HARDWARE-ORIENTED OPTIMIZATION AND BLOCK-LEVEL ARCHITECTURE DESIGN FOR MPEG-4 FGS ENCODER

Chih-Wei Hsu, Yung-Chi Chang, Wei-Min Chao, and Liang-Gee Chen

DSP/IC Design Lab, Graduate Institute of Electronics Engineering,
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
{jeromn, watchman, hydra, lgchen}@video.ee.ntu.edu.tw

ABSTRACT

MPEG-4 Fine Granularity Scalability (FGS) provides bandwidth adaptation and error resilience features for streaming applications. In this paper, by estimating the required computational power for FGS in a video encoding system, an efficient FGS implementation method is exploited. With the proposed hardware-oriented optimization approaches, a hardwired FGS block-level processing core is proposed to provide a cost-effective solution to FGS implementation. The proposed hardware core can support FGS profile level 5, frame size 720x576, 30Hz, for real-time streaming applications at 54 MHz.

1. Introduction

Multimedia delivery over Internet is an emerging application that needs to serve numerous users over network using various access media with different available bandwidth. In addition, suffering from unavoidable packet loss, which results from the best-effort nature for QoS of the Internet, bandwidth adaptation and error resilience are two key techniques to stream multimedia data over a range of bit rate on Internet. MPEG-4 Fine Granularity Scalability (FGS) can easily provide these two features and is suitable for streaming applications [1][2]. To further improve visual quality of FGS video, frequency weighting that uses different weighting for different frequency components is provided [2]. The streaming content can be either pre-coded video or live video. Due to the characteristic of separate coding and transmission procedures for FGS bitstream, it is easy to handle with the pre-coded video. The encoding system needs not finish encoding in real-time but just before the server to dispatch the coded bitstream upon users' requests. With coded FGS bitstream, the required computational power for server to perform rate allocation and adaptation is low. However, as to streaming live video, the streaming system should perform both coding and bitstream delivery in time. The required computational power is huge for video encoding and further raised when offering additional FGS function. There are many state-of-the-art MPEG-4 encoder/codec SOC chip to accelerate the encoding procedures with software/hardware co-design and co-optimization [3][4], which are ready for the real-time applications. Dedicated FGS accelerating core should be also implemented for specific streaming applications. Before FGS function is integrated into the encoding system, the most efficient implementation method for it should be exploited to achieve optimal system performance with least implementation cost.

In this paper, the required computational power for FGS in a video encoding system will be evaluated in section 2. Then detailed FGS coding flow will also be analyzed and modified with hardware/software optimization in section 3 to adapt FGS function into encoding system more efficiently. Then a cost-effective block-level hardware design for FGS encoder will be proposed in section 4. The conclusions will be drawn in section 5.

2. FGS Functional Descriptions

The basic idea of FGS is to code a video sequence into two layers. One is the base layer whose bit-rate is set at the least available bandwidth of the channel. Ideally, the base layer is received and decoded completely at the decoder's side to give the basic quality of the coded video. Another one is the enhancement that is ended to improve the video quality of the base layer. In FGS coding scheme, to achieve the fine-grained scalability of the enhancement layer, bit-plane coding method is used. With bit-plane coding, first the residues between the DCT coefficients and the inverse quantized ones of an 8x8 block are taken and scanned in zigzag order. Then, instead of being coded word-by-word, these residues are processed in bit-plane order, i.e. bits at the same significant position form one bit-plane from the MSB of this block to the LSB. In each bit-plane, only one and zero are left and run-length coding is performed to form the (RUN, EOP) symbols, where RUN means the number of consecutive zero before a one and EOP indicates the last one of a bit-plane. These symbols are further VLC-coded and finally packed into one enhancement bitstream from higher bit-plane to lower bit-plane. So, this enhancement bitstream can be decoded to recover the quantization error of the DCT coefficients from MSB, which makes more contributions to recover error, to LSB. Besides, it can be truncated at any point since this results in only some data loss in lower bit-plane. It can be referred to [2] for more details of the FGS functionality.

2.1 Profiling Results

FGS can be implemented using either software or dedicated hardware. So, before adding FGS function into encoding system, from system viewpoint, we must decide what kind of implementation is most suitable for FGS by estimating its required computational power. Table 1 shows the instruction level profiling of FGS in an MPEG-4 encoding system using MPEG-4 VM software [5]. The profiling condition is for foreman sequence, CIF format and 30 Hz and the profiling data can be divided into two parts, one is the required instruction analysis and another is for required memory bandwidth.

It is shown that when FGS function integrated into an MPEG-4 encoder targeting real-time applications, it consumes more than 2.7 GIPS of computational power and 3.4 GBytes memory access bandwidth, which are 13.6% and 10.3% of the system resources, respectively. Among others, 41.3% of instructions consumed are for Load/Store operations and up to 80% memory bandwidth is spent on loading data. This statistics show that implementing FGS

0-7803-7761-3/03/$17.00 ©2003 IEEE

Table1. Profiling FGS at UltraSparc-II 448MHz.

Functions	Instruction Analysis (MIPS)					Percentage (%)
	Arithmetic	Control	Load/Store	Others	Total	
ME/MC	2368	1965	6093	1311	11737	57.51
DCT/IDCT	888	419	1578	252	3137	15.37
Q/IQ	139	107	266	75	588	2.88
ACDCP	69	64	124	44	300	1.47
SCAN	50	60	124	42	276	1.35
VLC	207	402	723	256	1588	7.78
FGS	910	374	1151	350	2785	13.64
Total	4630	3393	10058	2330	20411	100

Functions	Required Memory Bandwidth			
	Memory Access (Mbytes/Sec)		Total Bandwidth	Percentage (%)
	Load	Store		
ME/MC	15940	5240	21179	63.60
DCT/IDCT	3414	1561	4975	14.94
Q/IQ	597	137	734	2.20
ACDCP	330	58	388	1.17
SCAN	331	54	385	1.16
VLC	1591	630	2221	6.67
FGS	2799	618	3418	10.26
Total	25002	8298	33300	100

characterizes massive but simple data processing and huge memory access bandwidth. The required computational complexity for FGS is not an easy task for the system based on a general-purpose processor. In the next section, a thorough analysis of FGS coding flow will be given. Besides proposing hardware-oriented optimized algorithms, a dedicated hardwired core for FGS will also be extracted and validated to perform the FGS function more efficiently. With this hardwired accelerator for FGS, more than 10% system resource can be saved.

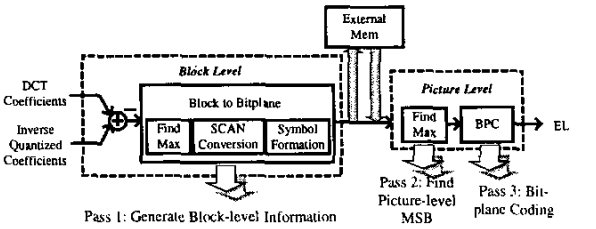
2.2 FGS Coding Flow Analysis

Figure 1(a) shows the detailed FGS coding flow in MPEG-4 VM. It can be roughly divided into block- and picture-level operations and a temporary buffer is used to bridge these two parts of operations.

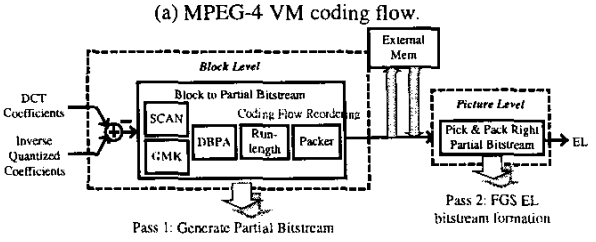
In block-level operations, SCAN, where zigzag scan and word-to-bit-plane conversion are performed, and Symbol Formation (SF) unit occupy the largest memory bandwidth and consume the largest amount of instructions and run-time. The reason is that a general-purpose processor is not suitable for the bit-level operations due to its word-based sequential processing property that cannot exploit the bit-level parallelism. In view of this, a dedicated hardware design for these block-level operations is more efficient in case FGS becomes a bottleneck for the whole system.

In picture-level operations, picture-level MSB information must be obtained first, which results in an extra picture-level passing. When entering bit-plane coding, this operation will transform the stored symbols into variable-length bitstream moving through the same significant position of the picture. Actually, only the final bitstream picking and packing operations demand sequential passing order, from higher bit-plane to lower bit-plane. However, as to bitstream formation, namely, VLC table lookup can be performed in parallel at block-level, which implies rearranging this part into a block-level core is possible and more efficient.

As the proposed FGS coding flow is shown in Figure 1 (b), the block-level core can cover a great part of work. With hardware-



(a) MPEG-4 VM coding flow.



(b) Proposed coding flow.

Figure.1 FGS coding flow.

oriented optimization approaches described in the following sections, the proposed hardwired block-level core can handle all the data-massive operations of FGS and leave only simple work to the encoding system.

As to the temporary buffering data type, in MPEG-4 VM it is (RUN, EOP) symbols for each bit-plane that are stored in memory. From system viewpoint, the amount of occupied system bus for memory access is more critical than the allocated external memory size. The most suitable temporary buffering data type will be evaluated to improve the system performance in the next section.

3. Hardware-oriented Approaches

3.1 Global Maximum Keeping (GMK)

In MPEG-4 VM implementation, it spends one extra picture-level pass to get the picture-level MSB information. However, the work of finding picture-level MSB can be performed in passing during block-level processing, that is, after finding out one block-level MSB, the picture-level MSB, which we define global maximum here, can be continually updated at the same time. Using this global maximum keeping method, the redundant picture-level pass can be saved and all the block-level MSB needn't be stored.

3.2 Dynamic Bit-Plane Adaptation (DBPA)

In FGS profile [1], maximum four coded bit-planes for one frame are supported. It implies that only the information of the top four bit-planes for each block needs to be saved and it is sufficient to generate the final bitstream regardless of the picture-level MSB. This substantially reduces the required computational complexity and implementation cost.

Refer to figure 2, DBPA functions as follows. Each block has maximum 11 bit-planes due to the dynamic range of the magnitude of the input DCT coefficients. Block-level MSB will be extracted first and GMK continually updates the picture-level MSB. Only the information of the top four bit-planes for each block will be dynamically kept according to the current picture-level MSB. Note that, the time GMK updates the picture-level MSB, which we call a MSB jump, means the blocks ahead it have the wrong picture-level MSB and the bit-plane level of the stored top four bit-planes needs

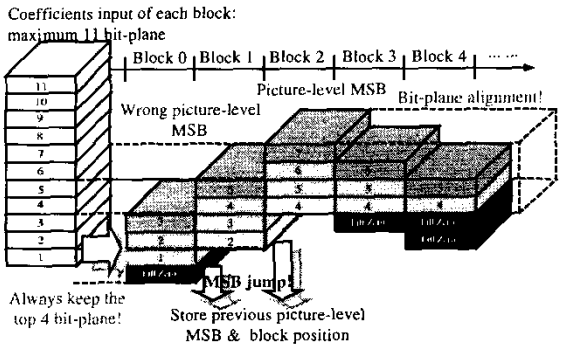


Figure.2 Concept of DBPA.

to be adjusted. So these block positions should be kept as well as the wrong picture-level MSB to aid the adjustment. However, after the picture-level MSB is found out sooner or later, the chosen top four bit-planes of the subsequent blocks will be aligned to the right position among the blocks. At this time some lower bit-plane information can be even replaced with zeros, which means there is no need to process these bit-planes.

3.3 Coding Flow Reordering

There are three candidates to estimate the most proper temporary buffering type, including bit-plane raw data, (RUN, EOP) symbol, which is adopted in MPEG-4 VM, and partial bitstream. The bit-plane raw data are just the residues obtained from taking the difference between DCT coefficients and the inverse quantized ones. However, proper packing should be carried out such that the data form independent bit-planes. As to partial bitstream, since each bit-plane in one block can perform VLC table lookup according to the significant position in that block regardless of picture-level MSB, this work can be advanced to perform at block-level to generate partial bitstream. And with GMK and DBPA, the final bitstream is generated by picking and packing the right partial bistreams. This coding flow reordering will store bitstream-level information of each block into buffer. Table 2 shows some comparisons between these two kinds of buffering types. As to the symbol-level, from implementation viewpoint, the performance is situated between the other two, so it is ignored here for briefness. As shown in the table, since the partial bitstream is formed through compressing the bit-plane raw data, the amount of data required to be stored is much less than the non-compressed one. However, note that if the bus bandwidth is 32 bit, the variable-length bitstream need to be packed into 32 bit units regardless of the truly bit-length of the bitstream and a header is needed to keep the information about it. Indeed, the amount of the reducing access bandwidth suffers from 32-bit packing and header insertion. As shown in table 2, average 81.55 bits are needed to coding a block and extra 32 bits are reserved for header. In spite of this, storing partial bitstream benefits the system performance by reducing the occupation of the memory access bandwidth to only 47% of non-compressed raw data. Besides, some required memory device for implementing these two methods, including local buffer and register, are listed also for comparing the total implementation cost. The most different between these two methods is that data before and after temporary buffer are belong to different data type, that is, residue words and bit-planes. When translating into bit-planes, the relation among a word is lost, for example, we have no idea about

Table 2. Required memory resources.

	External Memory Access Bandwidth	Scan Buffer	Required Memory Device: Serial-to-parallel Packer	BS Packer	Sign Coding Table	CBP Coding Table
Bit-plane Raw Data	2.72MBytes/s (320 bits/block)	(Cross-wiring)	V	V	V	V
Partial Bitstream	1.28 Mbytes/s (114bits/block)	V	(Merge to BS Packer)	V		

if a one in a bit-plane is the first one in a word. This impedes some bitstream coding operations, including sign coding and CBP coding, which all need the information of a one in the word. So extra picture-size tables are required to keep this important information. This validates the coding flow reordering to reduce the implementation cost. With coding flow reordering, sign bit adding can be performed in parallel when processing a word, multiple bit-planes. Some information, which we define pre-CBP, can be added into the partial bitstream header to aid the CBP coding. This pre-CBP information record if there exist any one in the upper bit-planes.

3.4 Picture-level Processing

As to generating the complete enhancement bitstream of FGS, it requires one more picture-level processing and from higher bit-plane to lower bit-plane to pack all the partial bitstream in order. All required information is generated in GMK unit so partial bitstream will be aligned in proper significant order. The required CBP information is also ready in the partial bitstream header to benefit the packing procedure. The picture-level processing is left simple and sequential in nature to be performed by the system. However, since the enhancement bitstream will be tailored to meet the users' conditions, this simple task of picking and packing partial bitstream can even be moved to server's side to generate the final bitstream according to the bit allocation plans.

4. Implementation

4.1 Proposed Architecture

With GMK, DBPA and coding flow reordering as described in the previous section, a cost-effective hardware core for implementing FGS block-level operations is proposed and the architecture is shown in Figure 3. In such a block-level processing core, all operations for FGS are performed in each block independently following the raster scan order. First preprocessing unit takes the residues between DCT coefficients and inverse quantized ones and transforms them into separate magnitude and sign data. Note that the frequency weighting function of FGS is also performed at block-level to adjust the transmission priority of coefficient residues in one block. An additional bit-plane shift module can be included in the proposed block-level core to support frequency weighting. The DCT and inverse quantized coefficients are both provided by the system and sent in first 64 clock cycles while scan buffer is used to hold this data temporarily and then dumps the data out in zigzag scan order in the later 64 cycles according to the global counter. During the scan procedure, the MSB in one block is extracted and GMK unit will update the picture-level MSB and keep some necessary information at the same time. Then the DBPA multiplexer selects the top four bits and send them into four parallel run-length coders. The run-length coder is simply a FIFO

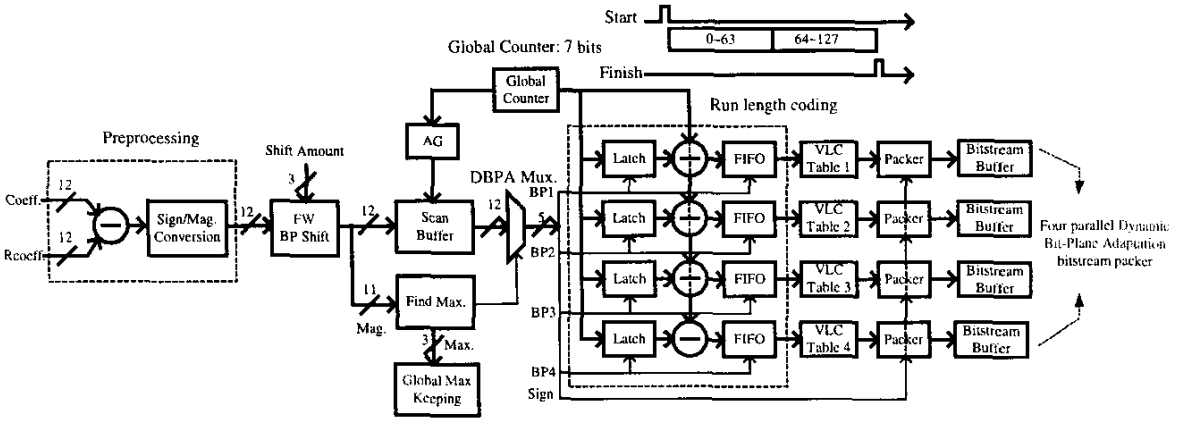


Figure 3. Proposed block-level processing core for FGS encoder.

that dumps new (RUN, EOP) symbol when another one is entering. There are four sets of subsystem working in parallel to generate partial bitstream, i.e. independent run-length coders, different VLC tables and bitstream packers, for each of the top four bit-planes. Finally, all the partial bitstreams will be stored in the partial bitstream buffers to wait until the system bus is available.

In summary, maximum four bit-planes are kept in trace to reduce the implementation complexity and four parallel bit-plane coders are adopted to exploit the bit-plane level parallelism to achieve the goal of hardware acceleration. GMK and DBPA algorithms are used to implement it with efficiency. Finally, partial bitstreams are generated to be stored in temporary buffer to minimize the occupation of the system bus for memory access.

4.2 Implementation Results

Table 3 shows the gate count synthesized at 54 MHz and the required memory size for the proposed FGS block-level processing core. There are four identical BPC units that perform all bit-, bit-plane- and bitstream- level operations in parallel. Since 4x32 bits are allocated for one partial bitstream for one bit-plane, the required buffer size is small and is realized as registers. The proposed block-level core is optimized for the processing data type and to exploit the parallelism between them.

When integrating the FGS block-level processing core into our previous-proposed Block Engine [6], which is responsible for the base layer texture coding, the partial bitstreams for one MB can be generated within 1,000 cycles. So, our proposed hardware core can support FGS profile level 5, frame size 720x576, 30Hz, for real-time streaming application at 54 MHz, which demands that one MB should be finished in 1,111 cycles. Compared with software implementation, this specification can be achieved at cost of 11 GIPS which implies higher than 11 GHz processor. Our proposed hardware core is a cost-effective solution to FGS implementation.

5. Conclusions

In this paper, a hardwired MPEG-4 FGS block-level processing core is proposed. Its required computation power in an encoding system is analyzed and several hardware-oriented approaches are discussed to achieve cost-effective implementation .The proposed hardware core can support FGS profile level 5, frame size 720x576, 30Hz, for real-time streaming application at 54 MHz.

Table 3. Implementation Results-Required gate count and memory size.

Module	Gate Count	Copy	Subtotal
SUB, ABS	130.67	1	130.67
BPC (Run Length Coder, Packer)	1254.67	4	5018.68
VLC Table(msb)	354	1	354
VLC Table(msb-1)	333.67	1	333.67
VLC Table(msb-2)	246.33	1	246.33
VLC Table(msb-3)	213.33	1	213.33
Zigzag Scan	178.33	1	178.33
Partial Bitstream Buffer	1210.33	4	4841.32
Others			2315.65
Total			13631.98
Size of Memory			
Scan Buffer	64x12, Single Port		

REFERENCES

[1] ISO/IEC 14496-2:1999/FDAM4, "Coding of Audio-Visual Objects – Part 2: Visual, Amendment 4: Streaming Video Profile," Pisa, Jan. 2001.

[2] W. Li, "Overview of Fine Granularity Scalability in MPEG-4 Video Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, pp. 301-317, Mar. 2001.

[3] M. Takahashi et al., "A 60-MHz 240-mW MPEG-4 Videophone LSI with 16-Mb Embedded DRAM," *IEEE Journal of Solid-State Circuit*, vol. 35, pp.1713-17121, Nov. 2000

[4] H. Nakayama et al., "An MPEG-4 Video LSI with an Error-Resilient Codec Core Based on a Fast Motion Estimation Algorithm," *IEEE International Solid-State Circuits Conference*, Sec. 22, Feb. 2002.

[5] MPEG-4 - MoMuSys - FPDAM1 (Version 2) 1.0, Dec. 2002.

[6] C. W. Hsu, W. M. Chao, Y. C. Chang, and L. G. Chen, "Texture Coder Design of MPEG-4 Video by Using Interleaving Schedule," *IEEE International Conference on Multimedia and Expo (ICME)*, Aug. 2002.