Using Code Composer Studio with OMAP1610

Ki–Soo Lee *Code Composer Studio Applications Engineering*

ABSTRACT

Extending on the widely-adopted OMAP™1510 processor, the more powerful OMAP1610 processor is a single-chip application processor that supports all cellular standards, and complements any modem or chipset and any air interface. To ensure the seamless scalability throughout the OMAP processor family, the OMAP161x devices feature the same dual-core architecture of the OMAP1510 processor, allowing device manufacturers to reuse software and other development efforts across multiple product lines.

Despite the similarities between the OMAP1610 and earlier OMAP devices, there are differences between the devices that users should be aware of when developing on the OMAP1610 with Code Composer Studio™ for OMAP. This application note addresses some of the issues that the developer may encounter when using Code Composer Studio with (but not always limited to) an OMAP1610 device. In addition, basic information on how to configure Code Composer Studio for development on an OMAP1610 device will be discussed.

Contents

List of Figures

1 Configuring Code Composer Studio for an OMAP1610 Target

Code Composer Studio Setup defines the target board or simulator you will use for your project. This definition is called the system configuration, and it consists of a device driver that handles communication with the target plus other information and files that describe the characteristics of your target. A system configuration is needed even before building an application because the configuration determines which tools will be used by Code Composer Studio.

Code Composer Studio Setup includes a set of standard OMAP1610 configuration files that are predefined for the most common system configurations. If one of the standard files matches your OMAP1610 system's configuration, simply load that file using the Import Configuration dialog and you are ready to start a Code Composer Studio session.

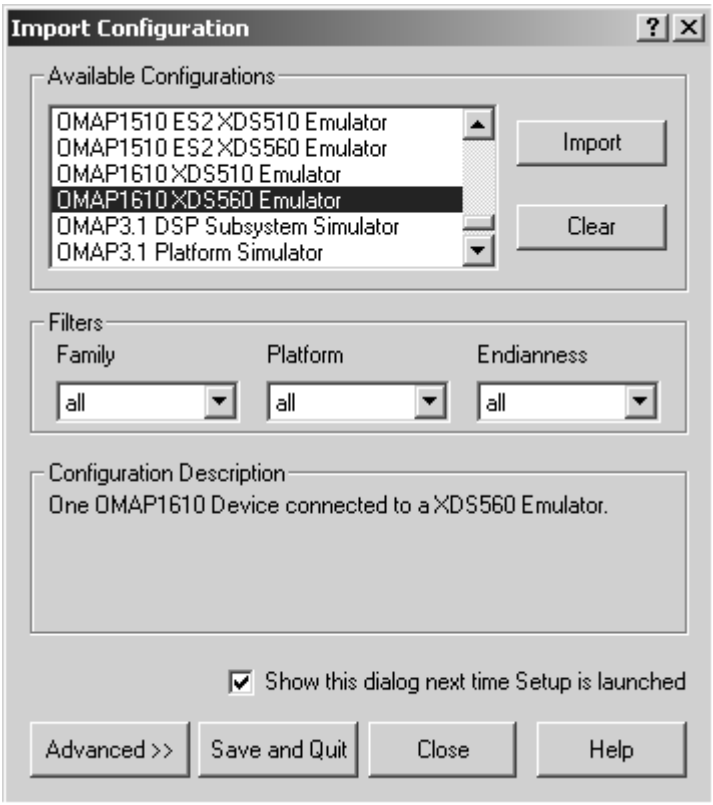

Figure 1. Import Configuration Dialog

Currently two standard configurations exist for the OMAP1610 target for Code Composer Studio Setup for OMAP. These configurations vary depending on the type of emulator you wish to use. In Figure 1 above, we are selecting an OMAP1610 target with an XDS560 emulator.

Once the correct configuration has been selected, the Code Composer Studio Setup interface will show the selected configuration in the System Configuration pane.

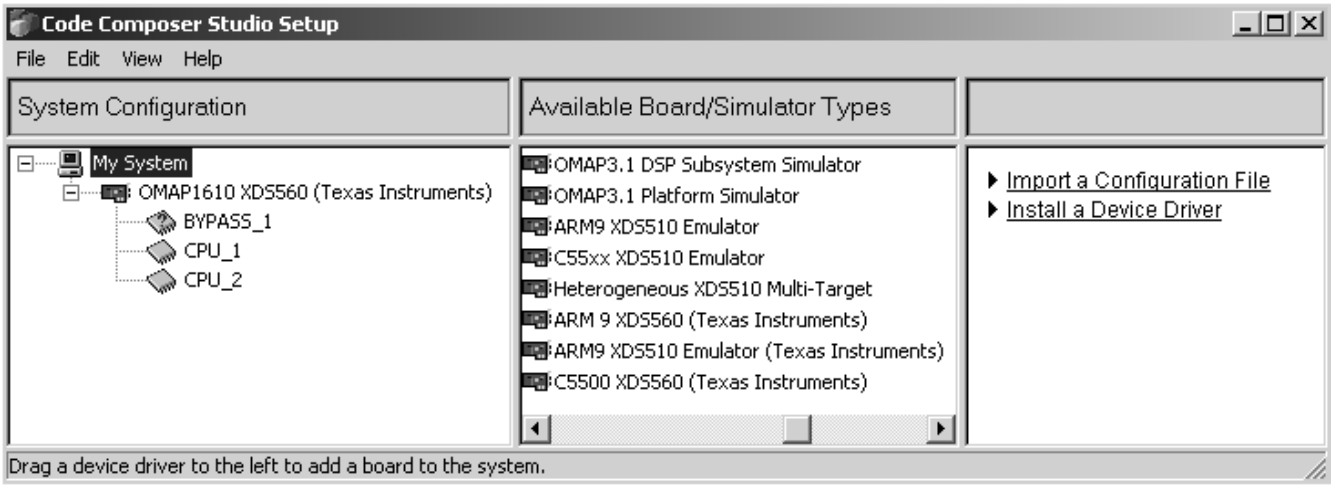

Figure 2. System Configuration for the OMAP1610 XDS560 Target

These standard OMAP1610 configurations are set up with the proper initialization order of the processors and configured with the correct GEL startup files to work with default OMAP1610 targets.

For custom targets, Code Composer Studio may not contain the proper predefined system configuration file for the custom OMAP1610 platform being used. In cases like this, the configuration file can be created from an existing OMAP1610 configuration file or created from scratch. It can then be exported (saved) and then imported each time Code Composer Studio Setup is invoked to create an OMAP1610 system configuration. It is very important to get the proper order of the processors on the scan chain and the initialization order of the processors correct. For more information on how to do this, please refer to application note, *Configuring Code Composer Studio for OMAP Debugging*, SPRA807.

NOTE: When using the OMAP1610 EVM, it is recommended to set the SLOWCLK setting in the ccbrd0.dat file if using an XDS510 emulator or set the TCLK to “legacy” if using the XDS560 emulator. These are already set up as the default settings when importing the existing OMAP1610 configuration files provided. However, if you configure Code Composer Studio to run on the OMAP1610 manually without using the import configuration option, you will need to configure these settings manually. Otherwise, you may encounter some issues running Code Composer Studio on the EVM. This is an OMAP1610 EVM board issue, not an OMAP1610 silicon issue.

2 Initialization and Connectivity Issues with the OMAP1610 Target

The following sections describe the initialization and emulation issues that a developer should be aware of when attempting to work with the OMAP1610 target.

2.1 Initialization Issues

This error may be given by Code Composer Studio when trying to initialize *any* OMAP target:

Can't Initialize target CPU:

Error 0x80000200/–2072

Fatal Error during:OCS

Cannot halt the processor

This issue results from the behavior of the ARM processor at initialization. The ARM hardware goes to the reset vector and begins execution at startup, and in many cases this address is in un-initialized RAM. In most cases this is not a problem, as long as you do not care what your system does at startup before downloading any code. However, with Code Composer Studio, this is not the case. When emulation attempts to get the RDY acknowledgement from the processor through emulation, the ARM attempts to finish execution of the instruction in the execute stage of the pipeline to be sure you are in the proper system state. But in this un-initialized RAM, we are often accessing memory which does not exist on the device or performing some other operation which will never "complete" to return the acknowledgement. Asserting reset through a power cycle clears this circumstance and returns the target to a good state to allow Code Composer Studio to launch.

Another initialization issue exists on the OMAP1610 when using the default GEL startup files. Code Composer Studio can not be launched twice without a power cycle in between because the GEL startup file initializes either the DDR or SDR and this can only be done once. You have the option to manually edit the GEL startup file to remove this initialization after the initial start of Code Composer Studio. You can then open and close Code Composer Studio without problems, but that initialization must still be done after each power cycle of the board. Do this by removing the SDR_enable() and DDR_enable() calls in the GEL Startup() function found in the GEL startup file for the ARM and creating GEL hotmenu items for SDR_enable() and DDR_enable() that can be called manually within the IDE when needed.

2.2 ARM926 RTCK Emulation Issues

The ARM926 does not support an independent test clock, and as a result, TCK must be synchronized with the ARM clock. This synchronized clock, RTCK, gets turned off when the ARM is reset or the ARM clock is turned off. Watchdog reset, or any warm reset, and idle are among the ways that this can happen. Also, the 0xFFs in the synchronizer are reset to 0 when the ARM is reset, which may cause a glitch to the RTCK signal since this reset is not synchronized with the ARM clock (causing emulation problems). Other than on reset, RTCK should be glitchless.

Sometimes there are problems with establishing a connection to the OMAP1610 target because the ARM926 does not halt due to execution of trash code on startup. If the ARM926 gets into a bad state, the reset trap vector will be reached. However, for this to take effect you have to reset the 1610, which causes the RTCK to go away (and with potential glitches). This situation then requires an emulator reset which puts the JTAG tap into TRST state which appears to clear the reset trap vector. The end result is that if ARM926 gets into a bad startup state it may be difficult to establish connection. Therefore, we recommend that you insure that the ARM does not execute trash code by having the system boot from flash which contains a small spin loop. When Code Composer Studio gains control, you can force the spin loop to exit.

2.3 Reset Issues

2.3.1 DSP Reset Issues Upon Power-up

The OMAP device is configured with the ARM core holding the DSP core in reset at power-up. The DSP will not be initialized while being held in reset. To open a debug session for the DSP, the DSP must be taken out of reset by the ARM and initialized. When Code Composer Studio is started, the GEL Startup() function in the default startup GEL file for the ARM releases the DSP from reset by setting bit 1 of ARM_RSTCT1 (at address 0xFFFECE10). This allows for a heterogeneous emulation session to be started on the device. This is all done by the time the Parallel Debug Manager or PDM appears (see section 2.4 below). When using a default OMAP configuration with the default GEL startup files, no extra action by the user is required to open a debug session for the DSP. If the user is not using the default OMAP configuration or the default startup GEL file, then they will need to be sure to release the DSP from reset in their startup GEL file for the ARM. Otherwise, initialization of the DSP will fail and an error will appear, complaining that the DSP is being held in reset.

NOTE: There is a known issue when the option to “Always Connect at Startup” is disabled for the Code Composer Studio session for the DSP. When the debug session starts up with the DSP disconnected and a debug session for the DSP is then opened with a GEL startup file that attempts to initialize the memory map, a series of “invalid page” errors will appear. This is due to an issue where upon startup of Code Composer Studio, the heterogeneous driver makes some assumptions on what the target may be if the device is not connected to Code Composer Studio on startup. In this case the assumption is incorrect in that it assumes the DSP has a unified memory map (explaining all the “invalid page” errors when running the GEL startup file). Thus it is recommended that users ALWAYS have the device connected at startup and have the ARM release the DSP from reset in its GEL Startup() function in the GEL startup file.

2.3.2 Device Reset During Debugging

It is important to be aware of the various ways the device can be reset since driving the chip through reset will cause Code Composer Studio to lose communication with the device during a debug session. The common ways the chip can be reset are:

- Power Cycle Reset
- System Reset
- Watchdog Reset (may be disabled – see Section 3.4)
- Global S/W Reset

A reset to just the DSP will cause the Code Composer Studio session for the DSP to lose connection but Code Composer Studio may still maintain the connection with the ARM (see section 2.4 below). However a reset to the ARM will most likely reset the DSP causing total loss of communication to the device by Code Composer Studio.

2.4 Parallel Debug Manager (PDM)

The Parallel Debug Manager (PDM) allows you to open a separate Code Composer Studio session for each target device (ARM and DSP for OMAP1610). Activity on the specified devices can be controlled in parallel using the PDM control. The PDM will be the first thing to appear when starting up Code Composer Studio for OMAP with a heterogeneous target configuration (OMAP1610). Please refer to the Code Composer Studio on–line help topic: *Parallel Debug Manager (PDM)* for more general information on how to use the PDM.

Figure 3. Parallel Debug Manager (PDM)

If a session is opened for each of the processors and if communication is somehow lost with one of the processors (say the DSP is reset), the Code Composer Studio session for that processor is no longer valid and can be closed. But the Code Composer Studio session for the other processor (ARM) should still have communication with the ARM and the user can continue their debug session with the ARM – although there may be problems running the ARM application after a DSP reset only because the DSP may change the ownership of shared peripherals and/or memory. However the user will not be able to establish connection when attempting to open another Code Composer Studio session for the DSP (in this case) from the PDM. They will need to exit Code Composer Studio completely by closing the PDM, most likely need to reset the device, and then restart Code Composer Studio.

3 Debugging the OMAP1610 with Code Composer Studio

There are several issues that the developer should be aware of during their debug session with the OMAP1610 target.

3.1 Accessing Secure Memory Resources

The OMAP1610 includes special-purpose security hardware that is used to activate a secure mode. The secure mode can be viewed as a third privilege level on the ARM. It is used to create an environment for protecting sensitive information from access by un-trusted software. The secure mode is set with the assertion of a dedicated signal (secure bit) that propagates across OMAP1610 and creates a boundary between resources that trusted software might access and those available to any software. When the secure bit is set, only then is access to the secure resources is unlocked and only for the MPU (and accessible through Code Composer Studio for ARM).

There are a set of features intended to support operations that require secure operations. These resources will only be accessible/functional when the device is in secure mode:

- Part of the ROM (48 kbytes of Secure ROM)
- 16 kbytes of SRAM (Secure RAM)
- 128–bit e–fuses & 256–bit eFuses
- The security watchdog timer
- The RNG (Random Number Generator)
- The SHA1/MD5
- DES/3DES
- The security control register

3.1.1 Accessing Secure ROM

64 kbytes of Boot ROM are available on the OMAP1610. The first 16 kbytes (precisely 16 kbytes + 128 bytes) of the Boot ROM can always be accessed whereas, for the remaining 48 kbytes (precisely 48 kbytes –128 bytes), access is restricted to secure mode (128 bytes of the secure ROM is a “secret” ROM which also requires an additional “secret” signal (*pi_rkey*) to be high to be accessed).

To access the standard (non “secret”) region of secure ROM, the secure bit must be set (*pi_secure* = HIGH), output enable must be active low (*pi_oe_n* = LOW), and access to Boot ROM must be given (*pi_bootcs_n* = LOW). As mentioned above, to access the “secret” 128 bytes in secure ROM, the “secret” signal must also be asserted (*pi_rkey* = HIGH) in addition to the requirements to access non–“secret” secure ROM.

When access to secure ROM is denied, the ROM returns dummy data represented as all zeros.

3.1.2 Accessing Secure RAM

16 kbytes of secure SRAM are part of the secure resources for the OMAP1610. Access to this secure RAM must be carefully controlled. It is accessible through the OMAP3.2 EMIFS (External Memory Interface Slow) only when the secure bit is set (*pi_secure* = HIGH) and supports byte addressing (*pi_cs_n* = LOW).

The last 8 bytes of secure RAM are associated with two 32 bit registers which support byte access. The least significant bit of the first register (MEM1) contains the status of BIST while the rest of the bits are used as a memory with reset along with all of the second register (MEM2). These registers are accessed when *pi_add[13:2]* (Address Bus from EMIFS) references the last 8 bytes of secure RAM (and the above conditions are met).

When access to secure RAM is denied, the RAM returns dummy data represented as all zeros.

3.1.3 The Security Control Register (SECCTRL)

The security control register (SECCTRL) controls various security settings, such as the secure watchdog, access to secure hardware accelerators, and debug/emulation capabilities. The register is accessible only in secure mode, otherwise access is denied and returns dummy data represented as all zeros.

The security control register is reset at power–on reset and its reset value depends on the SECURITY_DEVICE_TYPE[1:0] signals the configuration eFuse generates.

Several bits in the security control register are one–time programmable. Therefore, after the first write access, these bits are locked in a read–only mode and cannot be written again until the next power–up reset occurs.

3.2 Accessing Shared Peripherals/Memory

The OMAP1610 includes several peripherals and memory regions that can be considered shared by the MPU (ARM) and DSP. A public TIPB bus provides access to these shared peripherals and memory regions.

3.2.1 Accessing DSP Public Peripherals and Memory via MPU Interface

The MPU interface (MPUI) is a 16-bit parallel port that allows the MPU and the system DMA controller to communicate with the DSP and its peripherals, facilitating software downloads and data transfers. The MPUI provides the MPU with access to the full memory space of the DSP. In addition, the MPUI allows the MPU to access devices on the DSP public peripheral bus through duplicate memory-mapped peripheral registers in the MPU address space and also the control registers of the TIPB bridge module and the CLKM2 configuration registers.

The MPIU port supports four access modes:

- Single–access mode, memory (SAM_M): SARAM, DARAM and external memory interface are shared between the DSP domain and the MPU domain
- Single–access mode, peripheral (SAM_P): DSP public peripheral bus is shared between the DSP domain and the MPU domain.
- Host–only mode, memory (HOM_M): MPU has exclusive access to DSP SARAM, but it cannot access other DSP memory resources.
- Host–only mode, peripheral (HOM_P): MPU has exclusive access to the DSP public peripheral bus.

SAM is the normal operating mode in which all the DSP internal memory and the public peripherals are accessible by the MPUI interface as well as the DSP. If both the DSP and the MPU controllers access the same memory at the same time, priority is given to the DSP controllers.

HOM provides the MPU with exclusive access to the DSP SARAM or public peripherals, primarily to support high-speed transfers from/to DSP during DSP reset of IDLE conditions. During DSP reset condition, HOM_M and HOM_P are invoked. In HOM_M the MPUI interface does not have access to the DARAM (0x000000–0x00FFFF), but it has access to all the SARAM (0x010000–0x050000). The MPU must configure the MPUI_DSP_MPUI_CONFIG register to specify which block of SARAM is accessible in HOM before access. The API_SIZE bit field in the register sets the size using the formula, *(integer value of API_SIZE * 8K bytes)*, starting from the first SARAM block. The DSP is denied access to this specified host–only RAM portion; however, both MPU and DSP can access the other part of the remaining SARAM.

The host can not access the SARAM before releasing the MPU reset. After releasing the MPU reset and before releasing the DSP reset, the DSP is in HOM and all the SARAM is accessible only by the host as the default API_SIZE value is 0xFFFF. Then the MPUI_DSP_MPUI_CONFIG register can be programmed to give the host exclusive access to a portion or to all the SARAM. After the DSP reset is released, the DSP is automatically changed to SAM and all the SARAM is shared between the DSP and the host. MPUI_DSP_MPUI_CONFIG must be set before releasing the DSP from reset and after the MPU reset is deasserted.

Only the DSP can invoke a HOM/SAM change outside of reset. The mode is initiated by a DSP write to HOM_P (bit 8) and HOM_R (bit 9) of the ST3 register. The appropriate bit is written to request the SAM_M/HOM_M or SAM_P/HOM_P change. The mode change is not reflected on bits 8 and 9 in ST3 until the internal controller completes the mode switch. Therefore, the DSP polls bits 8 and 9 after requesting a mode change to ensure that the mode change is complete.

When HOM_P is set, and you attempt to access the DSP public peripherals through the Code Composer Studio session for the DSP, it will generate the following error message:

Trouble Reading Target CPU memory:

Error 0x00000002/–2140

Error during: Memory,

Cannot access memory address at 0x00000000

The reason this message is generated is because the MPU (and the Code Composer Studio session for the ARM) have exclusive access to the DSP public peripherals. When access is given back to the DSP (by clearing HOM_P), the error will go away and visibility will be given to the Code Composer Studio session for the DSP.

When HOM_M is set, the Code Composer Studio session for the ARM has visibility to DSP SARAM (memory mapped to 0xE0000000). However, when HOM_M is cleared, the Code Composer Studio session for the ARM will lose visibility to DSP SARAM and may cause Code Composer Studio for the ARM to be unable to access any memory. The Code Composer Studio session for the DSP will remain alright (see section 2.4).

3.2.2 DSP/MPU Shared Peripherals

The DSP/MPU shared peripherals are designed with two TIPB connections; one for the DSP public and another for the MPU public TIPB. This dual connection provides a flexible communications scheme where either the DSP domain or the MPU domain can access a peripheral without monopolizing the alternate processor public peripheral bus.

3.3 Breakpoints

There are several issues that the developer should be aware of when working with breakpoints during their debug session with the OMAP1610. Note that the same issues also apply when working with Probe Points and also the Profiler (since the Profiler needs to use breakpoints during a profiling session).

3.3.1 Setting Software Breakpoints in Secure RAM

Since software breakpoints need to access the target program in memory, Code Composer Studio will not be able to set software breakpoints on program code located in secure RAM unless secure RAM is made accessible by the steps mentioned in section 3.1.2. If you attempt to set a SW breakpoint without access, Code Composer Studio will generate a “Can’t Set Breakpoint” error.

3.3.2 Hardware Breakpoints

The OMAP1610 hardware resources allow two hardware breakpoints to be enabled simultaneously. If you try to enable more than two breakpoints simultaneously, the debugger will notify you of a resource conflict and display the hardware breakpoint using the resource. You then have the option to disable or remove either the existing breakpoint or the new one (see Figure 4). Just like in any device, when the maximum number of simultaneously enabled hardware breakpoints is reached, you will be restricted in terms of any form of debug stepping (e.g., single source/assembly stepping, run to cursor, etc.) until a hardware breakpoint is disabled.

NOTE: Since hardware breakpoints do not modify the target program like software breakpoints, there are no issues with setting hardware breakpoints in Secure ROM like there are with software breakpoints in secure RAM.

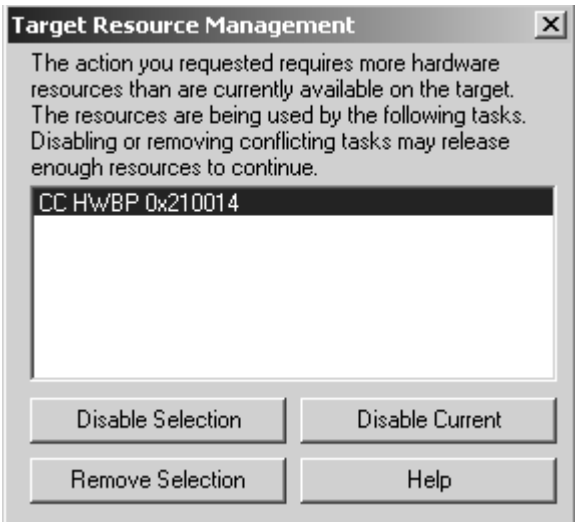

Figure 4. Target Resource Management Dialog

3.4 Watchdog Timers

In addition to the security watchdog timer, there is a separate 32-bit watchdog timer for the OMAP1610 and two private 16–bit watchdog timer peripherals (one for the ARM and one for the DSP). All the watchdog timers are disabled when the OMAP1610 device is initialized with the default GEL startup files (except for the security watchdog timer which is already disabled by default and must be manually enabled). The watchdog timers can also be disabled using custom GEL functions.

3.5 Idle Mode

The OMAP1610 device has two idle modes that it can enter to conserve power (big sleep and deep sleep) when the device is left dormant. The device can not actually go into true idle mode as long as the emulator (and Code Composer Studio) is connected because the emulator will always send the clock signal into the device.

4 References

1. Code Composer Studio v2 Online Help (SPRH199 bundled with the software)
2. *Configuring Code Composer Studio for OMAP Debugging* (SPRA807)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI 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 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

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

Following are URLs where you can obtain information on other Texas Instruments products & application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265