Application Report
SPRAA63–October 2004

# Video Encoding Optimization on TMS320DM64x/C64x ™

*Cheng Peng* *DSP Video Imaging Solutions*

**ABSTRACT**

Digital video encoding plays an important role in many applications such as digital video surveillance systems and video conference systems. This application report describes the optimization techniques for a general video encoder on TI TMS320DM64x/C64x DSP. The TMS320DM64x/C64x is a high-performance digital medial processor with 2-level memory/cache hierarchy and very-long-instruction-word (VLIW) architecture. Video encoding optimization on DM64x is a combination of multiple techniques including algorithm/system optimization, memory buffering optimization, enhanced direct memory access (EDMA) and cache utilization optimization.

**Contents**



**List of Figures**



**List of Tables**

# 1 Introduction to DM64x/ TMS320C64x

The TMS320DM64x/C64x™ device is based on the second-generation high-performance, very-long-instruction-word (VLIW) architecture VelociTI.2™ developed by Texas Instruments (TI). The key features of this device such as VLIW architecture, 2-level memory/cache hierarchy, and EDMA engine makes it an excellent choice for computationally intensive video/image applications such as video coding and analysis. When developing an application on DM64x, it is important to fully understand its features and memory architecture in order to achieve optimized performance.

Using a DM642 as an example, all DSP core and necessary features are listed:

- The enhanced functional units for video/imaging applications
  - The VelociTI.2™ extensions in the eight functional units of DM64x include new instructions which accelerate the performance in video and imaging applications.
- L1/L2 Memory hierarchy
  - 16K-byte direct mapped L1P program cache with 32-byte cache line. (8-cycle L1P cache miss penalty).
  - 16K-byte 2-way set-associative L1D data cache with 64-byte cache line. (6-cycle L1D cache miss penalty).
  - 256K-byte L2 unified mapped RAM/cache (flexible RAM/cache allocation, 8-cycle L2 cache miss penalty)
  - L2 4-way set associative cache has128-byte cache line
- Endianess: Little Endian, Big Endian
- 64-bit external memory interface (EMIF)
  - Glueless interface to synchronous and asynchronous memories
- 1024M-byte total addressable external memory space
- EDMA Controller (64 Independent Channels)

The on-chip peripheral set includes: three configurable video ports; a 10/100 Mb/s Ethernet MAC (EMAC); a management data input/output (MDIO) module and a VCXO interpolated control port (VIC). The video port peripherals provide a seamless interface to common video decoder and encoder devices. They support multiple video resolutions and standards (e. g., ITU-BT.656, BT.1120, SMPTE 125M, 260M, 274M, and 296M).

The features and overheads for memory accesses described above are important for all algorithm implementations including video coding. 2-level memory/cache hierarchy and EDMA engines essentially determine the architecture of a video encoder implementation.

Some basic concepts concerning memory/cache hierarchy and EDMA engine need to be considered for an algorithm implementation. When code size is bigger than the size of L1P, L1P caches misses can occur, CPU stalls until the required code is fetched. Similarly, L1D cache misses and CPU stalls occur when the data do not fit in the L1D. All L1P and L1D misses are serviced by L2 cache/SRAM. L2 cache misses occur if the code and data size is bigger than the size of L2 cache.

Cache-friendly program partitioning and data transfer handling (e.g. reducing L1/L2 misses) are two critical factors to guarantee video encoder optimal performance. EDMA is preferred to transfer code/data between L2 SRAM and off-chip memory.

## 2 Video Encoder System/Algorithm Optimization

The block diagram of a general video encoding algorithm is shown in Figure 1. Many video coding standards such as MPEG2, H.263 and MPEG4 can be derived from this algorithm diagram. In Figure 1, DCT and quantization (Q) reduce the spatial redundancy of the video. Motion estimation (ME) reduces the temporal redundancy of the video and VLC is entropy coding to pack the data efficiently.

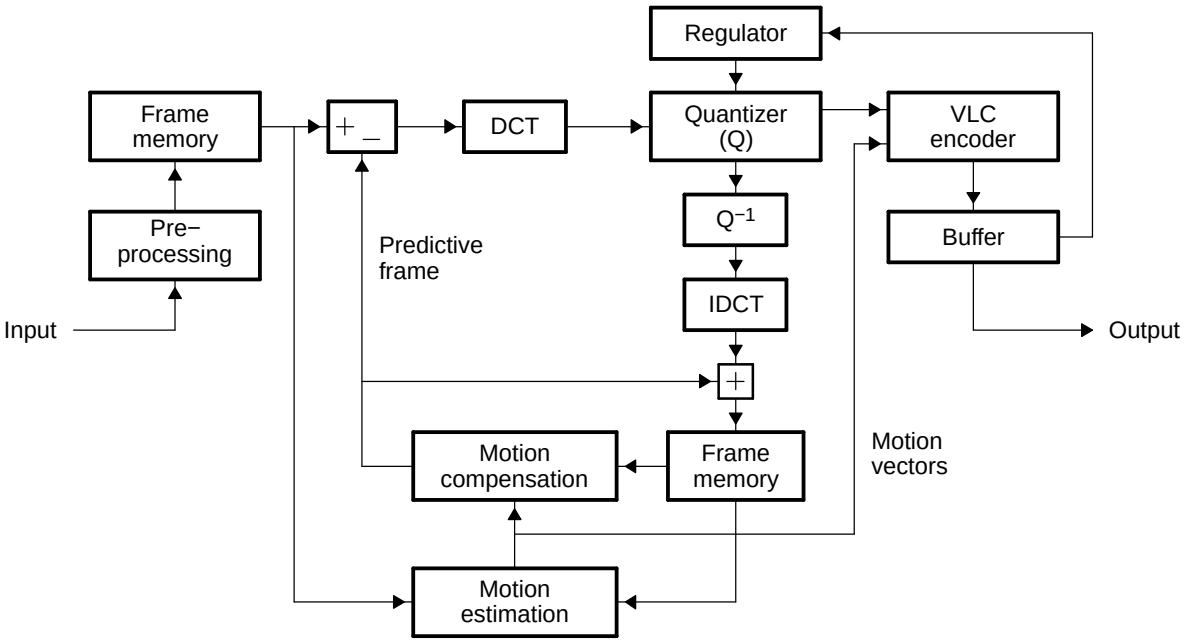


**Figure 1. Video Encoding Block Diagram [1]**

Conventional implementation of a video encoder is based on macroblock-level processing. The video encoder fetches a new macroblock (MB) only after the current MB goes through all the processing steps. This intuitive approach comes with two drawbacks:

- The overall code size of a video encoder is usually bigger than L1P. The code needs to swap between L1P and L2P every MB fetching period. It causes significant cache miss penalty.
- It is not efficient to transfer a small chunk of data such as a single MB from external video frame memory to internal memory by EDMA.

In order to avoid the huge cache miss penalty and CPU stalling, the algorithm can be broken into three loops/modules and each of them fits in L1P. M macroblocks (MB strip) are processed at a time in each loop instead of a single macroblock. M, the size of macroblock strip, is only restricted by the available L1D size. The bigger of M, the better EDMA performance we can expect for data throughput.

Three loops are:

- Marcoblock encoding loop
- Motion estimation loop
- Marcoblock reconstruction loop

As we emphasize above, M macroblocks are fetched and go through one of the three processing loops together. For example, in the macroblock encoding loop, when M macroblocks are fetched into internal memory, they are DCT transformed, quantized, and entropy coded. This set of macroblocks is not flushed out of L1D until the macroblock encoding loop is over. Corresponding programs including DCT, quantization and VLC kernels are also kept in L1P until all M macroblocks are processed completely in this loop. Ping/Pong memory buffering scheme driven by EDMA engine helps reducing the initial setup time needed to perform these loops for a set of macroblocks. It also ensures minimal CPU stalling cycles because the transfers are overlapped with processing.

## 2.1 Macroblock Encoding Loop

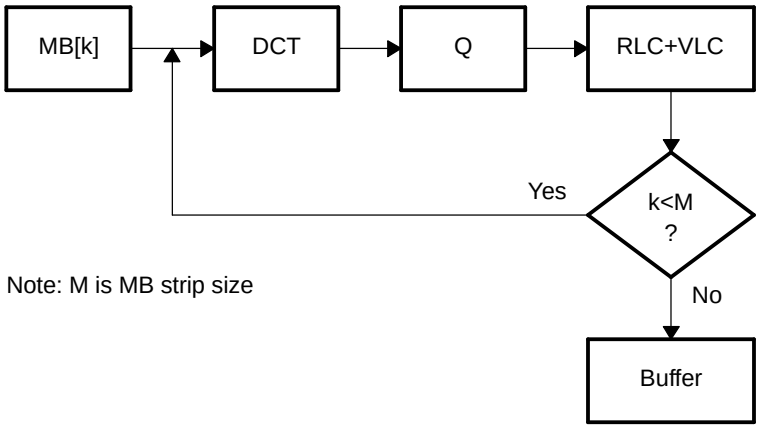


**Figure 2. Macroblock Encoding Loop**

MB[k] is one of M macroblocks (MB strip) in I frame or prediction residue blocks in P/B frame needs to pass the DCT, quantization, run length coding (RLC), and VLC. After DCT, the DC component and AC components go through differential encoding process separately. Quantization enables compression essentially and also introduces the quantization error. Run-length encoding scans the block in zigzag order. It is followed by variable length code assignment where each pair of run-level is treated as a symbol and translated into a code. VLC that is composed of table lookup and bit manipulating is a computationally heavy block in this loop. The lookup tables are formatted such that the code word and corresponding length are stored as a packed 32-bit quantity. VLC encoding directly writes the encoded bits into a bit-stream represented in 32 bit quantities. The buffer at the end of this loop depends on the number of macroblocks to be processed at a time. It is affected by the L1D cache size. The idea is to get all the data to be processed in the loop in L1D cache, which operates at CPU speeds, and at the same time the total size of the code operating on this fits L1P. M macroblocks in I frame or the prediction residual blocks in P/B frame need go through the process shown in Figure 2 together.

## 2.2 Motion Estimation Loop

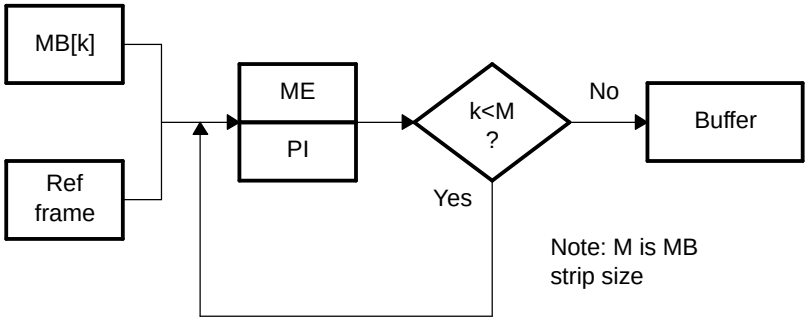


**Figure 3. Motion Estimation Loop**

DM64x provides a rich set of extensive video/image instructions that can implement effectively any motion estimation scheme. 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 in current frame and especially searching window in reference frame. It can easily fetch eight aligned pixels from current MB or non-aligned pixels from searching window. This non-aligned loading is more efficient than the aligned loading followed by shifting operation in this case. 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. The is the core for motion estimation. The idea of SAD kernel can be summarized in the following steps:

1. Two *LDNDW* fetches eight pixels from current frame and reference frame
2. Two *SUBABS4* calculate 8 Ads
3. Two *DOTPU4* accumulate 8 Ads

Both **SAD_16x16** and **MAD_16x16** functions of the C64x IMGLIB explain how to implement the above idea in details in *Image/Video Processing Library Programmer's Reference* (SPRU023).

Pixel interpolation (PI) is also a computationally intensive block in this loop. It is achieved efficiently with the C64x video processing instruction set. The AVG4 instruction performs 4 averaging operations on packed 8-bit data and the results are written in unsigned. The AVG2 instruction performs two averaging operations on packed 16-bit data and returns are written in unsigned integer values. Shift right and merge byte (SHRMB) shifts the second register right by one byte, and then the least significant byte of the first register is merged into the most significant byte position. In pixel interpolation, AVG4 calculates the average value and SHRMB packs the result tightly. The details of PI implementation can be found in Figure 4.

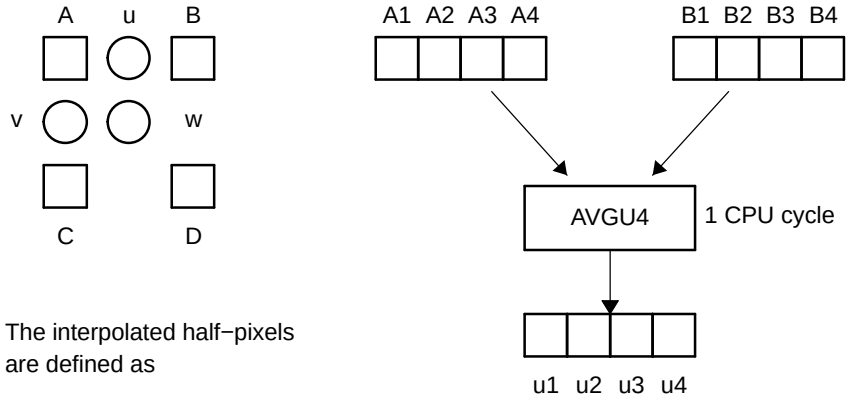


**Figure 4. Pixel Interpolation on DM64x**

## 2.3 Macroblock Reconstruction Loop

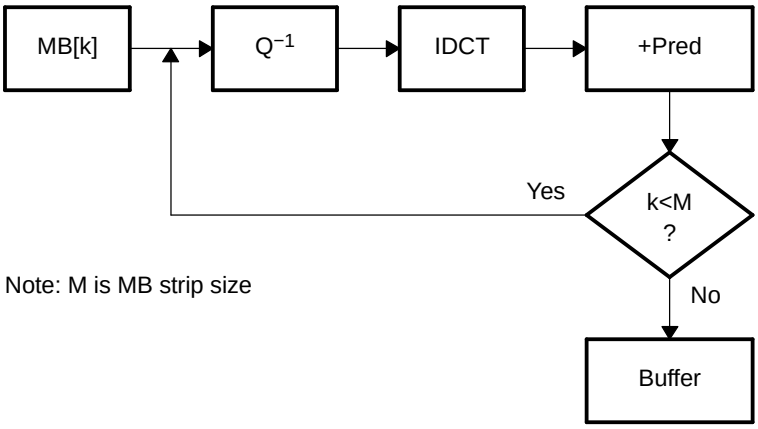


**Figure 5. Macroblock Reconstruction Loop**

The macroblock strip in I frame or the prediction residual strip in P frame need go through the process shown in Figure 5 together. Inverse quantization as the name suggests is an exact inverse process of the quantization process shown in Figure 2. TMS320C64x devices can perform 2400 million dual 16-bit MACs in one second at a CPU rate of 600 MHz. The MPY2 instruction performs two 16-bit by 16-bit multiplications between two pairs of signed, packed 16-bit values. The SPACK2 instruction takes two signed 32-bit quantities and saturates them to signed 16-bit quantities. To remove the overheads of signed arithmetic and rounding problems with signed values, the sign bit is extracted initially from the coefficient and inverse quantization operation is performed on the absolute value of the coefficient. The sign bit is applied to the end result directly.

## 2.4 Fast Motion Estimation Algorithm Optimization

Block-based motion estimation module shown in Figure 1 has been adopted by every common coding standard including H.261, H.263, MPEG2, MPEG4 and H.264. The block size of motion estimation may vary among standards. For example, the ME in H.264 may support up to seven block sizes (16x16, 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4). 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 need keep a good balance between computational intensity and coding efficiency. Exclusive search (full search) guarantees the globally best match result in motion estimation. Unfortunately, the computational cost is enormous. For example, a full search window of +/- 63 in horizontal and +/-63 in vertical would require a processor to calculate 500 Giga SADs per second [1]. Therefore, pure exclusive search ME is not realistic for an embedded solution. A real-world motion estimation is a combination of many fast search techniques together. A widely used 4-step fast searching algorithm is introduced in the rest of this section.

The block size is assumed to be 16x16 and searching window size is 48x48. The motion estimation by 4-step search is described by the following equations:

```
Setup:
                d={8,4,2,1} // d[i] is the distance between current MB and
                            //immediately succeeding MB (See Figure 6)

 Process:
                   for(i=0; i<4; i++)
                  {
                      Compute three upper SAD for d[i].
                      Compute three central SADs for d[i].
                      Compute three lower SADs for d[i].

                  Compute the minimum value of the 9−SAD table (see Figure 6)
                 Start above process around the minimum location for the new
                  distance d[i+1].
                    }
                    // The overall algorithm is shown in Figure 7.
```

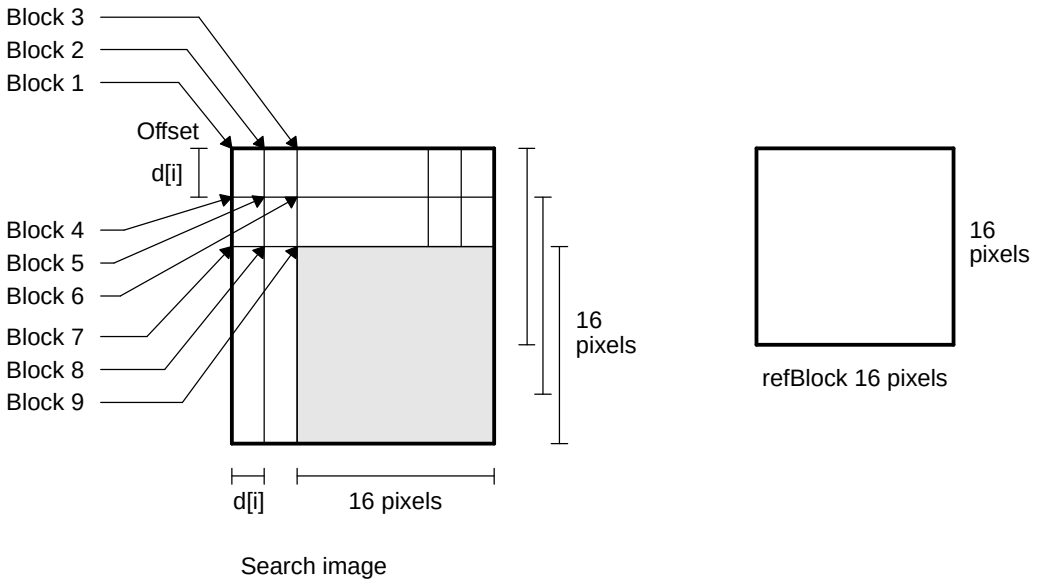


**Figure 6. 9-SAD Table**

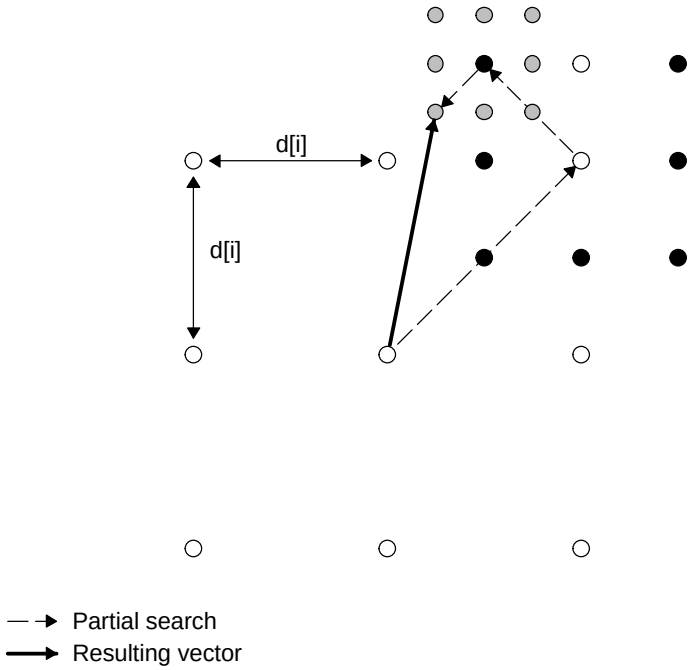


**Figure 7. 4-step Fast Search Scheme for ME**

The above 4-step fast searching technique reduces the computational cost significantly. However, this basic fast ME technique may not provide satisfactory performance alone. We have to combine many techniques together to achieve an optimized performance. Several potentially fast motion estimation optimization techniques are listed [1]:

- Hierarchical search using sub-sampled images [2]. N-step (N=2,3,4…) search introduced above is a good example of hierarchical search scheme.
- Using motion vectors of neighboring MB in the same frame as predictors. (Benefit from Spatial correlation)
- Using motion vectors of co-located MB in temporally adjacent frames as predictors. (Temporal Prediction) [3]
- Telescopic search [4]

For example, the combination of N-step searching and image sub-sampling technique can provide a good result with affordable computational cost. Using this combination as an example, we will go through the enhanced direct memory access (EDMA) activity of ME in Section 4 later.

## 3 Memory Buffering Scheme of a Video Encoder

In order to get the best performance, many lookup tables, state variables and data buffers associated with key kernels have to be located in internal memory. The internal SRAM varies on different C64x/DM64x devices. Except the memory consumed by above essential data, some TMS320DM64x still has enough internal memory to hold an entire video frame for the encoder to process, the others may not. Some applications need only run video encoder; the others may run other algorithms including video encoding. In order to provide a generic video buffering scheme for all DM64x devices in most video application scenarios, the whole video frames are located in external memory instead of internal memory. Each time, M macroblocks (MB strip) are transferred to internal buffer from external video buffer by EDMA . As we mentioned in previous section, M is only restricted by the size of L1D.

Group of picture (GOP) is an important concept in video coding because it defines the coding scheme of each video frame. Usually, I frame and P frame are defined in GOP for all video coding standard. B frame is included only in advanced video coding profile such as MPEG4 advanced simple profile and MPEG2 Main profile. In case of I-frames, JPEG-like coding scheme is implemented to remove the spatial

redundancy. In case of P frame, forward motion estimation is implemented and the previous I/P frame is used as a reference. In case of B frame, it need not only previous I/P frame for forward ME but also the following I/P frame for backward ME. The video encoder has to buffer all B frames between two P frames because of bidirectional ME strategy of B frame. The data dependence of video frames is shown in Figure 8.

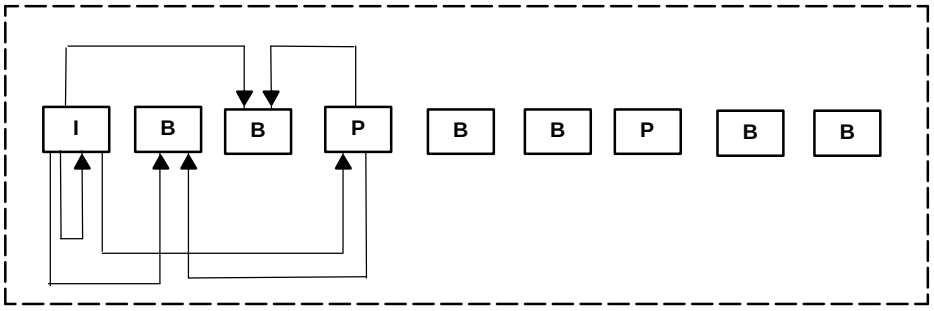


**Figure 8. Data Dependence of Video Frame in Video Encoder**

It is well known that video capture/display order is different with video coding order for a GOP with B frame. For example GOP=IBB PBB PBB PBB PBB is ordered for capturing/display and the coding order of this GOP is GOP'=IPBB PBB PBB PBB... The following table shows how to temporally partition a video coding algorithm efficiently. The key of video coding algorithm partition is to make CPU loading as constant as possible temporarily.

Suppose vp is a video frame for processing,

- **T(**vp **):** DCT transform and Quantization of vp (I,B,P frames).
- **F(**vp**):** Forward motion estimation of vp (B, P frames).
- **B(**vp**):** backward motion estimation of vp (B frames)
- **C(**vp**):** VLC coding of vp (I,B,P frames)

**Table 1. Video Processing Steps**

| Video frame in cap-ture/display order | I1 | B2 | B3 | P4 | B5 | B6 | P7 | B8 | B9 |
|---|---|---|---|---|---|---|---|---|---|
| Video pro-cessing steps | T(I1)+C(I1) | | | | | | | | |
| | | …. | | | | | | | |
| | | | …. | | | | | | |
| | | | | **F**(P4) + **C**(P4) | | | | | |
| | | | | | **B**(B2)+**B**(B3)+**C**(B2) | | | | |
| | | | | | | **C**(B3)+**F**(B5)+**F**(B6) | | | |
| | | | | | | | **F**(P7)+**C**(P7) | | |
| | | | | | | | | **B**(B5)+**B**(B6)+**C**(B5) | |
| | | | | | | | | | **C**(B6)+**F**(B8)+**F**(B9) |

An overall video encoding buffering scheme is shown in Figure 9.

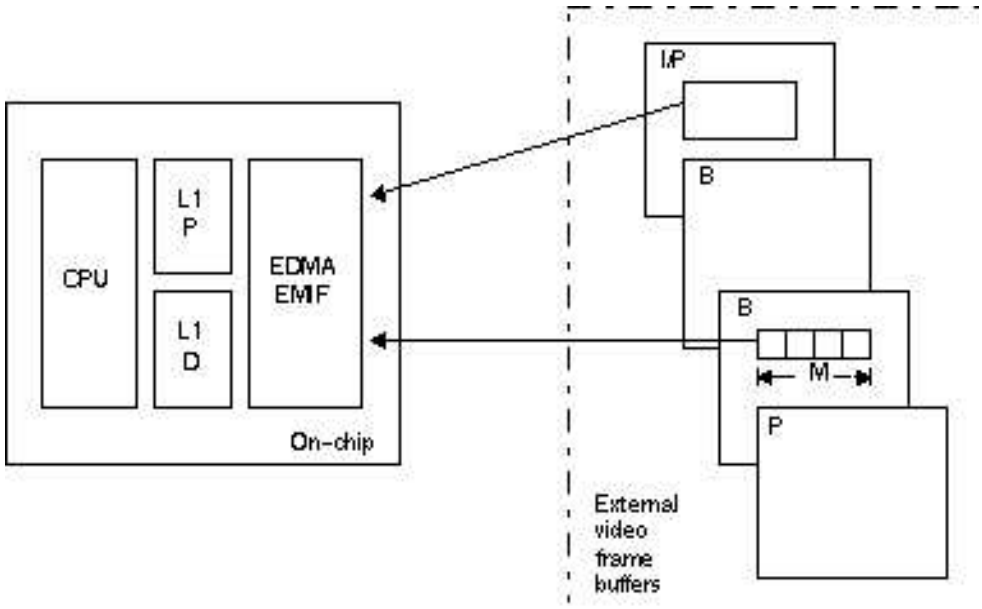


**Figure 9. Video Encoding Buffering Scheme**

Pixel interpolation in frame-based motion estimation loop requires one 18x18 luma block from the reference frame buffer. To ensure best performance, all EDMA data transfers are 32-bit aligned. It means that the starting address and transferring data size have to be a multiple of four bytes.

Therefore, an 18x24 luma block has to be transferred for required 18x18 block. Similar offset technique is implemented in case of chromo block. The additional offset of these blocks need be considered in the motion estimation loop. The offset of luma block is illustrated in Figure 10.

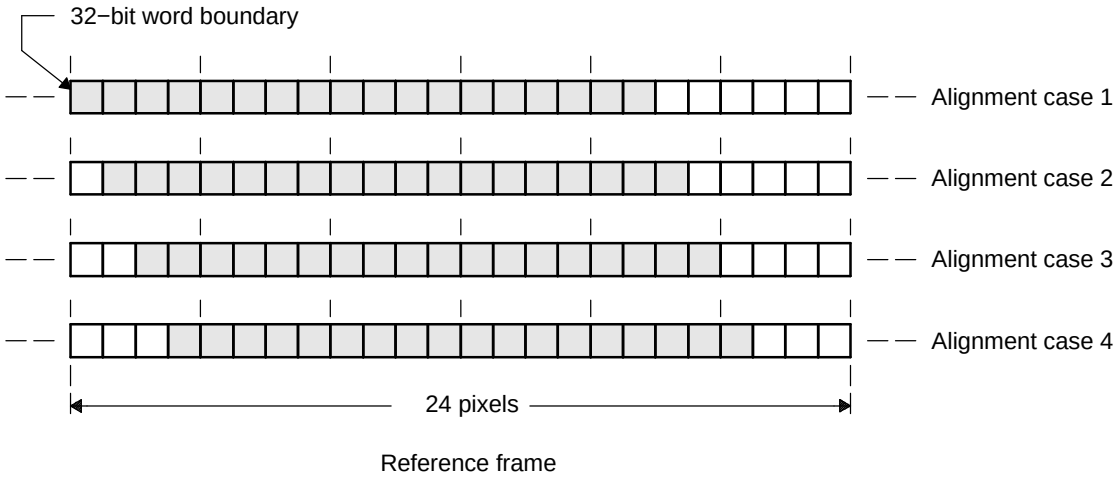


**Figure 10. The Offset of Memory Buffering for PI in Video Encoding**

## 4 Enhanced Direct Memory Access (EDMA) Usage

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

In order to utilize EDMA effectively, we need understand EDMA transfer process well. 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. All submitted TRs are sorted into queues based on priority. When a TR 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. Each transfer requestor is programmable such that it can submit TRs on any priority level. 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.

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. Each submits a transfer request to be processed by the EDMA. The request are queued according to priority, with higher priority requests services first. 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. Only one request is sent for any QDMA submission. 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. A whole QDMA implementation in ME is illustrated in Figure 11. In Figure 11, only one of double buffers is shown. The fast motion estimation shown in Figure 11 is combination of N-step search and image sub-sampling technique.

The EDMA channel and priority utilization is shown in Table 2.

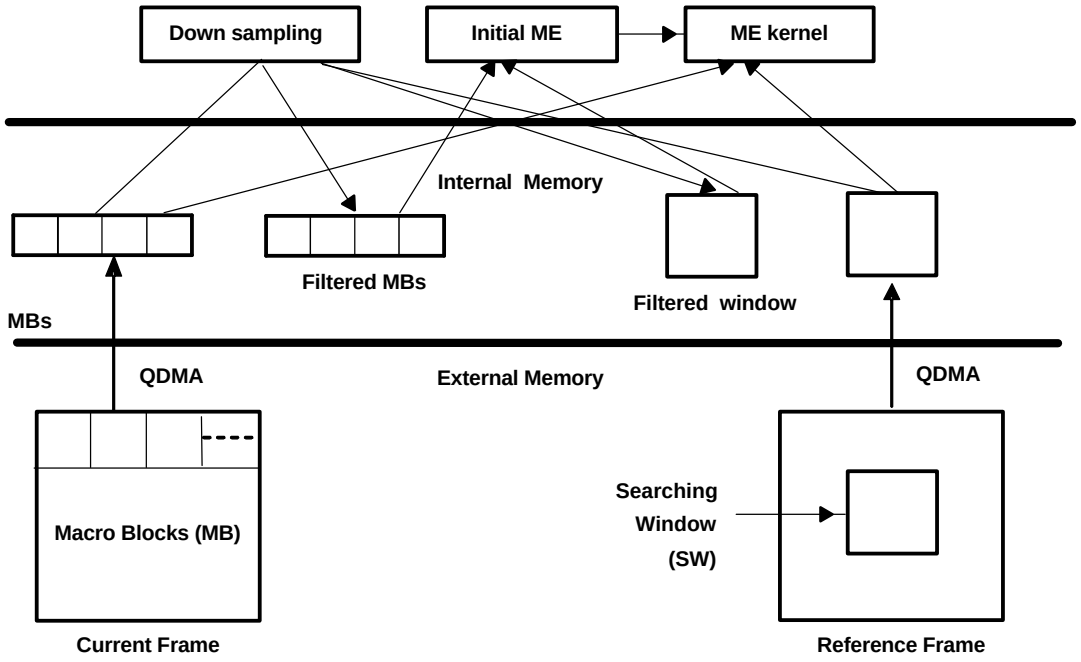


**Figure 11. QDMA Management for ME**

**Table 2. QDMA Channel and Priority Utilization in Frame-based ME**

| EDMA channel | Priority | Buffers Used in Transfer |
|---|---|---|
| I frame case | | |
| CH3 | Low | RMB(Cr)(1) |
| CH4 | Low | RMB(Y, Cr)(1) |
| CH6 | Low | Luma and Chroma info of CMB(2) and macroblock side information |
| CH7 | Low | Chroma(Cr) info of (CMB)(2) |
| P, B frame | | |
| CH2 | Medium | |
| CH5 | Medium | Frame reference region fetch |
| CH8 | High | |
| CH3 | Low | RMB(Cr)(1) |
| CH4 | Low | RMB(Y,Cr)(1) |
| CH6 | Low | Me data, mbinfo, CMB(2) |
| CH7 | Low | CMB(Cr)(2) |

(1) RMB: Reconstructed set of macroblocks after encoding
(2) CMB: Current set of macroblocks from the frame being encode

## 5 Cache Optimization

Maximizing cache effectiveness is one of the key factors for reaching the overall video coding performance expectation. Cache efficiency improves processor throughput by reducing CPU stall cycles due to memory activities. As we introduced in section 1, TM320C64x DSP (including DM64x) employs a highly efficient two level memory architecture for on-chip program and data accesses. In this hierarchy, the CPU interfaces directly to a dedicated level-one program (L1 P) and data (L1D) cache. These L1 caches operate at the same speed as the CPU., the direct-mapped L1P cache is readable only and the two-way set associate cache L1D can be read and written. The L1 memories are connected to a second level on-chip memory called L2. L2 is a unified memory block that contains both program and data. The L2 cache serves as a bridge between the L1 and off-chip memory. Please refer to *Cache Analysis for Code Composer Studio v2.3 User's Guide* (SPRU575) for a detailed documentation of this cache architecture.

Cache performance analysis and tuning up exists in the overall video coding development life cycle. Using TI's new cache analysis tools, we can identify cache efficiency problem areas and visualize them to facilitate making rapid improvements in cache performance for video coding. The cache optimization for video coding development can be divided into two distinct stages:

- Perform cache efficiency analysis at the video coding algorithm kernel level with a particular memory context. (See Figure 12)
- Perform cache efficiency analysis targeted at the whole-application level when CPU and memory cycles are measured after the integration of the algorithm into an application framework. (See Figure 13)

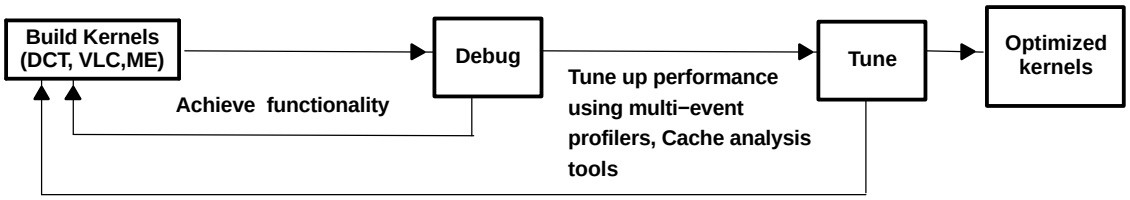


**Figure 12. Cache Efficiency Analysis of Algorithm Kernels**

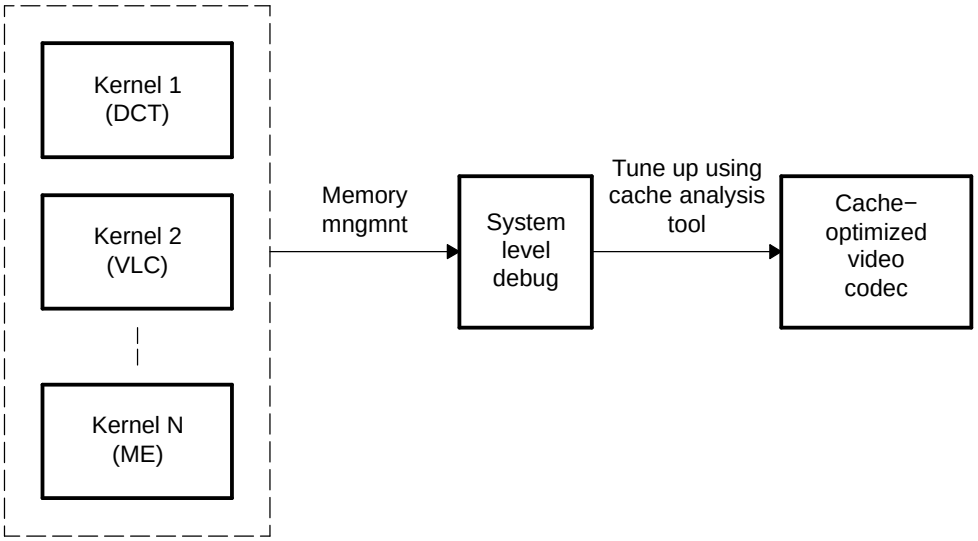


**Figure 13. Cache Efficiency System Level Analysis**

With the cache optimization techniques described above and algorithm optimization introduced in previous sections, we achieve a good cache performance for a MPEG2 encoder on DM642 (See Table 3). The L2 cache hit/miss number is not shown because it depends on the external memory type used in the system.

**Table 3. MPEG2 Encoder L1 Cache Performance on DM642**

| Event | Count/Status | Total Count | Type |
|---|---|---|---|
| l1d | | | |
| miss | | | |
| read | 845312 | 845312 | Count |
| write | 3233156 | 3233156 | Count |
| hit | | | |
| read | 102447507 | 102447507 | Count |
| write | 16918505 | 16918505 | Count |
| l1p | | | |
| miss | 2148568 | 2148568 | Count |
| hit | 102964973 | 102964973 | Count |

Above table tells us the cache miss is marginal for a video encoder benchmark with appropriate cache optimization.

## 6 References


1. Anurag Jain, Ratna Reddy, Jeremiah Golston, Jagadeesh Sankaran, Programmable Real-time MPEG-2 Encoding, GSPx 2002
2. Kyoung Won Lim, Jong Beom Ra. Improved hierarchical search block matching algorithm by using multiple motion vector candidates. Electronic Letters, Volume: 33 Issue:21, 9 Oct. 1997 Page(s): 1771-1772.
3. Ismaeil, I.R.; Docef, A., Kossentini, F., Ward, R. Motion estimation using long-term motion vector prediction. Data Compression Conference, Proc. 1999 Page(s): 531.
4. J. Lee and B.W. Dickinson. Temporally adaptive motion interpolation exploiting temporal asking in visual perception. IEEE Trans. Image Proc., 3(5): 513-526, Sep 1994.

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

| **Products** | | **Applications** | |
|---|---|---|---|
| Amplifiers | amplifier.ti.com | Audio | www.ti.com/audio |
| Data Converters | dataconverter.ti.com | Automotive | www.ti.com/automotive |
| DSP | dsp.ti.com | Broadband | www.ti.com/broadband |
| Interface | interface.ti.com | Digital Control | www.ti.com/digitalcontrol |
| Logic | logic.ti.com | Military | www.ti.com/military |
| Power Mgmt | power.ti.com | Optical Networking | www.ti.com/opticalnetwork |
| Microcontrollers | microcontroller.ti.com | Security | www.ti.com/security |
| | | Telephony | www.ti.com/telephony |
| | | Video & Imaging | www.ti.com/video |
| | | Wireless | www.ti.com/wireless |

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265