Software Defined Implementation of MPEG4 Decoder on Sandblaster SB3500 DSP

Vaidyanathan Ramadurai[1], Mayan Moudgill[2], Daniel Iancu[1], Gary Nacer[3], John Glossner[3]

ABSTRACT

MPEG4 decoder is predominantly used in video broadcast applications, mobile communications, surveillance and security, and multimedia streaming over the internet. Multimedia signal processing algorithms in MPEG4 and H.264 video codecs are characterized by high computational complexity and high memory bandwidth requirements. The decoder uses temporal, spatial and variable length decoding to reconstruct the original frames from the compressed video. The temporal decoding is based on motion compensation which is block based. The block sizes can be 16x16 or 8x8 with half pixel resolutions. Spatial decoding is done using an 8x8 IDCT. Huffman and run-length codes are used in variable length decoding. In addition, MPEG4 also provides various prediction modes for both temporally and spatially coded frames.

An MPEG4 simple profile decoder is implemented on Sandblaster SB3500 which is based on the Sandblaster 2.0 architecture [1] for high resolutions like VGA (640x480) and D1 (720x480). The Sandblaster SB3500 is a compact and power-efficient system-on-chip platform tailored for wireless and multimedia devices. The SB3500 provides a high level of parallelism at the instruction level and at the data level. Additionally the SB3500 employs simultaneous execution of 12 threads thereby allowing real time tasks to run in parallel. Multiple levels of parallelism have been exploited to implement an efficient high resolution MPEG4 decoder. The SB3500 also has facility for block DMAs (Direct Memory Access) as well as scatter/gather DMAs. The DMAs provide efficient movement of high bandwidth data like video reference frames from external memory to local memory.

We provide a brief overview of the architecture of SB3500. After an introduction on MPEG4 simple profile decoder, we provide the details of the software defined implementation of the decoder on SB3500 with focus on optimizations of key video kernels for compute and memory. The entire software is implemented in C without any hardware acceleration. The decoder has been optimized to utilize less than 9% of the total DSP signal processing capacity. We also compare our implementation on SB3500 to a similar implementation on SB3011 which incorporates the Sandblaster 1.0 architecture [2]. The SB3500 provides approximately 6 times better throughput performance, thereby providing the scalability to higher resolutions like 720p and 1080i HD (High Definition).

1. INTRODUCTION

MPEG4 is a multimedia standard adopted by the Moving Pictures Experts Group (MPEG) [3]. A simple profile MPEG4 decoder is defined by the standard for low complexity coding and decoding of rectangular video frames. Implementing an MPEG4 decoder on embedded processors has always been a challenging task. Moreover, the computational and memory requirements for decoding high resolution video streams like VGA is very demanding and requires substantial optimization. In this paper, we present the implementation of a simple profile MPEG4 decoder, executing in real time, on Sandblaster DSP for high resolution VGA encoded video streams. We focus on parallel processing; multithreading and efficient handling of video frames using Direct Memory Access (DMA). We also describe the partition of various blocks in the decoder across multiple threads based on their computational complexity and memory complexity.

This paper is organized as follows. In Section 2, we provide an overview of a simple profile MPEG4 decoder. In Section 3, we discuss the Sandblaster SB3500 multithreaded processor. In Section 4, we describe the optimization of high resolution MPEG4 decoder for SB3500. In Section 5 we compare our SB3500 implementation with the SB3011 implementation results. Section 6 concludes this paper.

2. SIMPLE PROFILE MPEG4 DECODER

The block diagram of an MPEG4 decoder is shown in Figure 1. A typical MPEG4 decoder unit consists of a

1 Optimum Semiconductor Technologies Inc, 120 White Plains Road, Tarrytown, NY 10523, vramadurai@optimumsemi.com

2 Goldman Sachs, NY

3 Sandbridge Technologies Inc, 120 White Plains Road, Tarrytown, NY 10523

Proceedings of the SDR '10 Technical Conference and Product Exposition, Copyright © 2010 Wireless Innovation Forum, Inc. All Rights Reserved

Variable Length Decoder (VLD), Inverse Scanning, an Inverse AC/DC prediction, Inverse Quantizer and Inverse Transform (IDCT). In case of an inter coded block/frame, it is motion compensated before being sent to output. The reconstructed output frame is available in YUV format to be displayed. This procedure is applied for every macroblock and thus for every frame.

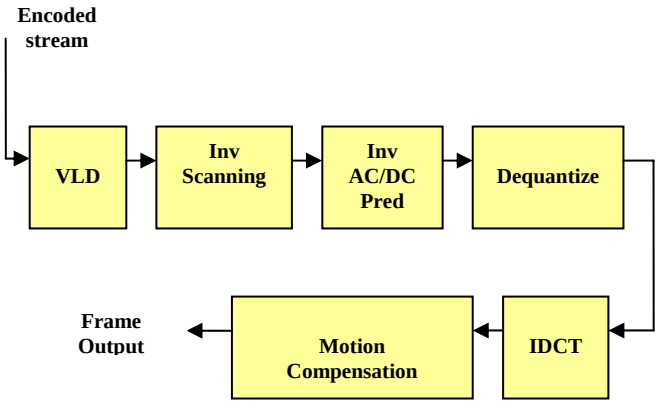

Figure 1: MPEG4 Decoder Block

The encoded data stream is demarcated into frames using start codes and end codes. Every frame is then subdivided into smaller units called macroblocks. Each macroblock is a 16x16 (4 8x8) unit of luminance (intensity) data and 2 8x8 units of chrominance (color) data. This subdivision facilitates in processing the video data in chunks of blocks rather than as an entire frame.

3. SANDBLASTER DSP

The SB3500® features the Sandblaster® DSP for execution of baseband in software – including physical layer. It has a programmable RF interface, with the capability to capture raw data at 240 mega samples per second (MSPS). It includes interfaces to LCD, keypad, USIM, Smartcard, Audio codec, IrDA, plus emerging 'critical' features such as add-on memory cards, camera interface, and USB.

Sandbridge Technologies has developed the Sandblaster architecture [1] for convergence devices. As handsets are converging to multimedia multi-protocol systems, the Sandblaster architecture supports the data types necessary for convergence devices including RISC control code, DSP, and Java.

Figure 2 shows the architecture design of Sandblaster. The design includes a unique combination of modern techniques such as a SIMD Vector/DSP unit, a parallel reduction unit, and a RISC-based integer unit. Each Sandblaster core provides support for concurrent execution for up to four threads of execution. All states may be saved from each individual thread and no special software support is required for interrupt processing. The machine is partitioned into a RISC-based control unit that fetches instructions from a set-associative instruction cache. Instruction space is conserved through the use of compounded instructions that are grouped into packets for execution.

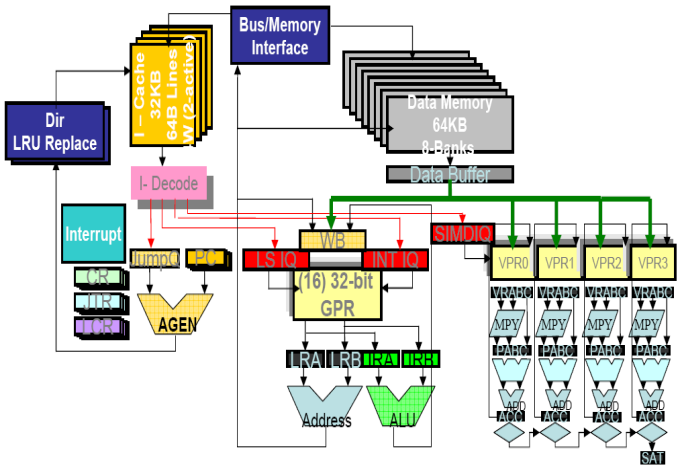

Figure 2 SB3500 DSP core

The memory subsystem has been designed carefully to minimize the power dissipation. The pipeline design in combination with the memory design ensures that all memories are single ported and yet the processor can sustain nearly 16 taps per cycle for a filter (the theoretical maximum) in every thread unit simultaneously. A RISC-based execution unit, shown in the center of Figure 1, assists with control processing. In the case of control code, a 16 entry, 32-bit register file per thread unit provides for very efficient control processing. Common integer data types are typically stored in register files. This allows for branch bounds to be computed and addresses to be generated efficiently. The SIMD/Vector unit depicted on the right side of Figure 2 performs intensive loop processing. Each cycle, a 16x16-bit vector may be loaded into the register file while two vectors are being multiplied, saturated, reduced (e.g. summed), and saturated again. The branch bound may also be computed and the instruction looped on itself until the entire vector is processed. This may be specified in as little as 64 bits.

To enable multimedia processing in software, the processor supports several levels of parallelism. Thread-level parallelism is supported by providing hardware support for up to 4 independent programs to be simultaneously active on a single Sandblaster core. This

Proceedings of the SDR '10 Technical Conference and Product Exposition, Copyright © 2010 Wireless Innovation Forum, Inc. All Rights Reserved

reduces the latency in physical layer processing. Since many algorithms have stringent requirements on response time, multithreading is an integral technique in minimizing latencies. The data-level parallelism (SIMD) is supported through the use of a SIMD Vector unit. Additionally, the compound word instruction set provides instruction level parallelism.

The SB3500 consists of 3 Sandblaster DSP cores with an integrated ARM9 core, peripherals and external DDR memory as shown in Figure 3. A DSP core consists of 4 threads with each thread running at a frequency of 150MHz. Every core has its own local memory of 256Kbytes and an instruction cache memory of 32 Kbytes.

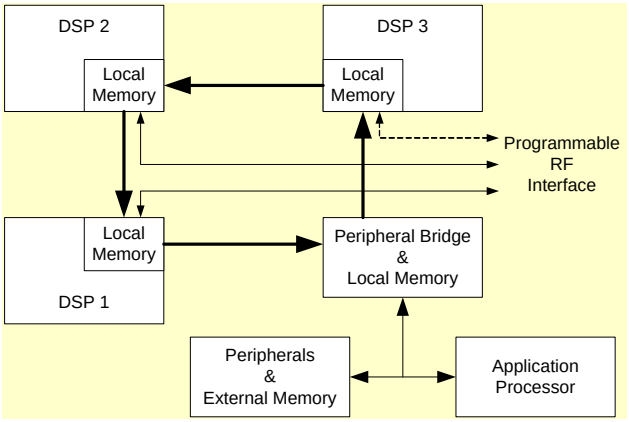

Figure 3 SB3500 DSP core

4. OPTIMIZING MPEG4 DECODER

The decoding routines are classified into parsing intensive functions like the VLD and compute intensive functions like the IDCT and motion compensation. We also subdivide the compute intensive functions into memory-intensive and non-memory intensive functions. For e.g. motion compensation is memory-intensive as it operates on current frame/pixels and reference frame/pixels. IDCT and Dequantize are non-memory intensive functions as they operate on smaller blocks and do not require reference pixels from memory. Memory access becomes intensive depending on the type or level of memory being accessed. The output frames and thus reference frames are stored in external memory or slow memory. Hence accessing these memory regions is extremely expensive. The next level of memory is the local memory or fast memory.

Table 1 shows the computational complexity of the simple profile MPEG4 decoder executed on a VGA (640x480) resolution clip which was encoded at 1mbps, 25 frames per second (fps).

Function	Computation (%)
Variable Length Decode	15%
Inverse Scan, Inv AC/DC	10%
Dequantize	15%
IDCT	20%
Motion Compensation	40%

Table 1: MPEG4 Computational Complexity

Each of these blocks has been optimized for SB3500 to fit in one thread running at 150MHz. Several optimization techniques have been used including 16 way vectorization for IDCT and motion compensation and efficient table lookups for VLD. Memory DMAs have also been effectively used to hide the latency of moving reference and decoded pixels across different memory regions.

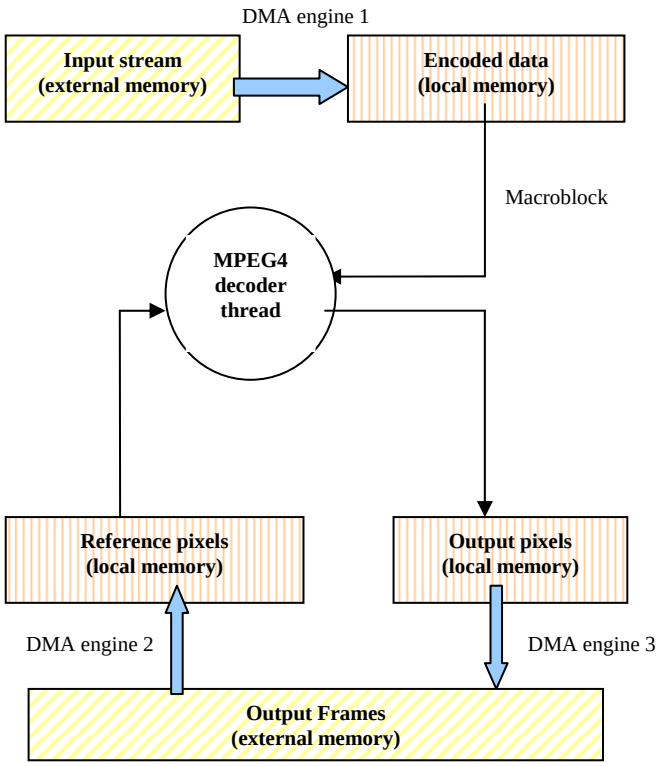

Figure 4: MPEG4 Decoder Architecture

Figure 4 shows the high level architecture for the decoder. The input encoded streams are stored in external or slow memory and are DMA transferred to fast memory or the local memory in smaller chunks. This local data is then processed by a single SB3500 DSP thread. The VLD block performs all the parsing and bit level decoding for one macroblock. The output from the VLD contains information

Proceedings of the SDR '10 Technical Conference and Product Exposition, Copyright © 2010 Wireless Innovation Forum, Inc. All Rights Reserved

to decode one macroblock, like the macroblock type, motion vectors, coded coefficients etc. This information is then transferred to a macroblock control block. The macroblock control block performs all the control routines based on information from the VLD. For example, it checks if the macroblock to be decoded is an intra-coded or an inter-coded macroblock and invokes the appropriate decoding routines like inverse Scanning, Inverse AC/DC prediction, dequantization, IDCT and motion compensation. We will focus on IDCT, motion compensation and frame buffer management for optimization techniques specific to SB3500.

4.1 DISCRETE COSINE TRANSFORM

A one dimensional DCT of the real-valued input vector is first taken. The input vector represents an 8x8 matrix that is stored sequentially one row after the other in a 64 length vector. Essentially, the 8 point 1D DCT of each column in this matrix is taken. This is performed using vector operations that utilize the commonality of various arithmetic operations in the DCT process to speed things up.

1. Given an input x = [x0 x1 ... x7], the 8 point 1D DCT then becomes:

Y0 = x0 + ... x7
Y1 = C1(X0-X7) + C3(x1-x6) + C5(x2-x5) + X7(x3-x4)
Y2 = C2(x0-x3-x4+x7) + C6(x1-x2-x5+x6)
Y3 = -C1(x2-x5) + C3(x0-x7) - C5(x3-x4) - C7(x1-x6)
Y4 = C4(x0-x1-x2+x3+x4-x5-x6+x7)
Y5 = C1(x6-x1) + C3(x3-x4) + C5(x0-x7) + C7(x2-x5)
Y6 = C2(x2-x1+x5-x6) + C6(x0-x3+x7-x4)
Y7 = C1(x4-x3) + C3(x2-x5) + C5(x6-x1) + C7(x0-x7)

Commonly used terms are computed first and then reused to speed things up.

2. Next, the transpose of the resultant DCT matrix is taken using SB3500 vector shuffle instructions.

3. Now, step 1 is repeated on the transposed result.

4. Finally, the result of step 3 is rescaled because the 8 point 1D DCT requires that Y0 be multiplied by sqrt(1/8) and Y1...Y7 be multiplied by sqrt(2/8). Therefore, a combined rescaling is performed for all of the DCT's in both steps 1 and 3. These operations are performed at once using the SB3500 vector instructions.
5. Lastly, the transpose of the rescaled result is taken to return the row/column ordering back to its original form.
The MPEG4 decoder utilizes Inverse Discrete Cosine Transform (IDCT) and the above DCT optimizations techniques are used in the IDCT as well.

4.2 MOTION COMPENSATION

Figure 5 shows an unoptimized code snippet for a motion compensation block. This function interpolates the buffer (which contains the data copied from the reference frame) and then adds it to the differential values decoded from the bitstream. The interpolation done here is both in the "x" and the "y" direction i.e. both in the horizontal and the vertical direction.

```
#define BLOCK_SIZE    8
#define CLIP(N,Max,Min) if((N) > (Max)) (N) =
(Max); \
else if((N) < (Min)) (N) = (Min)

void halfxhaly_mc
(
       UWORD8 *ref9x9,
       UWORD8 *BlkIn,
       UWORD8 *BlkOut,
       WORD32 VopRoundingType
)
{
 WORD16 temp_res;
 WORD8 *temp_ref;
 WORD16 temp;
 WORD32 j;
 WORD32 i;

 temp_ref = ref9x9 + (BLOCK_SIZE+1);

 for(i = 0; i < BLOCK_SIZE; i++)
 {
   for(j = 0; j < BLOCK_SIZE; j++)
   {
     WORD16 temp_res1;
     WORD16 temp_res2;

     temp_res1 =
      ref9x9[i*(BLOCK_SIZE+1)+j] +
      ref9x9[i*(BLOCK_SIZE+1)+j+1];

     temp_res2 =
       temp_ref9x9[i*(BLOCK_SIZE+1)+j] +
       temp_ref9x9[i*(BLOCK_SIZE+1)+j+1];

     temp_res = temp_res1 + temp_res2;

     temp = (BlkIn[i*BLOCK_SIZE+j] +
     ((temp_res + 2 –
       VopRoundingType)>> 2));

     BlkOut[i*BLOCK_SIZE+j] =
                       CLIP(temp, 0, 255);
   }
 }
}
```

Figure 5: Motion Compensation

The input 8x8 pixels are stored in BlkIn and the reference 9x9 pixels are stored in ref9x9. The horizontally interpolated values of a row (row1) are stored in temp_res1 and the horizontally interpolated values of the row below (row2) are stored in temp_res2. The vertically interpolated values from temp_res1 and temp_res2 are stored in

Proceedings of the SDR '10 Technical Conference and Product Exposition, Copyright © 2010 Wireless Innovation Forum, Inc. All Rights Reserved

temp_res. temp_res is then added to input pixels BlkIn, rounded, clipped to an 8 bit quantity and stored into BlkOut.

The SB3500 vector instructions are utilized to optimize this kernel in the following way. For every BLOCK_SIZE elements, temp_res1 is computed using a single vector add instruction. Similarly, temp_res2 and temp_res are each computed using a single vector add instruction. (2- VopRoundingType) is a constant that is broadcasted and stored in a 16 element vector register. Two vector adds and a vector shift thus computes temp. The final clipping is done using a vector minimum and a vector maximum instruction. Thus the inner BLOCK_SIZE loop is executed using only 7 SB3500 vector instructions. Similar optimizations have been used for all possible motion compensation cases, like horizontal only and vertical only interpolations.

4.3 FRAME BUFFER MANAGEMENT

The frame buffer management is also done by the decoder thread. The thread checks if a single row of macroblocks (in case of VGA, this is 40 macroblocks) has been decoded. Once a row of macroblocks has been decoded to local memory, the output DMA engine is started to transfer the decoded pixels to external memory, while the next row of macroblocks is decoded to a double buffered local memory. In this way, the latency of transferring the decoded pixels to external memory is very well hidden.

On the input side, the buffer management is implemented more like a cache. For example, a slice or a portion of the reference frame is DMA transferred from external memory to local memory based on demand. In our current implementation, we break the reference frame into M equal slices, {R1, R2, R3 ...RM} of N macroblocks each. Once the last macroblock is decoded, the first slice R1 of the reference frame is DMA transferred from external to local memory. The size M is calculated based on the video resolution and local memory availability. The video resolution is known beforehand once the header is decoded. The DMA transfer is hidden with actual DSP processing cycles. By the time the motion compensation stage is reached, these reference pixels are available in local memory. Depending on the motion vector, a check is made if the reference pixels are in the local memory or in external memory. If there is a hit, the pixels are taken from local memory, otherwise from external memory. We observed cache hit performance rate of more than 80% for most of the teststreams that were experimented with. Also, significant portion of cache misses were from boundary pixels that depended on reference pixels from the next or previous slice. This can be further improved by storing boundary pixels from the previous and the next reference frame slices.

5. COMPARISON RESULTS WITH SB3011

In [12], we had presented a similar implementation on Sandblaster SB3011. The SB3011 implementation utilized 12 threads, each running at 75MHz and 3 levels of local memory, external, L2 and L1.

Device	MHz	Threads	Local memory utilization (Kbytes)
SB3011	900	12	640
SB3500	150	1	256

Table 2. SB3500 - SB3011 MPEG4 (VGA @ 30fps) comparison

In summary, SB3500 provides approximately 6x performance improvement with less than 50% less memory requirement over Sandblaster 1.0 SB3011 device.

6. CONCLUSION

In this paper, we have described the optimization of a simple profile MPEG4 decoder on Sandblaster SB3500 DSP for high resolution video streams like VGA. By exploiting SB3500 vector instructions and using memory management techniques like DMA for frame buffer handling, we achieve real time performance requirements. The SB3500 implementation utilizes less than 9% of the total DSP signal processing capacity with the capability to scale to higher resolutions like D1 and 720p.

6. REFERENCES

[1] M. Moudgill, J. Glossner, S. Agrawal, and G. Nacer, “The Sandblaster 2.0 Architecture and SB3500 Implementation”, in *Proceedings of the Software Defined Radio Technical Forum (SDR Forum '08)*, Washington DC, October, 2008

[2] J. Glossner and D. Iancu, “The Sandbridge SB3011 SDR Platform” *Symposium on Trends in Communications (SympoTIC’06), Invited keynote, Bratislava, Slovakia, June 24-26, 2006.*

[3] ISO/IEC 14496-2:199/ Amd 1:2000, "Coding of Audio-Visual *Objects - Part 2: Visual, Amendement 1: Visual Extensions", Maui, December 1999.*

[4] P. Li, B. Veeravalli, and A. Kassim, “Design and Implementation of Parallel Video Encoding Strategies Using Divisible Load Analysis," *in IEEE Transactions on Circuits and Systems for Video Technology, No. 9, Vol.15, pp. 1098{1112, September 2005.*

Proceedings of the SDR ’10 Technical Conference and Product Exposition, Copyright © 2010 Wireless Innovation Forum, Inc. All Rights Reserved

[5] John Glossner et al, "Sandblaster low power DSP", IEEE 2004 *Custom Integrated Circuits Conference, 2004, pp 575-581.*

[6] S. Jinturkar, J. Glossner, M. Moudgill, E. Hokenek, "Programming the Sandblaster Multithreaded Processor", *in Proceedings of GSPx 2003.*

[7] Milan Pastrnak, Peter H. N. de With, Sander Stuijk, Jef van Meerbergen Parallel implementation of arbitrary-shaped MPEG-4 decoder for multiprocessor systems in Visual Communications and *Image Processing 2006 (VCIP 2006) , vol. 6077 p. 60771I-1 ... 60771I-10, January 2006, San Jose, California, USA*

[8] V. Ramadurai, S. Jinturkar, J. Glossner, M. Moudgill, "Implementation of H.264 decoder on Sandblaster DSP", *IEEE International Conference on Multimedia and Expo, July 2005.*

[9] Iain E.G. Richardson, "H.264 and MPEG-4 video compression", *Wiley 2003.*

[10] S. Agrawal, S. Jinturkar, V. Ramadurai, M. Moudgill, and J. Glossner, "Multithreading MPEG4 Encoder on Sandblaster DSP", *Proceedings at the 2005 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC), Santa Clara, California, October 24-27, 2005.*

[11] V. Ramadurai, S. Jinturkar, M. Moudgill, and J. Glossner, "Multithreading H.264 Decoder on Sandblaster DSP", *Proceedings at the 2005 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC), Santa Clara, California, October 24-27, 2005.*

[12] V. Ramadurai, S. Jinturkar, J. Glossner, M. Moudgill, "Design and Implementation of a high resolution MPEG4 decoder on Sandblaster DSP", *Proceedings at the Embedded Systems for Real-Time Multimedia, ESTIMedia 2007.*

Proceedings of the SDR '10 Technical Conference and Product Exposition, Copyright © 2010 Wireless Innovation Forum, Inc. All Rights Reserved