Demystifying digital signal processing (DSP) programming: The ease in realizing implementations with TI DSPs

Todd Hahn
Software Development Manager

Jonathan Humphreys
Software Senior Member Technical Staff

Andy Fritsch
Software and Tools Director

Debbie Greenstreet
Strategic Marketing

Texas Instruments

Overview

Introduced by Texas Instruments over thirty years ago, the digital signal processor (DSP) has evolved in its implementation from a standalone processor to a multicore processing element and has continued to extend in its range of applications. The breadth of software development tools for the DSP has also expanded to accommodate diverse sets of programmers. From small, low power, yet "smart" devices with applications such as voice and image recognition, to multicore, high-performance compute platforms performing real-time data analytics, the opportunities to achieve the low-power processing efficiencies of DSPs are nearly endless. The TI DSP has benefited from a relatively unique tool suite evolution making it easy and effective for the general programmer and the signal processing expert alike to quickly develop their application code. This paper addresses how TI DSP users are able to achieve the high performance afforded by the TI DSP architecture, in an efficient, easy-to-use development environment.

Introduction: The value of DSP

Initially developed to process audio, the early TI DSP was quickly leveraged by engineers for a wide variety of numerous applications. The use of a TI DSP, whether standalone or as part of a System-on-Chip (SoC) affords full software programmability and all of the benefits of software-based products. While essentially every algorithm or function that can be processed on a DSP can be executed on a general-purpose processor, the DSP, by design, performs math more efficiently. While digital signal processing functionality can certainly be implemented in FPGAs and ASICs, these devices are best utilized on applications that process data flow. Conversely, applications requiring algorithms that spend a majority of the time processing loops scale much better in terms of size, power and performance when implemented on DSPs compared to hardware-based implementations. To put it simply,

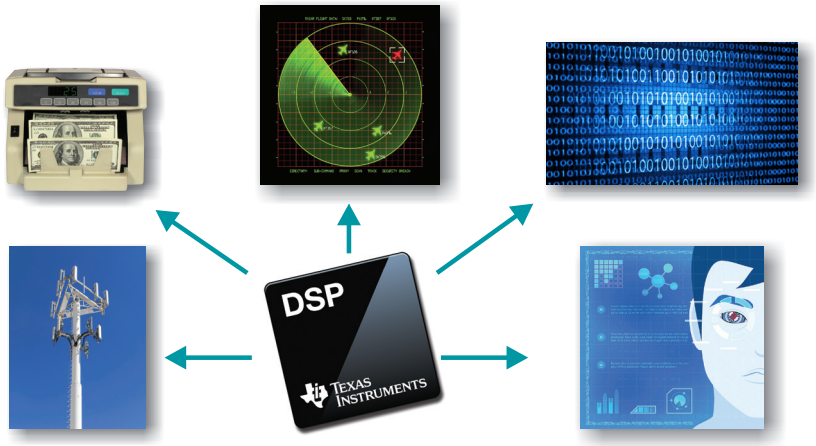


Figure 1 depicts an array of applications/end equipment that benefit from the efficiencies of a DSP

TI's DSPs offer a variety of efficiencies over other software-programmable processors, particularly for applications that include computation-intensive functions, such as analytics, FFTs and matrix math in a constrained environment. Be it machine vision, biometric analysis, video surveillance, audio processing, or data analytics, anywhere you find an intelligent automated system you are likely to find a DSP at the heart of it.

Designed for performance entitlement

Designed for high-performance processing of digital signals, including real-time mathematical computations of parallel data sets, the DSP CPU architecture is optimized to achieve the end application goals. TI's TMS320C6000™ platform of DSPs utilizes the very long instruction word (VLIW) architecture to achieve this performance, and affords lower space and power footprints to implement compared to superscalar architectures. As experienced software engineers know, the ability to obtain the theoretical maximum performance of a given CPU in an actual implementation is not a given. The ability to reach full performance entitlement with a given processor is a key consideration in selecting new CPUs for use in an application. Processor performance entitlement is afforded in TI's DSPs and TI's silicon/software design strategy is a key part of that this equation.

The process

TI was one of the first processor semiconductor manufacturers to have the DSP silicon designed in tandem with the DSP compiler. Enabling a cycle of iterative CPU development, TI CPU hardware/silicon architects, compiler designers and application system experts work hand-in-hand from design inception to product manufacturing. The systems team, along with other TI business experts, select applications and algorithms that represent a variety of potential end applications of the processor. Using TI's compiler technology, these applications and algorithms are then compiled and the results analyzed by the team to determine where to make modifications and improvements to the ISA and memory system. This prototyping cycle is depicted in Figure 2 and is repeated until the architecture is optimized for performance and efficiency, and the compiler can achieve that performance via C and C++. Combined with a rapidly re-targetable compiler and advanced compiler optimizations, this collaborative strategy also enables the compiler to effectively exploit the available performance of the DSP from C and C++. Employing this strategy has laid the ground work for TI to successfully develop generations of products over the lifetime of the DSP architecture, and as such, TI's customers often cite the compiler as being a key strength of their development chain.

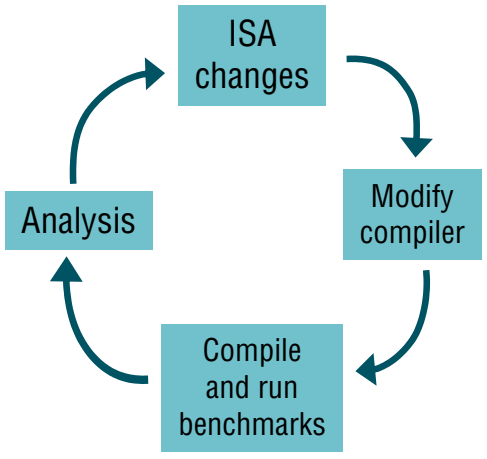


Figure 2. TI DSP silicon, software, tools co-design process results in an architecture that easily enables high-performance programs

Software pipelining

Instruction-level parallelism is critical in achieving real-time performance in TI's VLIW DSP architecture, and as such, software pipelining is a feature used to hone the CPU architecture and ISA for entitlement. Applications executed on DSPs commonly spend a lot of time executing loops, and as such, loop performance is critical to overall DSP processing performance. The TI DSP compiler is able to create instruction-level parallelism by overlapping iterations of a loop, thereby software pipelining them, as shown in Figure 3, which optimizes the use of CPU functional units and thus improves performance. The example in Figure 3 shows that, without software pipelining, loops are scheduled so that loop iteration i completes before iteration i+1 begins. Thus with software pipelining, as long as correctness can be preserved, iteration i+1 can start before iteration i finishes. This generally permits a much higher utilization of the machine's resources than might be achieved from non-software-pipelined scheduling techniques. In a software-pipelined loop, even though a single-loop iteration might take s cycles to complete, a new iteration is initiated every ii cycles.

The TI DSPs have multiple functional units and include a range of single instruction multiple data (SIMD) instructions. These features enable increased throughput per cycle and the TI compiler is designed to take full advantage of these features. In order to keep all eight functional units on the C6000™ DSP busy, the compiler often employs the technique of loop unrolling. Loop unrolling duplicates the body of a loop so multiple iterations are performed before branching back to the top of the loop. When legal and profitable, the compiler can perform loop unrolling and execute multiple iterations at the same time, increasing the utilization of the eight functional units and thereby increasing performance. The compiler also employs loop unrolling to automatically exploit the SIMD instructions on the C6000 devices. The compiler will unroll a loop to create the same-instruction, multiple-data situation that allows the usage of SIMD instructions, thereby exploiting throughput available in the SIMD instructions and increasing performance. While not always possible, these techniques highlight how the TI DSP compiler works to achieve optimal performance, in some cases achieving a 16× algorithm speedup over a naïve compiler translation of a natural C code routine.

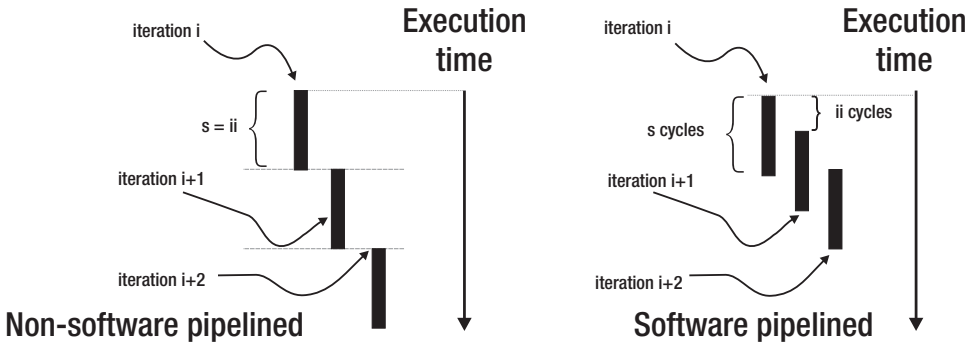


Figure 3. Leveraging software pipelined loop for code execution efficiency

Application example

As discussed earlier, the breadth of DSP applications have expanded over time. Already a key element of a wireless base station architecture, software architects looked to determine how they could leverage more of the DSP's low power consumption and real-time performance to take on more of the base station processing as wireless standards evolve to even more low latency requirements. Traditionally utilizing the DSP for Layer 1, physical layer processing, base station software architects began implementing some of the Layer 2 functionality for LTE solutions on the DSP in order to achieve the latency requirements. Layer 2 processing includes a significant amount of control code in the form of irregular loop-type algorithms. Irregular loops can be difficult to software pipeline because they contain complex, compound conditions both within the loop as well as at the exit condition, have unknown loop iteration counts, and contain complex memory accesses that make alias analysis difficult. As part of ongoing DSP performance enhancements, the compiler team, keeping close to customer activities, modified the compiler's ability to achieve high irregular loop performance.

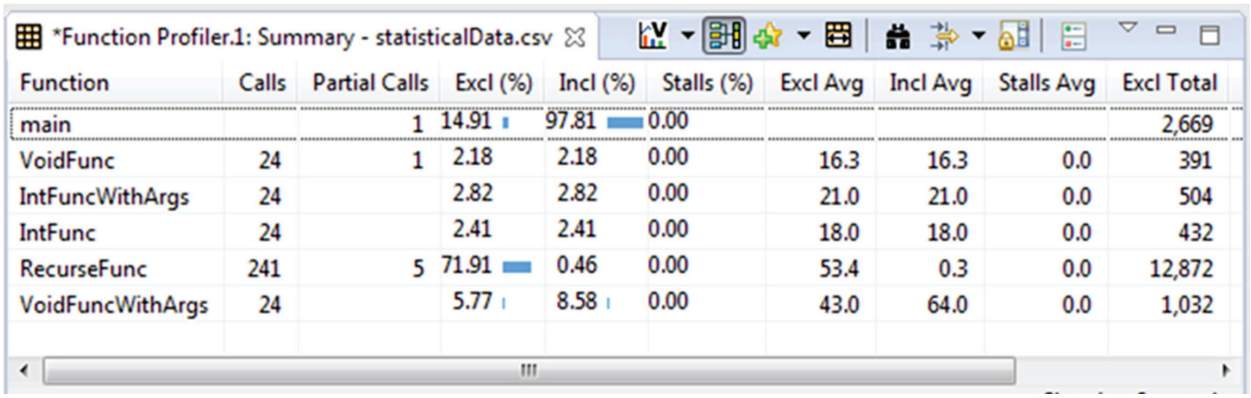
Achieving DSP performance with ease

As many software programmers will attest, there is a common software development paradox: achieving solution performance versus the effort, resources and time it takes to get there. This performance versus schedule tradeoff has become more amplified in today's software application environment, where the composition of the electronic product design team is increasingly in the software majority. Product schedule and resources costs regularly weigh in on product decisions. Hence, the ease of use of implementing and achieving desired performance of a selected processor is critical.

As mentioned previously, TI DSPs are co-designed by the team of CPU architects, compiler designers and system engineers, and their goal is not only to achieve DSP performance entitlement but to enable it in a realistic software environment with tools and languages familiar to the software developers. While historically the digital signal processor has had its share of assembly-level programmers, the TI DSP and its compiler are designed for use by the common language of today's software developers; C/C++. It supports standardized programming languages and extensions such as C99, C++, common GCC extensions, OpenMP and OpenCL. The TI DSP compiler and Code Composer Studio™ (CCStudio) IDE environment have a lot of inherent features that enable the developer to achieve efficient performance from the DSP code, and the developer is afforded state-of-the-art development tools, programming languages and extensions. The second half of the developer's challenge is how to achieve this application efficiency in a reasonable amount of time. We now explore the various performance tools and optimization features available to the DSP programmer from the feature-rich C6000 compiler, in conjunction with TI's CCStudio integrated development environment (IDE).

Function profiling

TI's CCStudio IDE supports a feature called function profiling that provides information on the number of times functions are called, as well as the inclusive and exclusive total cycle count each function took to execute. This feature can be invoked within CCStudio IDE using a hardware trace analysis tool, and can be configured to profile all



Function	Calls	Partial Calls	Excl (%)	Incl (%)	Stalls (%)	Excl Avg	Incl Avg	Stalls Avg	Excl Total
main		1	14.91	97.81	0.00				2,669
VoidFunc	24	1	2.18	2.18	0.00	16.3	16.3	0.0	391
IntFuncWithArgs	24		2.82	2.82	0.00	21.0	21.0	0.0	504
IntFunc	24		2.41	2.41	0.00	18.0	18.0	0.0	432
RecurseFunc	241	5	71.91	0.46	0.00	53.4	0.3	0.0	12,872
VoidFuncWithArgs	24		5.77	8.58	0.00	43.0	64.0	0.0	1,032

Figure 4. CCStudio IDE Function Profiler Summary details that the program spends the majority of its time in the function RecurseFunc, aiding the programmer to focus optimization efforts making RecurseFunc faster.

functions or those within a certain address range. Function profiling can be used early on in the code analysis process to help determine function areas to focus on to enhance performance. Figure 4 shows an example summary view of a CCStudio IDE function profile run.

Pragmas and restrict

The capability of the compiler to obtain critical information from pragmas and use of the restrict keyword further enhances the amount of tuning features in the TI DSP programmers' toolkit. For example, as discussed earlier, performing software pipelining is critical for optimal code performance. An example of the commonly used "MUST_ITERATE" pragma, which communicates to the compiler the lower bound of loop trip count is shown in Figure 5 below. An additional performance enhancer, the "restrict" keyword is heavily utilized by DSP programmers to communicate memory access independence, as it can dramatically improve the compiler's ability to software pipeline a loop.

Performance advice

It is important to note that software programmers, who are unfamiliar with DSP CPU architecture, should not be afraid to develop solutions using the TI DSP. Not only are all common language extensions supported as mentioned, but the TI compiler offers programming assistance in the form of performance advice. At compilation time, performance remarks can be enabled to provide feedback and ideas for further code insight and optimization. For example, if the programmer had enabled performance remarks he or she might get a warning message that the compiler couldn't fully

```
#pragma MUST_ITERATE(1000)          //  outer loop:  trip count  >=  1000
 for (i = 0; i < large_value; i++)
 {
        #pragma MUST_ITERATE (1,4)  //  inner loop;:  1   <=  trip count <=  4
        for  (j=0; j<small_value;  j++)
        {
          <stuff  for  iter  1,j>
        }
}
```

Figure 5. An example of the MUST_ITERATE pragma in use

```
- Command-line options: Should use -o3, should not use -g
- Function calls, switch stmt, etc. in loops disable pipelining - don't
  use them
- Suggestions on the use of restrict keyword
- Suggestions on the use of MUST_ITERATE, _nassert()
```

Figure 6: Performance advice guides programmers on ways to improve DSP code

optimize a loop because it couldn't determine if two pointers point to the same object in memory. However, if the programmer uses the following annotation, this loop can be further optimized by the compiler. Figure 6 depicts some example suggestions emitted from the performance advice feature.

Native vector types

Another performance feature of the C66x compiler is the support of native vector types. Examples of native vector types are int4 or float2, which are built-in types for a vector of four ints and vector of two floats, respectively. Being built-in types, they can be used with C operators like plus and multiply naturally. Native vector types allow the C/C++ programmer to more naturally express the ILP present in the algorithm and the SIMD opportunities that are available. This mitigates the need for vendor-specific C intrinsics or assembly language. An example of native vector types used in DSP code is shown in Figure 7.

```
void VECSUM_once(void *in1, void *in2, void out)
{
   unit64_t *restrict data1;
   unit64_t *restrict data2;
   unit64_t *restrict out1;
   int  i;

   data1 = (unit64_t * )in1;
   data2 = (unit64_t * )in2;
   out1 = (uint64_t * )out;
#pragma MUST_ITERATE(1)
   for (i = 0; i < SIZE_VECSUM_IN; i+=4)
   {
      double data1A, data2A;
      double data1B, data2B;
      data1A = _amemd8(data1++);
      data1B = _amemd8(data1++);
      data2A = _amemd8(data2++);
      data2B = _amemd8(data2++);
      _amemd8(out1++)= _daddsp(data1A, data2A);
      _amemd8(out1++)= _daddsp(data1B, data2B);
   }
}
```

Initial code

```
void VECSUM_newvec(float2 * restrict data1,
                   float2 * restrict data2,
                   float2 * restrict out1)
{
   int   i;

#pragma MUST_ITERATE(1)
   for (i = 0; i < SIZE_VECSUM_IN; i+=4)
   {
        *out1++ = (*data1++) + (*data2++);
        *out1++ = (*data1++) + (*data2++);
   }
}
```

Adding two arrays of floats and taking advantage of C66x's DADDSP instruction

Figure 7. An example of compiler support of native vector types

Performance vs. memory footprint

Not all software optimization is defined by runtime speed or latency. Some applications may have physical size, power and/or memory restrictions, and may require trading off runtime performance for overall code size. The TI DSP compiler has an option (-m[0-3]) to indicate a small code size preference and the compiler will subsequently optimize code size over performance.

In addition to reduced memory footprints, lower code size can improve performance by reducing cache conflict and capacity misses. By compiling non-performance critical code for reduced size, cache memory can be better utilized, as code is more likely to fit in the cache and less likely to conflict with existing cached code.

Intrinsics

As the C6000 compiler continues to evolve, performance optimization techniques are continually enhanced. In most cases, the techniques described so far are sufficient for achieving performance entitlement; however the C6000 compiler not only provides an easy programming environment for developing high-performance code, but also includes tools for advanced users to specialize their code for maximum performance. One such feature is the C6000 intrinsics, which allows the user access to specific C6000 DSP instructions through C in a natural way. By using built-in C functions (intrinsics), the user has access to specialized DSP capability that is not easily representable in the native C language.

DSP tool suite robustness

Last but not least, no discussion about the value of a processor compiler would be complete without the assessment of the compiler correctness itself. That is, does the generated code correctly execute as the developer specified. When one thinks about the complexity of the compilation task, the importance of robustness should be obvious. The TI DSP compiler offers execution robustness in a multitude of ways.

The TI DSP compiler is developed and produced as formally and rigorously as all TI products, and this begins with the development process. Industry best practices are leveraged, along with functional, design and source code reviews, as well as being managed through a formal software configuration system. A root cause analysis process is employed for defects.

Every version of the TI DSP compiler is thoroughly validated with in-house kernels, applications, regressions, feature tests and unit tests, along with all commercially available test suites before release. The commercial suites include:

- PlumHall C and C++
- Perennial CVSA and C++VS
- CodeSourcery C++ ARM® ABI test suite
- Dinkumware proofer for C++
- Nullstone optimizer test
- ACE supertest
- GNU torture suite

The DSP compiler process includes an automated nightly validation and an automated release validation process, both of which are run on a powerful collection of servers, leveraging thousands of processors to execute an extensive array of tests.

Since the compiler team works closely with TI's end equipment and systems teams, a large amount of applications-type code is included in the regression flow. This extends an additional level of robustness to the compiler and an additional level of assuredness to the programmer. TI's reusable

and retargetable compiler infrastructure, developed over nearly 30 years of DSP product development, has been applied to and validated with tens of diverse TI DSP architectures. Due to the wide variety of applications and architectures the compiler must support and has supported, the strategy of reuse and retargetability results in a more robust compiler, with high code reuse and code coverage. It represents a significant amount of development and support behind it due to nearly 30 years of DSP product deployment.

Join the DSP bandwagon

While no longer simply a standalone device, the DSP continues to grow and expand in function and application. Integrated as part of a heterogeneous SoC, the DSP is rapidly finding a home as a functional accelerator, along with more general-purpose code running on an ARM, in growing applications such as video surveillance, high-performance computing and anything requiring analytics algorithms. This is because the power, cost and size efficiencies afforded by the DSP architecture, along with its full software programmability, make the DSP an ideal processor choice for applications with intensive math computations in constrained environments. The TI DSP development tools support standardized programming languages and extensions such as C99, C++, common GCC extensions, OpenMP and OpenCL, enabling a variety of software programmer skill sets to leverage the performance efficiency of a DSP, while enjoying the ease of use in getting such a product to market quickly and effectively. The TI DSP compiler strategy is committed to providing a complete feature set and high-performance level, as part of the overall TI DSP product line. Hundreds of customers have successfully delivered differentiated products to the market based on TI's highly efficient DSPs. Chances are, your application can benefit from DSPs.

Important Notice: The products and services of Texas Instruments Incorporated and its subsidiaries described herein are sold subject to TI's standard terms and conditions of sale. Customers are advised to obtain the most current and complete information about TI products and services before placing orders. TI assumes no liability for applications assistance, customer's applications or product designs, software performance, or infringement of patents. The publication of information regarding any other company's products or services does not constitute TI's approval, warranty or endorsement thereof.

C6000, Code Composer Studio and TMS320C6000 are trademarks of Texas Instruments. All trademarks are the property of their respective owners.

© 2015 Texas Instruments Incorporated

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not 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 relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2015, Texas Instruments Incorporated