constrained encoder control [17].

The implementation of the H.264 video coding is a challenging process in development because of the high complexity, while it provides high video quality at low bit rates compared with previous compression standards. Additionally, in order to provide higher bit-rates and resolution with the complex tools such as HD (High Definition) video, multi-DSP systems are needed as the encoding gets much more complex.

2.4 H.264/MPEG-4 Part 10 Architecture

H.264/MPEG-4 Part 10 is organized into two different conceptual layers. In Figure 2.2, the Network Abstraction Layer (NAL) and Video Coding Layer (VCL) are drawn as the main concept of the H.264 architecture. In [15], an abstract about the architecture can be found.

2.4.1 Network Abstraction Layer (NAL)

A NAL unit is a syntax structure containing an indication of the type of data to follow and bytes containing that data in the form of a raw byte sequence payload (RBSP) interspersed as necessary with emulation prevention bytes [8]. The RBSPs, a set of data corresponding to VCL data or header information, are encapsulated into Network Abstraction Layer Unit (NALU) for storage or transmission.

H.264/AVC allows multiple sequences in one stream and a sequence containing multiple pictures with non-VCL NAL units. As described in [10], a sequence parameter set (SPS) contains important header information that applies to all NAL units in the coded video sequence and a picture parameter set (PPS) contains header information that applies to the coding of one or more picture within the coded video sequence. Therefore, non-VCL NAL units, SPS and PPS, are numerated to identify each sequence and picture. Besides, the VCL NAL units containing the data that represents the values of the samples in the video pictures are packaged after the related non-VCL parts.

The proposed encoder implementation outputs bitstream in Annex B byte stream file format using NALU syntax which is described in ITU-T H.264 Recommendation [8].

2.4.2 Video Coding Layer (VCL)

The typical block diagram for coding a macroblock is shown in Figure 2.5. The encoding process also includes decoding process (except for entropy decoding) because in encoding, the motion estimation has to use the same reconstructed reference picture with the decoder.

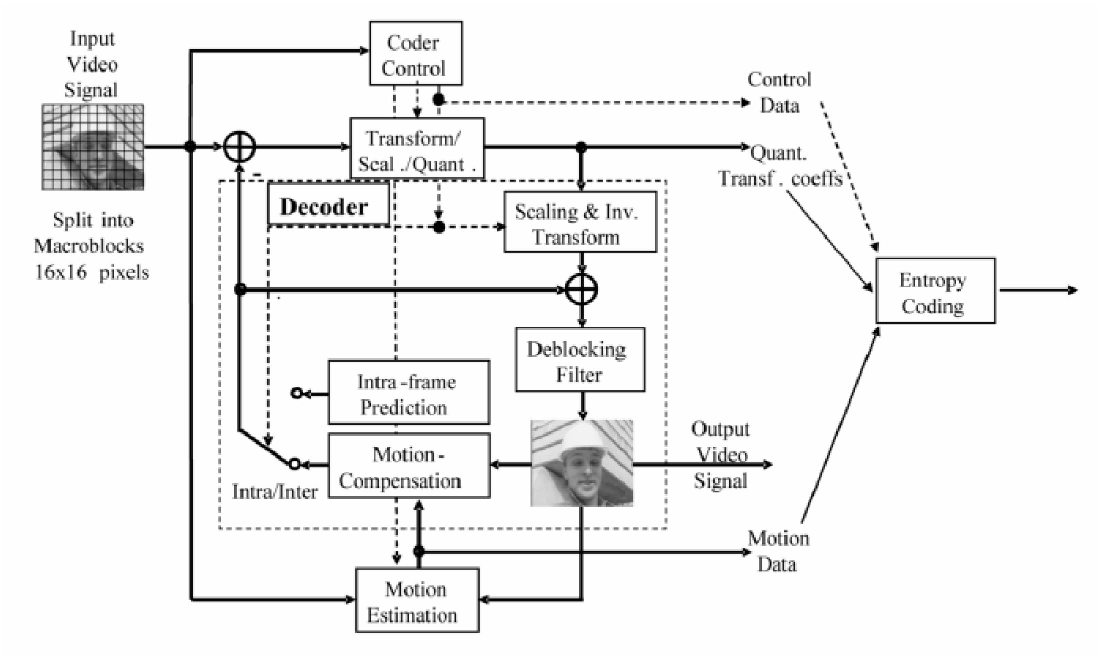


Figure 2.5: Block Diagram of encoding process in the VCL of H.264/AVC [1]

The VCL processes the frame prediction and/or motion estimation according to slice type and also decides intra/inter macroblock coding in inter slices based on coding cost using distortion and bit rate (Lagrange model). After decision, the prediction errors are coded using compression tools (transform, quantization etc.) to represent the value of video samples.

2.4.2.1 Intra Prediction Process

Intra prediction is derived from decoded samples of the same decoded slice in encoder and this process extracts the spatial redundancy between adjacent macroblocks in a slice. The intra predicted pictures usually give better quality and lower distortion than inter predicted picture, but intra prediction requires much more bits to represent the samples. Because of the higher bit rate requirement of an intra predicted slices, the number of the intra predicted slices is quite less than the inter slices for reduction of bits in stream.

At the beginning of a coded video sequence is an instantaneous decoding refresh (IDR) Access unit. An IDR access unit contains an intra picture and the presence of an IDR Access unit indicates that no subsequent picture in the stream will require reference to pictures prior to the intra picture it contains in order to be decoded [1]. The intra slices called IDR access unit are used after subsequent inter frames to decrease the propagated error in coded video. The intra frame period as IDR rate (default 50) is determined with a parameter in proposed implementation.

The H.264/AVC standard defines the intra 4x4 block (Intra_4x4) and intra 16x16 (Intra_16x16) modes for the luminance samples, intra 8x8 modes for the chrominance samples of a macroblock, and also the I_PCM mode. The Intra_4x4 mode is based on predicting each 4x4 luminance block separately and is well suited for coding parts of a picture with significant detail. The Intra_16x16 mode performs prediction of the whole 16x16 luminance macroblock and is more suited for coding very smooth areas of a picture. The I_PCM coding type allows bypassing the prediction and transform process and directly sending values of the encoded samples.

As shown in Figure 2.6 and Figure 2.7, there are nine possible optional prediction modes for Intra_4x4 luma block and four prediction modes for Intra_16x16 luma block. The Chroma prediction uses the all four Intra_16x16 prediction modes for 8x8 blocks, only numbering of modes differ as well.

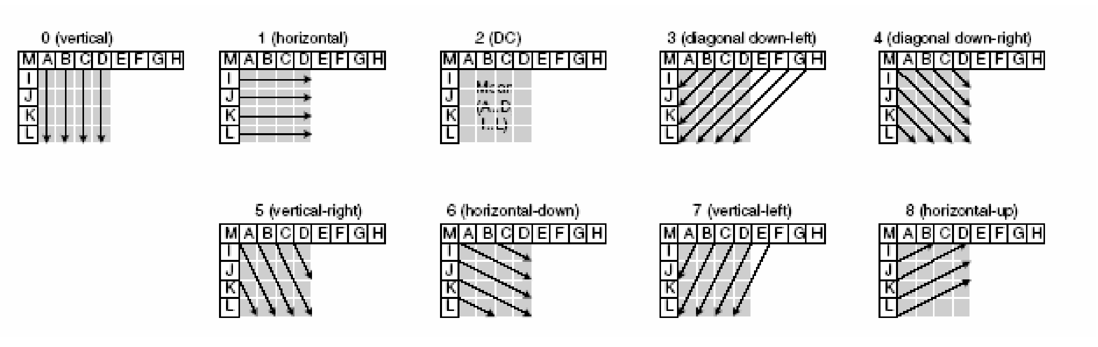


Figure 2.6: Intra 4x4 luma prediction modes [13]

In Figure 2.6, the Intra_4x4 luma block prediction modes (9 possible modes) are drawn with the prediction directions and also the descriptions of the predictions can be found in Table 2.4.

Table 2.4: Intra 4x4 luma prediction modes' descriptions [13]

Mode 0 (Vertical)	The upper samples A,B, C, D are extrapolated vertically
Mode 1 (Horizontal)	The left samples I, K, K, L are extrapolated horizontally
Mode 2 (DC)	All samples are predicted by the mean of Samples A...D and I...L
Mode 3 (Diagonal Down-Left)	The samples are interpolated at a 45^o angle between lower-left and upper-right
Mode 4 (Diagonal Down-Right)	The samples are extrapolated at a 45^o angle down and to the right
Mode 5 (Vertical-Right)	Extrapolation at an angle of approximately 26.6^o to left of vertical (width/height = 1/2)
Mode 6 (Horizontal-Down)	Extrapolation at an angle of approximately 26.6^o below horizontal
Mode 7 (Vertical-Left)	Extrapolation (or interpolation) at an angle of approximately 26.6^o to left of vertical
Mode 8 (Horizontal-Up)	Interpolation at an angle of approximately 26.6^o above horizontal

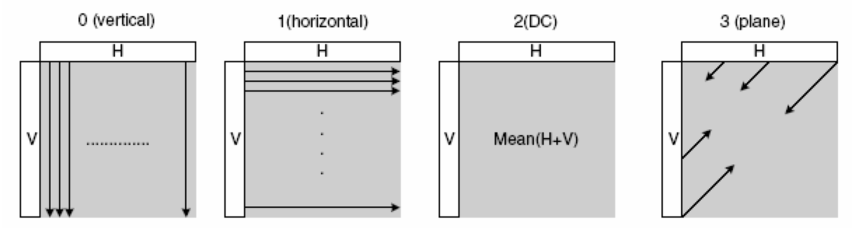


Figure 2.7: Intra 16x16 luma prediction modes [13]

In Figure 2.7, the Intra_16x16 luma block prediction modes are drawn and the descriptions of the predictions can be found in Table 2.5.

Table 2.5: Intra 16x16 luma prediction modes' descriptions [13]

Mode 0 (Vertical)	The upper samples H are extrapolated vertically
Mode 1 (Horizontal)	The left samples V are extrapolated horizontally
Mode 2 (DC)	All samples are predicted by the mean of Samples H and V
Mode 3 (Plane)	A linear 'plane' functions is fitted to the upper and left-hand samples H and V. this work well in areas of smoothly-varying luminance

For representing the intra prediction mode in bitstream with reduced number of bits, a predictive coding mechanism, called as "most probable mode", is applied. The most probable mode for the block is predicted from the neighbors and the coding of the actual mode is based on that prediction. A flag is used to determine if the actual block mode is the predicted most probable mode. If it is not, the intra prediction mode is coded and sent in addition to the most probable mode flag, as defined in the H.264/MPEG-4 Part 10 standard.

2.4.2.2 Inter Prediction process

In video coding, it is well known that temporal correlations of macroblocks are stronger than spatial correlations of macroblocks. Inter prediction is carried out on the decoded samples of reference pictures other than the current decoded picture [8], and this process eliminates the temporal redundancy between successive pictures for the compression.

Inter prediction in H.264/AVC supports variable block size from 16x16 to 4x4 as in Figure 2.3.b and fine ¼ sample motion vectors for luminance as well as ⅛ sample motion vectors for chrominance component. Although using multiple reference picture for motion estimation in inter prediction process enhances the compression efficiency, the proposed encoder implementation uses only one reference frame for real-time purpose. Multi-reference support can be adapted easily to the proposed software as a future work.

Variable block size for macroblock partitions and sub-macroblock partitions increase the number of possible combinations within each macroblock. As large partition sizes are ideal for homogenous areas and small partition sizes are ideal for detailed areas, an ideal encoder should select the optimal result according to distortion and bit rate. While the partition sizes of a macroblock become smaller, the number of bits to represent the coded data with each partition's motion vectors and reference indexes increase. However, distortion of macroblock partitions is reduced as the block size becomes smaller. Therefore, bit-rate and distortion should be considered together.

In video coding standards, the macroblock motion is found out in motion estimation block using reference picture(s). The motion estimation block can be summarized as finding the minimal difference for original block and reference picture. As the process has computational overhead, there are several methods and algorithms for approximated, restricted and fast motion estimation such as using search windows, stepped searches etc.

A H.264/AVC encoder should find out the best result for all possible block sizes. The motion estimation block outputs a motion vector for the macroblock mode. In decode process, these motion vectors are used for luma motion compensation operation and chroma motion compensation is derived by halving the corresponding luminance partition block sizes and motion vectors, since the resolution is half of the luma component.

2.4.2.3 Coefficient Transform and Reconstruction

The basic coefficient coding process in H.264/AVC is shown in Figure 2.8 that is similar to previous block-based video coding standards. The coefficient coding process includes the 4x4 2-D forward transform, hadamard transform of DC components if necessary, quantization, zigzag scanning and entropy coding for the NAL unit. Also a video encoder need references for prediction, it includes the decoding process as necessary inverse hadamard transform of DC components, dequantization and inverse transform of the coded coefficients. The block is reconstructed by the addition of motion compensated prediction and the decoded transform coefficients.

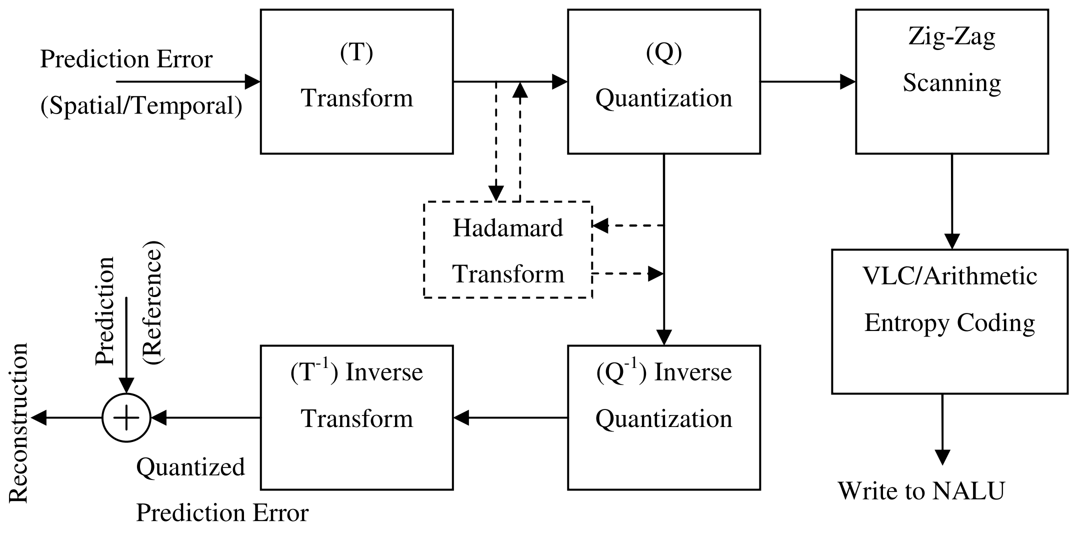


Figure 2.8: Basic coefficient/residual coding in H.264

In H.264/AVC standard, the 4x4 DCT transform is performed with integer arithmetic. The frequency domain post-scaling operation is embedded into quantization process as described in [13]. The 4x4 transform matrix [13] with integer approximations specified in the H.264/AVC standard have been designed for fast, efficient software and hardware implementations. In the encoder, each 4×4 block of quantized transform coefficients is mapped to a 16-element array in a zigzag order (Figure 2.9).

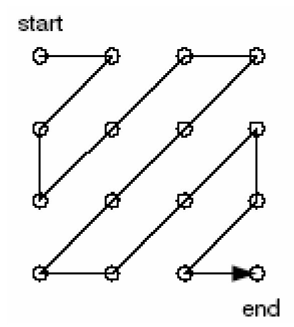


Figure 2.9: Zigzag scan for 4x4 luma block (frame mode) [13]

3. TEXAS INSTRUMENTS TMS320DM642 DSP

A digital signal processor (DSP) is a specialized microprocessor designed specifically for digital signal processing, generally in real-time computing. As stated in [18], Programmable DSPs are increasingly important in a wide range of video and imaging applications, such as medical imaging, security monitoring, digital cameras and printers, and a large number of consumer applications driven by digital video processing including DVDs, digital TV, video telephony, and many others.

As image and video processing is getting more popular, DSPs overcome the requirements due to the complexity and need of parallelism in nature. There are several special purpose multimedia processors, such as the Blackfin by Analog Devices, Philips' Trimedia and Texas Instruments digital media processors (e.g. C6000 family), and these multimedia processors are being used in low cost and/or low power embedded applications: mobile applications, digital TV, DVDs and set-top boxes.

Multimedia applications are characterized by requirements for processing flexibility, sophisticated algorithms, and high data rates [18]. For digital signal processing applications, DPSs are suited to exploit opportunities for efficient parallelism with very long instruction word (VLIW) architecture. With VLIW architecture, a flexible, high-level language programming environment has been developed in support of this processor paradigm. Also, the TMS320DM642 device that is based on VLIW architecture is an appropriate environment to implement a real-time flexible/programmable H.264 encoder.

3.1 Technical Overview of DM642 Core

The DM642 integrates a number of peripherals to address the development of video and imaging applications, including three configurable video ports capable of video input, video output, or transport stream input. The C6000 DSP family with the

VelociTI architecture addresses the needs of video and imaging applications. The C6000 family uses advanced very long instruction word (VLIW) architecture that contains multiple execution units running in parallel, which allow them to execute multiple instructions in a single clock cycle. Parallelism is the key to extremely high performance and C64x introduced VelociTI.2 extensions to the VelociTI architecture. These extensions allow more work to be done in each cycle by including new instructions to accelerate performance in key application areas including video and imaging [18,19].

The high performance very long instruction word (VLIW) architecture with VelociTI.2, 2-level memory/cache hierarchy and Enhanced Direct Memory Access (EDMA) engine makes it an excellent choice for computational intensive video/image applications such as video coding and analysis [11]. In Figure 3.1, the block diagram of the TMS320C64x DSP with CPU core, EDMA, 2-Level memory hierarchy with L1 and L2, and the peripherals is shown.

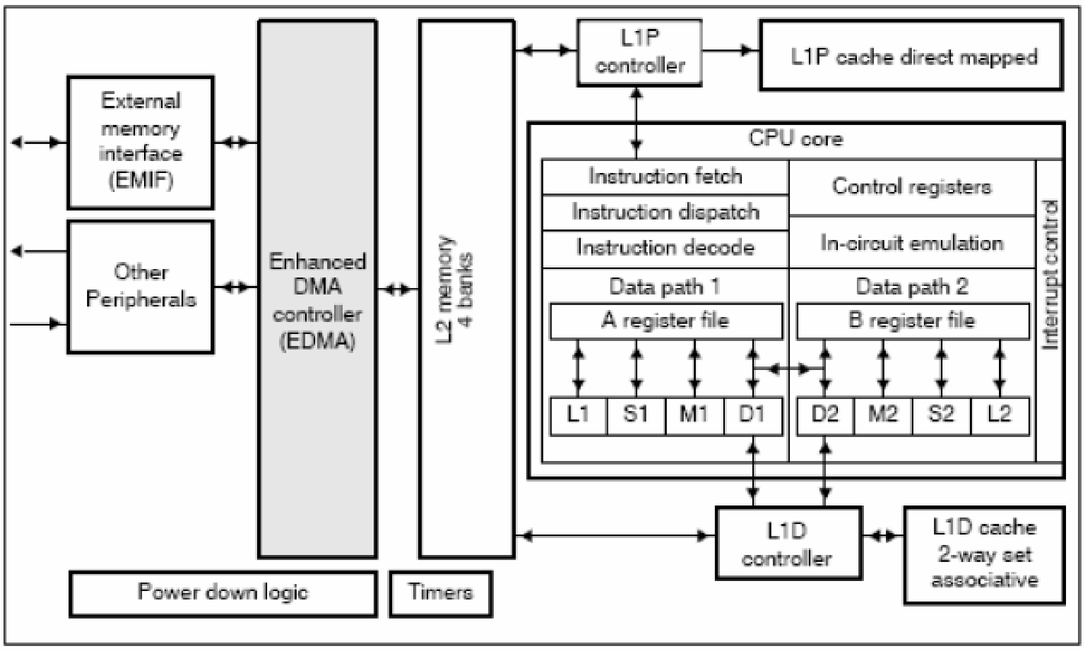


Figure 3.1: TMS320C64x DSP Block Diagram [18]

3.1.1 DM642 CPU

The DM642 is based on the C64x CPU, which is part of the C6000 DSP family that has VelociTI.2 extensions to the VelociTI architecture. The C6000 CPU components consist of:

- Two general-purpose register files (A and B)
- Eight functional units (.L1, .L2, .S1, .S2, .M1, .M2, .D1, and .D2)
- Two load-from-memory data paths (LD1 and LD2)
- Two store-to-memory data paths (ST1 and ST2)
- Two data address paths (DA1 and DA2)
- Two register file data cross paths (1X and 2X)

The DM642 has a 16 Kbytes direct mapped L1P program cache with 32-byte cache line size (8-cycle L1P cache miss penalty). The L1D cache is 16 Kbytes 2-way set-associative and has a 64 bytes cache line size (6-cycle L1D cache miss penalty). Additionally, 256 Kbytes of L2 internal memory can be configured as RAM and/or cache (flexible RAM/cache allocation, 8-cycle L2 cache miss penalty) and L2 4-way set associative cache has 128 bytes cache line size.

3.1.1.1 Register Files

There are two general-purpose register files (A and B) in the C6000 data paths. For the C64x, each of these files contains 32 32-bit registers (A0–A31 for file A and B0–B31 for file B). The general-purpose registers can be used for data; data address pointers, or condition registers. On the C64x, registers A0, A1, A2, B0, B1, and B2 can be used as condition registers. In all C6000 devices, registers A4–A7 and B4–B7 can be used for circular addressing [18].

The C64x register file supports data ranging in size from packed 8-bit data, packed 16-bit data, through 40-bit fixed-point, 64-bit fixed point, and 64-bit floating-point data. Values larger than 32 bits, such as 40-bit long and 64-bit float quantities are stored in register pairs, with the 32 LSBs of data placed in an even-numbered register and the remaining 8 or 32 MSBs in the next upper register (which is always an odd-numbered register). Packed data types store either four 8-bit values or two 16-bit values in a single 32-bit register or four 16-bit values in a 64-bit register pair [18].

3.1.1.2 Functional Units

The eight functional units in the C6000 data paths can be divided into two groups of four; each functional unit in one data path is almost identical to the corresponding unit in the other data path. The C64x contains many 8-bit and 16-bit instructions to support video and imaging applications [18].

3.1.1.3 Register File Paths

Each functional unit reads directly from and writes directly to the register file within its own data path. That is, the .L1, .S1, .D1, and .M1 units write to register file A, and the .L2, .S2, .D2, and .M2 units write to register file B [19].

Most data lines in the CPU support 32-bit operands, and some support long (40-bit) and double word (64-bit) operands. Each functional unit has its own 32-bit write port into a general-purpose register file. Each functional unit has two 32-bit read ports for source operands src1 and src2. Four units (.L1, .L2, .S1, and .S2) have an extra 8-bit-wide port for 40-bit long writes, as well as an 8-bit input for 40-bit long reads. Because each unit has its own 32-bit write port, all eight units can be used in parallel with every cycle when performing 32 bit operations. Since each C64x multiplier can return up to a 64-bit result, an extra write port has been added from the multipliers to the register file, as compared to the C62x.

The register files are also connected to the opposite-side register file's functional units via the 1X and 2X cross paths (Figure 3.2). These cross paths allow functional units from one data path to access a 32-bit operand from the opposite side's register file. The 1X cross path allows functional units from data path A to read its source from register file B. Similarly, the 2X cross path allows functional units from data path B to read its source from register file A.

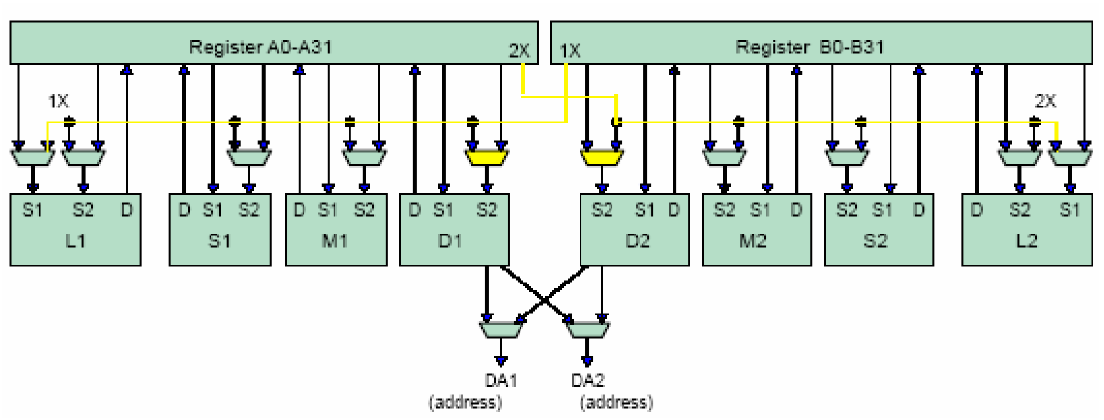


Figure 3.2: C64x Data Cross Paths [18]

On the C64x, all eight of the functional units have access to the register file on the opposite side via a cross path. Only two cross paths, 1X and 2X, exist in the C6000 architecture. Therefore, the limit is one source read from each data path's opposite register file per cycle, or a total of two cross-path source reads per cycle. The C64x pipelines data cross path accesses allow multiple units per side to read the same cross-path source simultaneously. The cross path operand for one side may be used by up to two functional units on that side in an execute packet.

3.1.1.4 Memory, Load and Store Paths

The data address paths named DA1 and DA2 are each connected to the .D units in both data paths. Load/store instructions can use an address register from one register file while loading to or storing from the other register file. The Figure 3.3 illustrates the C64x memory load and store paths.

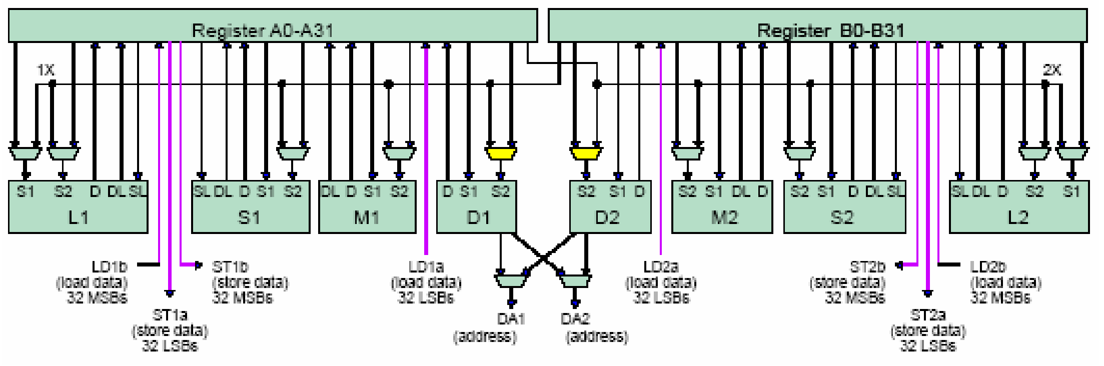


Figure 3.3: C64x Memory Load and Store Paths [18]

The C64x device can also access words and double words at any byte boundary using non-aligned loads and stores. As a result, word and double-word data does not always need alignment to 32-bit or 64-bit boundaries. This feature is particularly useful in motion estimation and video filtering operations, where one may need access to data from any arbitrary byte boundary in memory [18].

3.1.1.5 Additional Functional Unit Hardware

Additional hardware has been built into the eight functional units of the C64x. Each .M unit can perform two 16x16 bit multiplies or four 8x8 bit multiplies every clock cycle. Also, the .D units can access words and double words on any byte boundary by using non-aligned load and store instructions. In addition, the .L units can perform byte shifts and the .M units can perform bi-directional variable shifts in addition to the .S unit's ability to do shifts. The .L units can perform quad 8-bit subtracts with absolute value. This absolute difference instruction greatly aids motion estimation algorithms. Table 3.1 lists some of the special purpose instructions included in C64x for video and imaging applications.

It is important to note that the C64x provides a comprehensive set of data packing and unpacking operations to allow sustained high performance for the quad 8-bit and dual 16-bit hardware extensions. Unpack instructions prepare 8-bit data for parallel 16-bit operations. Pack instructions return parallel results to output precision including saturation support.

Table 3.1: C64x Special Purpose Instruction for Video and Imaging [18]

Instruction	**Description**	**Example Application**
LDNW	Load non-aligned double word	Motion estimation, image filtering
DOTPx	Dot product	Image filtering, image resizing, image transforms (e.g. DCT, wavelet)
MINUx/MAXUx	Minimum/Maximum computation	Nonlinear image filtering (e.g. median filter)
PACKx	Data packing and unpacking	Multiplexing/De-multiplexing interleaved data (e.g. Y/C data)
AVGx	Quad 8-bit, dual 16-bit average	Motion compensation
SUBABS4	Quad 8-bit absolute of differences	Motion estimation

These specialized instructions for video and image are used in video coding for optimization and parallelism purposes. The instruction set of the DM642 DSP is examined and the powerful ones are used for the H.264 encoder for real-time implementation.

3.1.2 DM642 Cache Architecture

On DM642 devices, the CPU interfaces directly to dedicated level-one program (L1P) and data (L1D) caches of 16 Kbytes each. These caches operate at the full speed of CPU access. A second level unified L2 program/data memory provides flexible storage. Figure 5 depicts an example L2 of size 256 Kbytes; the size and segmentation of the L2 cache in the DM64x family may change over time. One configuration for L2 is entirely mapped SRAM. The other configurations have both SRAM and a 4-way set associative cache of various sizes. Mapped SRAM can be used for streaming video data and critical sections of code such as interrupt service routines. Cache is useful for most of the program and data structures.

In DM642 core, which consists of two-level cache-based memory model that are L1 and L2 and an external memory [20] are shown in Figure 3.4 and Figure 3.5, also the 256 Kbytes L2 internal memory space can be set as both cache and RAM. The external memory interface is the EDMA for data transfer operation from/to internal memory space.

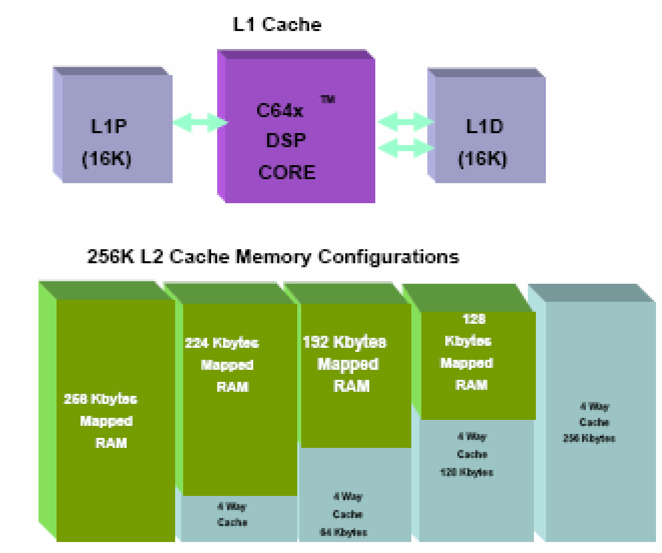


Figure 3.4: DM642 L1/L2 Cache Organization [18]

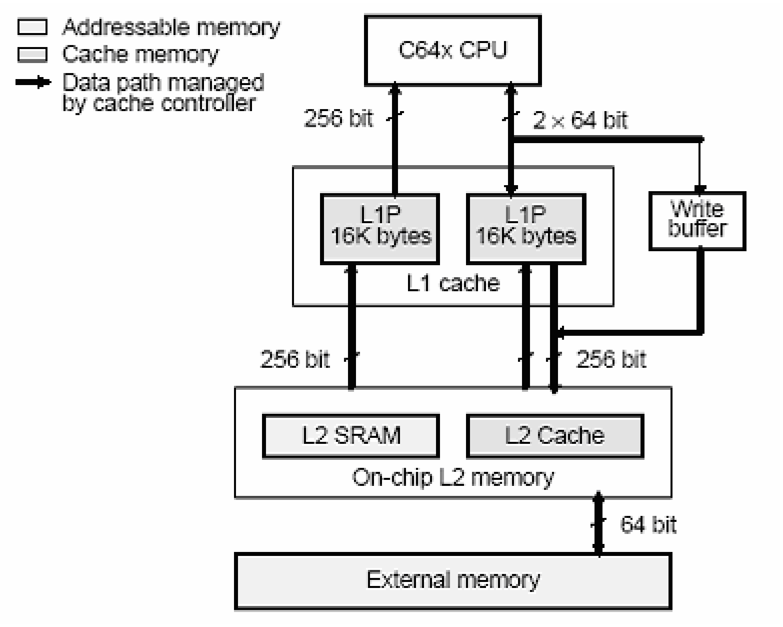


Figure 3.5: C64x Cache Memory Architecture [20]

3.1.3 DM642 Enhanced DMA (EDMA) Controller

The DM642 EDMA can provide over 2Gbytes/sec of external bandwidth on initial implementations. The EDMA supports up to 64 channels triggered by independent events. A total of 85 parameter sets are available for linking or chaining. Linking allows a sequence of transfers to be issued when a single event occurs. Chaining allows one EDMA channel to trigger another channel upon data transfer completion. Linking and chaining allow continuous auto-initialization of DMA operation with only initial configuration by the CPU. These features also allow circular buffers, ping-pong buffers, and transfers of complex data structures. Using 1-D and 2-D the user can transfer sub frames of an image as well as automatically interleave or de-interleave time-division multiplexed (TDM) digital streams. Byte, half-word, word, and double-word data sizes are supported.

The EDMA supports unsurpassed concurrency. Four independent transfer queues allow highly efficient operation. Channels on different queues can interleave transfers on a cycle-by-cycle basis. The key system benefit is that interactions between channels do not affect performance as much as much as in traditional DMA implementations.

The EDMA Controller gives an approach to perform memory transfer operation in background/parallel so that CPU is not stalled in that type of operation. There are two methods to initiate and perform an EDMA operation: synchronous method that is CPU initiated by using EDMA APIs and asynchronous CPU request to DMA channels and no communication signaling between operations. These two methods are appropriate for background memory transfer (i.e. memory copy operation) for the H.264 video coding.

3.1.4 DM642 Video Port

The video port peripheral can operate as a video capture port, video display port, or TSI capture port [18]. The video capture port is used for the input for the video coding and the video display port is the output of the coding process as well. For the proposed implementation

- Video capture mode:

The captured input sequences are encoded in the proposed H.264 encoder optimization. Therefore, the real time consideration can be evaluated experimentally.

- Video display mode:

The encoded video reconstruction of the proposed H.264 encoder is outputted from the DSP to composite video; as a result, the compression quality can be evaluated visually as well.

4. H.264 BASELINE ENCODER SOFTWARE AND DSP ADAPTATION

The design and software development of multimedia applications such as complex video/image processing requires an algorithmic infrastructure. The algorithmic infrastructure of a video processing concerns on performance: bit-rate, visual appearance, peak signal-to-noise ratio (PSNR), etc. As the H.264 JM reference software [4], the algorithmic specification of a typical video coding scheme with software and descriptions are released for developers for implementation and verification. Therefore, a video coding software implementer for an embedded platform can take assistance from JM reference software to understand the sub-parts of the big software picture. However, the reference software does not offer an optimal software solution for each routine, so the adaptation and optimization of the codec is developers' challenge.

Starting with the study and analysis of the JM reference software [4], an applied encoder software in [11] was reorganized and developed using a workstation (e.g. x86 based desktop computer) for the target platform. While the software development on a workstation includes platform independent realizations, the DSP adaptation of the pure C programming language code on workstation needs architectural modifications reasonable for the platform. Also, the proposed H.264 baseline encoder implementation outputs are verified with the reference decoder for conformance purpose. The encoder reconstruction and the decoder output comparison have to be exactly the same; that means the outputs should be same.

4.1 H.264 Baseline Encoder Software

The encoder software was developed using C programming language on a workstation with MS development environments such as Microsoft Visual Studio. For the target platform, the software was migrated to the embedded DSP development environment with additional coding for video capture/decoder and display/encoder devices. Texas Instruments Code Composer Studio (CCStudio) integrated development environment (IDE) is the environment for DSP-based embedded application development and adaptation.

As the DM642 DSP (720 MHz) clock speed is less than our x86-based workstation, the computational power and the parallelism of the DSP architecture have to be used for a real-time performance. Therefore, to provide a real-time H.264 encoder on DM642, both the H.264 encoder algorithm and the target system architecture have to be analyzed deeply and understood extensively.

4.1.1 Main Software Flow

The main encoder flow diagram (Figure 4.1) shows the basics of a sequence of pictures encoding process. The encoding process consists of generating parameter sets, starting the sequence of encoding and coding the pictures in VCL for NAL units according to coding sequence. The sequence of the pictures to be encoded is passed onto encoder in parameter set section.

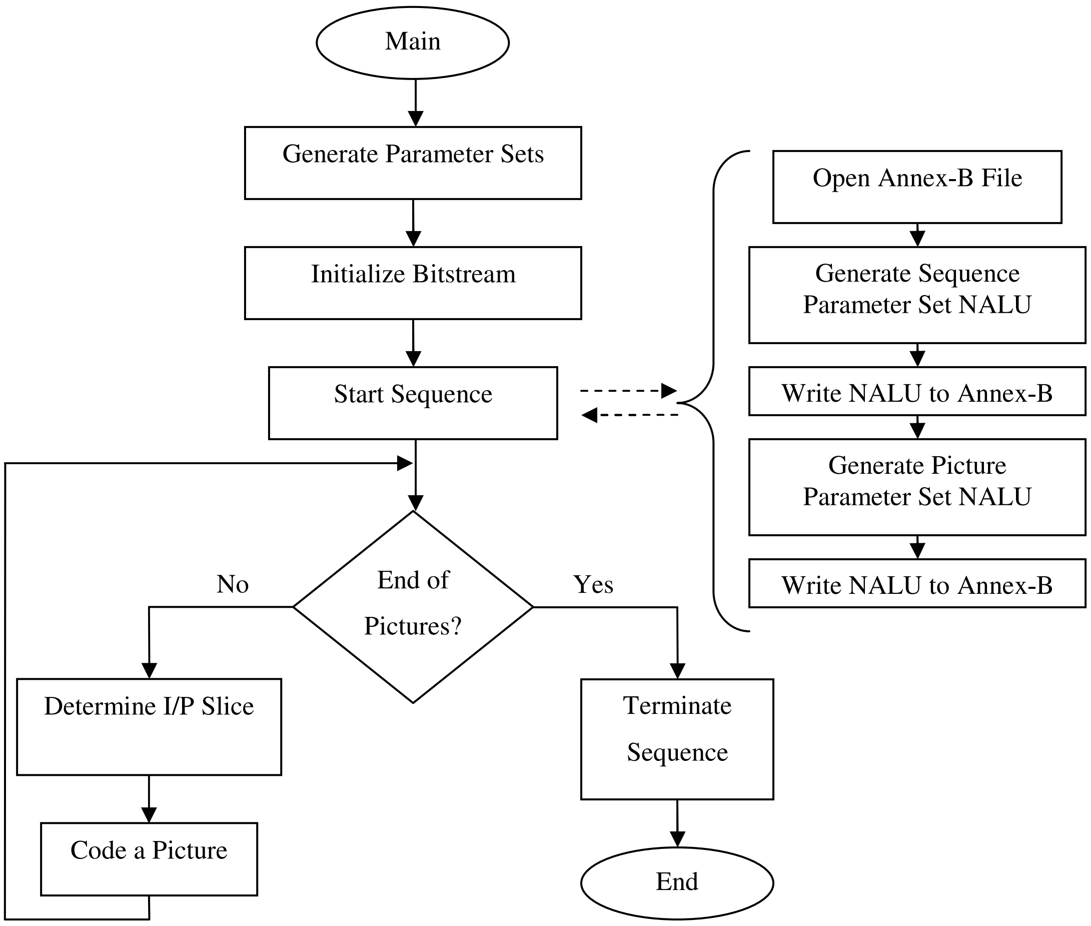


Figure 4.1: Main encoder flow diagram

In Figure 4.1, Generating parameter set includes the coding parameters of a picture sequence, such as input variables (i.e. width, height, etc.), profile type and encoding parameters as well. The parameters are applied for a determined picture sequence set, so the H.264 VCL parameters are passed to the encoding section. The bitstream initialization is required to create the output for writing the encoding results to NALU stream.

The output stream as Annex-B file requires sequence and picture parameter set NAL units at the beginning of the encoded picture NALs that are used in the rest of video sequence set. The start sequence block creates the Annex-B file and sequence and picture parameter set NAL units are constituted. Afterwards, the pictures in the raw video sequence are encoded in order with I and P slice type determination.

4.1.2 Coding a Picture

The main flow line of coding a picture is shown in Figure 4.2, that a picture is encoded macroblock by macroblock order. Code a picture routine is the core encoding process of the software. The subroutine gets a picture from the sequence, encodes, reconstructs and filters (if enabled).

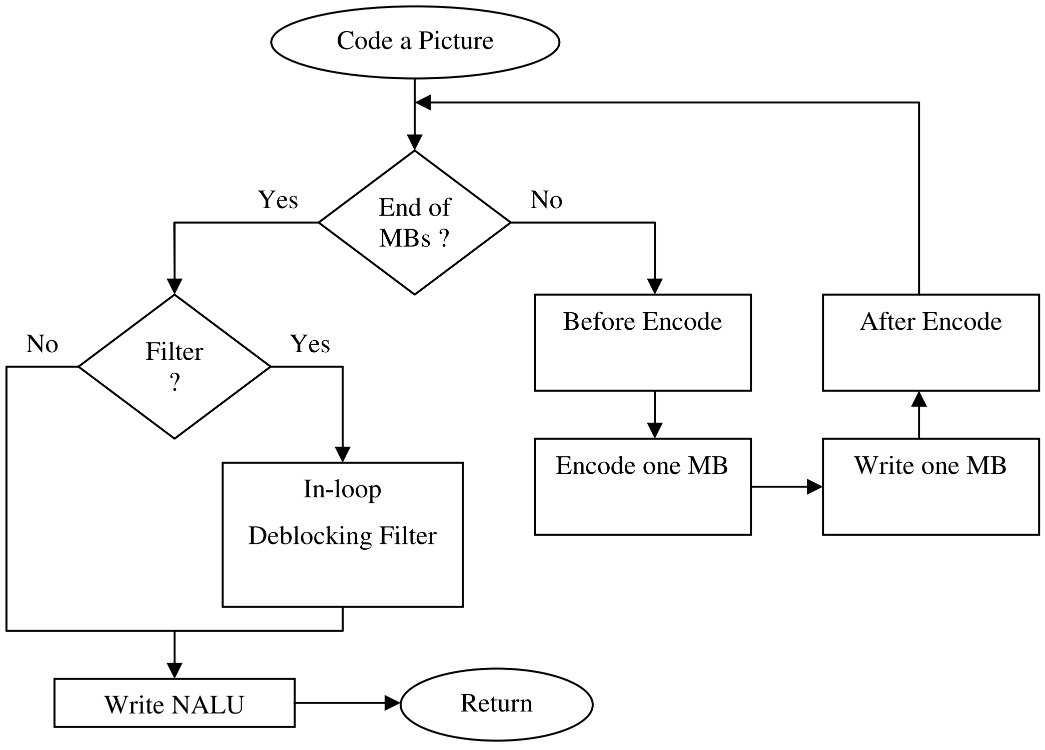


Figure 4.2: Flow diagram of coding a picture

In Figure 4.2, coding a picture's macroblock encoding consist of before, encode, write and after process. Before and after encode process includes the buffer transfer operations (i.e. Quick DMA data transfer process) such as original MB, search window, reconstructed MB, etc. The encoded macroblock's header, motion, CBP, luma and chroma coefficients information is written to the NAL unit.

4.1.3 Encoding a Macroblock

The macroblock encoding operation is different for inter and intra modes. An I slice only consists of intra predicted macroblocks, however a P slice can both contain inter and intra macroblocks.

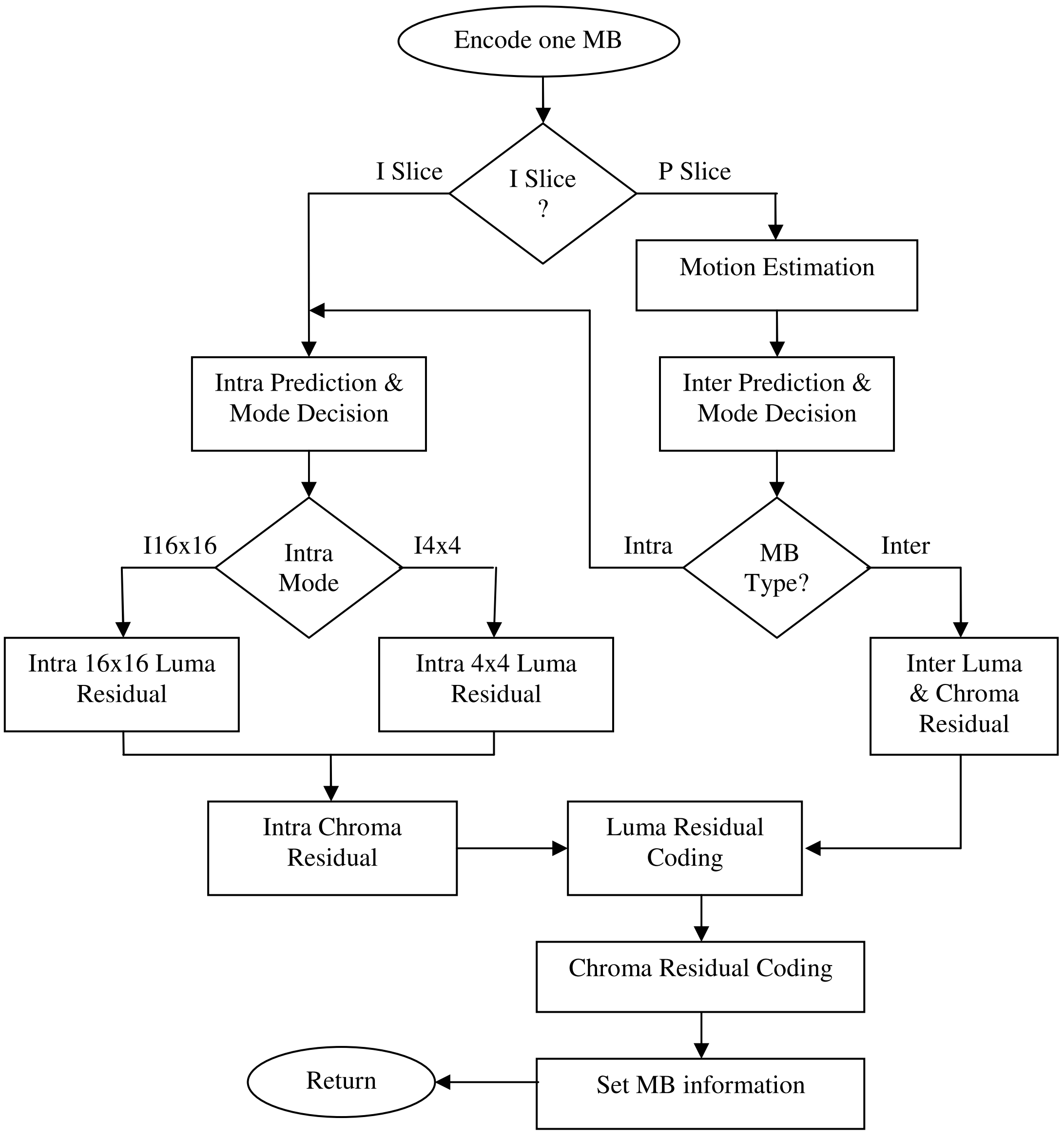


Figure 4.3: Flow diagram of encoding a macroblock

In Figure 4.3, the encoding process of a macroblock is determined whether it is in I or P slice. If the slice type is I, the macroblock are coded as intra (Intra_16x16 or Intra_4x4). For an inter macroblock, the motion estimation and mode decision blocks are processed, the luma and chroma residual operations are executed that the prediction for the chosen mode is generated and the difference is calculated. As P slice can contain intra macroblock, the inter prediction distortion and bits are considered as the cost; as a result, for an undesired (big) cost intra macroblock prediction and mode decision are included.

In the proposed encoder, the intra mode decision is based on Hadamard cost calculation. Hadamard transform computation covers both rate and distortion components, so the smallest cost is chosen as the intra mode. Only for Intra_4x4, the mode information for each 4x4 block have to be sent, so the mode information cost should be added to SATD for that mode.

The inter prediction and mode decision is based on the motion estimation entirely. The motion estimation for all possible inter macroblock mode is executed with SAD (Sum of Absolute Differences) calculation. For the inter macroblock cost, rate is added with all blocks motion vector costs that is:

$$cost\{x, y\} = cost_{mv\_x} + cost_{mv\_y} \tag{4.1}$$

$$cost_{mv\_x} = mv\_bits\{\, mv\_x_{current} - mv\_x_{pred}\} * \lambda_{motion} \tag{4.1.a}$$

$$cost_{mv\_y} = mv\_bits\{\, mv\_y_{current} - mv\_y_{pred}\,\} * \lambda_{motion} \tag{4.1.b}$$

The Lagrange formula is used for comparison of the modes to find out the optimal distortion and bitrate for the macroblock with a λ multiplier, such that the cost of a macroblock mode is:

$L = D + R.\lambda$ (L= Lagrange cost, D = Distortion, R = Rate) **(4.2)**

The mode is decided according to total cost, which is adding SAD and motion vector cost. The minimum of the mode costs is chosen as the best mode for the inter macroblock (Rate-Distortion model). If an inter macroblock cost is bigger than threshold, the intra modes are tried and the minimum of inter and intra cost is chosen as the macroblock mode. For inter and intra macroblock comparison, the intra macroblock SAD-based cost is calculated.

4.1.4 Motion Estimation

A motion estimation process is motion searching of an original block over a search area that is defined as reference. A block-based encoder employs a block-based motion search to find out an optimal matching block in its reference. For any macroblock mode in H.264 standard, the matching block is determined according to minimal SAD in the proposed encoder. In this respect, the motion estimation s searching the minimum SAD in the reference search area.

There are several methods for motion estimation that are called motion search algorithms. The first and the best resulting approach is full search, which is comparing all possible block in the search area. However, computational overhead is enormous for a real-time encoder because of the computation and comparison amount. A good video encoder algorithm implementation needs to keep a good balance between computational intensity and coding efficiency. That is to say, the motion estimation kernel needs a computationally reduced search technique as well as optimal matching block for coding efficiency

Another motion estimation method is N-Step fast search [13], which has fixed step with N, is a popular and used search algorithm in motion estimation. However, the fixed step size with different depth for each step can get caught to local minima in spatial domain. And the other search method is hexagon search [27], which is fixed depth (i.e. 2 pixels), is a fast search algorithm with inner (early) termination. Although there are several fast search algorithms for motion estimation, the distortion result can not be better than the full search's exact match approach.

In the proposed H.264 encoder, the 4-step fast search and then the hexagon motion search algorithms take a part for the motion estimation for various block size as in standard. The detailed description and results of the motion search algorithms are described in section 5.1.1.2

4.2 The Proposed Encoder Configuration

The proposed H.264 baseline encoder's main concentration is on real-time implementation on a programmable DSP. The H.264 standard defines the NAL and VCL layers structure and the profile tools as in Table 2.2. The tools are defined for the coding efficiency of the VCL layer with variable block size 16x16 to 4x4, half and quarter-pel luma motion vector accuracy, etc. However, they may need much more computational operation and memory space. For maximum coding efficiency on a real-time implementation, the proposed encoder is restricted in some tools as listed in Table 2.2.

The H.264 standard define 16x16 to 4x4 block sizes, however, 8x8 block mode and sub-partitions are for detailed video concepts. Therefore, the proposed encoder is restricted to 8x8 block size as the smallest size. The quarter pixel motion compensation for each possible direction needs 12 quarter-pel buffer arrays. That approach is unreasonable for the embedded platform. As a result, rather than pre-processing the whole picture, the computation of the quarter pixel position should be done on the fly during quarter-pel motion estimation. In the proposed encoder, the quarter-pel accuracy is not covered as it adds computational complexity that hinders the encoder to work in real-time. In addition, half pixel motion compensation in diagonal direction needs 2 times more calculation amount compared to vertical and horizontal directions. In the proposed encoder, half-pel motion compensation is implemented, but the diagonal half-pel motion compensation and the search of the diagonal direction are not implemented because of the real-time limitation on DM642 DSP target.

The embedded encoder support the resolution of 352x288 (CIF) format, and the approximation for the motion estimation is limited to motion with 16 pixels, so the search range is limited to -/+16, with a search window of 48x48. If the target encoding resolution is selected over CIF, the search range also should be enlarged for motion estimation accuracy and coding efficiency. Beside, the H.264 standard can take reference from more than one reference pictures. However, the motion estimation reference picture number is limited to 1 for fast motion estimation, but worse coding efficiency as well.

The propose encoder configuration and the features can be summarized as:

- H.264 baseline profile
    - I and P slices
    - Variable block size (16x16 to 8x8)
    - CAVLC Entropy coding
    - Full-Pel & Half-Pel motion compensation
    - Reference picture number = 1
    - Deblocking filter support
- Supported resolutions
    - CIF (352x288), QCIF (176x144) or below CIF
- Motion Estimation
    - 4-Step & Hexagon search
    - -/+ 16 search range
- CQP (Constant Quantization Parameter)

4.3 DSP Implementation

A typical embedded platform project development flow under Code Composer Studio (CCStudio) IDE is shown in Figure 4.4. Beginning with the application design, code creation, debugging and analyzing phases are fed-back to the desired optimization level for the design. So, The CCStudio development environment provides many tools for code generation, debugging and tuning as well.

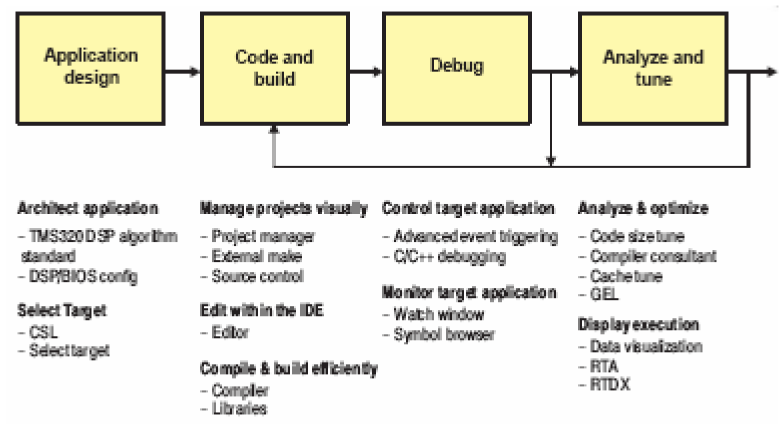


Figure 4.4: Development flow using CCStudio [21]

4.3.1 DSP Environment Set-up

The target device for the proposed H.264 software implementation is TI's DM642 DSP platform for an embedded application. A real-time development flow for an embedded platform needs the experimental setup with the target board as well as the development tools. The proposed H.264 real-time encoder is ported to the TMS320DM642 evaluation module using TI's Code Composer Studio IDE tools, and testing and verification is done under the workstation comparing the outputs of the encoder on DM642 and workstation PC.

4.3.1.1 TMS320DM642 Evaluation Module (EVM)

The DM642 EVM, an evaluation board designed by Spectrum Digital, is a low-cost standalone development platform that enables users to evaluate and develop applications for the TI C64xx DSP family. The developed software under TI's Code Composer studio development environment is ported to the EVM using JTAG or PCI interface for real-time implementation.

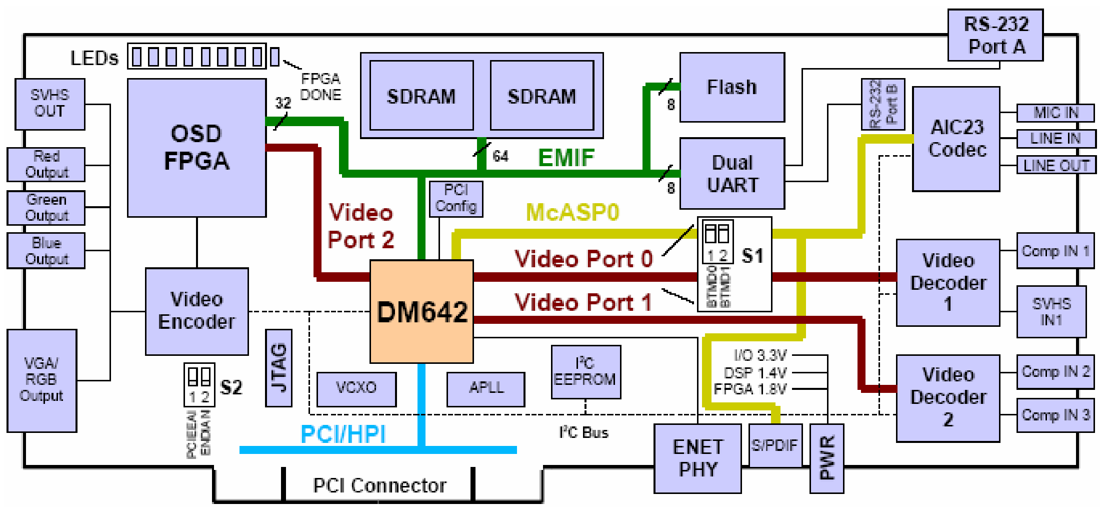


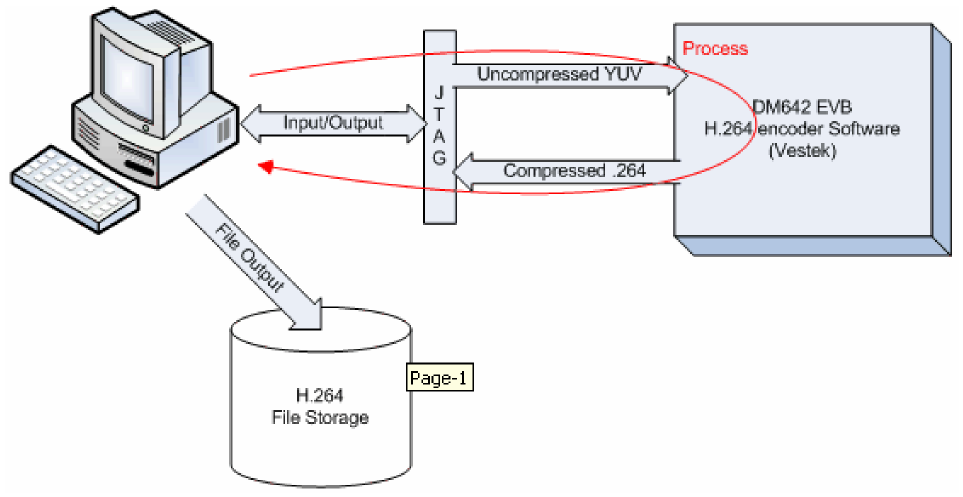
Figure 4.5: Block diagram of DM642 EVM [22]

In Figure 4.5 the block diagram of the DM642 evaluation module with the peripherals are shown. The EVM comes with a full complement of on board devices that suit a wide variety of application environments [22]. Key features include:

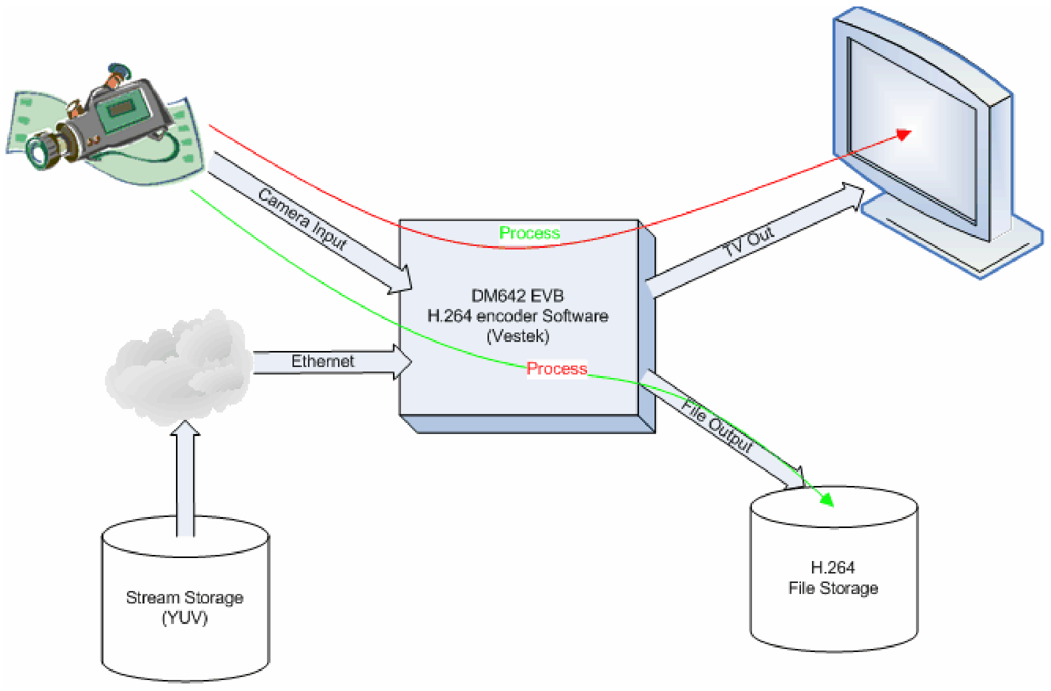
- A Texas Instruments TMS320DM642 DSP operating at 720 MHz.
- Standalone or standard PCI computer slot operation
- 3 video ports with 2 on board decoders and 1 on board encoder
- 32 Mbytes of synchronous DRAM
- On Screen display (OSD) via FPGA
- 4 Mbytes of non-volatile Flash memory
- Ethernet interface
- Software board configuration through registers implemented in FPGA
- Configurable boot load options
- JTAG emulation through on-board external emulator interface
- Expansion connectors for daughter card use

4.3.1.2 Experimental Set-up

A real world embedded application, such as a video encoder needs input for compression and output of the compressed data as bit stream or reconstruction. An experimental set-up should obtain the necessary interfaces for the real-time problem analyses and optimization. The experimental hardware setup, shown in Figure 4.6.a, is designed to realize the system.



(a)



(b)

Figure 4.6: The (a) experimental and (b) real-world hardware set-up

As shown in Figure 4.6.a, the hardware consists of the DM642 Evaluation Board, a desktop computer for Code Composer Studio IDE tools, XDS 560 JTAG emulator and a display device for video output. The Philips SAA1705 video encoder on evaluation board is used for analog video output.

Input sequences for encoding are read form host using JTAG port in the experimental set-up, which is only for using standard compression sequences in literature. In real world, the proposed implementation can not read uncompressed data from a storage device, so a video camera is inserted to the hardware for in final implementation as shown in Figure 4.6.b. The input sequences are captured from the video camera respectively by the video decoder on evaluation board, and then compressed in DM642 DSP core. The write operation is still driven over JTAG port, because the evaluation board is not directly attached to a storage device.

4.3.2 Code Composer Studio IDE

Development tools continue to grow in importance when choosing a processor platform. The TI's Code Composer Studio IDE provides useful development tools in code creation, debugging and analyzing steps according to programmers need for application development. Therefore, the proposed H.264 encoder adaptation using the Code Composer Studio v3 was practiced for the DM642 DSP evaluation board.

The Code Composer Studio (CCStudio) Integrated Development Environment (IDE) is a key element of the eXpressDSP Software and Development Tools strategy. CCStudio delivers all of the host tools and runtime software support for TMS320 DSP and OMAP based real–time embedded applications.

CCStudio integrated development environment includes host tools and target software that slashes development time and optimizes the performance for all real-time embedded DSP applications [21].

Code Composer Studio's host side tools include:

- Fully integrated CodeWright Editor,
- Source Code Debugger common interface for both simulator and emulator targets featuring with breakpoints and probe points

- Application Code Tuning Dashboard
    - Profiling and Analysis tools to understand and monitor code performance
    - CodeSizeTune, CacheTune and Compiler Consultant optimization tools
- TMS320 DSPs and OMAP Code Generation tools: C/C++ compiler, assembler and linker
- Drag and Drop CCStudio setup utility supporting:
    - XDS560 ™ high speed emulation drivers
    - XDS510 ™ emulation drivers
    - Simulators for full devices, CPU only and CPU plus memory for optimal performance
- DSP/BIOS Host Tooling Support (Configure, Real-time analysis and Debug)
- RTDX data transfer for real time data exchange between host and target

4.3.2.1 DSP/BIOS Real Time Kernel

The DSP/BIOS is a scalable real-time kernel which is designed to be used by applications that require real-time scheduling and synchronization, host-to-target communication, or real-time instrumentation. The DSP/BIOS provides preemptive multi-threading, hardware abstraction, real-time analysis, and configuration tools. The DSP/BIOS kernel also provides run-time services for developers use to build DSP applications and manage application resources. The DSP/BIOS kernel effectively extends the DSP instruction set with real-time, run-time kernel services that form the underlying architecture, or infrastructure, of real-time DSP-based applications [23].

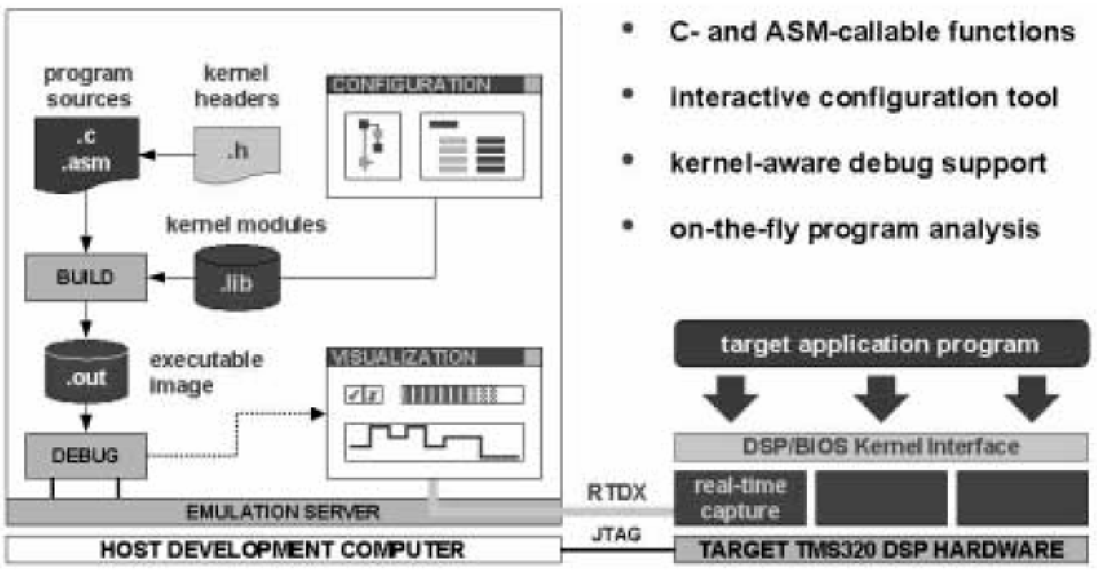

program sources
kernel headers
.c .asm
.h
CONFIGURATION
kernel modules
BUILD
.lib
.out
executable image
VISUALIZATION
DEBUG
EMULATION SERVER
HOST DEVELOPMENT COMPUTER
RTDX
JTAG
C- and ASM-callable functions
interactive configuration tool
kernel-aware debug support
on-the-fly program analysis
target application program
DSP/BIOS Kernel Interface
real-time capture
TARGET TMS320 DSP HARDWARE

Figure 4.7: Building DSP/BIOS based DSP application [23]

A DSP/BIOS configuration (Figure 4.7) allows optimizing an application by creating static objects and setting their properties, rather than at run-time. This both improves run-time performance and reduces the application footprint. Using the DSP/BIOS configuration necessary device drivers are determined and configured.

In the proposed H.264 encoder for video display and video capture, the video encoder chip and the video decoder chip device drivers for the hardware are included in the DSP/BIOS configuration. Additionally, the multithread support of the DSP/BIOS kernel is used for the tasks for display and capture operation with the H.264 encode task. Also, the required synchronization between the tasks are provided and implemented by the Synchronization Module (SCOM) of DSP/BIOS kernel as shown in Figure 4.8. The DSP/BIOS kernel details can be found at [23]

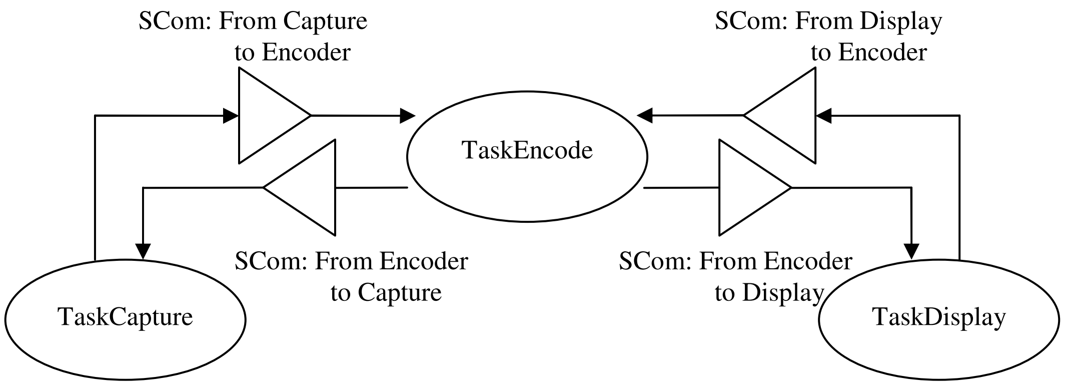

SCom: From Capture to Encoder
SCom: From Display to Encoder
TaskEncode
SCom: From Encoder to Capture
SCom: From Encoder to Display
TaskCapture
TaskDisplay

Figure 4.8: Synchronized Communication between tasks

4.3.3 Testing and Verification

As the H.264/AVC coding standard defines the syntax elements and the representation of them, the output of the proposed encoder conformance with the standard have to be done. Code Composer Studio environment supplies debugging and tracing features, that allows rapidly testing the output such as graphical view window. Also, the RTDX real-time data exchange properties with the DSP provides non-interrupted debug and trace feature.

The output streams can be visualized using software that provides file level and especially macroblock level information such us macroblock type, motion vector, coded block pattern, etc. Also, the conformance of the proposed encoder is achieved by comparing the JM Reference Software [4] decoder outputs and the proposed encoder's reconstructed frames. If there is no difference between reference decoder output and reconstruction, it can be said that the proposed H.264 encoder output is compliant to the H.264/AVC Mpeg-4 Part 10 standard.

5. THE PROPOSED H.264 BASELINE ENCODER OPTIMIZATION

The DM642 emulator and cycle accurate device simulator can be used for profiling the H.264 encoder under Code Composer Studio v3. For a real-time implementation, rather than a simulator, DM642 EVM board can be profiled using JTAG port emulator. As the emulator runs the code on the DM642 board in real-time, the profiling and cycle counting information are gathered from the JTAG port to the workstation.

While working with CCStudio v3, the statistical objects of the DSP/BIOS kernel [24] (the objects have to be created in kernel) are used to obtain cycle or time measurement and information for desired code area. In Figure 5.1, the usage of the statistical object is shown. The given code segment is attached to the proposed encoder and time and ratio measurements of optimization steps are calculated.

```
extern struct STS_Obj &STS_obj;

. . .

. . .

STS_get(STS_obj, GET_TICK_COUNT());

/* Code area to be measured */

STS_delta(STS_obj, GET_TICK_COUNT());
```

Figure 5.1: The H.264 encoder code segment for profiling

Using these STS (statistical) objects, the optimization steps' impacts are measured and a ratio between before and after process is calculated. The measurements are gathered in time units as milliseconds (ms), microseconds (µs) rather than instruction or cycle counts.

5.1 Software Optimization

Software optimization is the process of manipulating software code to achieve faster execution time and smaller code size [11]. For a software optimization process, the optimization phases should be determined and implemented. Code development steps [25] for the target system can be summarized as:

1. Compile and profile native C code

   - Validates original C code

   - Determines which loops are most important in terms of MIPS requirements

2. Add restrict qualifier, loop iteration count, memory bank, and data alignment information

   - Reduces potential pointer aliasing problems

   - Allows loop with indeterminate iteration counts to execute epilogs

   - Uses pragmas to pass count information to the compiler

   - Uses memory bank pragmas and _nassert intrinsics to pass memory bank and alignment information to the complier

3. Optimize C code using other C6000 intrinsics and other methods

   - Facilitates use of certain C64x instructions not easily represented in C.

   - Optimizes data flow bandwidth (double word access for a word data in C64x)

4a. Write Linear Assembly

   - Allows control in determining exact C6000 instructions to be used

   - Provides flexibility of hand-coded assembly without worry of pipelining, parallelism, or register allocation

- Can pass memory bank information to the tools
- Uses .trip directive to convey loop count information

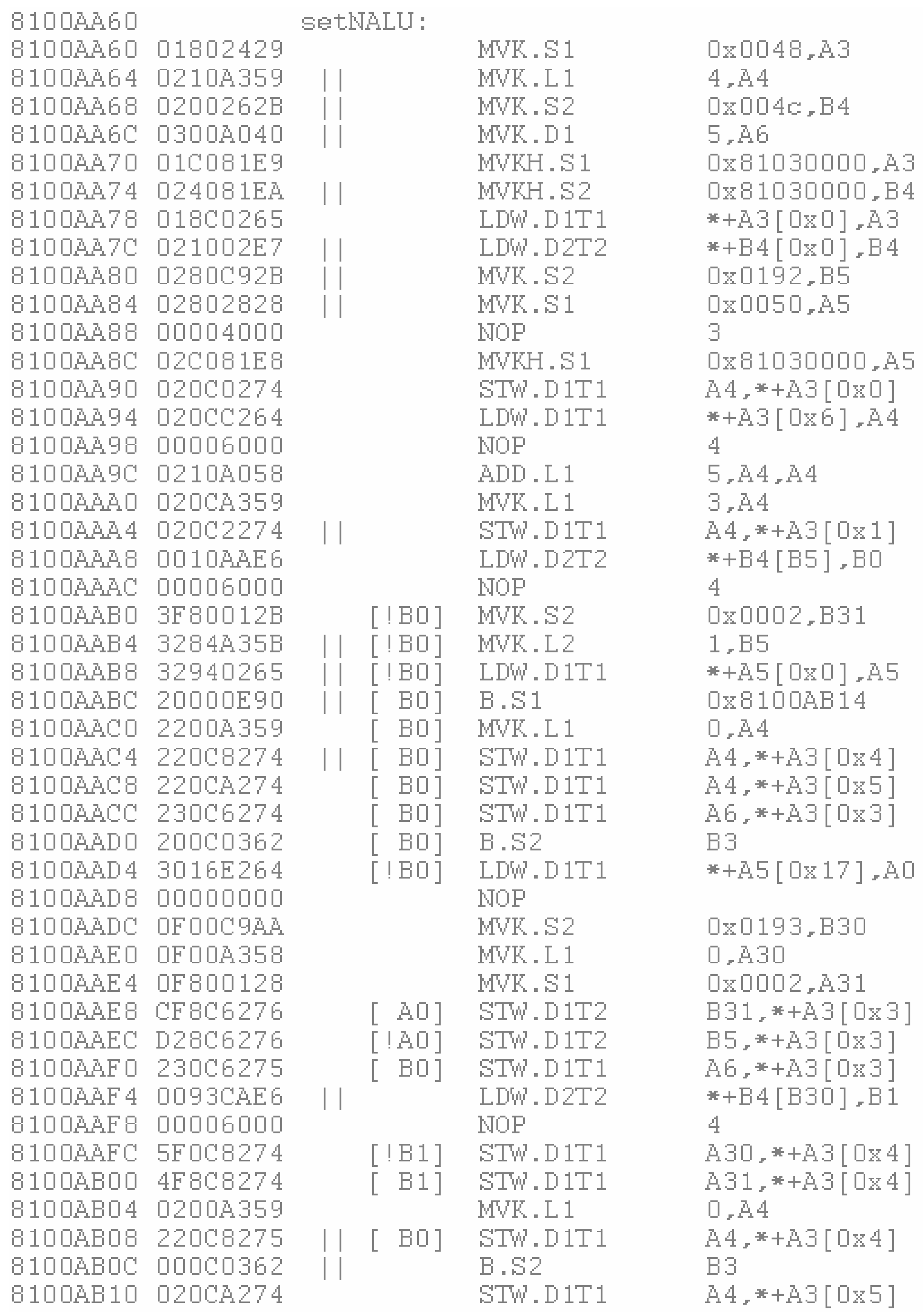
4b. Add partitioning information to the linear assembly

- Can improve partitioning of loops when necessary
- Can avoid bottlenecks of certain hardware resources

These development steps are proposed for the native and finalized software codes. The assumption for these phases is that the software code is optimized enough in the words of algorithms and program level structures. Therefore, there are several optimization phases such as algorithmic and the C level for the proposed encoder. In sections 5.1.1 and 5.1.2, the optimization steps of algorithmic and structural approaches are considered.

The CCStudio IDE's mixed source/assembly view feature is used to determine the pipeline fullness and parallelism for the time consuming code sections. The '[ ]' symbol before the instructions is the condition of the related instruction, which means it is a conditional instruction. The instructions that are tied with the pipe symbol '||' are executed in the same packet in the pipeline; as a result, maximum eight instructions can be executed in a packet because of the DM642 pipeline architecture. As shown in Figure 5.2, the code section is not parallelized enough and contains NOP (no operation) instructions.

The critical and time consuming code sections that are not using the hardware sources efficiently, especially VLIW architecture and pipeline of the DSP, are observed and profiled. The finalized C or assembly code has to fully utilize the opportunities of the DM642 DSP architecture.



```
8100AA60          setNALU:
8100AA60 01802429              MVK.S1        0x0048,A3
8100AA64 0210A359  ||          MVK.L1        4,A4
8100AA68 0200262B  ||          MVK.S2        0x004c,B4
8100AA6C 0300A040  ||          MVK.D1        5,A6
8100AA70 01C081E9              MVKH.S1       0x81030000,A3
8100AA74 024081EA  ||          MVKH.S2       0x81030000,B4
8100AA78 018C0265              LDW.D1T1      *+A3[0x0],A3
8100AA7C 021002E7  ||          LDW.D2T2      *+B4[0x0],B4
8100AA80 0280C92B  ||          MVK.S2        0x0192,B5
8100AA84 02802828  ||          MVK.S1        0x0050,A5
8100AA88 00004000              NOP           3
8100AA8C 02C081E8              MVKH.S1       0x81030000,A5
8100AA90 020C0274              STW.D1T1      A4,*+A3[0x0]
8100AA94 020CC264              LDW.D1T1      *+A3[0x6],A4
8100AA98 00006000              NOP           4
8100AA9C 0210A058              ADD.L1        5,A4,A4
8100AAA0 020CA359              MVK.L1        3,A4
8100AAA4 020C2274  ||          STW.D1T1      A4,*+A3[0x1]
8100AAA8 0010AAE6              LDW.D2T2      *+B4[B5],B0
8100AAAC 00006000              NOP           4
8100AAB0 3F80012B     [!B0]    MVK.S2        0x0002,B31
8100AAB4 3284A35B  || [!B0]    MVK.L2        1,B5
8100AAB8 32940265  || [!B0]    LDW.D1T1      *+A5[0x0],A5
8100AABC 20000E90  || [ B0]    B.S1          0x8100AB14
8100AAC0 2200A359     [ B0]    MVK.L1        0,A4
8100AAC4 220C8274  || [ B0]    STW.D1T1      A4,*+A3[0x4]
8100AAC8 220CA274     [ B0]    STW.D1T1      A4,*+A3[0x5]
8100AACC 230C6274     [ B0]    STW.D1T1      A6,*+A3[0x3]
8100AAD0 200C0362     [ B0]    B.S2          B3
8100AAD4 3016E264     [!B0]    LDW.D1T1      *+A5[0x17],A0
8100AAD8 00000000              NOP
8100AADC 0F00C9AA              MVK.S2        0x0193,B30
8100AAE0 0F00A358              MVK.L1        0,A30
8100AAE4 0F800128              MVK.S1        0x0002,A31
8100AAE8 CF8C6276     [ A0]    STW.D1T2      B31,*+A3[0x3]
8100AAEC D28C6276     [!A0]    STW.D1T2      B5,*+A3[0x3]
8100AAF0 230C6275     [ B0]    STW.D1T1      A6,*+A3[0x3]
8100AAF4 0093CAE6  ||          LDW.D2T2      *+B4[B30],B1
8100AAF8 00006000              NOP           4
8100AAFC 5F0C8274     [!B1]    STW.D1T1      A30,*+A3[0x4]
8100AB00 4F8C8274     [ B1]    STW.D1T1      A31,*+A3[0x4]
8100AB04 0200A359              MVK.L1        0,A4
8100AB08 220C8275  || [ B0]    STW.D1T1      A4,*+A3[0x4]
8100AB0C 000C0362  ||          B.S2          B3
8100AB10 020CA274              STW.D1T1      A4,*+A3[0x5]
```

Figure 5.2: Disassembly of example (setNALU) function

5.1.1 Algorithmic Optimizations

During software optimization, the used software algorithms can be redesigned or modified according to the complexity and performance consideration. The algorithmic optimization is the way of adjusting the software to a reasonable payload while maintaining the desired objective.

After profiling the H.264 encoder, the computational load of sub operations and processes should be optimized according to implementation area and intention. According to profile results, the motion estimation kernel that includes motion search has the heaviest computational cost in the encoder because of the video encoding nature. Besides, the variable block size support in H.264 increases the search amount for each block type as well.

The motion estimation kernel computational intensity has to be decreased with an appropriate fast search algorithm and early termination determination that eliminates the unnecessary or extra search operations. From that point of view, the algorithmic optimization of the motion estimation kernel is proposed in the H.264 encoder considering the real-time limitation on DM642.

5.1.1.1 Early Skip Detection

According to the nature of the video sequence, the picture or a picture fragment can contain global motion. In H.264, an improved skip macroblock mechanism is provided to represent the global motion of macroblocks when they do not contain residues after vector prediction and coding. Since the skip motion vector [8] is predicted and the skip macroblock is motion compensated at the decoder side, a skip macroblock does not consume any bitrate.

To increase the fast motion estimation performance, early skip macroblock is determined using zero-block detection criteria in motion search. The detection is performed by the SAD comparison using equation in [26]. Before the motion search with the possible modes, the skip predicted macroblock according to skip motion vector prediction is compared with the current macroblock. If the difference is smaller than a threshold, which depends on QP (Quantization Parameter) value, the macroblock is marked as skip macroblock and no more motion search is performed. This approach may cause a small PSNR drop, but improves the motion estimation performance significantly. The QP based threshold equation is:

$threshold = M.N.25.QP/192$ (M = block size x, N= block size y) **(5.1)**

$SAD(MB_{Skip}) < 100.QP / 3$ (for skip MB)

5.1.1.2 Motion Search Algorithm

As described in [7], it is well-known that video coding derives most of its coding efficiency advantage from motion estimation because it removes the huge video redundancy in temporal domain significantly. On the other hand, the motion estimation contributes the heaviest computational load for the whole video encoding. A good video encoder algorithm implementation needs to keep a good balance between computational intensity and coding efficiency.

Even with the use of early termination, Full Search motion estimation is too computationally intensive for many practical applications. In computation or power limited applications, so-called 'fast search' algorithms are preferable. These algorithms operate by calculating the energy measure (e.g. SAE, SAD) at a subset of locations within the search window.

In section 4.1.4, the motion estimation process is described with the implemented algorithms. The motion estimation kernel is responsible for finding out the minimal distortion for various blocks sizes of H.264. In the H.264 encoder, to decrease the computational over-head and consumed time against the Full Search, the Three Step Search (TSS, sometimes described as N-step search) algorithm (Figure 5.3) is implemented with N = 4 for -/+ 16 search area. In Figure 5.3, the step locations are numerated with the step number as well. In TSS's each step, search center is set as the position with minimum energy value and in each step the search distance is halved till it is no longer divisible by 2 (the search termination condition).

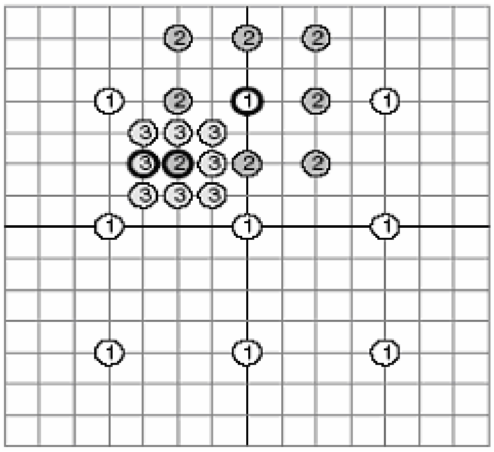


Figure 5.3: Three Step Search [13]

The TSS is considerably simpler than Full Search (8N + 1 searches compared with $(2^{N+1} - 1)^2$ searches for Full Search) but the TSS (and other fast search algorithms) do not usually perform as well as Full Search. A block containing complex detail and/or different moving components may have several local minima. While the Full Search will always identify the global minimum, a fast search algorithm may become 'trapped' in a local minimum, giving a suboptimal result [13].

In the proposed encoder, for better performance and results rather than the TSS, the hexagon-based search [27] pattern, illustrated in Figure 5.4, is used in motion estimation. At the beginning, search center is set as either the (0, 0) vector or the median prediction vector within the defined search window, depending on whichever gives lower distortion. Afterwards, estimation is performed with hexagon pattern with a fixed distance (i.e. 2 pixels), and by choosing the lowest energy position as the new search center. The hexagon search is terminated if the minimum energy is at the center of the hexagon pattern, so that this approach contains self early termination condition.

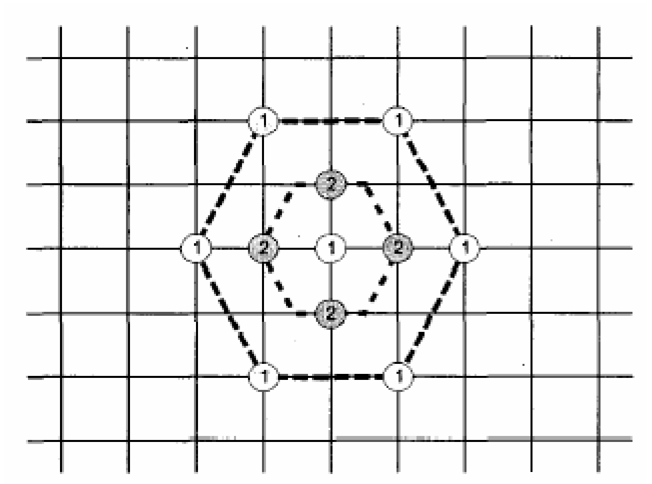


Figure 5.4: Hexagon-based search with large (1) and small (2) patterns [27]

The 4-step search and then hexagon-based search algorithms are implemented in motion estimation kernel, and they are also ported to the embedded software for fast estimation with minimal computational power. The result on 'foreman' sequence shows that the hexagon-search pattern with self-termination gives better performance result that the fixed step size step search algorithm. In Table 5.1, the search algorithm real-time result for ‘foreman’ sequence is given with time and bitrate results on the target platform.

Table 5.1: TSS and Hexagon Search results on DM642 DSP

Search Algorithm	**Total Elapsed Time**	**Average MB search**	**Total Encode Time**	**Compression ratio**
TSS (N = 4)	538.15 ms	468.6 µs	885.86 ms	60
Hexagon-based	459.75 ms	400.3 µs	763.42 ms	70
Foreman CIF video sequence with 30 frames (1 I-Slice and 29 frame ME)				

As shown in Table 5.1, with migration from the N-step to hexagon search, the proposed encoder is optimized nearly 18% in motion estimation search process, 13.8% in MB encoding process respectively. Consequently, 10% of the overall performance enhancement and better bitrate compression on the target embedded platform is very satisfactory for a real-time implementation.

5.1.2 C Level Optimization

The development steps are considered as the software is going to run on an embedded DSP platform. In program level, there are platform independent C level arrangements and optimizations, such as memory allocation and pointer exchange mechanism in reference frames for avoiding unnecessary copy operations. Besides, the structures and variables are arranged according to embedded architecture.

The platform independent software level optimization includes variable types, constants and structures as well as the locality of frequently used sections and also function in-lining. The platform dependent intrinsics, library functions are the other C level optimization performed on the proposed encoder software. Moreover, the compiler options of the DSP code compiler is the common phase of program level optimizations.

5.1.2.1 Structure and Variable Level Optimization

In an embedded platform, dynamic memory allocation can vary according to memory management unit. Frequent memory allocations can be prevented with a single allocation and pointer exchanging rather than copying the data. As shown in Figure 5.5, reference frame and reconstructed frames are allocated at the beginning of the program and pointers are used to exchanging the data because reconstructed frame is reference frame for the next sequence respectively.

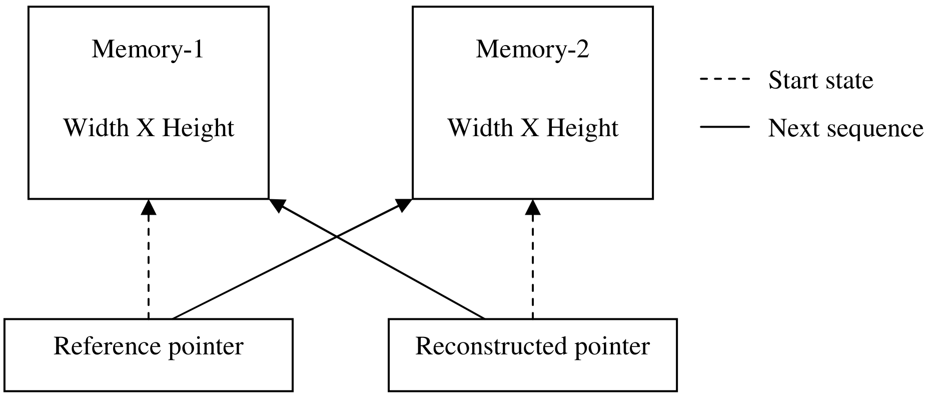


Figure 5.5: Memory/Pointer structure for reference and reconstruction

In C6000 DSP architecture, integer is 4 bytes, short integer is 2 bytes and character type is 1 byte long respectively. If a load and store operation differs in variable type, as you read from an integer and write to character type, these operations are by default performed element by element (one data element within an instruction), even though it can be performed for multiple data elements. Therefore, the variable types and structures are arranged to the minimum required sizes for processing more than one data element within a single instruction using the DSP architecture (e.g. special SIMD instructions of the architecture).

In the proposed encoder structure, 16x16 reconstruction, 16x16 prediction and 16x16 original macroblock arrays are arranged and cast to unsigned character type as they need one byte elements. As a result, multiple load, store or SIMD instruction can be performed in an instruction. These block arrays are copied from/to the big picture using EDMA controlled as described in section 5.2.2.

5.1.2.2 Inline Functions

When an inline function is called, the C/C++ source code for the function is inserted at the point of the call. This is known as inline function expansion. Inline function expansion is advantageous in short functions for the following reasons:

- It saves the overhead of a function call.
- Once inlined, the optimizer is free to optimize the function in context with the surrounding code.

There are several types of inline function expansion [28]:

- Inlining with intrinsic operators (intrinsics are always inlined)
- Automatic inlining
- Definition-controlled inlining with the unguarded inline keyword
- Definition-controlled inlining with the guarded inline keyword

Expanding functions inline increases code size, especially inlining a function that is called in a number of places. Function inlining is optimal for functions that are called only from a small number of places and for small functions.

In the proposed H.264 encoder, some functions are inlined with 'static __inline' keyword to speed up with the inlining advantages. In the proposed encoder, the MVcost function, which calculates motion vector costfor Lagrange formula, sign function, and clipping operation for saturation functions are inlined for performance optimization.

5.1.2.3 C level Compiler Intrinsics

The first optimization step that can be performed on C source code for the TMS320C64x is to use intrinsic operators. Intrinsics are used like functions and produce assembly language statements that would otherwise be inexpressible in C. The problem is that once you have performed the first optimization step, your C source code is no longer ANSI C compatible.

The code proposed within application report [29], allows writing C code using intrinsic operators keeping the possibility to validate the code on a workstation. The C6000 compiler recognizes a number of intrinsic operators that allow expressing the meaning of certain assembly statements that would otherwise be cumbersome or inexpressible in C/C++. The intrinsics are specified with a leading underscore '_', and are accessed by calling them as a function. They correspond to the indicated C6000 assembly language instruction(s) [28,30]. By using the C intrinsic, access to assembly statements from C level is provided; as a result, optimized code instructions are executed. In [28], a table list for C intrinsics can be found.

```
/* C code*/
int abs(a)
{
if(a < 0)
      return -a;
else
      return a;
}

/* C Intrinsic */
int x = _abs(a);
```

```
/* C code */
int min(a, b)
{
if(a < b)
      return a;
else
      return b;
}

/* C Intrinsic */
int x = _min2(a, b);
```

Figure 5.6: C program code and corresponding intrinsics

In the proposed H.264 encoder software as shown in Figure 5.6, _abs intrinsic is used for absolute of an integer value, _abs2 and _dotp2 intrinsics are used for parallel addition of two absolute values for hadamard transform. The _min2 and _max2 intrinsics are used for minimum and maximum value comparison, and lastly, the _mem4 intrinsic is used for various unaligned memory load/store operations. These intrinsics slightly increases the encoder performance as they are in-lined with the corresponding instruction to the software.

5.1.2.4 Fast Library Functions

The Texas Instruments C64x Image Library [31] and DSP Library [32] are optimized Image/Video Processing and DSP Functions Library for C programmers using TMS320C64x devices. It includes many C-callable, assembly-optimized, general-purpose image/video processing and DSP routines. These routines are typically used in computationally intensive real-time applications where optimal execution speed is critical. By using these routines, achieving execution speeds considerably faster than equivalent code written in standard ANSI C language becomes possible. In addition, by providing ready-to-use Image and DSP functions can significantly shorten the image/video processing application development time.

In a video encoder, motion estimation or motion search has great computational overhead, because the original macroblock is searched over the reference picture in pixel by pixel approach. For the resulting motion, SAD (Sum of Absolute Differences) calculation is used in the proposed encoder. TI's image library provides 8x8 and 16x16 block size SAD calculation functions so that SAD calculation on a single 8x8 block with IMG_sad_8x8 speeds up the encoder.

Function	*IMG_sad_8x8(unsigned char *src, unsigned char *ref, int pitch)*
Arguments	*src[64]* 8x8 source block. Must be double-word aligned.
	ref[] Reference image.
	pitch Width of reference image.

Another C-callable DSP function is DSP_dat_mul [32] (DSP matrix multiplication), which can be used in hadamard transform, hadamard based SATD, integer transform and inverse transform in the H.264 encoder.

Function	*DSP_mat_mul(short *x, int r1, int c1, short *y, int c2, short *r, int qs)*
Arguments	*x [r1*c1]* Pointer to input matrix of size r1*c1.
	r1 Number of rows in matrix x.
	c1 Number of columns in matrix x. Also number of rows in y.
	*y [c1*c2]* Pointer to input matrix of size c1*c2.
	c2 Number of columns in matrix y.
	*r [r1*c2]* Pointer to output matrix of size r1*c2.
	qs Final right–shift to apply to the result.

The source block must be double word aligned [31] for the correct calculation of SAD function. The pragma directive ‘DATA_ALIGN’ should be used for the source block for the double word alignment.

However, the IMG_sad_8x8 function is restricted to source blocks that have to be copied from the original picture with 8x8 sizes. In the proposed H.264, macroblocks are copied (e.g. data transferred with EDMA) in 16x16 size. In section 5.1.5.1, linear assembly of SAD function is written using C64x instruction set (as fast as image library function) for flexibility.

5.1.3 Compiler Options for Optimization

In C64x, which have VLIW architecture, having a powerful compiler eases the implementation process for the H.264 encoder. Compiler of C64x DSPs has different options which can be set through CCStudio and creates the executables from the source code according to options. The strategy for optimizing the code using these options determines the H.264 encoder performance. In Table 5.2, some of the critical and efficient compiler options and their descriptions are listed.

Table 5.2: C64x Compiler Options for Performance

Option	**Description**
Speed Most Critical (no -ms)	The first strategy to determine the optimization type (code-size vs. speed)
-o3	File-level optimization option for the highest level of optimization available. Software pipelining, loop unrolling, SIMD are applied (-oN determines the optimization level)
-pm	Program-level optimization, combines source files for full pipeline utilization and performance
-mt	Allows the source code and assembly optimizer to assume there is no memory aliases in code, i.e., no memory references ever depend on each other
No Debug	Exclude the debug info from the output file, so provides much more parallelized code

In the C64x compiler, the compiling and optimizing are worked out combining the options together. For the proposed H.264 encoder, the optimization for speed is much more critical than the code size formed by the C64x compiler. First of all, the compiler optimization strategy is set to Speed Most Critical (no -ms) for compiler performance. The following combination of the optimization levels offers the best performance for the proposed encoder.

The -o3 instructs the compiler to perform file-level optimization. Even though the optimization level can be used alone for general file-level optimization, there are several options to perform specific optimizations. Also, Software pipelining is turned on in this compiler option, which parallelizes instructions, fills delay slots and maximizes functional unit [11]. More detailed information about file level optimization can be found in [28].

In Code Composer Studio, program-level optimization is specified by using the -pm option with the -o3 option. With program-level optimization, all of source files are compiled into one intermediate file called a module. The module moves to the optimization and code generation passes of the compiler. Because the compiler can see the entire program, it performs several optimizations that are rarely applied during file-level optimization:

- If a particular argument in a function always has the same value, the compiler replaces the argument with the value and passes the value instead of the argument.
- If a return value of a function is never used, the compiler deletes the return code in the function.
- If a function is not called directly or indirectly by main, the compiler removes the function.

The -mt option informs the compiler that it can make certain assumptions about how aliases are used for memory addresses and pointers in your code. These assumptions allow the compiler to improve optimization. The -mt option also specifies that loop-invariant counter increments and decrements are non-zero. Loop invariant means the value of an expression does not change within the loop.

The -mt option indicates that a pointer to an object type does not alias (point to) an object of another type. Indirect references on two pointers do not alias, so each subscript expression in an array reference is in the range and there are no loop-invariant counter increments and decrements of loop counters [28]. If source code contains any of these aliasing techniques the -mt option should not be used and unexpected results may occur. The proposed H.264 encoder is rearranged not to contain any of these aliasing, so -mt option is use to improve the optimization.

After specifying the compiler options for the highest performance for the code, the code size and speed is improved with passing the debug information with 'No Debug' option. The debug information kept in the compiled output file is excluded, as a result, the code size decreases. Additionally, the compiler can perform better software pipelining and instruction-level parallelism, while there is no user information partitions in compiled output file.

5.1.4 Linear Assembly

When the compiler does not fully exploit the potential of the 'C6000 architecture, better performance may be obtained by writing the loop/function in linear assembly, and then linear assembly code is the input for the assembly optimizer [25]. To enhance the performance of a video/image process on such a DSP architecture, the linear assembly is the key coding feature for using the pipeline parallelism with powerful SIMD instructions (i.e. Table 3.1) as the inexpressible C program level operations and calculation can be specified in linear assembly easily.

Linear assembly is similar to regular 'C6000 assembly code that it uses 'C6000 instructions to write your code. With linear assembly, however, it is not needed to specify all of the information that you need to specify in regular 'C6000 assembly code. Software expert decides whether to specify the information or let the assembly optimizer specify.

Here is the information that is not needed to specify in linear assembly code:

- Parallel instructions
- Pipeline latency

- Register usage
- Which functional unit is being used

If they are not specified, the assembly optimizer determines the information that is not included, based on the information that it has about the code. As with other code generation tools, linear assembly code might be needed to be modified up to a satisfactory performance. During linear assembly coding, much more detail to assembly can be added, such as specifying which functional unit should be used. The important regulars of linear assembly code writing are:

- A linear assembly file must be specified with a '.sa' extension.
- Linear assembly code should include the '.cproc' and '.endproc' directives. The .cproc and .endproc directives delimit a section of code that is optimized by assembly optimizer. Use .cproc at the beginning of the section and .endproc at the end of the section.
- Linear assembly code may include a '.reg' directive. The .reg directive allows using descriptive names for values that will be stored in registers. When .reg directive is used, the assembly optimizer chooses a register whose use agrees with the functional units chosen for the instructions that operate on the value.
- Linear assembly code may include a '.trip' directive. The .trip directive specifies the values indicating how many times a loop will iterate.

In the proposed encoder, the performance optimization of the critical and unutilized code segments and functions are written in linear assembly. Furthermore, the SIMD instructions of the C6000 DSP are used with balanced side effect (DSP's A/B side operational units and registers) for pipelining.

5.1.4.1 Linear Assembly of Critical Functions

By writing the linear assembly, the pipeline utilization and parallel video/image processing with SIMD instructions can be provided. In H.264 encoder, the time consuming operations are profiled and the appropriate linear assembly codes are written for C64x DSP. Also, the SIMD instructions are covered in that part for the best performance.

The motion estimation has the highest computational complexity for a video encoder that the original macroblock is searched over a reference picture area. DM64x provides a rich set of extensive video/image instructions that can implement effectively SAD based motion estimation scheme. In the proposed encoder, the 8x8 SAD function is written in linear assembly, and this function can be expressed as in Figure 5.7.

```
      ZERO  SAD

      . . .

LDDW  *org_ptr++[ORG_WIN], org8:org4  ; org[7-4]:org[3-0]
LDNDW *win_ptr++[SRCH_WIN], win8:win4  ; win[7-4]:win[3-0]

      SUBABS4     org4, win4, absdif4
      SUBABS4     org8, win8, absdif8

      DOTPU4      absdif4, dot_sad, SAD_tmp4
      DOTPU4      absdif8, dot_sad, SAD_tmp8

      ADD         SAD_tmp4, SAD, SAD
      ADD         SAD_tmp8, SAD, SAD
```

Figure 5.7: Linear assembly fragment of SAD function

In Figure 5.7, load non-aligned double word (LDNDW) may read a 64-bit value with any byte boundary. This instruction is important to accelerate the data fetching from the MB especially in searching window in reference frame. It can easily fetch eight aligned pixels from non-aligned pixels from searching window, which allows expanding memory bandwidth usage.

On the other hand, subtract with absolute value (SUBABS4) instruction calculates four absolute values of the difference between the packed 8-bit data contained in the source registers. DOTPU4, an important video/image instruction returns the dot-product between four pairs of packed 8-bit values. Since two DOPTPU4 can run in parallel in a single cycle, this instruction accelerates the sum of absolute difference (SAD) process significantly that is the core for motion estimation.

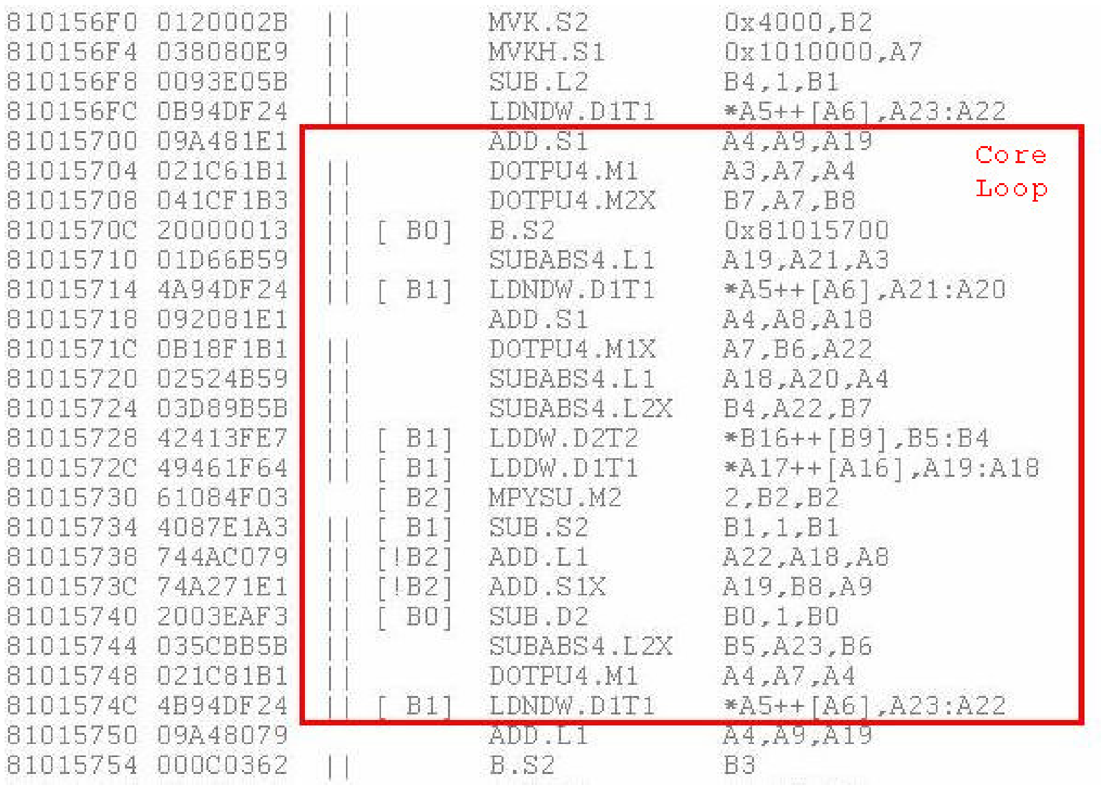
The idea of SAD kernel can be summarized in the following steps:

1. LDDW and LDNDW fetches 8 pixels from current and reference frame

2. Two SUBABS4 calculate 8 absolute difference

3. Two DOTPU4 accumulate 8 result addition

The given code for a SAD line is expanded for 8x8 block with looping, and the '.trip' directive is used to specify the loop will iterate at defined amount. In Figure 5.8, the core SAD iteration code is disassembled that fully utilizes the architecture: SIMD instructions and pipeline fullness.



```
810156F0 0120002B  ||         MVK.S2       0x4000,B2
810156F4 038080E9  ||         MVKH.S1      0x1010000,A7
810156F8 0093E05B  ||         SUB.L2       B4,1,B1
810156FC 0B94DF24  ||         LDNDW.D1T1   *A5++[A6],A23:A22
81015700 09A481E1             ADD.S1       A4,A9,A19
81015704 021C61B1  ||         DOTPU4.M1    A3,A7,A4
81015708 041CF1B3  ||         DOTPU4.M2X   B7,A7,B8
8101570C 20000013  || [ B0]   B.S2         0x81015700
81015710 01D66B59  ||         SUBABS4.L1   A19,A21,A3
81015714 4A94DF24  || [ B1]   LDNDW.D1T1   *A5++[A6],A21:A20
81015718 092081E1             ADD.S1       A4,A8,A18
8101571C 0B18F1B1  ||         DOTPU4.M1X   A7,B6,A22
81015720 02524B59  ||         SUBABS4.L1   A18,A20,A4
81015724 03D89B5B  ||         SUBABS4.L2X  B4,A22,B7
81015728 42413FE7  || [ B1]   LDDW.D2T2    *B16++[B9],B5:B4
8101572C 49461F64  || [ B1]   LDDW.D1T1    *A17++[A16],A19:A18
81015730 61084F03     [ B2]   MPYSU.M2     2,B2,B2
81015734 4087E1A3  || [ B1]   SUB.S2       B1,1,B1
81015738 744AC079  || [!B2]   ADD.L1       A22,A18,A8
8101573C 74A271E1  || [!B2]   ADD.S1X      A19,B8,A9
81015740 2003EAF3  || [ B0]   SUB.D2       B0,1,B0
81015744 035CBB5B  ||         SUBABS4.L2X  B5,A23,B6
81015748 021C81B1  ||         DOTPU4.M1    A4,A7,A4
8101574C 4B94DF24  || [ B1]   LDNDW.D1T1   *A5++[A6],A23:A22
81015750 09A48079             ADD.L1       A4,A9,A19
81015754 000C0362  ||         B.S2         B3
```

Figure 5.8: Disassembly of SAD linear assembly's core loop

In Table 5.3, the SAD 8x8 function's performance results are given. The overall motion search kernel is optimized with 4x speed up factor that is the core computationally intensive part of the encoder. The intermediate and after results differ because SAD function is rewritten by changing LDNDW to LDDW instruction for original macroblock read operation, which is double word aligned in internal memory with pragma directives.

Table 5.3: SAD 8x8 linear assembly results

Function	Before	Intermediate	After	Speed Ratio
SAD_8x8	195.7 μs	65.1 μs	48.2 μs	4x

In the proposed encoder the other time consuming operations, transform and inverse transform, are also written in linear assembly. The transform process is parallelized with butter-flying the transform matrix for using ADD2, SUB2, SHR2, PACK2, PACKH2 instructions that can calculate two pairs of data with only an instruction. Beside the SATD process is unlocked for parallelization and appropriate SIMD instructions such as DOTP2, ABS, ADD2, etc. The linear assembly code of 4x4 integer transform in residual coding and SATD in intra mode decision accelerates the processes slightly compared to SAD function.

5.1.4.2 Half-Pel Motion Compensation Optimization

As the encoder supports half-pel motion search and compensation, the half-pel motion compensation [8] is optimized for H.264 encoder by pre-calculation. The whole image is interpolated in horizontal and vertical direction before the motion estimation process. This approach eliminates the unnecessary recalculation of macroblock motion interpolation in post processing.

The half-pel motion compensation is written in linear assembly for parallel calculation of half-pel positions using SIMD instructions. The 6-tap filter in H.264 is used to calculate the half-pel position and sub-pel predictions can be parallelized with line based filtering using SIMD and reduced cache latency is obtained using L2 internal SRAM memory. To improve the half-pel motion compensation, the lines to be filtered are fetched to internal memory using EDMA. The ping-pong buffer mechanism as illustrated in Figure 5.9 is used to parallel (hidden) transfer of compensation buffers.

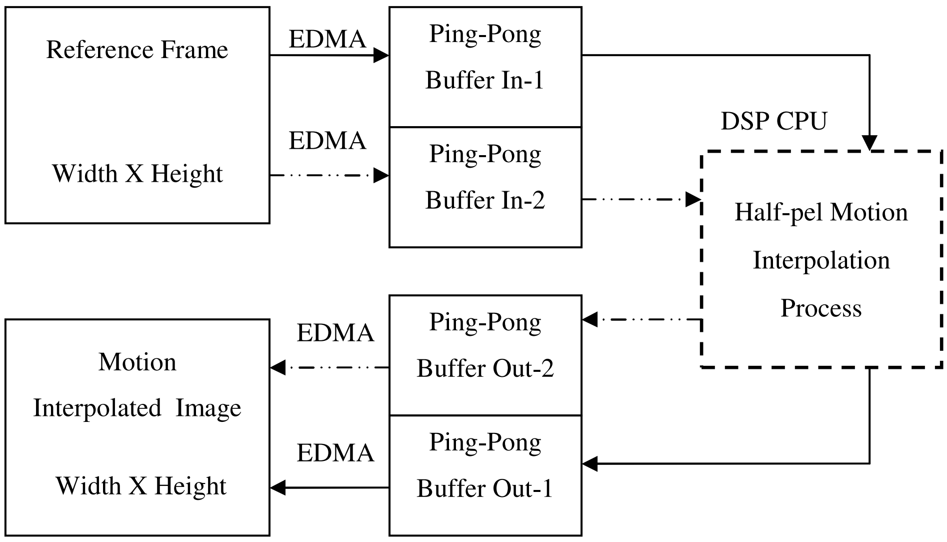


Figure 5.9: Half-pel motion compensation and ping-pong buffers

The half-pel motion compensation takes the fetched memory line and outputs the compensated line to internal memory as shown in Figure 5.9. The two separate Ping-Pong buffers are used for background EDMA transfer. After a line is fetched by EDMA, the second line fetch is started. At that time, the compensation of the first line is started in parallel. As a result, the fetch of second and the other lines are parallelized with the compensation process. For vertical interpolation, the column-based interpolation is converted to line-based compensation by using the EDMA column transfer policy.

Table 5.4: Half-pel motion compensation optimization results

Compensation Direction	**Before (C code)**	**After (ASM + EDMA)**	**Optimization Ratio**
Horizontal	693.1 µs	268.3 µs	~2.6x
Vertical	5815 µs	393.5 µs	~14.8x
Diagonal	Not implemented in proposed encoder		

In Table 5.4, the result of horizontal and vertical motion compensation optimization is given. The optimization ratio is much more aggressive in vertical motion compensation because the column-based interpolation is turned to line-based interpolation using the EDMA controller's 2-D data transferring feature.

5.2 Memory/Cache Optimization

As the memory requirement for the encoder is significant, accessing lots of data and spending much more time for waiting data that is stored in off-chip external memory occurs because of the video encoder nature. As told in [20], using cache is the easiest way to solve the problem, but suffers high memory latency in case of memory misses. In DSP cores, as in DM642, with smaller cache size, cache misses significantly effects the encoder performance. The DM642 on-chip internal L2 SRAM (256 Kbytes) can be set as both internal RAM and Cache together. For avoiding the cache misses caused by the small L1 cache, 128 Kbytes L2 cache space is set as specified in section 5.2.1.

Another way to avoid cache misses of external memory accesses; Direct Memory Access (DMA) is used to fetch required data to internal memory before processing. As DM642 core includes EDMA and Quick DMA that is designed to transfer required data to temporary buffers in internal memory, so that the core should not encounter any memory latency. For the temporary buffer usage, the L2 internal memory should be configured to contain RAM space as well.

As described in section 5.2.2, the temporary buffers and the flow of the H.264 encoder are adjusted to use Quick DMA for data transfer in parallel (hidden) with core processing. Additionally, non-transferred frequently used temporary buffers, such as 4x4 block difference buffer for integer transform, 16x16 prediction buffer, etc. are put in to internal memory for locality and low-delay access time for those buffers in encoder flow.

5.2.1 Internal L2 SRAM Configuration

The DM642 two-level cache architecture, as in Figure 3.5**,** is a combination of L1 (total 32 Kbytes) cache and a programmable 265 Kbytes L2 SRAM memory. As the proposed encoder needs room for temporary buffers for DMA transfer operations and low latency access, the L2 cache is configured as a combination of cache and internal memory. The optimal cache size for the proposed encoder on DM642 is selected as 128 Kbytes. Therefore, the maximum internal memory space for the encoder is 128 Kbytes mapped SRAM within the DM642 internal memory as shown in Figure 3.4.

In DM642 128 Kbytes L2 internal memory, the original macroblock, -/+ 16 search window and luma/chroma reconstruction temporary buffers are used in DMA data transfers. Besides, the prediction (intra or inter), difference and ping-pong buffers are internal memory elements for low latency. In Table 5.5, the temporary buffers used in the proposed encoder are listed with the usage description.

Table 5.5: Data elements in DM642 L2 SRAM section

Data Element	**Data Type&Size**	**Description**
Original luma & chroma Macroblock	unsigned char [16][16] [2][8][8]	The original MB values are encoded and used in motion search, prediction error calculation, SAD etc. The buffers are transferred via (Quick) DMA.
Luma Search window	unsigned char [48][48]	As the MB size is 16x16, -+ 16 search window for luminance component in motion estimation kernel. Also, The buffer is transferred via (Quick) DMA.
Luma prediction buffer	unsigned char [4][4][4]	Four 4x4 predictions of a sub-block 8x8.
Chroma prediction buffer	unsigned char [2][8][8]	Two 8x8 prediction buffer for U and V components of chrominance and used in reconstruction.
Luma difference buffer	short int [4][4][4]	Four 4x4 prediction error buffers for sub-block 8x8. The quantization/de-quantization, zigzag scan, integer and inverse transforms are performed on that buffer.
Luma & Chroma reconstruction buffer	unsigned char [16][16] [2][8][8]	The reconstruction buffers of decoding process of MB's YUV components. Also, the buffers are transferred via EDMA to reconstructed (next reference) picture.

In the linker command the internal L2 SRAM memory section name is defined as 'mysect' related to the DSP/BIOS memory/cache configuration. In Figure 5.10, the pragma directive 'DATA_SECTION' is used to allocate a data array or temporary buffer (Table 5.5) within the internal memory, and also pragma directive 'CACHE_LINE_SIZE' (128 bytes for DM642) is used to align the data address to cache line size boundary to increase the cache hits for the line pages and decrease the cache miss penalties as possible as can be.

```
#pragma DATA_SECTION(window,".mysect");
#pragma DATA_ALIGN(window, CACHE_L2_LINESIZE)
unsigned char window[SRCH_WIN][SRCH_WIN];

#pragma DATA_SECTION(orgMB_16x16,".mysect");
#pragma DATA_ALIGN(orgMB_16x16,
CACHE_L2_LINESIZE)
unsigned char orgMB_16x16[16][16];

#pragma DATA_SECTION(diff4x4,".mysect");
#pragma DATA_ALIGN(diff4x4, CACHE_L2_LINESIZE)
short diff4x4[4][4][4];

#pragma DATA_SECTION(pred4x4,".mysect");
unsigned char pred4x4[4][4][4];

#pragma DATA_SECTION(luma_rec,".mysect");
#pragma DATA_ALIGN(luma_rec, CACHE_L2_LINESIZE)
unsigned char luma_rec[16][16];

// Chroma Elements
#pragma DATA_SECTION(orgUV8x8,".mysect");
#pragma DATA_ALIGN(orgUV8x8, CACHE_L2_LINESIZE)
unsigned char orgUV8x8[2][8][8];

#pragma DATA_SECTION(chr_rec,".mysect");
#pragma DATA_ALIGN(chr_rec, CACHE_L2_LINESIZE)
unsigned char chr_rec [2][8][8];

#pragma DATA_SECTION(chr_pred,".mysect");
#pragma DATA_ALIGN(chr_pred, CACHE_L2_LINESIZE)
unsigned char chr_pred[2][8][8];
```

Figure 5.10: Allocation of frequently accessed buffers within internal memory

While the memory sections' efficient usage is crucial for a video encoder design, allocating the frequently accessed arrays within the on-chip memory reduces the data read/write cache miss penalties as well as the CPU stalls because of the memory request. As a result, the proposed encoder performance is optimized using the memory architecture of the DSP core with an appropriate configuration.

5.2.2 EDMA Controller Usage

The H.264 encoder must access significant amount of data that is stored in off-chip external memory. For optimizing the performance for latency, on-chip L2 SRAM is configured as a cache, the instruction and data elements are transferred from/to the external memory by the cache controller before they are used. However, as declared in [5,6] if data exchange depends only on the cache controller, it is hard to schedule the transfer processes in reason. Besides, there is no data superiority for the cache controller; the frequently used data may be replaced with the rarely employed data because of the code flow.

Since the frequently used one is going to be re-rolled in soon, the system's efficiency is declined. An alternate method for data scheduling is directly controlling the EDMA controller [6]. By using the EDMA controller of DM642 [35], the background (hidden) data transfers between external and internal memory are maintained. Therefore, high latency memory accesses are limited to internal memory cache miss penalty respectively.

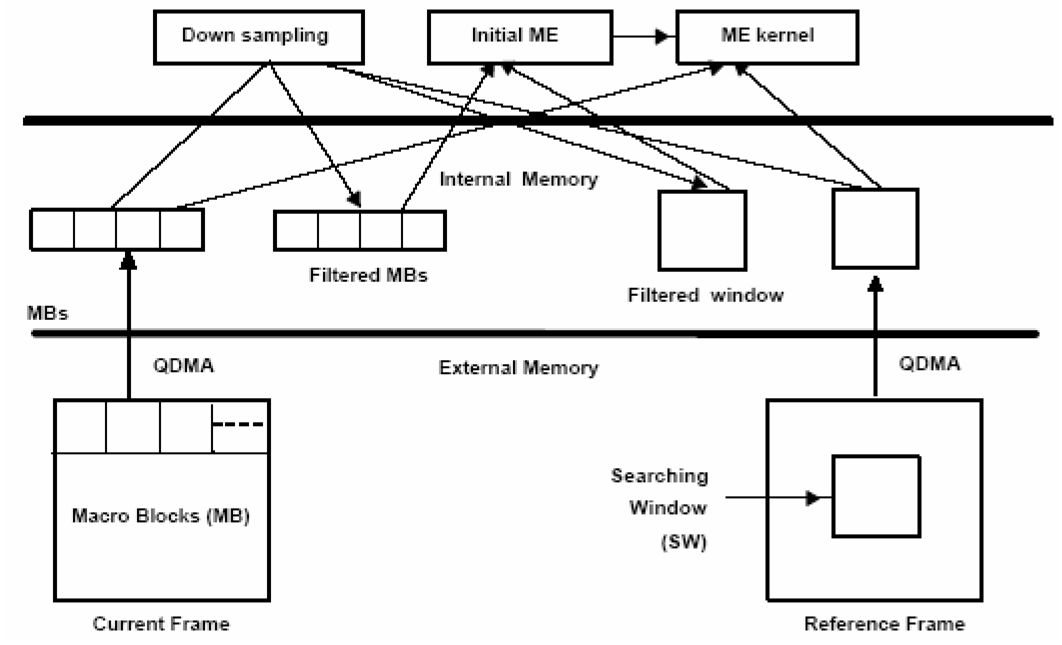


Figure 5.11: QDMA management for Motion Estimation [7]

In Figure 5.11, a main flow of a video encoder's motion estimation is figured out using Quick DMA and internal temporary buffers [7]. The parallelism of the CPU and EDMA architecture is used for data transfer scheduling for the performance of the video encoder.

The enhanced direct memory access (EDMA) controller of the DM64x devices is a highly efficient data transfer engine, capable of handling up to 8 bytes per EDMA cycle, resulting 2.4GB per second of total data throughput at a CPU rate of 600 MHz.

To make a video encoder application fully benefit from the bandwidth in the transfer engine, it is best to use 32-bit element size whenever possible.

In EDMA communication, each data transfer is initiated by a transfer request (TR), which contains all the information required to perform the transfer: source address, destination address, transfer property, element count, etc. When a transfer request is shifted into one of the transfer request queues to wait for processing, the transfer priority level determines the queue to which it is sorted. There are four queues: Q0(urgent), Q1(high), Q2(medium), and Q3 (low) corresponding to four priority levels, each with a depth of 16 entries. Only one TR from each priority queue can be serviced at a time by the address generation/transfer logic. When a TR arrives at the head of queue, it is moved into the EDMA transfer Controller queue registers, which perform the actual data movement defined by the TR [7].

The EDMA has the capability of performing unsynchronized transfers through the use of a QDMA request by the CPU. In other word, QDMA transfer is synchronized by the CPU. In video encoder, the EDMA transfer is synchronized by data flow of the algorithm instead of external events. The QDMA is better to issue a single, independent transfer to quickly move data, rather than to perform periodic or repetitive transfers like the other EDMA channels. The requests are queued according to priority, with higher priority requests services first processed by the EDMA. Because of the EDMA structure, all QDMA transfers are submitted using frame synchronization. Therefore, the QDMA always requests a transfer of one complete frame of data. A good video encoder should use all three priority queues (low, medium and high) in parallel to transfer data between external memory and internal on-chip buffers.

In the proposed encoder the QDMA transfer scheme for current macroblock, search window and reconstructed macroblock is drawn in Figure 5.12. As well as the luminance component, the halved two chrominance component buffers are allocated within internal memory and the QDMA is used to transfer required data to temporary buffers in internal memory beforehand, so that the DM642 core should not face any memory latency. The code flow of the H.264 encoder is adjusted to make EDMA memory fetch in parallel (hidden) with core processing.

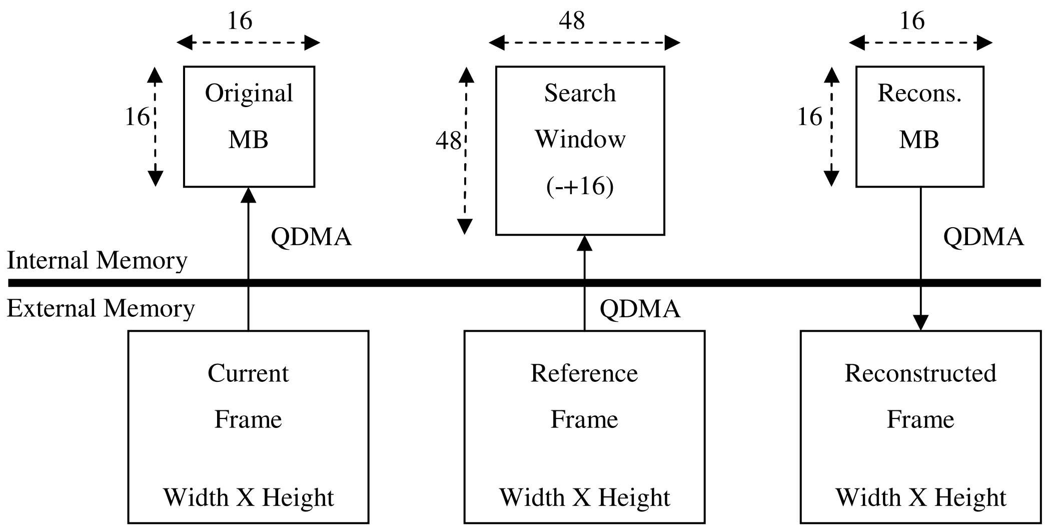


Figure 5.12: The proposed encoder QDMA scheme for luminance

5.3 The H.264 Encoder Optimization Results

The DM642 DSP core is capable of video/image processing using the hardware architecture and also instruction level video/image operations. The proposed H.264/AVC baseline encoder is targeted to real-time application with CIF (352x288) resolution above 25 fps encoding speed as well. For achieving the target rate, a video encoder has to fully utilize the core's resources as much as possible.

The H.264/AVC encoder is optimized under the DSP environment from software (C language) level to hardware level using the DSP APIs, register level programming and intrinsics as well as linear assembly language for SIMD instructions and parallel processing the image pixels respectively. The software encoder performance is improved for achieving the target encoding rate in real-time implementation.

5.3.1 The H.264 Encoder Optimization Summary

The proposed H.264/AVC encoder on DM642 DSP performance optimization steps cover program level optimizations and DSP hardware usage such as EDMA controller and VLIW architecture (e.g. SIMD instructions in linear assembly). In section 5.1.2 the C program level optimizations; structures, function/intrinsic inlining and fast library functions achieve software optimization with little effort on the proposed encoder. However, the DSP hardware usage needs expertise especially in

EDMA and data scheduling and occupation on DSP software with linear assembly, but serves significant results on the encoder optimization.

The 2-level cache/memory hierarchy of DM642 core and configurable L2 internal memory space have to be considered as the bottleneck for speed optimization for video/image processing that requires high memory bandwidths. Therefore, a L2 configuration as both cache and internal RAM is used to maximize the cache hits with larger cache memory and minimize the cache miss penalty with internal memory for locality and frequently accesses data elements. As describe in section 5.2.1, the 256 Kbytes L2 SRAM is configured as 128 Kbytes cache and 128 Kbytes internal RAM.

Table 5.6: Optimization results for CIF [352x288] 'Foreman' sequence

Functional Block		**Total Count @ 30 frames**	**Total Elapsed Time (ms)**	**Average Time (ms/frame)**	**Percentage (%)**
Before & after encode (EDMA usage)		All MBs	133.38 ms	4.45 ms	11.16 %
Encode MB		All MBs	763.42 ms	25.45 ms	63.89 %
Encode MB	Motion Estimation	11484 MBs @ 29 frames	459.75 ms	15.85 ms	Encode MB: 60.22 %
	Residual Coding	All MBs	228.50 ms	7.62 ms	Encode MB: 29.93 %
	Intra Predict & Others	All MBs	75.17 ms	2.51 ms	Encode MB: 9.85 %
Deblocking Filter		30 frames	132.96 ms	4.43 ms	11.13 %
VLC (write NALU)		All MBs	77.98 ms	2.60 ms	6.53 %
Motion Compensation		@ 29 frames	87.09 ms	3.00 ms	7.29 %
Total		30 frames	1194.8 ms	39.93 ms	All

In Table 5.6, the optimized encoder on DM642 DSP results are measured using CIF (352x288) format for 'foreman' video sequence. The average picture encoding time for this sequence is 35.5ms (Deblocking off) and 39.93 ms (Deblocking on) for 30 frames, so the proposed encoder runs on DM642 at 28.57 fps without deblocking filter and 25.04 fps with deblocking filter. Besides, a real-time application frame rate can be 25 to 30 fps, the H.264 encoder is optimized enough for real-time purpose, achieving encode rate over 25 fps.

5.3.2 Simulation Results with PSNR and Compression Ratio

The average PSNR values for video sequences of 30 frames are measured with the proposed encoder speed. For the compression efficiency in bitrate, constant quantization parameter with a value 28 is used. Average PSNR shows that the picture qualities of the encoder compressed '.264' outputs are high enough for a real-time application. The compression rates change with the video sequence properties with the motion types. In Table 5.7, the encoder compression efficiency is given with encoding speed, PSNR and compression rate with constant QP value at 28 while the deblocking filter is on. The results show that the proposed encode speed is above 25 fps and the compression efficiency is well suited for a real-time implementation such as video surveillance.

Table 5.7: Compression efficiency of the implemented H.264 encoder

Sequence [CIF 352x288]	**Encoder Speed (fps)**	**Y-PSNR (dB)**	**U-PSNR (dB)**	**V-PSNR (dB)**	**Compression Ratio**
Akiyo	40.42 fps	39.50	42.46	44.05	258
Container	30.17 fps	35.96	42.16	41.89	105
Foreman	25.04 fps	36.02	40.69	43.27	70
Mother&Daughter	38.95 fps	38.85	43.63	44.53	281
News	34.29 fps	37.65	40.00	41.66	128
Paris	26.08 fps	35.21	38.63	38.71	48

The medium motion low detail and low motion medium detail video sequences, such as 'Foreman' and 'Paris' respectively, and also low motion low detail video sequences are profiled on DM642 EVM platform as shown in Table 5.7. These types of streams are similar to video conferencing and mobile applications environment, so the corresponding sequences are chosen for profiling in an embedded platform.

6. CONCLUSION AND FUTURE WORK

In this thesis, a real-time H.264 baseline encoder on TI TMS320DM642 digital signal processor at CIF (352x288) resolution is implemented and verified with reference decoder. According to the performance measurements over video sequences, 25 to 40 fps encoding performance is possible and the PSNR measurements are sufficient for embedded applications such as video conferencing and mobile applications.

As the video/image processing system accommodates parallelism, digital signal processors can provide much more parallelized implementation with high performance and flexibility. However, computational complexity and memory accesses are restrictive for the encoder performance especially in embedded targets; the DM642 DSP can overcome the computational complexity with the VLIW architecture as well as the access limitation with the EDMA controller.

The realization and optimization of the proposed encoder on DM642 target platform with the given optimization phases are carried out. From algorithmic to architectural optimizations, the platform independent software optimizations and the platform dependent memory and linear assembly optimizations are derived. After all optimization steps, the overall performance of the proposed H.264 encoder above 25 fps is qualified enough for a real world implementation.

For future work, the proposed encoder efficiency can be improved by adding quarter pixel motion compensation support and error resilience tools. Besides, motion estimation kernel can be improved by using and adapting more appropriate motion search algorithms. As the proposed encoder achieves real-time performance at CIF resolution, development of the encoder performance for higher resolution especially at D1 (720x576) is a challenging future study.

REFERENCES

[1] **Wiegand, T. and Sullivan, G.J.,** July 2003. Overview of the H.264/AVC Video Coding Standard, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, 560-576.

[2] **Werda I., Kossentini F. and Massmoudi N.**, 2006. Analysis and Optimization of UB Video's H.264 Baseline Encoder Implementation on Texas Instruments' TMS320DM642 DSP, *International Conference on Image Processing*, Atlanta, Georgia, USA, 8-11 October, 3277-3280.

[3] **Liang M. and Chen J.**, 2004. A DSP-coprocessor Architecture for Image/Video Processing, 7th *International Conference on Solid-State and Integrated Technology*, Beijing, China, 18-21 October, Vol. 3, 1601-1604.

[4] Joint Video Team (JVT) of ITU-T VCEG and ISO IEC MPEG. Joint Model (JM) Reference Software Version 8.6, http://iphome.hhi.de/suehring/tml

[5] **Wei Z. and Cai C.**, 2006. Realization And Optimization of DSP-based H.264 Encoder, *International Symposium on Circuits and Systems*, Island of Kos, Greece, 21-24 May, 1921-1924.

[6] **Kant S., Mithun U. and Gupta P.S.S.B.K.**, 2006. Real-Time H.264/AVC Video Encoder on A Programmable DSP processor for Video Applications, *8th International Conference on Consumer Electronics*, Las Vegas, USA, 7-11 January, Digest of Technical Papers, 93-94.

[7] **Peng C.,** 2004. Video Encoding Optimization on TMS320DM64x/C64x, *DSP Video Imaging Solution Application Report,* **SPRAA63,** Texas Instruments.

[8] **ITU-T Rec. H.264/ISO/IEC 14496-10 AVC**, 2003. Draft ITU-T recommendation and final draft international standard of joint video specification, *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG.*

[9] **Kamaci, N. and Altunbasak, Y.**, 2003. Performance Comparison of the Emerging H.264 Video Coding Standard With the Existing Standards, *ICME '03 Proceedings 2003 International Conference on Multimedia and Expo*, Baltimore, Maryland, USA, 6-9 July, Vol. 1, 345-348.

[10] **Tamhankar, A. Rao, K.R.**, 2003. An Overview of H.264 / MPEG-4 Part 10, *4th EURASIP Conference Focused on Video/Image Processing and Multimedia Communications*, Zagreb, Croatia, 2-5 July, Vol. 1, 1-51.

[11] **Güney, M.**, 2006. H.264 Baseline Encoder Implementation and Optimization on TMS320DM642 Digital Signal Processor, *Master Thesis*, Sabancı Uni. Graduate School of Engineering and Natural Sciences, İstanbul.

[12] **Golomb S.W.**, 1966. Run-length encoding, *IEEE Transcations on Information Theory*, IT-12, 399–401.

[13] **Richardson, E.G.,** 2003. H-264 and MPEG-4 Video Compression, Wiley, London.

[14] **Kwon, S., Tamhankar A. and Rao, K.R.**, 2005. Overview of H.264/MPEG-4 Part 10, Dongeui University, T-Mobile, University of Texas at Arlington, USA, March 15.

[15] **Sullivan G.J., Topiwala P. and Luthra A.**, 2004. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions, *SPIE Conference on Applications of Digital Image Processing XXVII Special Session on Advances in the New Emerging Standard: H.264/AVC*, Denver, USA, 2 August, Vol. 5558, 454-474.

[16] **Raja G. and Mirza M.J.**, 2004. Performance Comparsion of Advanced Video Coding H.264 Standard with Baseline H.263 and H.263+ Standards, *Internation Symposium on Communications and Information Technologies*, Sapporo, Japan, 26-29 October, Vol. 2, 743-746.

[17] **Ostermann, J., Bormans J. and Marpe D.,** 2004. Video Coding with H.264/AVC: Tools, Performance, and Complexity, *IEEE Circuits and Systems Magazine*, Vol. 4, 7-28.

[18] **Marcandey V. and Rao D.,** 2002. TMS320DM642 Technical Overview, *Application Note*, **SPRU615,** Texas Instruments, TX.

[19] **TI,** 2001. TMS320C64x Technical Overview, *Application Note*, **SPRU395b,** Texas Instruments, TX.

[20] **TI,** 2003. TMS320C6000 DSP Cache Users' Guide, *User's Guide*, **SPRU656A,** Texas Instruments, TX.

[21] **TI**, 2006. Code Composer Studio Development Tools v3.3, *Getting Started Guide,* **SPRU509H**, Texas Instruments, TX.

[22] **Digital Spectrum Inc**., 2003. TMS320DM642 Evaluation Module, *Technical Reference.*

[23] **Dart D.,** 2001. DSP/BIOS Kernel Technical Overview, *Application Note*, **SPRA780,** Texas Instruments, TX.

[24] **Stevenson J.,** 2004. Code Composer Studio IDE v3 White Paper, *Application Report,* **SPRAA08,** Texas Instruments, TX.

[25] **TI,** 2002. TMS320C6000 Optimizing C Compiler Tutorial, *Technical Document*, **SPRU425A,** Texas Instruments, TX.

[26] **Cheng Y., Dai K. and Wang Z.**, 2004. Motion Search Method Based on Zero-block detection in H.264/AVC, *8th International Conference on Computer Supported Cooperative Work in Design*, Xiamen, China, 26-28 May, Vol. 2, 739-743.

[27] **Zhu, C., Lin X. and Chau L.,** May 2002. Hexagon-based search algorithm for fast block motion estimation, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 12, Issue 5, 349-355.

[28] **TI,** 2004. TMS320C6000 Optimizing Compiler User's Guide, *Technical Document*, **SPRU187L,** Texas Instruments, TX.

[29] **Biscondi E.,** 1999. C Implementation of the TMS320C62xx Intrinsic Operators, *Digital Signal Processor Application Report,* **SPRA616,** Texas Instruments, TX.

[30] **TI,** 2006. TMS320C6000 CPU and Instruction Guide, *Reference Guide*, **SPRU189G,** Texas Instruments, TX.

[31] **TI,** 2003. TMS320C64x Image/Video Processing Library, *Programmer's Reference*, **SPRU023B,** Texas Instruments, TX.

[32] **TI,** 2002. TMS320C64x DSP Library, *Programmer's Reference*, **SPRU565A,** Texas Instruments, TX.

[33] **TI,** 2006. TMS320C6000 Enhanced Direct Memory (EDMA) Controller, *Reference Guide*, **SPRU234C,** Texas Instruments, TX.

AUTOBIOGRAPHY

Ender Meriç was born in Uşak at 04.02.1982. He received B.S. degree in computer engineering from Istanbul Technical University. His graduate education is still continuing in computer engineering at Istanbul Technical University. His research interests include real-time implementations and embedded systems.