White Paper
SPRA581C - May 2002

The TMS320 DSP Algorithm Standard

Steve Blonstein Technical Director

ABSTRACT

The TMS320 DSP Algorithm Standard™, also known as XDAIS, is part of TI's eXpressDSP™ initiative. The purpose of the standard is to reduce those factors that prohibit an algorithm from being easily integrated into a system without significant re-engineering. Many of the unknowns in such a situation relate to resource allocation and consumption on a DSP. Bugs often occur during system integration as a result of the algorithm designer's unfounded assumptions about the system into which the algorithm is to be integrated. The standard, therefore, focuses on a set of general rules and guidelines that should be applied to all algorithms. In addition, all algorithms must comply with a memory management API, called IALG. For those algorithms utilizing DMA, the IDMA interface must be implemented. Finally, specific rules and guidelines are provided for each family of TI DSPs. A XDAIS developer's kit provides the standard itself, example code, and a demonstration.

Contents

List of Figures

1 Background

Texas Instruments has long understood the importance of good development tools that typically include a C compiler, linker, emulator, and debugger. Since improvements in each of these tools translate to an improvement in productivity for the developer, TI invests significant time to continuously improve these tools. Our 1998 acquisition of GO–DSP and the 1999 release of Code Composer Studio™ illustrated TI's commitment to push the tool environment forward. Additionally in 1999, TI launched eXpressDSP with a major focus on improving the developer experience with the code base on the target DSP. Before this, DSP applications typically revolved around proprietary technologies instead of standards. The result was a splintered mass market for DSP, whereby the entire target code implementation was left to the customer who, despite the progress of C compilers, often had to write much of the code in assembly to achieve the performance required to justify selecting a DSP. With the advent of many telecommunication, imaging, and video standards, a market developed for TI third parties to develop, market, and distribute commercial off-the-shelf (COTS) algorithms.

Since today's applications often require the use of several such COTS algorithms, it is feasible to conceive of DSP applications that can provide the infrastructure that enable multiple standard algorithms to operate on a single platform. The question arises as to what infrastructure TI has put in place to support this type of environment. There are two significant advances that have been made by TI in the past few years. This white paper discusses one of those two advances, the TMS320 DSP Algorithm Standard. In addition, in 2002, TI introduced Reference Frameworks for eXpressDSP. These frameworks act as starting points and illustrations for how to best construct an application on the TMS320 DSPs. They are also textbook examples for how algorithms should be integrated into the final application. These reference frameworks are discussed in detail in several application notes such as SPRA791 and SPRA793.

The premise for creating XDAIS was to directly address the issues that cause problems in system integration, and in particular the issues revolving around the use of existing algorithm IP that is supplied from another party. XDAIS is the result of a tremendous support effort from many TI third parties and several key customers. We wish to thank them for their dedicated efforts to make this standard a reality.

2 Terminology

Below is a sampling of definitions for some of the more common terms that we encounter when discussing the TMS320 DSP Algorithm Standard.

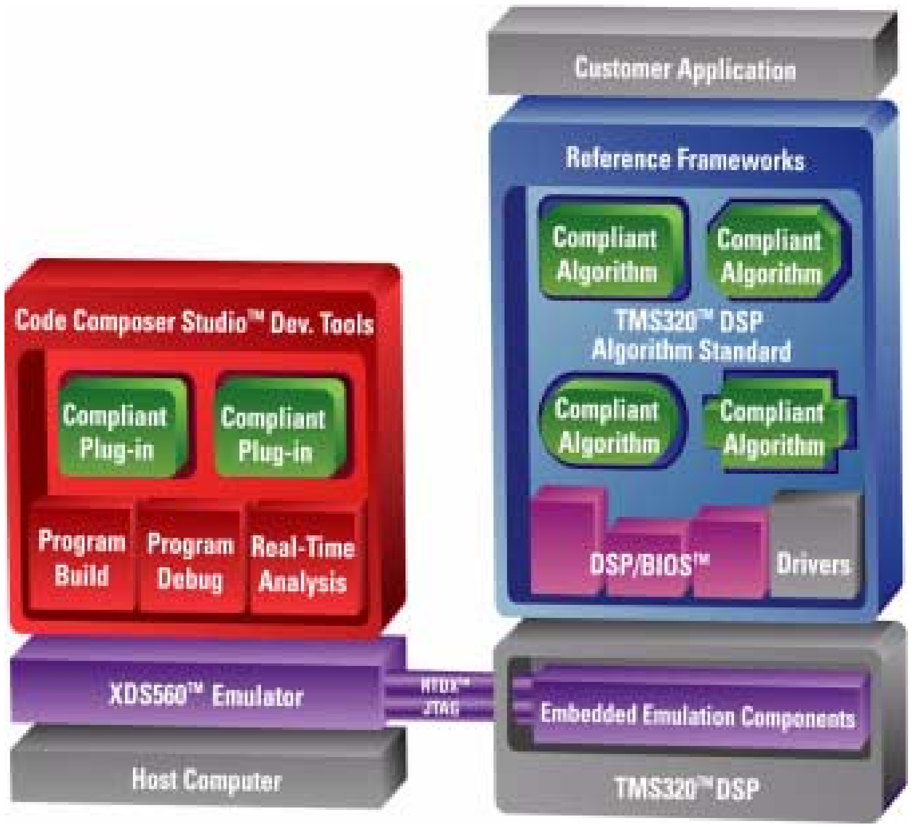
Algorithm: A module of code that consumes a data stream, processes it, and outputs a resultant stream. Examples include vocoders, modems, audio compression, video decompression, etc.

Reference Framework: The "glue" code that holds together the drivers, the algorithms, resource managers, and DSP kernel. Reference Frameworks start out as application-agnostic. Upon the addition of application-specific algorithms, the Framework takes on an application-specific nature.

DSP Kernel: A low-level software layer that provides hardware abstraction and manages low level physical resources. It provides threading, interrupt support, pipes, signals, and several other functions. In addition, DSP/BIOS offers data logging and statistical accumulation that enable real-time analysis of the system.

Application: The definition depends upon the use of some or all of the other components. If a customer writes all the code from scratch including a kernel, algorithms, and a framework, then the entire software system may be described as the application. However, in an environment where DSP/BIOS, a reference framework, and COTS algorithms have been deployed, the application programmer may see no further down into the system than the APIs to the controlling framework.

Figure 1. eXpressDSP elements on a T320 DSP



2.1 Description of the eXpressDSP Elements

The left-hand side of the diagram shows the Code Composer Studio Development Tools environment. It is not the intent of this white paper to discuss this area of technology. More detail on this can be found at www.ti.com/sc/ccstudio. The right-hand side of the diagram represents the target DSP in the eXpressDSP environment. It assumes the use of the DSP/BIOS real-time kernel (shown as purple components) with the possible inclusion of threads and tasks as required by the system designer. It also assumes that the system is built upon a Reference Framework (shown as the surrounding blue box). Algorithms that have been written to comply with XDAIS, referred to as eXpressDSP-compliant, are shown here as the green shapes “plugging” into the Reference Framework. The key here are the “sockets” that the algorithms plug into. It is the separation between the plugs and sockets that defines XDAIS.

XDAIS focuses on the interfaces between the algorithm and the rest of the system, rather than the ability of algorithm writers to exploit their individual talents to achieve their goals of fastest, smallest, and cheapest. Essentially, the core of the standard focuses on an abstraction of DSP resource management away from the algorithms themselves. Typically, resources on a DSP refer to memory usage and placement, along with I/O control such as the use of DMA channels, and possibly the use of key control registers. When only a single algorithm runs on the DSP, one can make broad assumptions about the use of the DSP resources. Even in the case of a multiple-channel instantiation of that single algorithm, provided basic re-entrancy exists, then the system should run just fine. However, when several algorithms are combined, problems may occur; for example, when one algorithm assumes the use of certain resources that are then “stolen” or “borrowed” by another algorithm in real time. It can be very difficult to pinpoint the source of a problem if the first algorithm was never designed to run in such a way and then doesn’t perform to its specification, or worse, performs sporadically. In addition, algorithms cannot make direct calls to the underlying DSP kernel for most of its services. If this were true, then that algorithm could not be used in any other environment where the same kernel was not present. However, the standard does allow limited use of the DSP/BIOS. For example, the real-time analysis modules may be called to provide for visualization of the algorithm in real time. None of the threading or tasking APIs may be called. With these basic premises, the algorithm standard enables a system integrator to more easily assemble production-quality systems from one or more algorithms.

3 Who Stands to Gain From Using The Algorithm Standard?

With the basic premise for the standard being reduction in system integration time, everyone in the development cycle stands to win. Let’s briefly touch on some of the highlights.

3.1 The Algorithm Writer

Until recently, many system integrators had their own algorithm integration methodologies and the algorithm writer had to adhere to that methodology. This required the algorithm vendor to support multiple interfaces. This issue is resolved by getting everyone to agree to one standard. A second major advantage to supporting the algorithm standard is that the system integrator can more quickly monitor the performance of an algorithm. If the algorithm vendor believes that he or she has a better solution than the one currently in use, the system integrator can quickly make an exchange to test out the new algorithm. This is very difficult if each algorithm follows a different interface standard.

3.2 System Integrators and OEMs

Many hard-to-find bugs simply go away because algorithm “black boxes” become a lot more predictable. The system integrator now has more choices. With several standardized algorithms available, it becomes much easier to compare algorithm A to algorithm B in order to gauge performance, robustness, and size. Individual algorithm interface methodologies adopted by system integrators can be eliminated. Also, when combined with Reference Frameworks, standardized algorithms enable rapid prototyping.

4 Scope of TMS320 DSP Algorithm Standard

The algorithm standard is comprised of thirty-nine rules that must be adhered to in order to be fully eXpressDSP-compliant. An additional fifteen guidelines are also recommended. Many of the rules are common-sense programming practices that have been in use for a long time and apply to all algorithms, regardless of the application area. Besides the general rules, there are instruction set architecture (ISA)-specific rules (e.g., use of specific control registers) for each of the major TMS320 families of DSPs. These general rules and guidelines, along with the ISA-specific rules and guidelines, are contained in the *TMS320 DSP Algorithm Standard Rules and Guidelines*, literature number, SPRU352.

A second document, *TMS320 DSP Algorithm Standard API Reference*, literature number, SPRU360, contains information on the implementation of the generic APIs that all algorithms must follow. The primary API, referred to as IALG, is basically responsible for taking the memory management function away from the algorithm and placing it in the hosting framework. Thus, a negotiation occurs between the algorithm and framework to assign memory for that algorithm. Additional functions within this API allow such functions as shared memory blocks between algorithms and for the framework to move memory around while an algorithm is operating in the system. For algorithms that require or choose to utilize on-chip DMA resources, the IDMA interface must also be implemented. A third optional, but highly recommended API in this document is called IRTC. This is designed to standardize the algorithm's test mode.

Finally, to assure the practical use of an algorithm, it must be supplied with a specific API, appropriate to the function being performed. The algorithm standard does not require the use of any particular set of APIs. However, to increase acceptance of the standard, TI will be distributing one or more algorithm-standard demonstration applications which use algorithms that are eXpressDSP-compliant. Specific API documentation is provided for each of the included algorithms. By using eXpressDSP-compliant algorithms, a customer can “switch out” one of the algorithms and “link–in” another version of the same algorithm without having to recompile. This can only be done when object code compatibility is ensured, and this can only be achieved if the second algorithm also uses the same specific API.

Note: An algorithm vendor may want to provide a version of the algorithm that follows the specified example APIs, so that it can be evaluated in one of these algorithm standard demonstrations.

TI provides a unique testing environment that checks the conformance of algorithms to the rules of the standard. Algorithms that pass this automated test are awarded the right to be called “eXpressDSP-compliant” and to display the eXpressDSP Compliant logo.

Figure 2. eXpressDSP-Compliant Logo

4.1 New versus Retrofit?

There are clearly hundreds and perhaps thousands of algorithm implementations that exist for use on TMS320 DSPs. The question often arises as to whether it makes sense to go back and retrofit an algorithm to be eXpressDSP-compliant. Clearly, for new algorithms it is prudent to follow the rules and guidelines right away. The answer to the retrofit question should be handled on a case by case basis. Issues to address include whether or not the algorithm will be used in different applications in the future. Will the algorithm be sold and integrated into a larger application? Will a particular end customer require that all algorithms be eXpressDSP-compliant? If the answer to these or similar questions is yes, then it probably is necessary to convert the algorithms. To assist with this effort, TI provides a selection of tools that will make this task easier. One additional scenario is that old and new algorithms must be integrated. If some algorithms are eXpressDSP-compliant, then it is probably most expeditious to convert the old algorithms to be eXpressDSP-compliant to help simplify the overall integration task.

5 Summary

The TMS320 DSP Algorithm Standard is a bold program being undertaken by Texas Instruments as part of the overall its eXpressDSP initiative. TI has clearly recognized that as system software complexity grows at an exponential pace, this kind of standard initiative is vital if we are to maintain the time-to-market goals of our customers. While we recognize that many system integrators have internal methodologies already, it is advantageous for everyone in the marketplace to share and promote a single standard.