SYSTEM AND ALGORITHM IMPLEMENTATION TECHNIQUES ON THE TMS320 FAMILY

Panos Papamichalis, Jay Reimer, Jon Rowlands
Texas Instruments
P.O. Box 655474, MS 446
Dallas, TX 75265, USA

ABSTRACT

The widespread use of DSP techniques in many signal processing applications has made algorithm implementations a task that engineers face frequently. However, the real-time constraints that most of these applications have, often clash with the need to do the implementation quickly and with minimal pain. This paper examines tools and techniques that have been created for and applied on the TMS320 family of digital signal processors to facilitate such development. The use of C compilers speeds up the implementation, while software and hardware development tools make debugging easier. Some techniques of algorithm implementation are examined in the context of specific applications.

1. INTRODUCTION

Digital signal processing is no longer a term used only by experts. DSP techniques have been used in consumer products, such as CDs, answering machines, modems, and toys, and now they have made a strong entry into cellular telephony and, the latest buzzword, the Information Superhighway. As the breadth of the applications increases, more and more engineers need to design systems using such techniques. In such cases, the simplest approach is to use existing off-the-shelf programmable DSP devices. The reason for needing fast DSP devices is real-time implementation, a common feature of all the examples above. To make sure that there is minimal waste of computational resources, the algorithm encoding should be done in assembly language. However, this makes code development and debugging a time-consuming and tedious task. So, high-level language availability is very desirable, and almost all of the currently available DSP devices come with such high-level language support, typically in the form of C compilers. The C compilers for the TMS320 family of processors, have reached a very high efficiency level, which minimizes the need of assembly code, as will be discussed below.

Development of efficient algorithms also requires the ability to understand which areas of the code are the most important. It is also quite often valuable to do the development in situ. Both of these capabilities are made available through the debugger and emulation products that support the TMS320 family.

2. DSP ARCHITECTURES

There are certain architectural characteristics that define the digital signal processors. Figure 1 shows the block diagram of such a processor, the TMS320C30, that can use floating-point arithmetic [1]. These characteristics include a hardware multiplier, single-cycle execution of instructions, high execution speed and on-chip memory. The objective behind all of them is to supply a high throughput for real-time applications.

Also, these devices typically use a Harvard architecture, where the program and data spaces are separated in order to avoid bus conflicts that would have an impact on the execution speed. However, newer devices have been returning to a von Neuman architecture, but with multiple buses accessing the same memory space. As an example, the TMS320C2x and TMS320C5x generations of TI DSPs have a modified Harvard architecture on-chip, while the TMS320C3x and TMS320C4x generations have a hybrid Harvard/von Neuman character.

The early generations of DSP devices were envisioned more as embedded

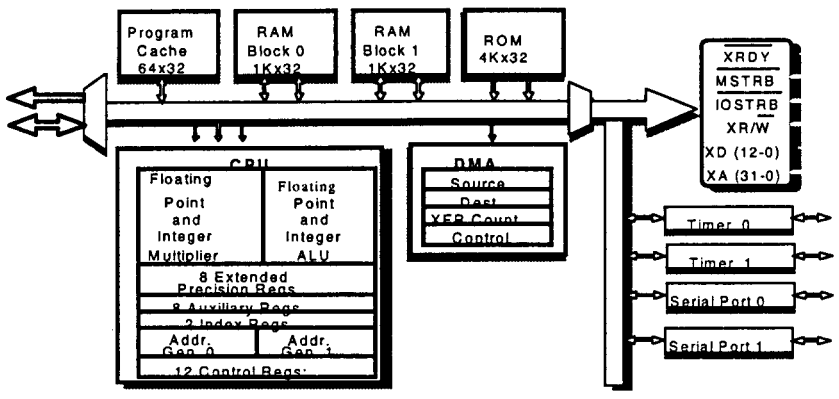

Figure 1. Block Diagram of the TMS320C30 Architecture.

0-7803-2431-5/95 $4.00 © 1995 IEEE

processors to be programmed primarily in assembly. Now, though, C compilers have become almost indispensable as development tools and the newer architectures have been enhanced to make compilation more efficient. Register files, a rich set of addressing modes, and the availability of a software stack help the compiler performance.

3. AN APPLICATION EXAMPLE

A large variety of algorithms have been implemented on the TMS320 family of processors, and many of these algorithms are available either as freeware or as licensable code. To discuss the aspects of algorithm implementation on DSP devices an application example, the MPEG-1 audio standard [2], will be used.

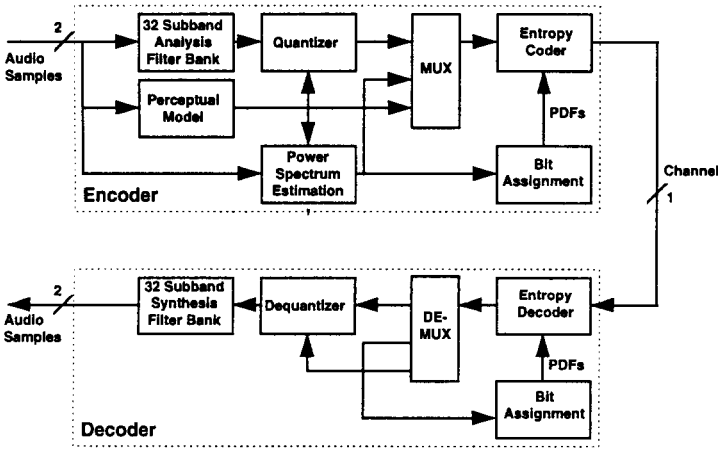

Figure 2. Block Diagram of the MPEG-1 Encoder and Decoder

The MPEG-1 audio standard (encoding and decoding for algorithm layers 1 and 2) has been implemented in real time on TMS320C3x floating-point processors. Figure 2 presents the encoder and the decoder of the MPEG-1 audio standard. In this example the implementation is on a single processor, and it took some ingenuity to make this computationally intensive algorithm fit into the available resources of the device.

The encoder and decoder were implemented mostly in the C language, using assembly language in critical blocks. Figure 3 compares the use of C and assembly language in the core algorithms. The functions chosen for optimization in assembly language are clearly execution "hot spots".

Both the encoder and decoder require "traditional" DSP operations such as linear filtering, cosine and Fourier transforms, operations which make good use of the specialized DSP architectural features of the 'C3X. Several of these functions were reused directly from the TMS320 applications library, where they are available already optimized. Conversely, there are many operations with little structural regularity, such as the encoder bit assignment and perceptual model. Their execution time is dominated by branches and data structure access. These operations cannot use the DSP architectural features as efficiently as dataflow operations, and the C compiler produced comparable results to hand optimization in these cases, with reduced effort. Finally, there are operations which fall between these extremes. It was found that the compiler's understanding of the 'C3X architecture allowed it to produce efficient code in most such cases, particularly once the strengths and limitations of the optimizer were understood.

	Instrs	Instrs %	Cycles	Cycles %	# Execs per Instr
Encoder C	4558	86%	194864	50%	42.8
Encoder Asm	744	14%	191773	50%	257.8
Decoder C	2296	87%	56905	29%	24.8
Decoder Asm	347	13%	137135	71%	395.2

Figure 3. Comparison of C and Assembly Language in the MPEG-1 Encoder and Decoder

The internal code and date memory and the instruction cache of the 'C3X devices were used to reduce the external memory bandwidth requirements of the algorithms. External data was copied into the internal memory before processing, and code loops were optimized to fit inside the instruction cache. Figure 4 summarizes the effect of these optimizations. With the cache on, the cost of a wait state was significantly reduced. (The normalized cost of a wait state with the cache off was less than one cycle because of idle states in the memory interface.)

Cache	Normalized Execution Time
Off	1.00 + 0.78 * number of wait states
On	0.91 + 0.25 * number of wait states

Figure 4. Normalized Execution Time of the MPEG-1 Decoder vs Number of External Memory Wait States

4. PROGRAMMING IN C VERSUS ASSEMBLY

Since the DSP devices were developed for computationally-intensive, real-time applications, it was initially envisioned that all the programming would be done in assembly. And this still holds true when we come to the most time-critical sections of the algorithm. However,

the emphasis has been gradually shifting towards the use of high-level language for most of the non-repetitive and non-time-critical code. This makes the flow of the implementation much easier to follow (and debug). The compiler technology has made significant strides to get close to the hand-assembled code for a device. Figure 5 shows an example of that evolution for the C compiler for the TMS320C30. The code size and the execution time in cycles are shown for an audio decoding algorithm. The levels of optimization, going from zero (no optimization) to two, roughly correspond to the evolution of the compiler over time.

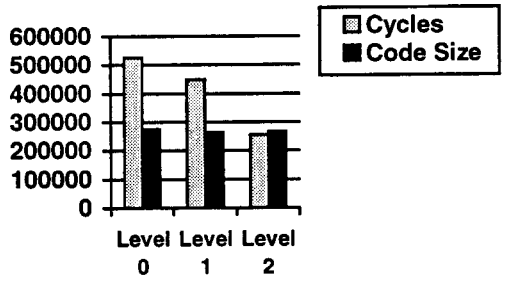

Figure 5. Performance of the TMS320C30 C Compiler on an Audio Decoding Algorithm.

5. ALGORITHM IMPLEMENTATION ISSUES

To write efficient code for a real-time application, there are some recommended practices that presume a good understanding of the available resources of the device. The TMS320 series devices include a number of architectural resources for DSP applications. Some of these can be optimized tactically, or locally, such as:

- fast multiply and multiply-accumulate operations
- arithmetic and address register files
- parallel and pipelined execution units
- delayed branches
- zero overhead loops

Compilers are increasingly effective at tactical optimizations, particularly of the register file usage, instruction scheduling and control flow. For recent TMS320 devices, the C compiler generates code which can approach the efficiency of hand optimized code using all of the resources from the above list.

Some resources must be optimized strategically, at the level of the system design. This is usually because they must be dedicated to one function over an extended period. On the TMS320 devices such resources include:

- fast internal memory
- instruction cache, due to size and replacement policy
- bit reversed and circular addressing modes, due to data address alignment requirements
- peripherals such as DMA controllers, timers, serial ports and interprocessor communications links
- interrupts

Strategic optimizations by definition may have widespread implications for the rest of the system, making them difficult for a compiler to optimize using only local information. However an initial C language algorithm implementation is useful in confirming the effect of strategic optimizations, since it provides the most flexibility for experimentation.

Fast internal memory often allows two accesses per instruction cycle, required by many DSP operations including the multiply-accumulate. This often means that internal memory should be reserved for data. If external data accesses can be predicted, then the internal memory can be used as a cache by copying data to it for processing. If external accesses are not predictable, then executing code from internal memory can increase the bus bandwidth available for data. The 'C3X instruction cache also increases available bus bandwidth. Inner loops can execute entirely from the cache if optimized appropriately. Interrupts disrupt the cache by causing multiple reloads, however this can be minimized by placing interrupt code in internal memory, which does not require or use the cache.

The use of fixed-point versus floating-point is a complex subject that should take into account the need for precision in the algorithm, the available capabilities in the processor, and the computational needs of the implementation. More often than not, the choice is dictated by chip cost considerations, since the fixed-point devices are typically cheaper than the floating-point ones.

6. SYSTEM IMPLEMENTATION ISSUES

When dealing with the overall system that is handling an application, there are issues associated with external interfaces of the DSP device. Fast and slow-memory arrangement, use of serial and parallel ports, as well as DMA should be taken into consideration. To facilitate such interfaces while permitting the developer to focus on the development of the algorithm, operating systems for DSPs have been developed. Most TMS320 devices use the SPOX operating system, developed by Spectron Microsystems. You can also use libraries of functions available with such operating systems or independently,

but you should be careful with the overhead imposed by such functions. Occasionally, this overhead is more than is necessary in a particular application because the library developers want to make a function more generic.

Two current trends in applications are multitasking and multiprocessing, pointing to the two different ends of the applications spectrum. In multitasking, you try to have multiple applications running concurrently. This is attractive in multimedia environments where you want to use the same device for multiple purposes. However, partitioning the resources so that no application runs out of MIPS or memory is a challenging task. On the other hand, there are applications that have such a high computational load that no single processor can keep up with them. In this case, the algorithm is partitioned to run on multiple processors. Careful selection of processors with rich interconnection capabilities, such as the TMS320C40 can help significantly in the implementation.

7. DEBUGGING TOOLS

In implementing an algorithm, the available debugging tools can make a significant difference in the ease of development and the elimination of bugs. The TMS320 family of processors comes with Windows-based tools that provide debugging capabilities in both the C and the assembly level. The supplied profiler can help to identify the segments of the code where most of the computation is done, and pursue optimization there. This is beneficial, since the code can be developed initially in C. The obvious benefits are a shorter amount of time is spent in developing an initial functioning version of the code, adaptations or variations can be more readily developed, and a maintainable reference for the final optimized version is available.

For the board-level implementations, there is scan-based emulation available with an interface similar to the simulator. This benefits the developer since there is no need to learn a different debugging environment at any stage from initial simulation to final target system debug. It further benefits the target system design by not imposing higher speed design consideration due to probes on the external busses or due to multiplexing of a monitor interface with the external memory interface. Also, since control of the devices is accomplished via the scan interface there is no need to reserve any of the on-chip memory or interrupt resources specifically for debug support.

8. AVAILABLE APPLICATIONS ON TMS320

A significant number of applications have been developed for the TMS320 family of digital signal processors. These algorithms cover the areas of speech coding and recognition, control, telecommunications, etc. Assembly code for fundamental functions, such as FIR and IIR filters, FFTs, and extended-precision arithmetic can be found in the applications chapters of the corresponding User's Guides. The same code plus freely available algorithm implementations, such as the G.721 standard, are available in electronic form on the TI Bulletin Board System. A number of application notes are available on several volumes of application books [3]. Furthermore, there is a list of algorithms available from third parties that can be licensed [4].

9. FUTURE TRENDS

The trend seems to be an acceleration in technology which in turn often means that product design-cycles must be shorter because product life tends to be shorter. The result is that it must be simple and effective to design and develop systems. Programmable DSPs offer an effective means to quick development and meet performance objectives because of several reasons. One is that the technology of the devices themselves has led to greater performance and greater levels of integration. A second is that the tools used in development have also improved, providing greater insight and understanding of how a particular system operates. A third is that the increasing amount of DSP literacy and the maturity of the technology in the market has produced a resource of algorithms and peripherals that can be utilized in the next round of designs.

REFERENCES

[1] *TMS320C30 User's Guide,* Texas Instruments, Inc., 1991.

[2] *ISO/IEC IS 11172-3 - Coding of Moving Pictures and Associated Audio for Digital Storage Media At Up To About 1.5 Mbit/s - Part 3: Audio,* ISO 1993

[3] *Digital Signal Processing Applications with the TMS320 Family,* Prentice Hall, vol. 1 (Kun Lin, Ed., 1987), vol. 2 (P. Papamichalis, Ed., 1991), vol. 3 (P. Papamichalis, Ed., 1991)

[4] TMS320 Software Cooperative, Texas Instruments, Inc., 1994.