Guidelines for Software Development Efficiency on the TMS320C6000 VelociTI Architecture

WHITE PAPER: SPRA434

Authors: Marie Silverthorn
Leon Adams
Richard Scales

Digital Signal Processing Solutions
April 1998

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI and VelociTI are trademarks of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Guidelines for Software Development Efficiency on the TMS320C6000 VelociTI Architecture

Abstract

This white paper recommends programming and reuse techniques designed to develop efficient software applications for the Texas Instruments (TI™) TMS320C6000 digital signal processor (DSP). The TMS320C6000 DSP belongs to a powerful generation of DSPs designed with the TI VelociTI™ advanced VLIW architecture.

As DSP capability increases, product designers create more complex software applications. Responding to the desire of our clients to create and reuse code efficiently, Texas Instruments is committed to creating software development tools that support increasingly efficient, powerful, and easy to use programming environments.

A new feature of the Texas Instruments C6000 code is its compatibility with future C6000 platforms. This includes ANSI C code, ANSI C code with TI C6000 Language extensions, and TI C6000 Linear Assembly code.

The programming guidelines recommended in this paper include an efficient C6000 programming technique and offer references for additional detail. A code reuse efficiency diagram and development flow chart are included to aid understanding.

We highly recommend using C with TI C6000 Language Extensions as the software development language for TI C6000 software development.

Product Support on the World Wide Web

Our World Wide Web site at **www.ti.com** contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

Introduction

Texas Instruments developed the TMS320C6000 DSP architecture with performance and high level programming capability as key criteria. VLIW (Very Long Instruction Word) techniques are used to maximize parallelism and minimize overhead, which in turn maximize the performance capabilities of the architecture. The architecture was tuned to the C compiler. The DSP hardware and software development tools were developed jointly. TI's programming tools enable development teams to program in C, which reduces development time and enables software to be reused on multiple DSP VelociTI platforms.

The efficient optimizers in the tool set schedule code to match the intricacies of the highly parallel pipeline without requiring the programmer to understand the hardware architecture. Programming can be completed in reusable C or Assembly Language. Tools optimize the performance and schedule directly to the parallel pipeline.

The guidelines presented in this paper aim to:

- Reduce software development cycle time
- Reduce software development cost
- Increase software quality
- Achieve the software performance required by the application

To achieve these goals, the guidelines

- Enable the reuse of software assets through the use of ANSI C, ANSI C with TI C6000 Language Extensions, and TIC6000 Linear Assembly language on multiple DSP VelociTI chipsets
- Use an abstract yet more powerful language for programming (ANSI C and ANSI C with TI C6000 Language Extensions)
- Automate the tedious task of optimizing the code to the C6000 pipeline with the C and Linear Assembly optimizers
- Reducing errors (and rework) in coding and optimizing assembly code through the more abstract language and automation of tasks

DSP Product Line Compatibility

The C6000 is compatible with the following five types of code:

- ANSI C code
- ANSI C code with TI C6000 Language Extensions
- TI TMS320C6000 Linear Assembly code
- Scheduled assembly code
- Object binaries

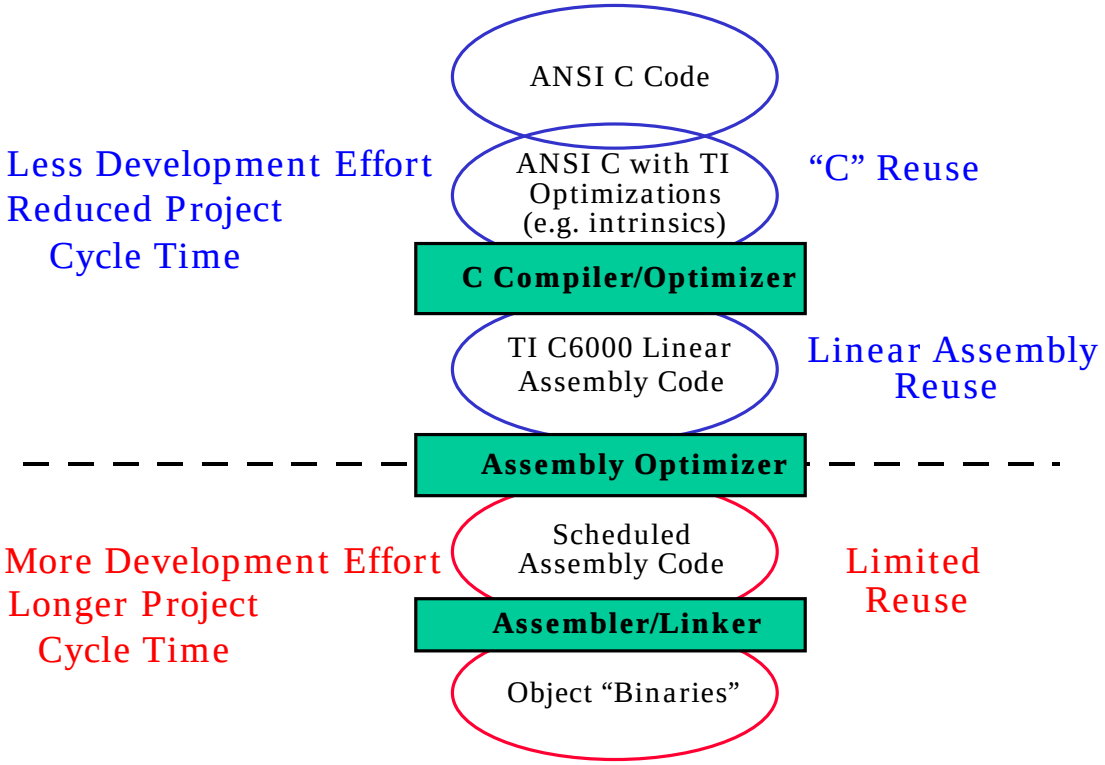
Beginning with the C6000 Instruction Set Architecture (ISA), Texas Instruments supports ANSI C, ANSI C with TI C6000 Language Extensions (including intrinsics), and a C6000 Linear Assembly Language that can all be reused on future C6000 platforms. The investment in code you write today can be reused on future C6000 DSPs.

Figure 1 shows the three levels of source code for software development and reuse of software on the C6000:

- ANSI C
- ANSI C with TI C6000 Language Extensions (including intrinsics)
- TI C6000 Linear Assembly

ANSI C source code is an industry standard that can be input into the TI C compiler and compiled/optimized (see Figure 1). It is more abstract than assembly code and takes less programming resources. The TI compiler optimizes and schedules C code for the TI DSP. This offers many code development efficiencies but, in some specific cases (for example, tight loops) may not generate sufficient performance.

Figure 1. TMS320C6000 Code Reuse Efficiency Diagram



When additional performance or control needs addressing, the programmer can include TI Optimization Techniques (including C6000 Language Extensions) directly in the ANSI C code stream. TI Language Extensions are C functions that map directly to the C6000. This moves control of the code from the C compiler to the programmer. The combination of C and TI C6000 Language Extensions is a powerful combination for efficiently developing optimized and reusable software while maintaining direct control or literal performance tuning access.

We highly recommend using C with TI C6000 Language Extensions as the software development language for TI C6000 software development.

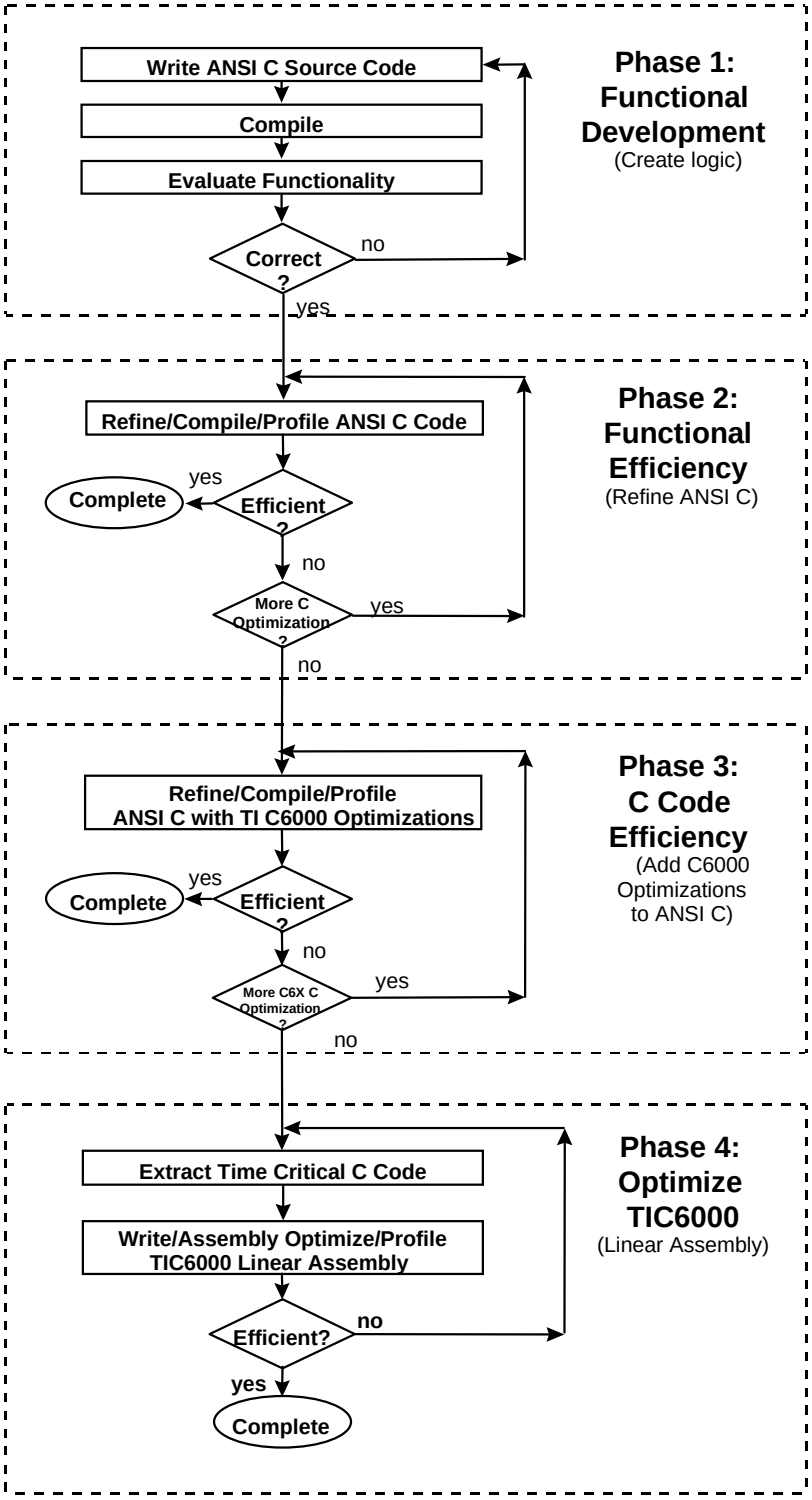
If more control is required to meet performance goals, another language, TIC6000 Linear Assembly, can be used. Linear Assembly is the C6000 machine instructions written in a serial, or linear, fashion with variable operands. The assembler optimizer takes the Linear Assembly as input and schedules the code for parallel pipeline of the DSP.

Because the C6000 is highly parallel and flexible, the task of scheduling code in an efficient way is best completed by the TI code generation tools (compiler and assembly optimizer) and not by hand. This results in a convenient C framework to maintain code development on current products as well as reuse the same code on future products (C code, C with TI C6000 Language Extensions, and the Linear Assembly source can all be reused on future TI C6000 DSPs).

The final territory of programming is the hand-scheduled assembly language. At this level, the programmer is literally scheduling the assembly code to the DSP pipeline. Depending on the programmer's ability, she may achieve results similar to those achieved by the C6000 tools; however, she risks “man-made” pipeline scheduling errors that can cause functional errors and performance degradation. In addition, scheduled code may not be reusable on future C6000 family members, unlike the three levels of C6000 source code. Consequently, avoid the hand-coded assembly level “Limited Reuse” programming if at all possible.

Recommended DSP Software Development Flow

Figure 2. Recommended DSP Software Development Flow



Developing software code is one phase of the product development process. Understanding the product requirements and creating a high level design before coding ensures the product goals are understood and reduces rework during code development. When these phases are complete, the software code can be developed using these steps:

NOTE:
If at any point in the flow, the desired performance is achieved, there is no need to move to the next stage. Each of the stages in the code development involves passing more information to the C6000 tools. Even at the final stage, development time is greatly reduced from that of hand coding, and performance approaches the best that can be coded by hand.

1) **Develop product functions** (logic, interface) with high level ANSI C language. Obtain (create or reuse) standard portable C code that meets the functional requirements of the product. This may begin on a workstation (PC, Sun, HP, etc.), and migrate to the C6000. A floating-point model may be built first, then the corresponding fixed-point model. Evaluate the code for functionality (that is, logic, input/output interfaces, security, etc.).

2) **Refine, compile, and profile ANSI C code for functional efficiency**. The C6000 profiling tools enable you to check the code's performance. If your code meets your performance requirements, the development is complete.

   If your code is not as efficient as you need, change compiler options (to enable software pipelining, reduce redundant loops, etc.). Compiler experts have built extensive algorithms and heuristics into this optimizer to increase product performance.

   All of phase one and two can be accomplished by editing only the ANSI C code. (For details on optimizing ANSI C code on the C6000, see Section 3.1 and Section 3.2 in the *TMS320C62X/C67X Programmer's Guide* and Chapter 2, "Using the C Compiler," in the *Optimizing C Compiler User's Guide*.)

3) If more performance is needed, **refine your ANSI C code by adding the TI C6000 Language Extensions.** TI Language Extensions let the compiler make use of symbolic references and better use of registers than can be accomplished with ANSI C or in-line assembly. Although unique to the VelociTI product family (and portable to other TI VelociTI product lines), TI Language Extensions can quickly improve performance of

your C code. TI Language Extensions include the following examples:

- Saturated instructions Language Extensions (see Section 3.3.1 in the *TMS320C62X/C67X Programmer's Guide* for details).
- Unrolling the loop (expanding small loops so that each iteration of the loop appears in your code. This increases the number of instructions available to execute in parallel (see Section 3.3.3.4 in the *TMS320C62X/C67X Programmer's Guide* for more detail).
- Using word (int) access to read two short values at a time with Language Extensions to operate on the data. This can double the performance of some loops (for more information, see Section 3.3.2.1 in the *TMS320C62X/C67X Programmer's Guide* and Chapter 3, “Optimizing Your Code," in the *Optimizing C Compiler User's Guide*)

After applying these techniques, analyze your performance with optimizer options. If you meet your product’s functional and performance requirements, your development is complete.

4) **Optimize the TIC6000 Linear Assembly code**. (Those who prefer to code in Linear Assembly begin here.)

If the required performance is not achieved after following these steps, write Linear Assembly by extracting the time-critical areas from your C code and rewriting the code in Linear Assembly.

Expert DSP assembly programmers may be able to achieve greater optimization than the tool; however, optimal implementations require exhaustive searches that machines can perform many times faster and generally are not portable to other DSP platforms. (For additional details, see Section 4, “Using the Assembler Optimizer," in the *Optimizing C Compiler User's Guide*.)

Benefits

DSP software developers often jump into coding at the assembly language level at the project start without understanding either the product requirements or the alternative design solutions. Issues with product functions, performance, and interfaces with hardware are not resolved and cannot be reworked until found during testing.

At Texas Instruments we have developed a Product Development Process: requirements, high level design, and low level design phases are complete *before* linear assembly coding. Although these activities add time at the beginning of software development, they typically reduce overall cycle time by one-third to one-half. The final product arrives to market sooner.

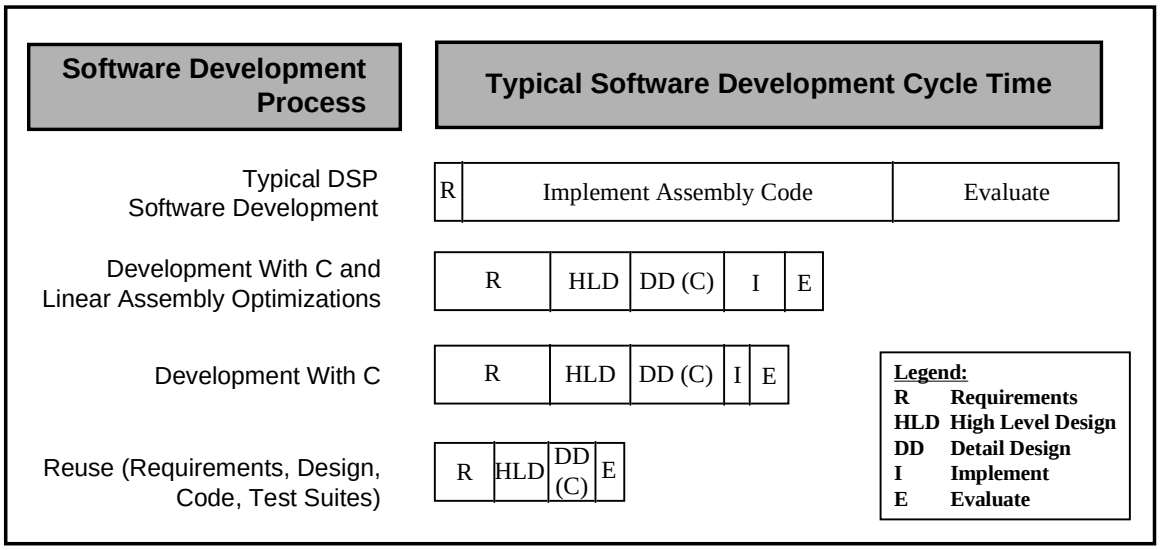
In the design phase, the developer can use ANSI C to design the code. This enables the developer to work **in a more abstract and powerful language** than assembly language. The developer can focus on developing the product functions and algorithm instead of tuning the algorithm to the hardware platform. TI's compilers and optimizers translate C source code into the assembly code instructions. The programmer no longer needs to learn the intricate details of the C6000 architecture.

The code development may be complete at this point. If additional performance is required, the developer can add TI C6000 Language Extensions to the ANSI C before working at the assembly level. Because the developer has created fewer lines of C code than assembly for the same capability, less work was done, reducing product development cycle time and effort.

Reuse: Not just code but requirements and designs can be reused on other DSP products. This eliminates work and reduces product development time.

Code developed with TI tools in ANSI C, C with C6000 Language Extensions, or in TIC6000 Linear Assembly can be reused with future generations of the C6000 architecture family.

Figure 3. Software Development Cycle Time Results



Appendix A. TI TMS320C6000 Optimiztion Checklist

Feedback	Solution
Loop Carried Dependency Bound is much larger than Unpartitioned Resource Bound	C Code Use -pm program level optimization to reduce memory pointer aliasing *(Ref 3.2.2.2 in TMS320C62X/C67X Programmer's Guide)* Add const declarations to all pointers passed to a function that are read only *(Ref 3.2.2.1 in TMS320C62X/C67X Programmer's Guide)* Use -mt option to assume no memory pointer aliasing *(Ref 3.2.2 in TMS320C62X/C67X Programmer's Guide)* Linear Assembly Make sure instructions accessing memory at the beginning of the loop do not use the same pointer variables as instructions accessing memory at the end of the loop *(Ref 5.6.3 in TMS320C62X/C67X Programmer's Guide)*
Partitioned Resource Bound is higher than Unpartitioned Resource Bound	Write code in Linear Assembly with partitioning/functional unit information *(Ref 5.3.4 in TMS320C62X/C67X Programmer's Guide)*
Too many instructions, or inefficient instructions were generated by the compiler	Use intrinsics in C code to pick force more efficient C6000 instructions *(Ref 3.3.1 in TMS320C62X/C67X Programmer's Guide)* Write code in Linear Assembly to pick exact C6000 instruction to be executed *(Ref 4.3 in Optimizing C Compiler User's Guide)*
Failed to software pipeline due to Register live too long	Write Linear Assembly and insert MV instructions to split register lifetimes that are live too long *(Ref 5.9.4.4 in TMS320C62X/C67X Programmer's Guide)*
Failed to software pipeline due to register allocation	Try splitting the loop into two separate loops If multiple conditionals are used in the loop register allocation of the condition registers could be the reason for the failure. Try writing Linear Assembly and partition all instructions writing to a condition register evenly between the A and B sides of the machine. If there is an uneven number, put more on the B side as there are 3 condition registers on the B side and 2 on the A side.

Feedback	Solution
T address paths are Resource Bound	C Code Use word access for short arrays – declare int * and use mpy Language Extensions to multiply upper and lower halves of registers *(Ref 3.3.2 in TMS320C62X/C67X Programmer's Guide)* Try to employ redundant load elimination technique if possible *(Ref 5.10 in TMS320C62X/C67X Programmer's Guide)* Linear Assembly Use LDW/STW instructions for accesses to memory *(Ref 5.3 in TMS320C62X/C67X Programmer's Guide)*
There are memory bank conflicts (specified in the memory analysis window of simulator)	Write Linear Assembly and use the .mptr directive *(Ref 5.11 in TMS320C62X/C67X Programmer's Guide)*
Large outer loop overhead in nested loop	Unroll inner loop *(Ref 3.3.3.4 and 5.8 in TMS320C62X/C67X Programmer's Guide)* Make one loop with outer loop instructions conditional on an inner loop counter *(Ref 5.13 in TMS320C62X/C67X Programmer's Guide)*
Uneven resources (i.e., 3 multiplies per loop iteration)	Unroll loop to make even number of resources *(Ref 5.8 in TMS320C62X/C67X Programmer's Guide)*
Two loops are generated, one not software pipelined.	Use _nassert statement to specify loop count info *(Ref 3.3.3.3 in TMS320C62X/C67X Programmer's Guide)*
Loop will not software pipeline for other reasons	Make sure there are no function calls, branches to other code, or conditional break statements in loop *(Ref 3.3.3.5 in TMS320C62X/C67X Programmer's Guide)* Try making the loop counter downcounting *(Ref 3.3.3.1 in TMS320C62X/C67X Programmer's Guide)* Remove any modifications to the loop counter inside the loop *(Ref 3.3.3.5 in TMS320C62X/C67X Programmer's Guide)*

Appendix B. Reuse libraries

One way to reduce development time is to avoid developing new code by reusing existing proven code. Texas Instruments continues to promote libraries.

The Texas Instruments TMS320 **Third Party Program** is the most extensive collection of Digital Signal Processing development support in the industry. Over 250 independent companies and consultants provide development boards, operating systems, software algorithms, function libraries, and system consulting services around the world. Types of applications include vocoders, speech recognition/ synthesis, audio, telecommunications, image/video, and run-time support libraries. For a current listing of available software, visit the world wide web at:

http://www.ti.com/sc/docs/dsps/develop/freeman.htm

The DSP North American Business Development organization is developing the following libraries:

- **General 6Cx kernels**: FFTs (block floating point, complex, Radix-2, Radix-4, real), FIRs (real block, single-sample, complex block, LMS adaptive), IIRs, vector manipulation (dot product, add, maximum), convolution encoder, and finite state machine
- **Math functions**: arithmetic (single-precision, double precision, 32-bit, pseudo-double precision), arctangent, auto-covariance, autocorrelation, convolution and correlation, convolution encoder, cross-correlation, cross covariance, log, matrix manipulation, trig functions
- References to www freeware/shareware. Software with unknown levels of quality control, including university software. Examples include: Vocoders (G.711, G.721 ADPCM, etc.), Parametric coders (CELP, G.728, etc.), Waveform coders (G.711, ITU G.726 ADPCM, etc.), Speech Recogniton/Synthesis (Text-to-Speech, Speaker Independent, etc.), Audio (Karaoke, Graphic Equalizers, etc.), Control, Telecom (Acoustic Echo Canceller, Line Echo Canceller, etc.), Image (JPEG Still Image, H.324, etc.)

For additional information, contact your Texas Instruments field sales technical staff.

Appendix C. References

Texas Instruments, *TMS320C62x/C67x Programmer's Guide*
http://www-s.ti.com/sc/psheets/spru198b/spru198b.pdf

”C6x Code Optimization Checklist” by Richard Scales, pp 3-4 and 3-5,
T*MS320C62x/C67x Programmer's Guide*
http://www-s.ti.com/sc/psheets/spru198b/spru198b.pdf

Texas Instruments, *TMS320Cx Optimizing C Compiler User's Guide*
http://www-s.ti.com/sc/psheets/spru187c/spru187c.pdf