

Optimal Porting of Embedded Software on DSPs

Benix Samuel and Ashok Jhunjunwala

ADI-IITM DSP Learning Centre, Department of Electrical Engineering

Indian Institute of Technology Madras, Chennai 600036, India

Email: - benixsamuel@gmail.com, ashok@tenet.res.in

Abstract: Signal processors are evolving more and more powerful with rich instruction set and algorithm driven hardware units. Superscalar, SIMD and VLIW core architectures of DSPs enhance their instruction level parallelism. Blended core architecture, encompassing both microcontroller and signal processor increases the versatility of DSPs. Memory architectures like modified and super Harvard eases multiple memory accesses. Most of the applications/codecs are written in high level languages like c and c++. When they are ported blindly on DSP processor they are highly inefficient, as the DSP compilers have not evolved enough to exploit the tailored resources. Hence optimal porting of embedded software to DSP is indispensable. This paper presents a methodology to port embedded software optimally to a DSP, given the hardware, compiler and profiler. As an example the most common MP3 decoder algorithm is ported on Blackfin processor and optimized stage by stage. The MIPS has been reduced to tens from hundreds.

I. INTRODUCTION

A. Blackfin core

Blackfin's Micro Signal Architecture (MSA) core is jointly developed by Analog Devices, Inc and Intel Corporation. The Blackfin processor core architecture combines a dual MAC signal processing engine, an orthogonal RISC-like microprocessor instruction set, flexible Single Instruction, Multiple Data (SIMD) capabilities, and multimedia features into a single instruction set architecture.

B. Experimental Setup

MP3 decoder algorithm code from underbit technologies is ported on ADSP-BF533 EZ-KIT LITE using the VDSP++ 4.5 tool [1]. ADSP-BF533 EZ-KIT LITE is an evaluation board from Analog Devices, Inc for Blackfin BF533 processor [2]. It has support for audio and video applications with an audio codec and video encoder/decoder. MP3 decoder is ported and executed with the most common bit rate index of 128 kbps, sampling rate of 44.1 kHz and stereo mode.

MIPS calculation

$$MIPS = \frac{\frac{\text{cycles}}{\text{frame}} * \frac{\text{samples}}{\text{second}}}{\frac{\text{samples}}{\text{frame}} * 10^6}$$

Where

Samples/second = 88200 (44.1 K, stereo mode)

Samples/frame = 2304 (stereo)

II. PROCESS TO OPTIMIZE

DSPs have Direct Memory Access (DMA), Cache and efficient memory architectural features [3]. Additionally, high speed ports for serial and parallel data communication for audio and video applications are present. For optimal performance all these system features should be tapped properly. In DSP systems the computing performance depends on the data memory layout significantly i.e. where the data lies. DSP systems have various levels of memory. Based on the speed they are classified as L1, L2, L3 and cache. L1 is the internal memory on chip and closer to the core. L2 is also on chip but farther to core whereas L3 is the external memory (SDRAM). Cache is to improve the performance when data is present in external memory (SDRAM). Blackfin ADSP-BF533 does not have L2 memory. The clock ratio to fetch a data from internal to external memory is 1:8, so diligent data placement is needed for performance benefits. During the code optimization process we must take care of the Instruction set, hardware features and pipeline of the target processor. This can be accomplished by assembly code optimization. Based on the consideration above, a process is defined in Figure 1.

A. Process flow chart

First step, offline implementation and validation involve code acquisition, offline implementation and functionality testing. In the second step, the system is analyzed and the most effective interface for real time is identified and implemented. In the third step, the expensive functions are identified from the profiler and their data dependency is analyzed. Based on the results from step 3, an effective data memory layout is brought up and caching is employed.

Finally the code is traversed, to find the functions which can be ported efficiently in assembly exploiting the architectural features of the processor and instruction set. The identified functions are code ported in processor's assembly language.

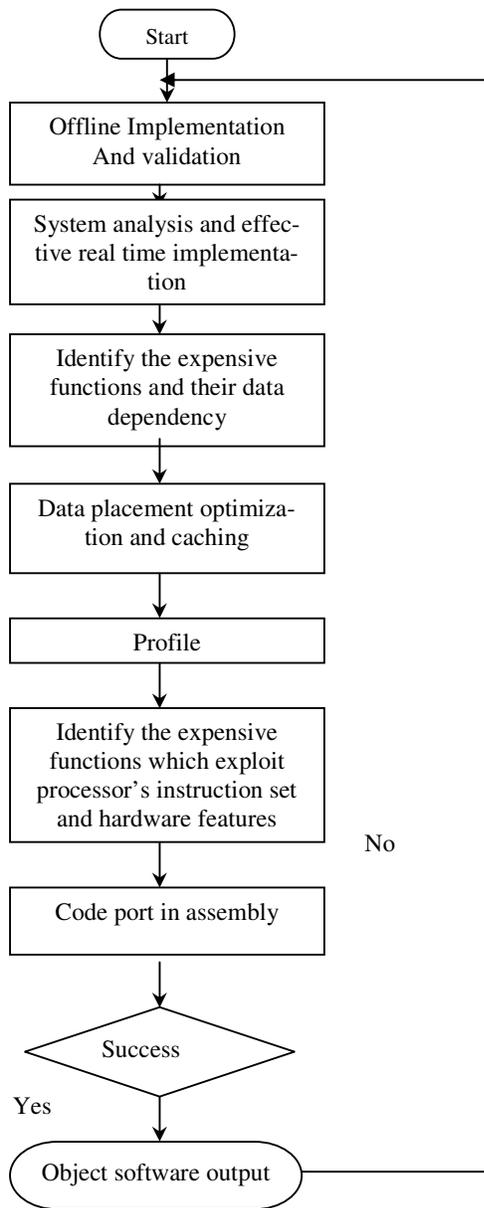


Figure1: Process flow chart

III. OFFLINE IMPLEMENTATION AND VALIDATION

MAD (MPEG Audio Decoder) code built for GCC compiler on Linux platform from underbit technologies, is migrated to DSP integrated development environment, known as VDSP++ (Visual DSP++). VDSP++ supports only ANSI C. MAD code is modified, that all the Linux specific commands and its referred libraries are removed and Blackfin libraries are linked, for this a new linker is created skilfully. It is first implemented offline i.e. the code is executed; *the decoded PCM samples are fully stored in memory and exported through PC sound card.*

Table 1: Offline implementation results

| Mode | Code Memory | Data memory | MIPS |
|------------------------|--------------|---------------|------|
| Offline Implementation | 29,864 Bytes | 347,264 Bytes | 783 |

The decoded results are not played in real time. This is just to confirm the functionality of code and to check any run time error. Offline check was successful. The implementation results are given in Table 1.

IV. SYSTEM ANALYSIS AND EFFECTIVE REAL TIME IMPLEMENTATION

To make the MAD real time, the decoded samples have to be played back through audio codec, interfaced through SPORT of Blackfin, simultaneously as the decoding process goes on. MP3 decoder gives out 1152(2304 for stereo) PCM samples per frame. SPORT offers two modes of operation, multichannel and stereo serial mode. Problem now is to identify the efficient mode for SPORT operation. The important characteristics of multichannel mode are, RFS (Receive Frame sync) signals start of frame, TFS (Transmit Fame sync) is used as "Transmit Data Valid" for external logic, true only during transmit channels and minimum window size is eight[3].

| | | | | | | | |
|----|---|---|---|----|---|---|---|
| LS | 0 | 0 | 0 | RS | 0 | 0 | 0 |
|----|---|---|---|----|---|---|---|

Data layout for multichannel mode

LS Left sample of stereo output
 RS Right sample of stereo output
 Total buffer size for a frame = 36,864 Bytes --- (1)

Whereas in stereo serial mode (I2S format)

| | | | | |
|----|----|----|----|----|
| LS | RS | LS | RS | LS |
|----|----|----|----|----|

Data layout for I2S mode

LS Left sample of stereo output
 RS Right sample of stereo output
 Total buffer size for 2304 samples = 9,216 Bytes --- (2)

So from (1) and (2) I2S mode is selected. The data buffer needed to store decoded samples from one frame is less. The next problem is transferring data from data buffer to SPORT. Two identical buffers, Buffer1 and Buffer2 of same size, 1152 * 4 bytes (a frame is 1152 samples, each sample is 4 bytes) are chosen. When core is writing into Buffer1, DMA will transfer

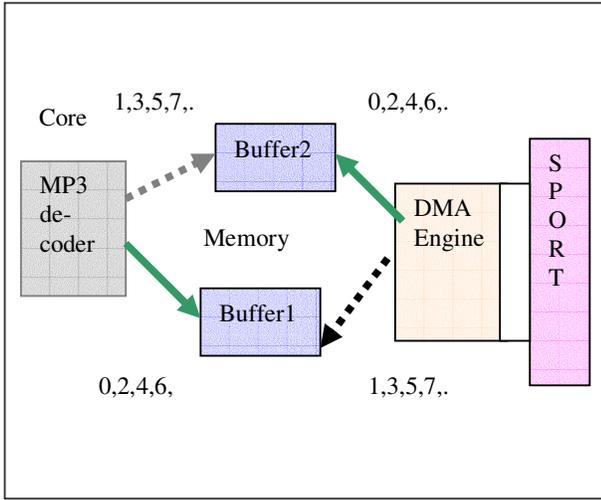


Figure 2: Ping-Pong Buffer

buffer2 to SPORT and both DMA and core will toggle between these two buffers. Core after filling Buffer1, will wait for DMA to finish Buffer2 and then fills Buffer2. Collision of core and DMA bus is now totally eradicated. This technique is called *ping-pong* buffer implementation, shown in Figure 2. This is accomplished by *small descriptor mode* DMA in Blackfin where two descriptors, one for each Buffer (specifying start address, number of bytes to transfer etc) are used to toggle between the two buffers [3]. Now the MP3 code is made real time and run. The output was clear and pleasant. The real time MP3 decoder implementation results are given in Table 2.

Table 2: Real time implementation results for 32 bit PCM

| Mode | Code Memory | Data memory | MIPS |
|-----------------------|--------------|--------------|------|
| Real time MP3 decoder | 29,864 Bytes | 75,080 Bytes | 245 |

The above implementation is for 24 bit PCM output. It is good enough if the output PCM samples are 16 bit wide. Hence the PCM output was rounded to 16 bits and played back. The enhanced results are shown in Table 3.

Table 3: Real time implementation results for 16 bit PCM

| Mode | Code Memory | Data memory | MIPS |
|-----------------------|--------------|--------------|------|
| Real time MP3 decoder | 29,864 Bytes | 70,080 Bytes | 182 |

Table 4: First Profile

| S.No | Functions | % Processor time |
|------|-------------------|------------------|
| 1 | Subband Synthesis | 49.30% |
| 2 | dct32 | 7.47% |
| 3 | Aliasreduce | 6.26% |
| 4 | Overlap | 4.77% |
| 5 | Huffmandecode | 4.15% |
| 6 | Imdct | 3.41% |

V. FIRST PROFILE OF MP3 CODE

Statistical profiler available with VDSP++ measures the performance of a processor program by sampling the target's Program Counter (PC) register at random intervals while the target runs the processor program. Areas of the program where most PCs are concentrated are where most of the time is spent executing the program. The top six functions of the profiled results for MP3 code is shown in Table 4.

VI. DATA PLACEMENT/ACCESS OPTIMIZATION

From Table 4 it is safely concluded that subband synthesis consumes most of the processor resources. When we look into subband synthesis, it transforms the 32 subbands of 18 time domain samples in each granule (576 samples) to 18 blocks of 32 PCM samples, which is the final decoded result. To calculate 32 PCM samples, a large number of calculations are needed as below.

$$V(m) = \sum_{i=0}^{31} N(m,i)S(i), 0 \leq m \leq 63$$

$$N(m, i) = \cos \{ \pi/4 (16+m) (2i+1) \}$$

$$out(i) = \sum_{m=0}^{15} U(i+32m)D(i+32m), 0 \leq i \leq 31$$

where $V(m)$: matrixing results
 $N(m, i)$: coefficients
 $S(i)$: 32 subband samples
 $U(k)$: data from $V(m)$ as $U(64i+j) = V(128i+j)$
 $i = 0$ to $7, j = 0$ to 31
 $Out(i)$: reconstructed samples
 $D(k)$: window coefficients

Initially the D (k) window coefficients of size 32 * 17 * 4 bytes is placed in external SDRAM memory with out enabling the cache. Let us calculate the cycles saved by placing D (k) in internal memory.

No of memory access for D = 512 per 32 output samples
 Cycles gained = 2,006,752
 MIPS reduction = 77

Similar work is done on all the expensive functions. The frequently fetched data is placed internally and less frequently fetched data are kept in the external SDRAM. One bank of 16k bytes of internal memory configurable as SRAM/cache is configured as SRAM to place more data internally and the other bank of 16k bytes is used as cache. The results are given in Table 5.

Table 5: Results after Memory optimization

| Mode | Code Memory | Data memory | MIPS |
|-----------------------|--------------|--------------|------|
| Real time MP3 decoder | 29,864 Bytes | 70,080 Bytes | 91 |

The performance enhancement of the expensive functions after the data placement and access optimization are given in Table 6. MIPS after the data placement and access optimization without cache is equal to 91. Further the MP3 encoded song usually in Mega bytes and the requantization table of size 32kilo bytes are put in SDRAM and *cached* internally. This reduces the MIPS to 76.

Table 6: Cycles taken by expensive functions before vs after data placement optimization are given below

| | | |
|----------------|------------|----------|
| Aliasreduce | 11,863,096 | 7,89,523 |
| Synthesis | 21,087 | 5,100 |
| dct32 | 18,539 | 1,209 |
| Overlap | 2,99,616 | 40,000 |
| Huffman decode | 3,44,612 | 32,516 |

VII. ASSEMBLY CODE OPTIMIZATION

Blackfin's compiler-compiled code is not very efficient as it does not use the resources effectively, so assembly code optimization is indispensable. After data placement and access

optimization, the code is profiled again to choose functions, which when coded in assembly yield better performance enhancement.

A. Second profiling of MP3 decoder

Table 7: Second Profile

| S.No | Functions | % Processor time |
|------|--------------------|------------------|
| 1 | Synthesis | 9.70% |
| 2 | Huffmandecode | 2.40% |
| 3 | Dct32 | 2.3% |
| 4 | III_imdct_1 | 2.2% |
| 5 | Imdct36 | 1.8% |
| 6 | III_stereo | 1.7% |
| 7 | fastsdct | 1.6% |
| 8 | dctIV | 1.6% |
| 9 | Aliasreduce | 1.3% |
| 10 | Overlap | 1.1% |

Among these functions, alias reduction, overlap and fastsdct are chosen to be written in assembly as blackfin's instruction set and features can be exploited to a great extent [4]. As an example, to exhibit assembly code structure optimization, aliasreduce function's assembly code structure optimization is demonstrated below.

B. Assembly optimization of aliasredecce function

Let us have a look into the aliasreduce function shown in Figure3 below. There are 32 butterflies in each column and 8 in each row, each can be computed independent of each other. The weight coefficients are calculated and stored as look up tables. In the original c code, the internal loop runs from 1 to 8 computing the butterflies along the row and outer loop runs vertically from 1 to 32, but the horizontal butterflies need consecutive values of ca and cs, whereas vertical butterflies (each column) relies on the same set of ca and cs value. Number of memory access for butterfly coefficients (cs and ca) when calculating row wise is (32*8) and when calculating column wise is (8*2) is shown in Similarly the other two functions too are written in assembly. The comparison of these functions in C and assembly are given in Table 8.

Table 8: Comparison of functions in C and assembly

| S.No | Functions | Calls /frame | Cycles taken in C | Cycles taken in asm |
|------|-------------|--------------|-------------------|---------------------|
| 1 | Aliasreduce | 1 | 32,516 | 1,758 |
| 2 | Overlap | 84 | 1,209 | 470 |
| 3 | Fastsdct | 4064 | 302 | 100 |

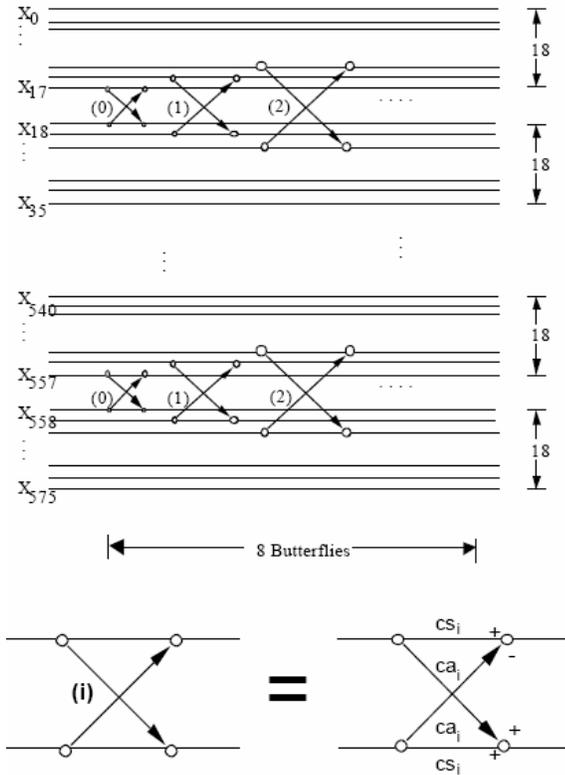


Figure 3: Aliasreduce diagram

MIPS after assembly code optimization is equal to 49. Finally the compiler optimizer switch is enabled by which the compiler itself optimizes the code to some extent. This reduces the MIPS to 22.

VIII. CONCLUSION

Thus the MP3 decoder is optimized to a great extent. The code is tested with 20,000 frames encompassing MPEG-1 and MPEG-2 encoded files. The above results are the average of all. The peak MIPS observed is 24. Table 9 lists the optimization stages and the corresponding MIPS and cycles taken to decode one frame.

Table 9: Final results

| S.No | Optimization stages | Cycles | MIPS |
|------|--|------------|------|
| 1 | Offline Implementation | 20,453,877 | 783 |
| 2 | Real time (32 bit output) implementation with I2S format | 6,400,000 | 245 |
| 3 | Real time (16 bit output) implementation with I2S format | 4,500,000 | 172 |
| 4 | With memory placement Optimization (w/o cache) | 2,469,600 | 91 |
| 5 | With cache enabled | 2,000,600 | 76 |
| 6 | With assembly functions | 1,300,600 | 49 |
| 7 | With compiler Optimization enabled | 480,000 | 22 |

REFERENCES

- [1] Analog Devices, Inc. "VDSP 4.0++" Rev 1.0 Part Number 70-000450-01
- [2] Analog Devices, Inc. "ADSP BF533 EZ-KIT LITE User Manual "Revision
- [3] Analog Devices, Inc."ADSP-BF533 Blackfin® Processor Hardware Reference" Revision 3.1, June 2005 Part Number 82-000555-01.
- [4] Analog Devices, Inc. "Blackfin Processor Instruction Set Reference" Revision 3.0, July 2004 Part Number 82-000410-14.