TMS320x2834x Delfino Boot ROM

Reference Guide

Literature Number: SPRUFN5A
March 2009–Revised October 2009

List of Figures

List of Tables

Read This First

This reference guide is applicable for the code and data stored in the on-chip boot ROM on the TMS320C2834x Delfino™ processors. This includes all devices within this family.

The boot ROM is factory programmed with boot-loading software. Boot-mode signals (general purpose I/Os) are used to tell the bootloader software which mode to use on power up. The boot ROM also contains standard math tables, such as SIN/COS waveforms, for use in IQ math related algorithms found in the *C28x™ IQMath Library - A Virtual Floating Point Engine* (literature number SPRC087). Floating-point tables for SIN/COS are also included for use with the Texas Instruments™ *C28x FPU Fast RTS Library* (SPRC664).

This guide describes the purpose and features of the bootloader. It also describes other contents of the device on-chip boot ROM and identifies where all of the information is located within that memory.

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h or with a leading 0x. For example, the following number is 40 hexadecimal (decimal 64): 40h or 0x40.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

Related Documentation From Texas Instruments

The following documents describe the related devices and related support tools. Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

Data Manual—

SPRS516 — TMS320C28346, TMS320C28345, TMS320C28344, TMS320C28343, TMS320C28342, TMS320C28341 Delfino Microcontrollers Data Manual. This document contains the pinout, signal descriptions, as well as electrical and timing specifications for the C2834x devices.

SPRZ267 — TMS320C2834x Delfino MCU Silicon Errata. This document describes the advisories and usage notes for different versions of silicon.

CPU User's Guides—

SPRU430 — TMS320C28x CPU and Instruction Set Reference Guide. This document describes the central processing unit (CPU) and the assembly language instructions of the TMS320C28x fixed-point digital signal processors (DSPs). It also describes emulation features available on these DSPs.

SPRUEO2 — TMS320C28x Floating Point Unit and Instruction Set Reference Guide. This document describes the floating-point unit and includes the instructions for the FPU.

Peripheral Guides—

SPRU566 — TMS320x28xx, 28xxx DSP Peripheral Reference Guide. This document describes the peripheral reference guides of the 28x digital signal processors (DSPs).

SPRUFN1 — TMS320x2834x Delfino System Control and Interrupts Reference Guide. This document describes the various interrupts and system control features of the x2834x microcontroller (MCUs).

SPRUFN4 — TMS320x2834x Delfino External Interface (XINTF) Reference Guide. This document describes the XINTF, which is a nonmultiplexed asynchronous bus, as it is used on the x2834x device.

SPRUFN5 — TMS320x2834x Delfino Boot ROM Reference Guide. This document describes the purpose and features of the bootloader (factory-programmed boot-loading software) and provides examples of code. It also describes other contents of the device on-chip boot ROM and identifies where all of the information is located within that memory.

SPRUG80 — TMS320x2834x Delfino Multichannel Buffered Serial Port (McBSP) Reference Guide. This document describes the McBSP available on the x2834x devices. The McBSPs allow direct interface between a microcontroller (MCU) and other devices in a system.

SPRUG78 — TMS320x2834x Delfino Direct Memory Access (DMA) Reference Guide. This document describes the DMA on the x2834x microcontroller (MCUs).

SPRUFZ6 — TMS320x2834x Delfino Enhanced Pulse Width Modulator (ePWM) Module Reference Guide. This document describes the main areas of the enhanced pulse width modulator that include digital motor control, switch mode power supply control, UPS (uninterruptible power supplies), and other forms of power conversion.

SPRUG77 — TMS320x2834x Delfino High-Resolution Pulse Width Modulator (HRPWM) Reference Guide. This document describes the operation of the high-resolution extension to the pulse width modulator (HRPWM).

SPRUG79 — TMS320x2834x Delfino Enhanced Capture (eCAP) Module Reference Guide. This document describes the enhanced capture module. It includes the module description and registers.

SPRUG74 — TMS320x2834x Delfino Enhanced Quadrature Encoder Pulse (eQEP) Module Reference Guide. This document describes the eQEP module, which is used for interfacing with a linear or rotary incremental encoder to get position, direction, and speed information from a rotating machine in high performance motion and position control systems. It includes the module description and registers.

SPRUEU4 — TMS320x2834x Delfino Enhanced Controller Area Network (eCAN) Reference Guide. This document describes the eCAN that uses established protocol to communicate serially with other controllers in electrically noisy environments.

SPRUG75 — TMS320x2834x Delfino Serial Communication Interface (SCI) Reference Guide. This document describes the SCI, which is a two-wire asynchronous serial port, commonly known as a UART. The SCI modules support digital communications between the CPU and other asynchronous peripherals that use the standard non-return-to-zero (NRZ) format.

SPRUG73 — TMS320x2834x Delfino Serial Peripheral Interface (SPI) Reference Guide. This document describes the SPI - a high-speed synchronous serial input/output (I/O) port - that allows a serial bit stream of programmed length (one to sixteen bits) to be shifted into and out of the device at a programmed bit-transfer rate.

SPRUG76 — TMS320x2834x Delfino Inter-Integrated Circuit (I2C) Reference Guide. This document describes the features and operation of the inter-integrated circuit (I2C) module.

Tools Guides—

SPRU513 — TMS320C28x Assembly Language Tools v5.0.0 User's Guide. This document describes the assembly language tools (assembler and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the TMS320C28x device.

SPRU514 — TMS320C28x Optimizing C/C++ Compiler v5.0.0 User's Guide. This document describes the TMS320C28x™ C/C++ compiler. This compiler accepts ANSI standard C/C++ source code and produces TMS320 DSP assembly language source code for the TMS320C28x device.

SPRU608 — TMS320C28x Instruction Set Simulator Technical Overview. This document describes the simulator, available within the Code Composer Studio for TMS320C2000 IDE, that simulates the instruction set of the C28x™ core.

SPRU625 — TMS320C28x DSP/BIOS 5.32 Application Programming Interface (API) Reference Guide. This document describes development using DSP/BIOS.

Application Reports—

SPRAB26 — TMS320x2833x/2823x to TMS320x2834x Delfino Migration Overview. This application report describes differences between the Texas Instruments TMS320x2833x/2823x and the TMS320x2834x devices to assist in application migration.

Piccolo Boot ROM

1 Boot ROM Memory Map

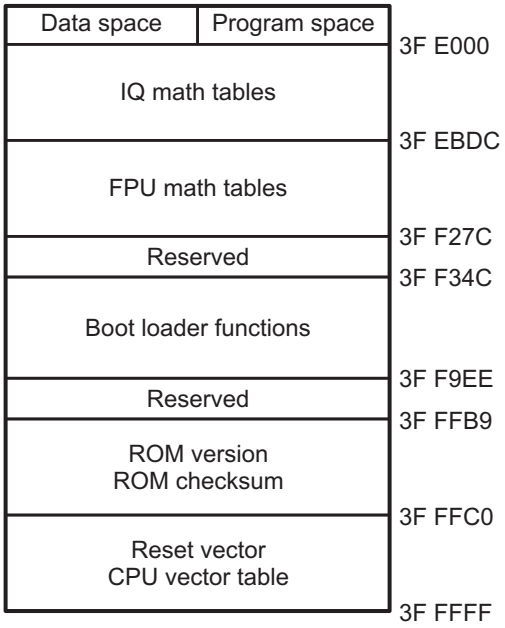
The boot ROM is an 8K x 16 block of read-only memory located at addresses 0x3F E000 - 0x3F FFFF.

The on-chip boot ROM is factory programmed with boot-load routines and math tables. These are for use with the *C28x™ IQMath Library - A Virtual Floating Point Engine* (SPRC087) and the *C28x FPU Fast RTS Library* (SPRC664). This document describes the following items:

- Bootloader functions
- Version number, release date and checksum
- Reset vector
- Illegal trap vector (ITRAP)
- CPU vector table (Used for test purposes only)
- IQmath Tables
- Floating-point unit (FPU) math tables

Figure 1 shows the memory map of the on-chip boot ROM. The memory block is 8Kx16 in size and is located at 0x3F E000 - 0x3F FFFF in both program and data space.

Figure 1. Memory Map of On-Chip ROM



1.1 On-Chip Boot ROM IQmath Tables

Approximately 4K of the boot ROM is reserved for floating-point and IQmath tables. These tables are provided to help improve performance and save SARAM space.

The floating-point math tables included in the boot ROM are used by the Texas Instruments™ *C28x FPU Fast RTS Library* (SPRC664). The C28x Fast RTS Library is a collection of optimized floating-point math functions for C programmers of the C28x with floating-point unit. Designers of computationally intensive real-time applications can achieve execution speeds considerably faster than what are currently available without having to rewrite existing code. The functions listed in the features section are specifically optimized for the C28x + FPU controllers. The Fast RTS library accesses the floating-point tables through the FPUmathTables memory section. If you do not wish to load a copy of these tables into the device, use the boot ROM memory addresses and label the section as “NOLOAD” as shown in Example 1. This facilitates referencing the look-up tables without actually loading the section to the target.

The following floating-point math tables are included in the Boot ROM:

- **Sine/Cosine Table, Single-precision Floating-point**
  - Table size: 1282 words
  - Contents: 32-bit floating-point samples for one and a quarter period sine wave
- **Normalized Arctan Table, Single-Precision Floating Point**
  - Table Size: 388 words
  - Contents: 32-bit second order coefficients for line of best fit.
- **Exp Coefficient Table, Single-Precision Floating Point**
  - Table size: 20 words
  - Contents: 32-bit coefficients for calculating exp (X) using a taylor series

Example 1. Linker Command File to Access FPU Tables

```
MEMORY
{
    PAGE 0 :
    ...
    FPUTABLES : origin = 0x3FEBDC, length = 0x0006A0
    ...
}
    SECTIONS
{
    ...
    FPUmathTables : > FPUTABLES, PAGE = 0, TYPE = NOLOAD
    ...
}
```

The fixed-point math tables included in the boot ROM are used by the Texas Instruments™ *C28x™ IQMath Library - A Virtual Floating Point Engine* (SPRC087). The 28x IQmath Library is a collection of highly optimized and high precision mathematical functions for C/C++ programmers to seamlessly port a floating-point algorithm into fixed-point code on TMS320C28x devices.

These routines are typically used in computational-intensive real-time applications where optimal execution speed and high accuracy is critical. By using these routines you can achieve execution speeds that are considerably faster than equivalent code written in standard ANSI C language. In addition, by providing ready-to-use high precision functions, the TI IQmath Library can shorten significantly your DSP application development time.

IQmath library accesses the tables through the IQmathTables and the IQmathTablesRam linker sections. The IQmathTables section is completely included in the boot ROM. From the IQmathTablesRam section only the IQexp table is included and the remainder must be loaded into the device if used. If you do not wish to load a copy of these tables already included in the ROM into the device, use the boot ROM memory addresses and label the sections as “NOLOAD” as shown in Example 2 . This facilitates referencing the look-up tables without actually loading the section to the target.

Example 2. Linker Command File to Access IQ Tables

```
MEMORY
{
   PAGE 0 :
...
   IQTABLES : origin = 0x3FE000, length = 0x000b50
   IQTABLES2 : origin = 0x3FEB50, length = 0x00008c
   ...
}
SECTIONS
{
   ...
   IQmathTables : load = IQTABLES, type = NOLOAD, PAGE = 0
   IQmathTables2 > IQTABLES2, type = NOLOAD, PAGE = 0
   {
      IQmath.lib<IQNexpTable.obj> (IQmathTablesRam)
   }
   IQmathTablesRam : load = DRAML1, PAGE = 1
   ...
}
```

The following math tables are included in the Boot ROM:

- **Sine/Cosine Table, IQ Math Table**
  - Table size: 1282 words
  - Q format: Q30
  - Contents: 32-bit samples for one and a quarter period sine wave

  This is useful for accurate sine wave generation and 32-bit FFTs. This can also be used for 16-bit math, just skip over every second value.
- **Normalized Inverse Table, IQ Math Table**
  - Table size: 528 words
  - Q format: Q29
  - Contents: 32-bit normalized inverse samples plus saturation limits

  This table is used as an initial estimate in the Newton-Raphson inverse algorithm. By using a more accurate estimate the convergence is quicker and hence cycle time is faster.
- **Normalized Square Root Table, IQ Math Table**
  - Table size: 274 words
  - Q format: Q30
  - Contents: 32-bit normalized inverse square root samples plus saturation

  This table is used as an initial estimate in the Newton-Raphson square-root algorithm. By using a more accurate estimate the convergence is quicker and hence cycle time is faster.
- **Normalized Arctan Table, IQ Math Table**
  - Table size: 452 words
  - Q format: Q30
  - Contents 32-bit second order coefficients for line of best fit plus normalization table

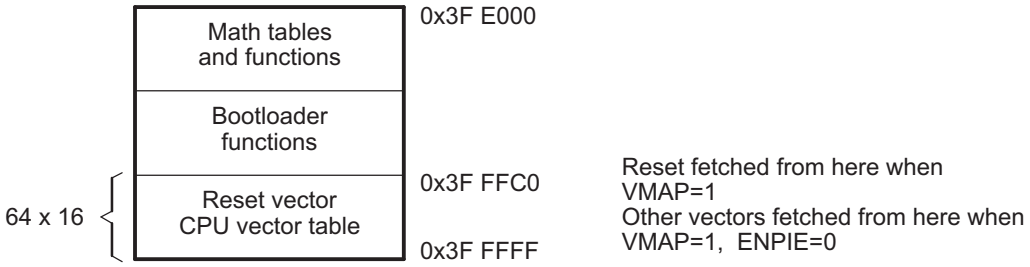
  This table is used as an initial estimate in the Arctan iterative algorithm. By using a more accurate estimate the convergence is quicker and hence cycle time is faster.
- **Rounding and Saturation Table, IQ Math Table**
  - Table size: 360 words
  - Q format: Q30
  - Contents: 32-bit rounding and saturation limits for various Q values
- **Exp Min/Max Table, IQMath Table**
  - Table size: 120 words

  - Q format: Q1 - Q30
  - Contents: 32-bit Min and Max values for each Q value
- **Exp Coefficient Table, IQMath Table**
  - Table size: 20 words
  - Q format: Q31
  - Contents: 32-bit coefficients for calculating exp (X) using a taylor series

1.2 CPU Vector Table

A CPU vector table resides in boot ROM memory from address 0x3F E000 - 0x3F FFFF. This vector table is active after reset when VMAP = 1, ENPIE = 0 (PIE vector table disabled).

Figure 2. Vector Table Map



A The VMAP bit is located in Status Register 1 (ST1). VMAP is always 1 on reset. It can be changed after reset by software, however the normal operating mode will be to leave VMAP = 1.

B The ENPIE bit is located in the PIECTRL register. The default state of this bit at reset is 0, which disables the Peripheral Interrupt Expansion block (PIE).

The only vector that will normally be handled from the internal boot ROM memory is the reset vector located at 0x3F FFC0. The reset vector is factory programmed to point to the InitBoot function stored in the boot ROM. This function starts the boot load process. A series of checking operations is performed on General-Purpose I/O (GPIO I/O) pins to determine which boot mode to use. This boot mode selection is described in Section 2.9 of this document.

The remaining vectors in the boot ROM are not used during normal operation. After the boot process is complete, you should initialize the Peripheral Interrupt Expansion (PIE) vector table and enable the PIE block. From that point on, all vectors, except reset, will be fetched from the PIE module and not the CPU vector table shown in Table 1.

For TI silicon debug and test purposes the vectors located in the boot ROM memory point to locations in the M0 SARAM block as described in Table 1. During silicon debug, you can program the specified locations in M0 with branch instructions to catch any vectors fetched from boot ROM. This is not required for normal device operation.

Table 1. Vector Locations

Vector	Location in Boot ROM	Contents (i.e., points to)	Vector	Location in Boot ROM	Contents (i.e., points to)
RESET	0x3F FFC0	InitBoot	RTOSINT	0x3F FFE0	0x00 0060
INT1	0x3F FFC2	0x00 0042	Reserved	0x3F FFE2	0x00 0062
INT2	0x3F FFC4	0x00 0044	NMI	0x3F FFE4	0x00 0064
INT3	0x3F FFC6	0x00 0046	ILLEGAL	0x3F FFE6	ITRAPIsr
INT4	0x3F FFC8	0x00 0048	USER1	0x3F FFE8	0x00 0068
INT5	0x3F FFCA	0x00 004A	USER2	0x3F FFEA	0x00 006A
INT6	0x3F FFCC	0x00 004C	USER3	0x3F FFEC	0x00 006C
INT7	0x3F FFCE	0x00 004E	USER4	0x3F FFEE	0x00 006E
INT8	0x3F FFD0	0x00 0050	USER5	0x3F FFF0	0x00 0070
INT9	0x3F FFD2	0x00 0052	USER6	0x3F FFF2	0x00 0072
INT10	0x3F FFD4	0x00 0054	USER7	0x3F FFF4	0x00 0074
INT11	0x3F FFD6	0x00 0056	USER8	0x3F FFF6	0x00 0076
INT12	0x3F FFD8	0x00 0058	USER9	0x3F FFF8	0x00 0078
INT13	0x3F FFDA	0x00 005A	USER10	0x3F FFFA	0x00 007A
INT14	0x3F FFDC	0x00 005C	USER11	0x3F FFFC	0x00 007C
DLOGINT	0x3F FFDE	0x00 005E	USER12	0x3F FFFE	0x00 007E

2 Bootloader Features

This section describes in detail the boot mode selection process, as well as the specifics of the bootloader operation.

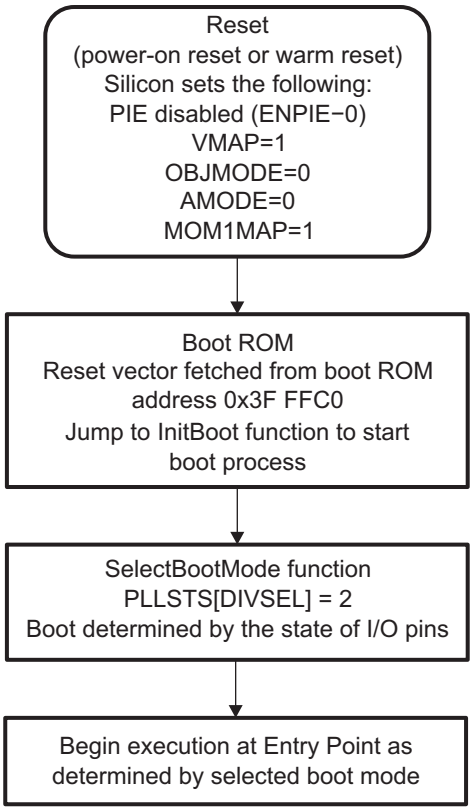
2.1 Bootloader Functional Operation

The bootloader is the program located in the on-chip boot ROM that is executed following a reset.

The bootloader is used to transfer code from an external source into internal memory following power up. This allows code to reside in slow non-volatile memory externally, and be transferred to high-speed memory to be executed.

The bootloader provides a variety of different ways to download code to accommodate different system requirements. The bootloader uses various GPIO signals to determine which boot mode to use. The boot mode selection process as well as the specifics of each bootloader are described in the remainder of this document. Figure 3 shows the basic bootloader flow.

Figure 3. Bootloader Flow Diagram



The reset vector in boot ROM redirects program execution to the InitBoot function. After performing device initialization the bootloader will check the state of GPIO pins to determine which boot mode you want to execute. Options include: jump to SARAM, jump to XINTF, or call one of the on-chip boot loading routines.

After the selection process and if the required boot loading is complete, the processor will continue execution at an entry point determined by the boot mode selected. If a bootloader was called, then the input stream loaded by the peripheral determines this entry address. This data stream is described in Section 2.10. If, instead, you choose to boot directly to XINTF or SARAM, the entry address is predefined for each of these memory blocks.

The following sections discuss in detail the different boot modes available and the process used for loading data code into the device.

2.2 Bootloader Device Configuration

At reset, any 28x™ CPU-based device is in 27x™ object-compatible mode. It is up to the application to place the device in the proper operating mode before execution proceeds.

On the 28x devices, when booting from the internal boot ROM, the device is configured for 28x operating mode by the boot ROM software. You are responsible for any additional configuration required.

For example, if your application includes C2xLP™ source, then you are responsible for configuring the device for C2xLP source compatibility prior to execution of code generated from C2xLP source.

The configuration required for each operating mode is summarized in Table 2.

Table 2. Configuration for Device Modes

	C27x Mode (Reset)	28x Mode	C2xLP Source Compatible Mode
OBJMODE	0	1	1
AMODE	0	0	1

Table 2. Configuration for Device Modes (continued)

	C27x Mode (Reset)	28x Mode	C2xLP Source Compatible Mode
PAGE0	0	0	0
M0M1MAP[1]	1	1	1
Other Settings			SXM = 1, C = 1, SPM = 0

(1) Normally for C27x compatibility, the M0M1MAP would be 0. On these devices, however, it is tied off high internally; therefore, at reset, M0M1MAP is always configured for 28x mode.

2.3 PLL Multiplier and DIVSEL Selection

The Boot ROM changes the PLL multiplier (PLLCR) and divider (PLLSTS[DIVSEL]) bits as follows:

- **XINTF parallel loader:**
  PLLCR and PLLSTS[DIVSEL] are specified by the user as part of the incoming data stream.
- **eCAN Timing 1 loader:**
  PLLCR is not modified. PLLSTS[DIVSEL] is set to 3 for SYSCLKOUT = CLKIN/1. Refer to Section 2.21 for details.
- **All other boot modes:**
  PLLCR is not modified. PLLSTS[DIVSEL] is set to 2 for SYSCLKOUT = CLKIN/2 . This increases the speed of the loaders.

NOTE: The PLL multiplier (PLLSTS) and divider (PLLSTS[DIVSEL]) are not affected by a reset from the debugger. Therefore, a boot that is initialized from a reset from Code Composer Studio™ may be at a different speed than booting by pulling the external reset line ($\overline{XRS}$) low.

NOTE: The reset value of PLLSTS[DIVSEL] is 0. This configures the device for SYSCLKOUT = CLKIN/8. The boot ROM will change this to SYSCLKOUT = CLKIN/2 or CLKIN/1 to improve performance of the loaders. PLLSTS[DIVSEL] is left in this state when the boot ROM exits and it is up to the application to change it before configuring the PLLCR register.

NOTE: The eCAN Timing 1 loader leaves PLLSTS[DIVSEL] in the CLKIN/1 state when the boot ROM exits. This is not a valid configuration if the PLL is used. Thus the application must change it before configuring the PLLCR register.

2.4 Watchdog Module

When branching directly to M0 single-access RAM (SARAM) or external interface (XINTF) the watchdog is not touched. In the other boot modes, the watchdog is disabled before booting and then re-enabled and cleared before branching to the final destination address. In the case of a reserved boot mode being selected, or an incorrect key value passed to the loader, the watchdog will be enabled and the device reset.

2.5 Taking an ITRAP Interrupt

If an illegal opcode is fetched, the 28x will take an ITRAP (illegal trap) interrupt. During the boot process, the interrupt vector used by the ITRAP is within the CPU vector table of the boot ROM. The ITRAP vector points to an interrupt service routine (ISR) within the boot ROM named ITRAPIsr(). This interrupt service routine attempts to enable the watchdog and then loops forever until the processor is reset. This ISR will be used for any ITRAP until the user's application initializes and enables the peripheral interrupt expansion (PIE) block. Once the PIE is enabled, the ITRAP vector located within the PIE vector table will be used.

2.6 Internal Pullup Resisters

Each GPIO pin has an internal pullup resistor that can be enabled or disabled in software. The pins that are read by the boot mode selection code to determine the boot mode selection have pull-ups enabled after reset by default. In noisy conditions it is still recommended that you configure each of the boot mode selection pins externally.

The peripheral bootloaders all enable the pullup resistors for the pins that are used for control and data transfer. The bootloader leaves the resistors enabled for these pins when it exits. For example, the SCI-A bootloader enables the pullup resistors on the SCITXA and SCIRXA pins. It is your responsibility to disable them, if desired, after the bootloader exits.

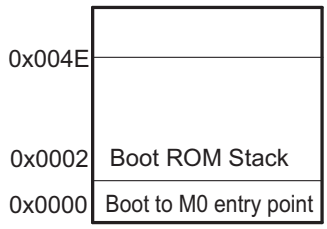
2.7 PIE Configuration

The boot modes do not enable the PIE. It is left in its default state, which is disabled.

2.8 Reserved Memory

The M0 memory block address range 0x0002 - 0x004E is reserved for the stack and .ebss code sections during the boot-load process. If code is bootloaded into this region there is no error checking to prevent it from corrupting the boot ROM stack. Address 0x0000-0x0001 is the boot to M0 entry point. This should be loaded with a branch instruction to the start of the main application when using "boot to SARAM" mode.

Figure 4. Boot ROM Stack



Boot ROM loaders on older C28x devices had the stack in M1 memory. This is a change for this boot loader.

NOTE: If code or data is bootloaded into the address range address range 0x0002 - 0x004E there is no error checking to prevent it from corrupting the boot ROM stack.

2.9 Bootloader Modes

To accommodate different system requirements, the boot ROM offers a variety of boot modes. This section describes the different boot modes and gives brief summary of their functional operation. The states of four GPIO pins are used to determine the desired boot mode as shown in Table 3. In the case of I2C and eCAN, there are two timing options in order to accommodate different input clock frequencies. Each of the timing options is evoked by selecting a different boot mode.

Table 3. Boot Mode Selection

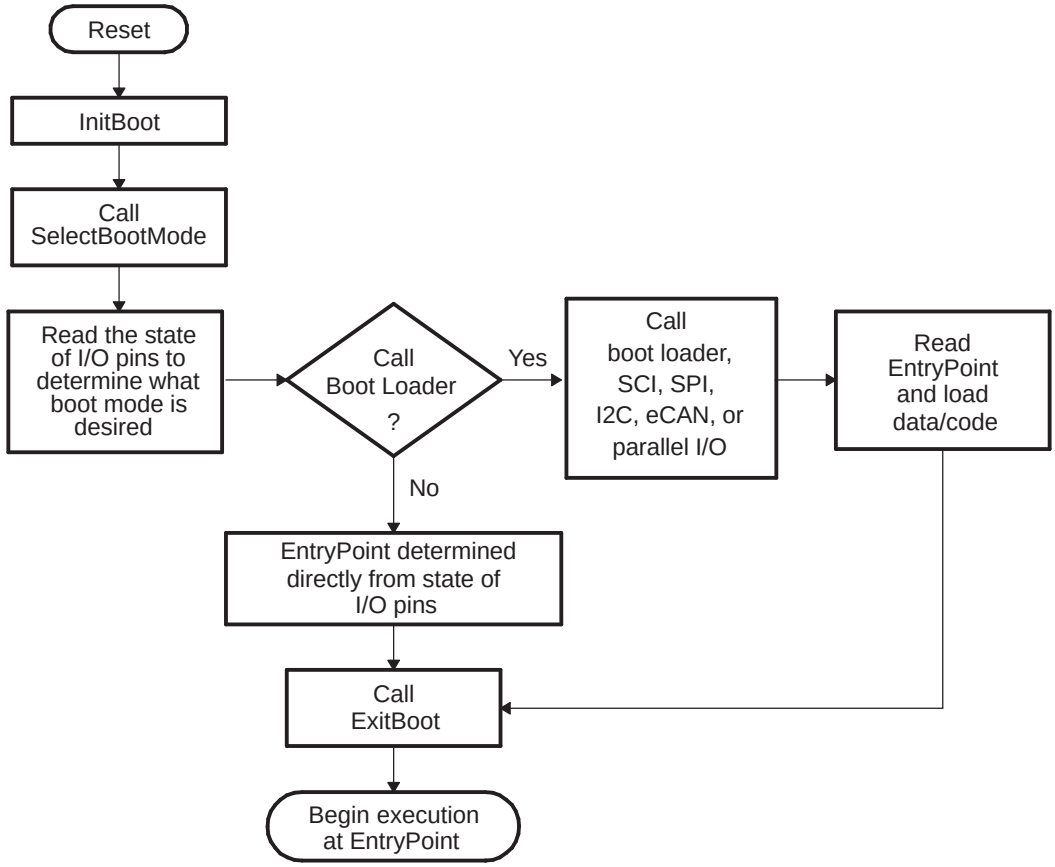
MODE	GPIO87/XA15	GPIO86/XA14	GPIO85/XA13	GPIO84/XA12	MODE (1)
F	1	1	1	1	Secure Boot(2)
E	1	1	1	0	SCI-A boot
D	1	1	0	1	SPI-A boot
C	1	1	0	0	I2C-A boot Timing 1
B	1	0	1	1	eCAN-A boot Timing 1

(1) All four GPIO pins have an internal pullup.
(2) This mode is available on secure devices only. Refer to the data manual or contact support@ti.com for more information.

MODE	GPIO87/XA15	GPIO86/XA14	GPIO85/XA13	GPIO84/XA12	MODE (1)
A	1	0	1	0	McBSP-A boot
9	1	0	0	1	Jump to XINTF x16
8	1	0	0	0	TI Test Only
7	0	1	1	1	eCAN-A boot Timing 2
6	0	1	1	0	Parallel GPIO I/O boot
5	0	1	0	1	Parallel XINTF boot
4	0	1	0	0	Jump to SARAM
3	0	0	1	1	Branch to check boot mode
2	0	0	1	0	I2C-A boot Timing 2
1	0	0	0	1	Reserved
0	0	0	0	0	TI Test Only

Figure 5 shows an overview of the boot process. Each step is described in greater detail in following sections.

Figure 5. Boot ROM Function Overview



The following boot mode is used for debug purposes:

- **Branch to check boot mode**

  When initially debugging a device the emulator takes some time to take control of the CPU. During this time, the CPU will start running and may execute some portion of the application. During debug this may not be desirable. Two solutions to this problem exist:

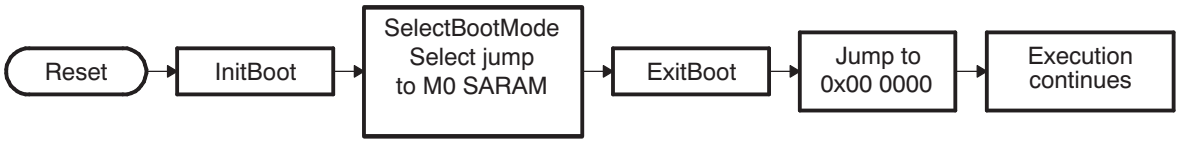
  – The first is to use the Wait-In-Reset emulation mode, which will hold the device in reset until the emulator takes control. The emulator must support this mode for this option.
  – The second option is to use the “Branch to check boot mode” boot option. This will sit in a loop and continuously poll the boot mode select pins. The user can select this boot mode and then exit this mode once the emulator is connected by re-mapping the PC to another address or by changing the boot mode selection pin to the desired boot mode.

The following boot modes do not call a bootloader. Instead, they jump to a predefined location in memory:

- **Jump to M0 SARAM**

  In this mode, the boot ROM software configures the device for 28x operation and branches directly to 0x00 0000. This is the first address in the M0 SARAM memory block.

Figure 6. Flow Diagram of Jump to M0 SARAM



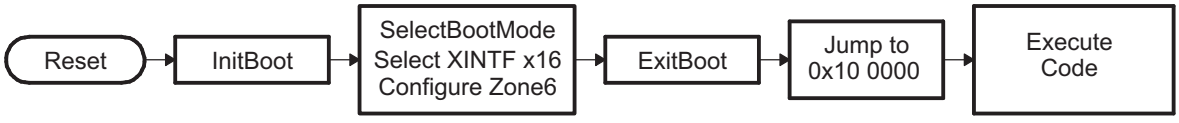
- **Jump to XINTF Zone 6 Configured for 16-bit Data**

  The boot ROM configures XINTF zone 6 for 16 bit wide memory, maximum wait states, and sample XREADY in asynchronous mode. This is the default values list after reset:

  – XTIMCLK = ½ SYSCLKOUT
  – XCLKOUT = 1/4 XTIMCLK
  – XRDLEAD = XWRLEAD = 3
  – XRDACTIVE = XWRACTIVE = 7
  – XRDTRAIL = XWRACTIVE = 3
  – XSIZE = 16-bit wide
  – X2TIMING = 1. Timing values are 2:1.
  – USEREADY = 1, READYMODE = 1 (XREADY sampled asynchronous mode)

  The boot ROM will then jump to the first location within zone 6 at address 0x10 0000.

Figure 7. Flow Diagram of Jump to XINTF x16



The following boot modes call a boot load routine that loads a data stream from the peripheral into memory:

- **Standard serial boot mode (SCI-A)**

  In this mode, the boot ROM will load code to be executed into on-chip memory via the SCI-A port.

- **SPI EEPROM or Flash boot mode (SPI-A)**

  In this mode, the boot ROM will load code and data into on-chip memory from an external SPI EEPROM or SPI flash via the SPI-A port.

- **I2C-A boot mode (I2C-A)**

  In this mode, the boot ROM will load code and data into on-chip memory from an external serial EEPROM or flash at address 0x50 on the I2C-A bus. The EEPROM must adhere to conventional I2C EEPROM protocol with a 16-bit base address architecture. To accommodate different input clock frequencies, there are two timing options for the I2C loader. Each timing option is evoked by selecting

a specific boot mode.

- **eCAN Boot Mode (eCAN-A)**

  In this mode, the eCAN-A peripheral is used to transfer data and code into the on-chip memory using eCAN-A mailbox 1. The transfer is an 8-bit data stream with two 8-bit values being transferred during each communication. To accommodate different input clock frequencies, there are two timing options for the eCAN-A loader. Each timing option is evoked by selecting a specific boot mode.
- **McBSP Boot Mode (McBSP-A)**

  Synchronously transfers code from McBSP-A to internal memory. McBSP-A is configured for slave mode operation. i.e. it receives the frame sync and clock from the host. Upon receiving a word, the McBSP echoes the data back to the host.
- **Boot from GPIO Port (Parallel Boot from GPIO0-GPIO15)**

  In this mode, the boot ROM uses GPIO port A pins GPIO0-GPIO15 to load code and data from an external source. This mode supports both 8-bit and 16-bit data streams.
- **Boot From XINTF (Parallel Boot From XD[15:0])**

  This mode is similar to the GPIO parallel boot mode except the boot ROM uses XINTF data lines XD[15:0] to load code and data from an external source instead of GPIO pins. This mode supports both 8-bit and 16-bit data streams. The user can specify the PLL configuration as well as XINTF timing through the input data stream.

2.10 Bootloader Data Stream Structure

The following two tables and associated examples show the structure of the data stream incoming to the bootloader. The basic structure is the same for all the bootloaders and is based on the C54x source data stream generated by the C54x hex utility. The C28x hex utility (hex2000.exe) has been updated to support this structure. The hex2000.exe utility is included with the C2000 code generation tools. All values in the data stream structure are in hex.

The first 16-bit word in the data stream is known as the key value. The key value is used to tell the bootloader the width of the incoming stream: 8 or 16 bits. Note that not all bootloaders will accept both 8 and 16-bit streams. Please refer to the detailed information on each loader for the valid data stream width. For an 8-bit data stream, the key value is 0x08AA and for a 16-bit stream it is 0x10AA. If a bootloader receives an invalid key value, then the load is aborted. In this case, the watchdog is enabled and a device reset is forced through software.

The next 8 words are used to initialize register values or otherwise enhance the bootloader by passing values to it. If a bootloader does not use these values then they are reserved for future use and the bootloader simply reads the value and then discards it. Currently only the SPI and I2C and parallel XINTF bootloaders use these words to initialize registers.

The tenth and eleventh words comprise the 22-bit entry point address. This address is used to initialize the PC after the boot load is complete. This address is most likely the entry point of the program downloaded by the bootloader.

The twelfth word in the data stream is the size of the first data block to be transferred. The size of the block is defined for both 8-bit and 16-bit data stream formats as the number of 16-bit words in the block. For example, to transfer a block of 20 8-bit data values from an 8-bit data stream, the block size would be 0x000A to indicate 10 16-bit words.

The next two words tell the loader the destination address of the block of data. Following the size and address will be the 16-bit words that makeup that block of data.

This pattern of block size/destination address repeats for each block of data to be transferred. Once all the blocks have been transferred, a block size of 0x0000 signals to the loader that the transfer is complete. At this point the loader will return the entry point address to the calling routine which in turn will cleanup and exit. Execution will then continue at the entry point address as determined by the input data stream contents.

Table 4. General Structure Of Source Program Data Stream In 16-Bit Mode

Word	Contents
1	10AA (KeyValue for memory width = 16bits)
2	Register initialization value or reserved for future use
3	Register initialization value or reserved for future use
4	Register initialization value or reserved for future use
5	Register initialization value or reserved for future use
6	Register initialization value or reserved for future use
7	Register initialization value or reserved for future use
8	Register initialization value or reserved for future use
9	Register initialization value or reserved for future use
10	Entry point PC[22:16]
11	Entry point PC[15:0]
12	Block size (number of words) of the first block of data to load. If the block size is 0, this indicates the end of the source program. Otherwise another section follows.
13	Destination address of first block Addr[31:16]
14	Destination address of first block Addr[15:0]
15	First word of the first block in the source being loaded
...	...
...	...
.	Last word of the first block of the source being loaded
.	Block size of the 2nd block to load.
.	Destination address of second block Addr[31:16]
.	Destination address of second block Addr[15:0]
.	First word of the second block in the source being loaded
.	…
.	Last word of the second block of the source being loaded
.	Block size of the last block to load
.	Destination address of last block Addr[31:16]
.	Destination address of last block Addr[15:0]
.	First word of the last block in the source being loaded
...	...
...	...
n	Last word of the last block of the source being loaded
n+1	Block size of 0000h - indicates end of the source program

Example 3. Data Stream Structure 16-bit

```
10AA                ; 0x10AA 16-bit key value
0000 0000 0000 0000 ; 8 reserved words
0000 0000 0000 0000
003F 8000           ; 0x003F8000 EntryAddr, starting point after boot load completes
0005                ; 0x0005 - First block consists of 5 16-bit words
003F 9010           ; 0x003F9010 - First block will be loaded starting at 0x3F9010
0001 0002 0003 0004 ; Data loaded = 0x0001 0x0002 0x0003 0x0004 0x0005
0005
0002                ; 0x0002 - 2nd block consists of 2 16-bit words
003F 8000           ; 0x003F8000 - 2nd block will be loaded starting at 0x3F8000
7700 7625           ; Data loaded = 0x7700 0x7625
0000                ; 0x0000 - Size of 0 indicates end of data stream
After load has completed the following memory values will have been initialized as follows:
Location     Value
0x3F9010     0x0001
0x3F9011     0x0002
0x3F9012     0x0003
0x3F9013     0x0004
0x3F9014     0x0005
0x3F8000     0x7700
0x3F8001     0x7625
PC Begins execution at 0x3F8000
```

In 8-bit mode, the least significant byte (LSB) of the word is sent first followed by the most significant byte (MSB). For 32-bit values, such as a destination address, the most significant word (MSW) is loaded first, followed by the least significant word (LSW). The bootloaders take this into account when loading an 8-bit data stream.

Table 5. LSB/MSB Loading Sequence in 8-Bit Data Stream

Byte		Contents	
		LSB (First Byte of 2)	MSB (Second Byte of 2)
1	2	LSB: AA (KeyValue for memory width = 8 bits)	MSB: 08h (KeyValue for memory width = 8 bits)
3	4	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
5	6	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
7	8	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
...	...	...	...
...	...	...	...
17	18	LSB: Register initialization value or reserved	MSB: Register initialization value or reserved
19	20	LSB: Upper half of Entry point PC[23:16]	MSB: Upper half of entry point PC[31:24] (Always 0x00)
21	22	LSB: Lower half of Entry point PC[7:0]	MSB: Lower half of Entry point PC[15:8]
23	24	LSB: Block size in words of the first block to load. If the block size is 0, this indicates the end of the source program. Otherwise another block follows. For example, a block size of 0x000A would indicate 10 words or 20 bytes in the block.	MSB: block size
25	26	LSB: MSW destination address, first block Addr[23:16]	MSB: MSW destination address, first block Addr[31:24]
27	28	LSB: LSW destination address, first block Addr[7:0]	MSB: LSW destination address, first block Addr[15:8]
29	30	LSB: First word of the first block being loaded	MSB: First word of the first block being loaded
...	...	...	...
...	...	...	...
.	.	LSB: Last word of the first block to load	MSB: Last word of the first block to load
.	.	LSB: Block size of the second block	MSB: Block size of the second block
.	.	LSB: MSW destination address, second block Addr[23:16]	MSB: MSW destination address, second block Addr[31:24]
.	.	LSB: LSW destination address, second block Addr[7:0]	MSB: LSW destination address, second block Addr[15:8]
.	.	LSB: First word of the second block being loaded	MSB: First word of the second block being loaded
...	...	...	...
...	...	...	...
.	.	LSB: Last word of the second block	MSB: Last word of the second block
.	.	LSB: Block size of the last block	MSB: Block size of the last block
.	.	LSB: MSW of destination address of last block Addr[23:16]	MSB: MSW destination address, last block Addr[31:24]
.	.	LSB: LSW destination address, last block Addr[7:0]	MSB: LSW destination address, last block Addr[15:8]
.	.	LSB: First word of the last block being loaded	MSB: First word of the last block being loaded
...	...	...	...
...	...	...	...
.	.	LSB: Last word of the last block	MSB: Last word of the last block
n	n+1	LSB: 00h	MSB: 00h - indicates the end of the source

Example 4. Data Stream Structure 8-bit

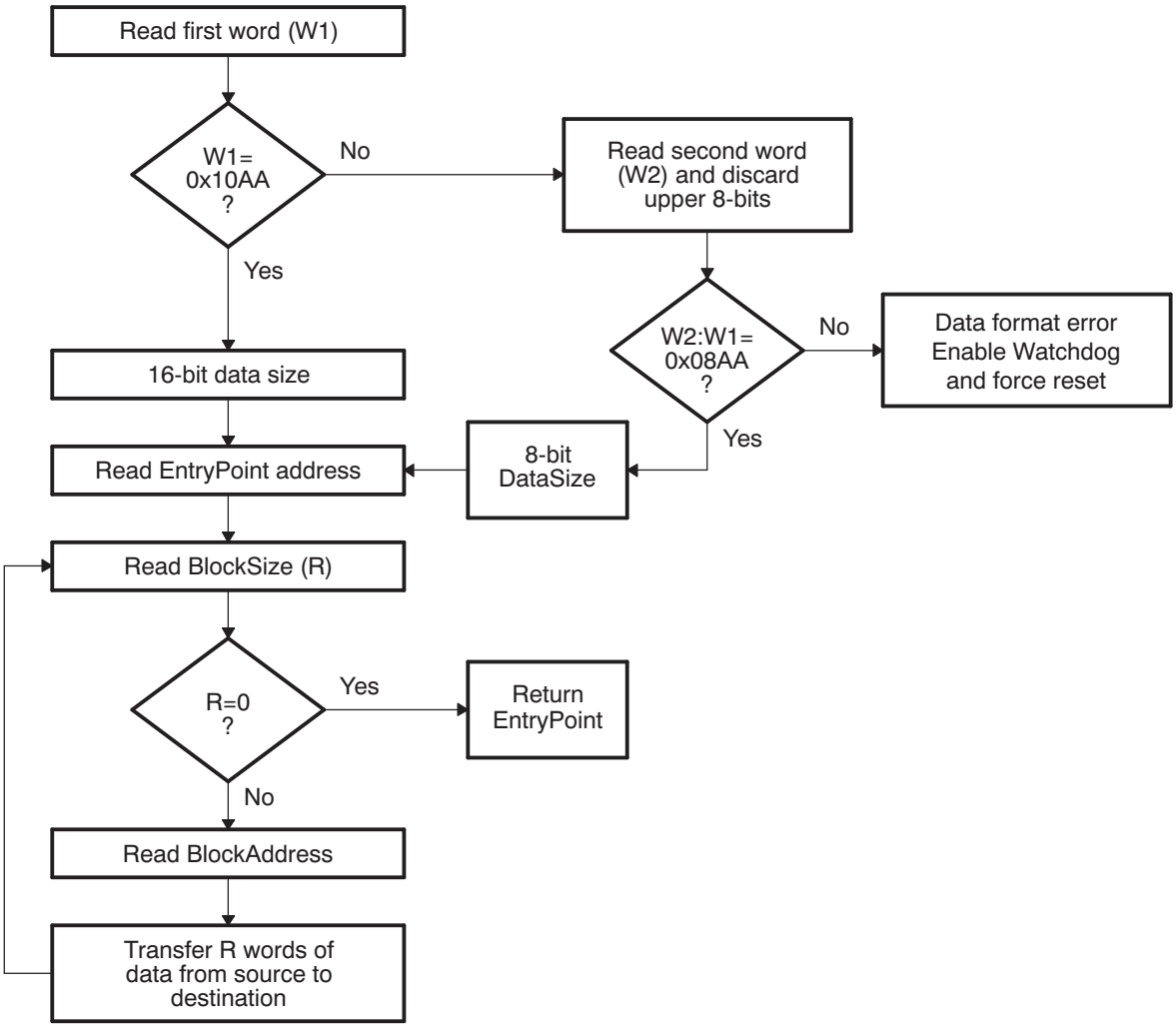
```
AA 08          ; 0x08AA 8-bit key value
00 00 00 00    ; 8 reserved words
00 00 00 00
00 00 00 00
00 00 00 00
3F 00 00 80    ; 0x003F8000 EntryAddr, starting point after boot load completes
05 00          ; 0x0005 - First block consists of 5 16-bit words
3F 00 10 90    ; 0x003F9010 - First block will be loaded starting at 0x3F9010
01 00          ; Data loaded = 0x0001 0x0002 0x0003 0x0004 0x0005
02 00
03 00
04 00
05 00
02 00          ; 0x0002 - 2nd block consists of 2 16-bit words
3F 00 00 80    ; 0x003F8000 - 2nd block will be loaded starting at 0x3F8000
00 77          ; Data loaded = 0x7700 0x7625
25 76
00 00          ; 0x0000 - Size of 0 indicates end of data stream

After load has completed the following memory values will have been initialized as follows:

Location   Value
0x3F9010   0x0001
0x3F9011   0x0002
0x3F9012   0x0003
0x3F9013   0x0004
0x3F9014   0x0005
0x3F8000   0x7700
0x3F8001   0x7625
PC Begins execution at 0x3F8000
```

2.11 Basic Transfer Procedure

Figure 8 illustrates the basic process a bootloader uses to determine whether 8-bit or 16-bit data stream has been selected, transfer that data, and start program execution. This process occurs after the bootloader finds the valid boot mode selected by the state of GPIO pins.

The loader first compares the first value sent by the host against the 16-bit key value of 0x10AA. If the value fetched does not match then the loader will read a second value. This value will be combined with the first value to form a word. This will then be checked against the 8-bit key value of 0x08AA. If the loader finds that the header does not match either the 8-bit or 16-bit key value, or if the value is not valid for the given boot mode then the load will abort. In this case the loader will enable the watchdog and force a device reset through software.

Figure 8. Bootloader Basic Transfer Procedure



8-bit and 16-bit transfers are not valid for all boot modes. If only one mode is valid, then this decision tree is skipped and the key value is only checked for correctness. See the info specific to a particular bootloader for any limitations.

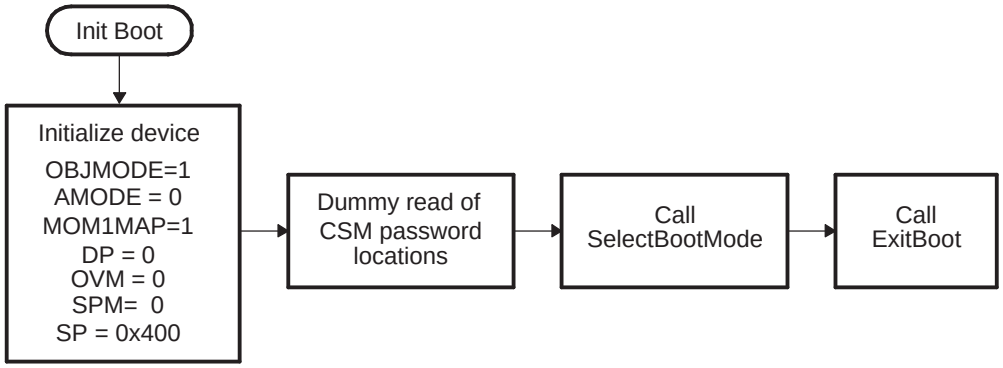
In 8-bit mode, the LSB of the 16-bit word is read first followed by the MSB.

2.12 InitBoot Assembly Routine

The first routine called after reset is the InitBoot assembly routine. This routine initializes the device for operation in C28x object mode. InitBoot also performs a dummy read of the Code Security Module (CSM) password locations. The 128-bit password locations on these devices will always read back 0xFFFF. To preserve compatibility with other C28x designs with code security, the password locations must be read after a device reset; otherwise, certain memory locations will be inaccessible. The Boot ROM code performs this read during startup. If during debug the Boot ROM is bypassed, then it is the responsibility of the application software to read the password locations after a reset.

After the dummy read of the CSM password locations, the InitBoot routine calls the SelectBootMode function. This function determines the type of boot mode desired by the state of certain GPIO pins. This process is described in Section 2.13. Once the boot is complete, the SelectBootMode function passes back the entry point address (EntryAddr) to the InitBoot function. EntryAddr is the location where code execution will begin after the bootloader exits. InitBoot then calls the ExitBoot routine that then restores CPU registers to their reset state and exits to the EntryAddr that was determined by the boot mode.

Figure 9. Overview of InitBoot Assembly Function



2.13 SelectBootMode Function

To determine the desired boot mode, the SelectBootMode function examines the state of 3 GPIO pins as shown in Table 3.

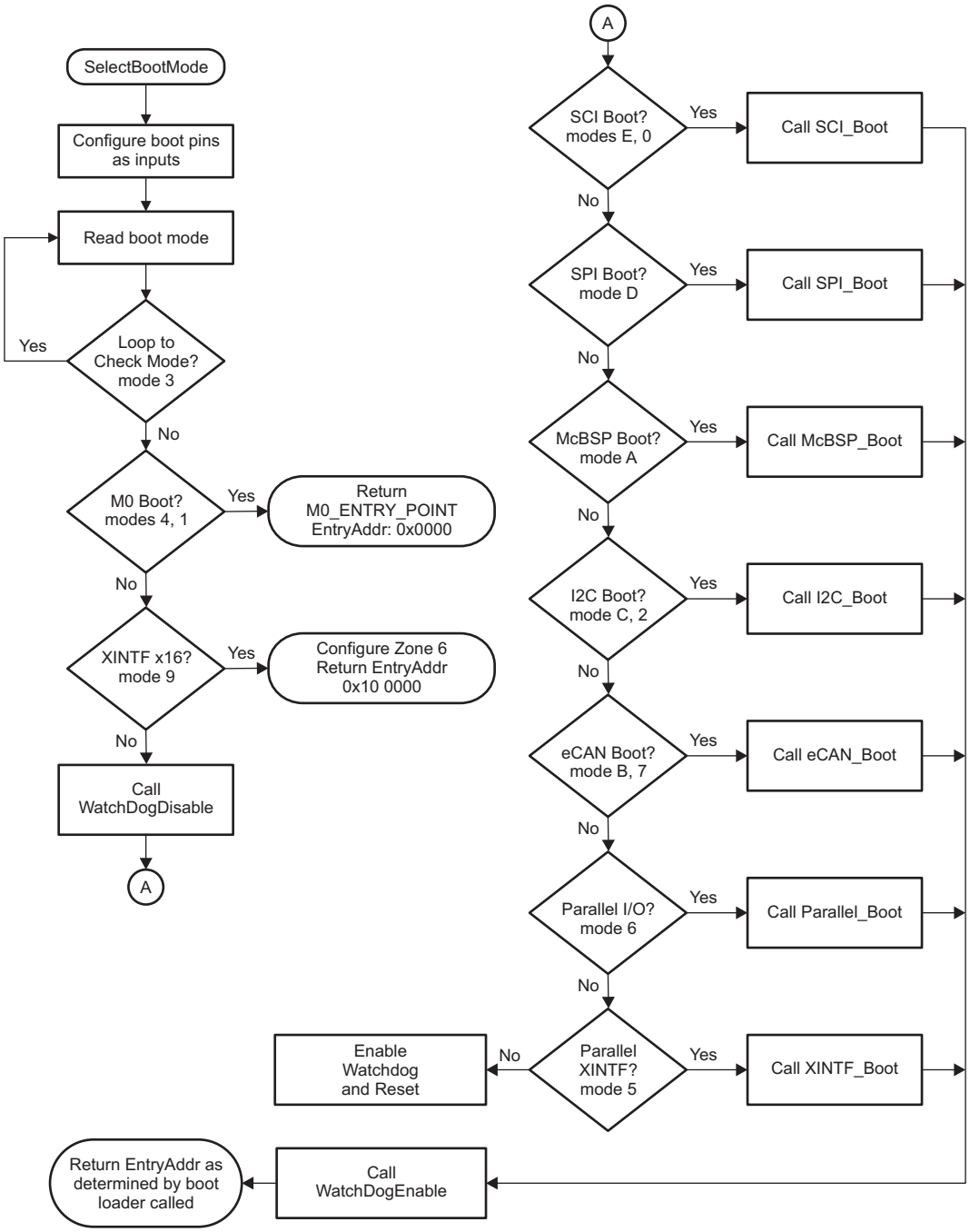
For a boot mode to be selected, the pins corresponding to the desired boot mode have to be pulled low or high until the selection process completes. Note that the state of the selection pins is not latched at reset; they are sampled some cycles later in the SelectBootMode function. The internal pullup resistors are enabled at reset for the boot mode selection pins. It is still suggested that the boot mode configuration be made externally to avoid the effect of any noise on these pins.

NOTE: The SelectBootMode routine disables the watchdog before calling the SCI, I2C , eCAN, SPI , McBSP, or parallel bootloaders. The bootloaders do not service the watchdog and assume that it is disabled. Before exiting, the SelectBootMode routine will re-enable the watchdog and reset its timer.

If a bootloader is not going to be called, then the watchdog is left untouched.

When selecting a boot mode, the pins should be pulled high or low through a weak pulldown or weak pull-up such that the device can drive them to a new state when required.

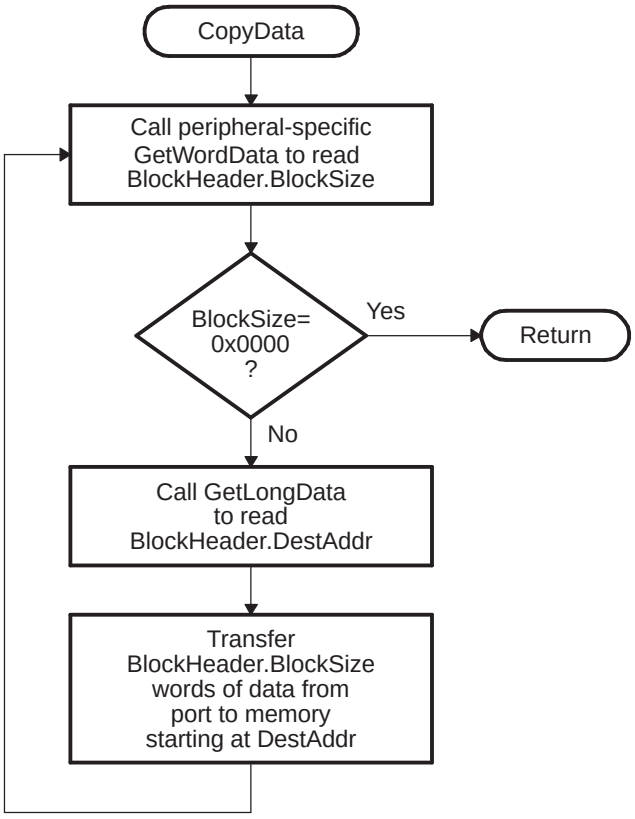
Figure 10. Overview of the SelectBootMode Function



2.14 CopyData Function

Each of the bootloaders uses the same function to copy data from the port to the DSP SARAM. This function is the CopyData() function. This function uses a pointer to a GetWordData function that is initialized by each of the loaders to properly read data from that port. For example, when the SPI loader is evoked, the GetWordData function pointer is initialized to point to the SPI-specific SPI_GetWordData function. Thus when the CopyData() function is called, the correct port is accessed. The flow of the CopyData function is shown in Figure 11.

Figure 11. Overview of CopyData Function



2.15 McBSP_Boot Function

The McBSP bootloader synchronously transfers code from McBSP-A to internal memory. McBSP-A is configured for slave mode operation. i.e., it receives the frame sync and clock from the host. Upon receiving a word, the McBSP echoes the data back to the host. The host could use this feature to ensure that the previous word was received and copied by the McBSP before transmitting the next word. The host can download a kernel to reconfigure the McBSP if higher data throughput is desired. This can be done by choosing a faster PLL multiplier and also by choosing the /1 divider for the PLL output.

The McBSP-A loader uses pins shown in Table 6.

Table 6. Pins Used by the McBSP Loader

C28x Slave	Device Pin Number	Host Signal
MDXA	GPIO20	MDR
MDRA	GPIO21	MDX
MCLKXA	GPIO22	CLKX
MFSXA	GPIO23	FSR
MCLKRA	GPIO7	CLKX
MFSXA	GPIO5	FSXA

The bit rates achieved for different XCLKIN values as shown in Table 7. The SYSCLKOUT values shown are for the default PLLCR of 0 and PLLSTS[DIVSEL] set to 2.

Table 7. Bit-Rate Values for Different XCLKIN Values

XCLKIN	SYSCLKOUT	LSPCLK	CLKG
30 MHz	15 MHz	3.75 MHz	1.875 MHz
15 MHz	7.5 MHz	1.875 MHz	937.5 KHz

The host should transmit MSB first and LSB next. For example, to transmit the word 0x10AA to the device, transmit 10 first, followed by AA. The program flow of the McBSP bootloader is identical to the SCI bootloader, with the exception that 16-bit data is used. The data sequence for the McBSP bootloader follows the 16-bit data stream and is shown in Table 8

Table 8. McBSP 16-Bit Data Stream

Word	Contents	Description
1	10AA	10AA (KeyValue for memory width = 16bits)
2	0000	8 reserved words (words 2-9)
...	...	...
9	0000	Last reserved word
10	AABB	Entry point PC[22:16]
11	CCDD	Entry point PC[15:0] (PC = 0xAABBCCDD)
12	MMNN	Block size (number of words) of the first block of data to load = 0xMMNN words
13	AABB	Destination address of first block Addr[31:16]
14	CCDD	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
15	XXXX	First word of the first block in the source being loaded
...		...
...		... Data for this section.
.	XXXX	Last word of the first block of the source being loaded
.	MMNN	Block size of the 2nd block to load = 0xMMNN words
.	AABB	Destination address of second block Addr[31:16]
.	CCDD	Destination address of second block Addr[15:0]
.	XXXX	First word of the second block in the source being loaded
.		…

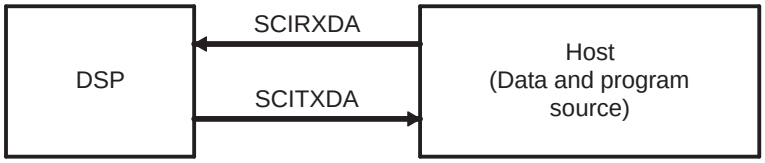
Table 8. McBSP 16-Bit Data Stream (continued)

Word	Contents	Description
n	XXXX	Last word of the last block of the source being loaded
n+1	0000	Block size of 0000h - indicates end of the source program

2.16 SCI_Boot Function

The SCI boot mode asynchronously transfers code from SCI-A to internal memory. This boot mode only supports an incoming 8-bit data stream and follows the same data flow as outlined in Example 4.

Figure 12. Overview of SCI Bootloader Operation



The SCI-A loader uses following pins:

- SCIRXDA on GPIO28
- SCITXDA on GPIO29

The 28x device communicates with the external host device by communication through the SCI-A Peripheral. The autobaud feature of the SCI port is used to lock baud rates with the host. For this reason the SCI loader is very flexible and you can use a number of different baud rates to communicate with the device.

After each data transfer, the 28x will echo back the 8-bit character received to the host. In this manner, the host can perform checks that each character was received by the 28x.

At higher baud rates, the slew rate of the incoming data bits can be effected by transceiver and connector performance. While normal serial communications may work well, this slew rate may limit reliable auto-baud detection at higher baud rates (typically beyond 100kbaud) and cause the auto-baud lock feature to fail. To avoid this, the following is recommended:

1. Achieve a baud-lock between the host and 28x SCI bootloader using a lower baud rate.
2. Load the incoming 28x application or custom loader at this lower baud rate.
3. The host may then handshake with the loaded 28x application to set the SCI baud rate register to the desired high baud rate.

Figure 13. Overview of SCI_Boot Function

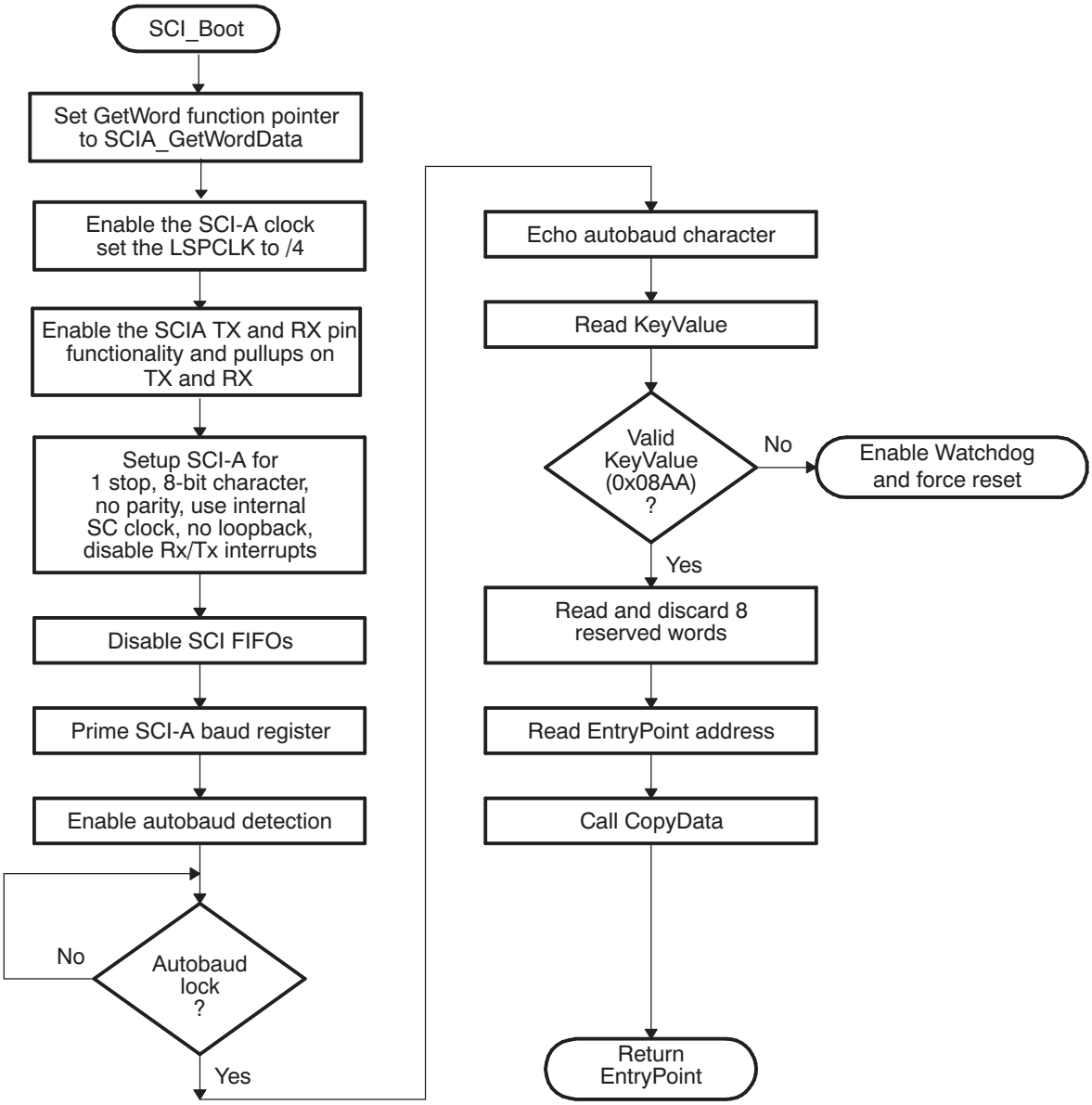
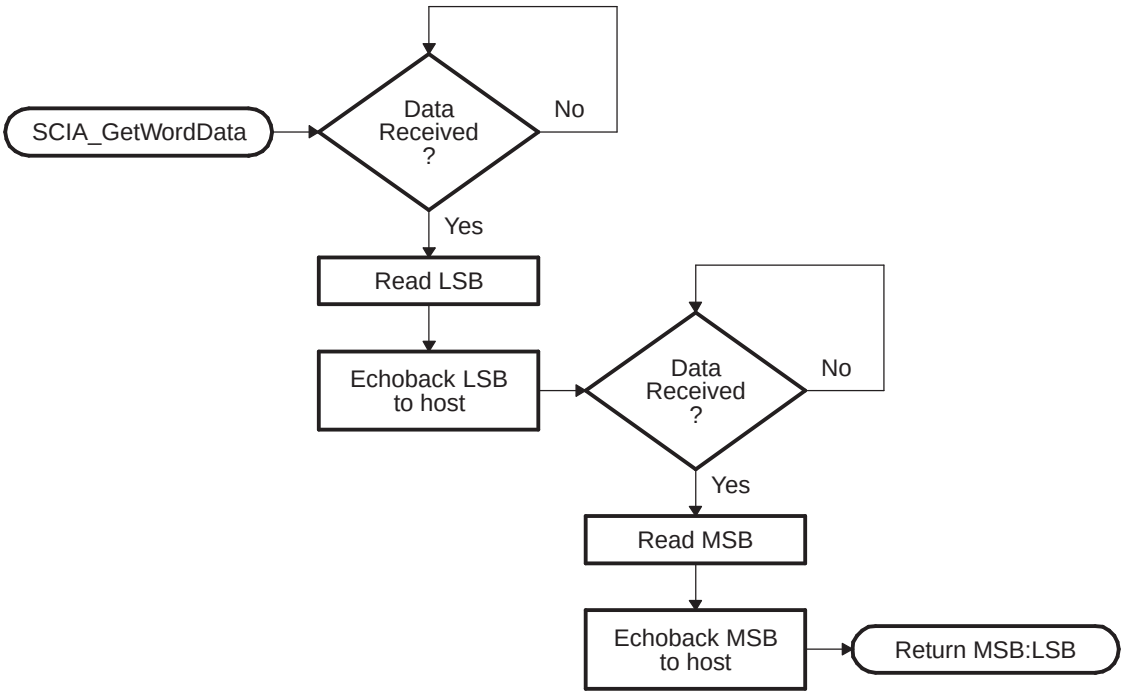


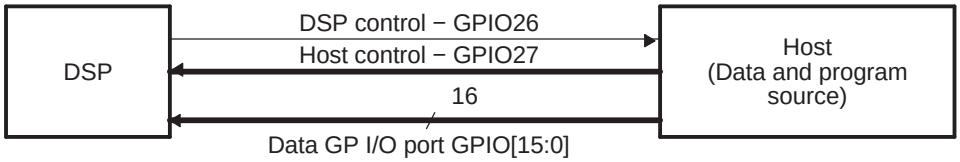
Figure 14. Overview of SCI_GetWordData Function



2.17 Parallel_Boot Function (GPIO)

The parallel general purpose I/O (GPIO) boot mode asynchronously transfers code from GPIO0 -GPIO15 to internal memory. Each value can be 16 bits or 8 bits long and follows the same data flow as outlined in Section 2.10.

Figure 15. Overview of Parallel GPIO bootloader Operation



The parallel GPIO loader uses following pins:

- Data on GPIO[15:0] or
- 28x Control on GPIO26
- Host Control on GPIO27

The 28x communicates with the external host device by polling/driving the GPIO27 and GPIO26 lines. The handshake protocol shown in Figure 16 must be used to successfully transfer each word via GPIO[15 :0]. This protocol is very robust and allows for a slower or faster host to communicate with the DSP .

If the 8-bit mode is selected, t wo consecutive 8-bit words are read to form a single 16-bit word. The most significant byte (MSB) is read first followed by the least significant byte (LSB). In this case, data is read from the lower eight lines of GPIO[7:0] ignoring the higher byte .

The 16-bit data stream is shown in Table 9 and the 8-bit data stream is shown in Table 10.

Table 9. Parallel GPIO Boot 16-Bit Data Stream

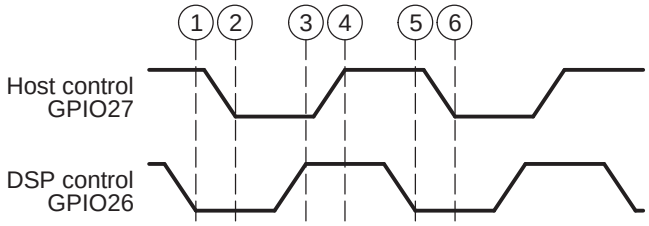
Word	GPIO[15:0]	Description
1	10AA	10AA (KeyValue for memory width = 16bits)
2	0000	8 reserved words (words 2 - 9)
...	...	...
9	0000	Last reserved word
10	AABB	Entry point PC[22:16]
11	CCDD	Entry point PC[15:0] (PC = 0xAABBCCDD)
12	MMNN	Block size (number of words) of the first block of data to load = 0xMMNN words
13	AABB	Destination address of first block Addr[31:16]
14	CCDD	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
15	XXXX	First word of the first block in the source being loaded
...		...
...		Data for this section.
		...
.	XXXX	Last word of the first block of the source being loaded
.	MMNN	Block size of the 2nd block to load = 0xMMNN words
.	AABB	Destination address of second block Addr[31:16]
.	CCDD	Destination address of second block Addr[15:0]
.	XXXX	First word of the second block in the source being loaded
.		…
n	XXXX	Last word of the last block of the source being loaded (More sections if required)
n+1	0000	Block size of 0000h - indicates end of the source program

Table 10. Parallel GPIO Boot 8-Bit Data Stream

Bytes		GPIO[7 :0] (Byte 1 of 2)	GPIO[7 :0] (Byte 2 of 2)	Description
1	2	AA	08	0x08AA (KeyValue for memory width = 16bits)
3	4	00	00	8 reserved words (words 2 - 9)
...	...	...	...	...
17	18	00	00	Last reserved word
19	20	BB	00	Entry point PC[22:16]
21	22	DD	CC	Entry point PC[15:0] (PC = 0x00BBCCDD)
23	24	NN	MM	Block size of the first block of data to load = 0xMMNN words
25	26	BB	AA	Destination address of first block Addr[31:16]
27	28	DD	CC	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
29	30	BB	AA	First word of the first block in the source being loaded = 0xAABB
...				...
...				Data for this section.
				...
.		BB	AA	Last word of the first block of the source being loaded = 0xAABB
.		NN	MM	Block size of the 2nd block to load = 0xMMNN words
.		BB	AA	Destination address of second block Addr[31:16]
.		DD	CC	Destination address of second block Addr[15:0]
.		BB	AA	First word of the second block in the source being loaded
.				…
n	n+1	BB	AA	Last word of the last block of the source being loaded (More sections if required)
n+2	n+3	00	00	Block size of 0000h - indicates end of the source program

The 28x device first signals the host that it is ready to begin data transfer by pulling the GPIO26 pin low. The host load then initiates the data transfer by pulling the GPIO27 pin low. The complete protocol is shown in the diagram below:

Figure 16. Parallel GPIO Boot Loader Handshake Protocol



1. The 28x device indicates it is ready to start receiving data by pulling the GPIO26 pin low.
2. The bootloader waits until the host puts data on GPIO[15 :0]. The host signals to the 28x device that data is ready by pulling the GPIO27 pin low.
3. The 28x device reads the data and signals the host that the read is complete by pulling GPIO26 high.
4. The bootloader waits until the host acknowledges the 28x by pulling GPIO27 high.
5. The 28x device again indicates it is ready for more data by pulling the GPIO26 pin low.

This process is repeated for each data value to be sent.

Figure 17 shows an overview of the Parallel GPIO bootloader flow.

Figure 17. Parallel GPIO Mode Overview

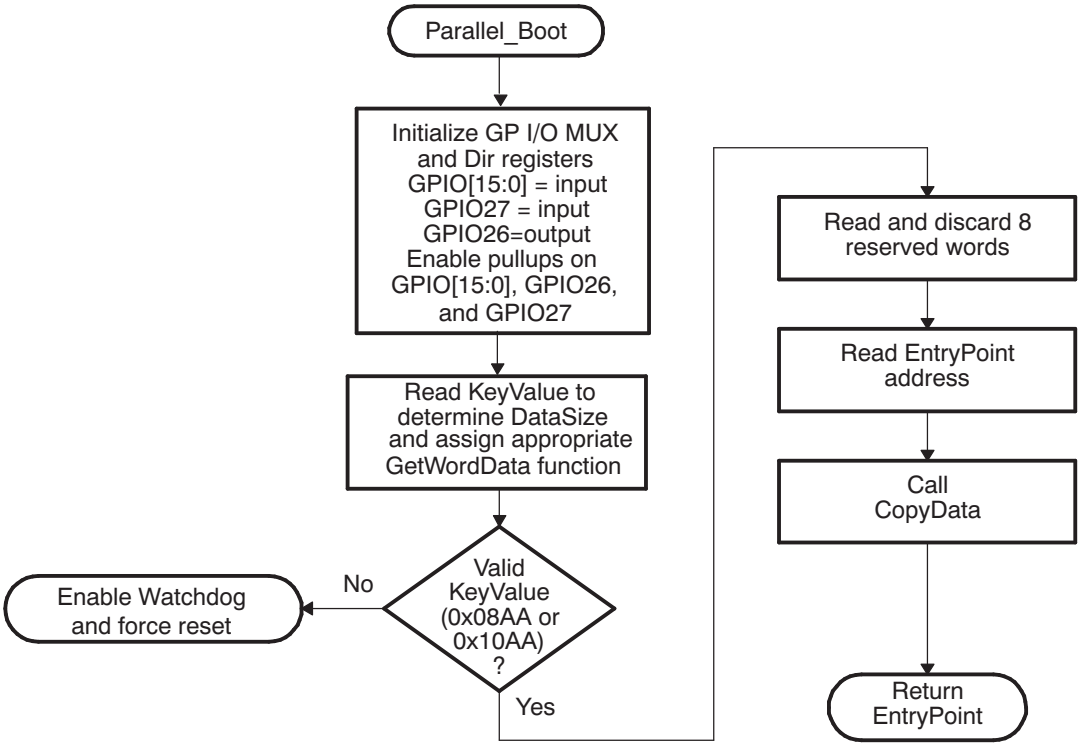


Figure 18 shows the transfer flow from the host side. The operating speed of the CPU and host are not critical in this mode as the host will wait for the 28x and the 28x will in turn wait for the host. In this manner the protocol will work with both a host running faster and a host running slower then the 28x.

Figure 18. Parallel GPIO Mode - Host Transfer Flow

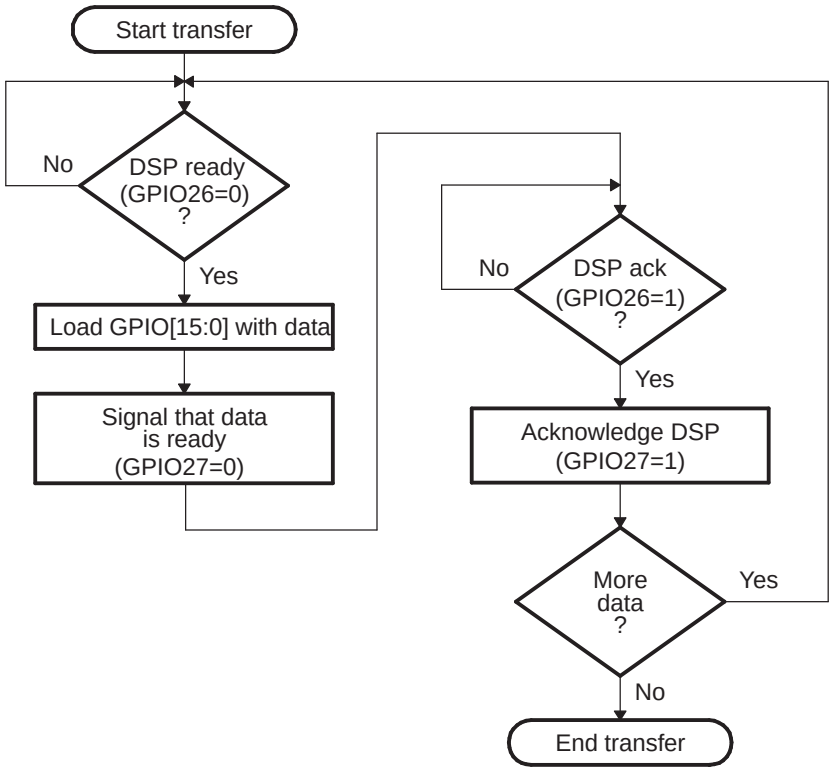


Figure 19 and Figure 20 show the flow used to read a single word of data from the parallel port. The loader uses the method shown in Figure 8 to read the key value and to determine if the incoming data stream width is 8-bit or 16-bit. A different GetWordData function is used by the parallel loader depending on the data size of the incoming data stream.

- **16-bit data stream**

  For an 16-bit data stream, the function Parallel_GetWordData16bit is used. This function reads all 16-bits at a time. The flow of this function is shown in Figure 19.

- **8-bit data stream**

  The 8-bit routine, shown in Figure 20, discards the upper 8 bits of the first read from the port and treats the lower 8 bits as the least significant byte (LSB) of the word to be fetched. The routine will then perform a second read to fetch the most significant byte (MSB). It then combines the MSB and LSB into a single 16-bit value to be passed back to the calling routine.

Figure 19. 16-Bit Parallel GetWord Function

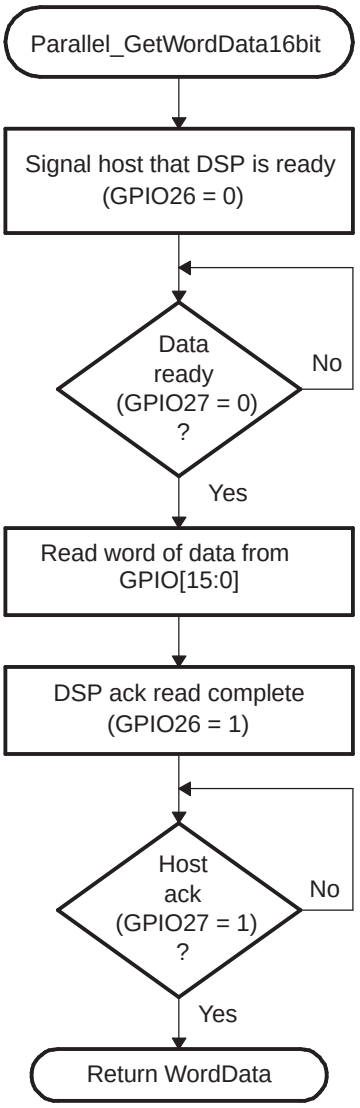
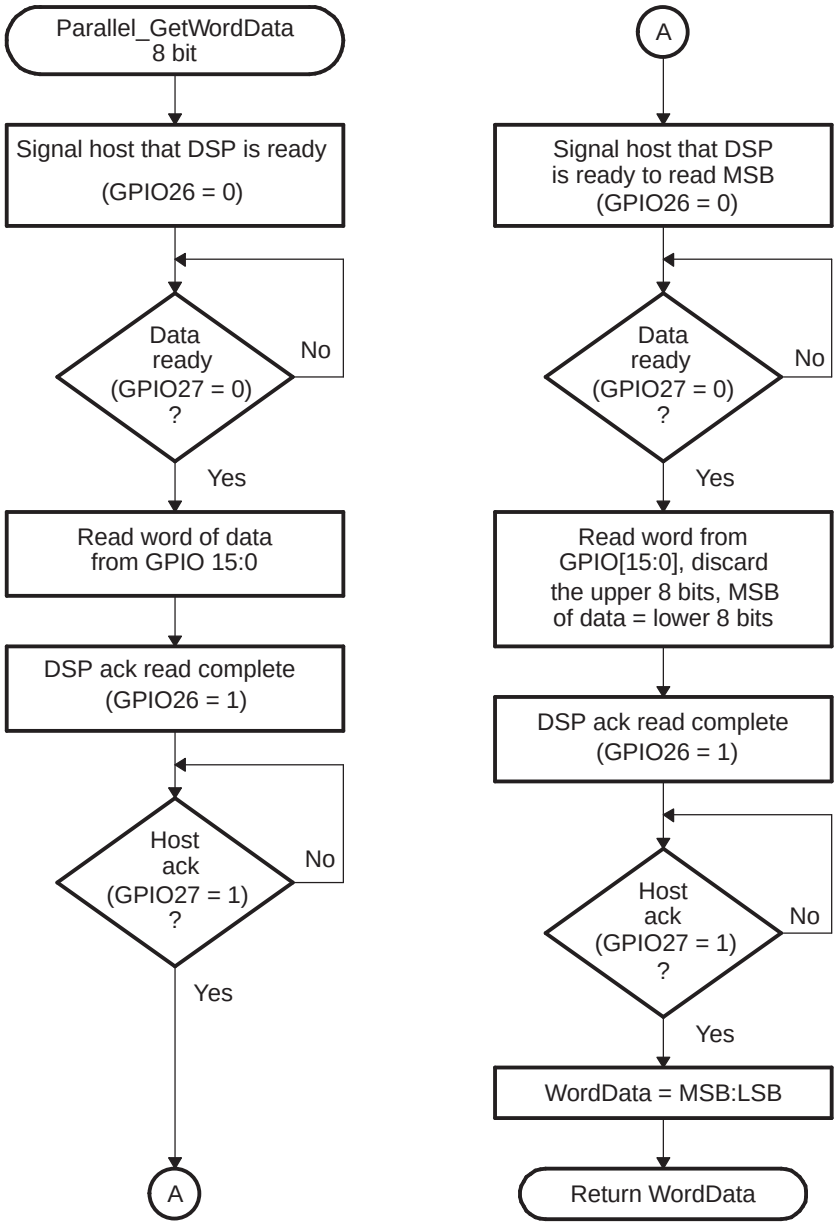


Figure 20. 8-Bit Parallel GetWord Function



2.18 XINTF_Parallel_Boot Function

The parallel general purpose I/O (GPIO) boot mode asynchronously transfers code from XD[15:0] to internal memory. Each value can be 16 bits or 8 bits long and follows the same data flow as outlined in Section 2.10. The each word or byte of data is read from address 0x100000 in XINTF zone 6.

NOTE: This mode loads a stream of data into the SARAM of the device using XINTF resources. If you instead want to configure and jump to the XINTF then use the "Jump to XINTF x16" boot mode.

The parallel XINTF loader uses following pins:

- Data on XD[15:0] or XD[7:0]
- 28x Control on GPIO13
- Host Control on GPIO12

The 28x communicates with the external host device by polling/driving the GPIO13 and GPIO12 lines. The handshake protocol shown in Figure 16 must be used to successfully transfer each word via XD[15:0]. This protocol is very robust and allows for a slower or faster host to communicate with the DSP.

If the 8-bit mode is selected, two consecutive 8-bit words are read to form a single 16-bit word. The most significant byte (MSB) is read first followed by the least significant byte (LSB). In this case, data is read from the lower eight lines of XD[7:0] ignoring the higher byte.

The DSP first signals the host that the DSP is ready to begin data transfer by pulling the GPIO12 pin low. The host load then initiates the data transfer by pulling the GPIO13 pin low. The complete protocol is shown in Figure 21.

Figure 21. Overview of the Parallel XINTF Boot Loader Operation

The DSP communicates with the external host device by polling/driving the GPIO13 and GPIO12 lines. The handshake protocol shown below must be used to successfully transfer each word via the first address location within XINTF zone 6. This protocol is very robust and allows for a slower or faster host to communicate with the DSP.

If the 8-bit mode is selected, two consecutive 8-bit words are read to form a single 16-bit word. The most significant byte (MSB) is read first followed by the least significant byte (LSB). In this case, data is read from the lower eight lines of XD[7:0] ignoring the higher byte.

To begin the transfer, the DSP will use the default XINTF timing for zone 6. This is the maximum wait states, slowest XINTF timing available. That is:

1. XTIMCLK = ½ SYSCLKOUT
2. XCLKOUT = 1/4 XTIMCLK
3. XRDLEAD = XWRLEAD = 3
4. XRDACTIVE = XWRACTIVE = 7
5. XRDTRAIL = XWRACTIVE = 3
6. XSIZE = 3 for 16-bit wide
7. X2TIMING = 1. Timing values are 2:1.
8. USEREADY = 1, READYMODE = 1 (XREADY sampled asynchronous mode)

The first 7 words of the data stream are read at this slow timing. Words 2 – 7 include configuration information that will be used to adjust the PLLCR/PLLSTS and XINTF XTIMING6. The rest of the data stream is read using the new configuration.

The 16-bit data stream is shown in Table 11 and the 8-bit data stream is shown in Table 12.

Table 11. XINTF Parallel Boot 16-Bit Data Stream

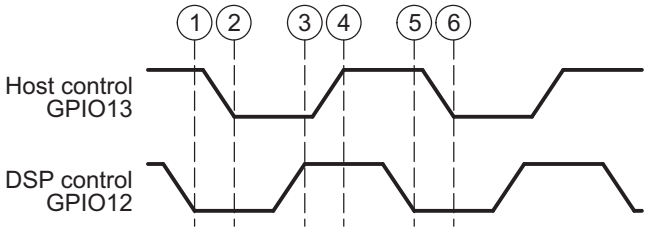
Word	XD[15:0]	Description
1	10AA	10AA (KeyValue for memory width = 16bits)
2	AABB	PLLCR register = 0xAABB
3	000B	PLLSTS[DIVSEL] bits = 0xB
4	AABB	XTIMING6[31:16]
5	CCDD	XTIMING6[15:0] (XTIMING6 = 0xAABBCCDD)
6	EEFF	XINTCNF2[31:16]
7	GGHH	XINTCNF2[15:0] (XINTCNF2 = 0xEEFFGGHH)
8	0000	reserved
9	0000	reserved
10	AABB	Entry point PC[22:16]
11	CCDD	Entry point PC[15:0] (PC = 0xAABBCCDD)
12	MMNN	Block size (number of words) of the first block of data to load = 0xMMNN words
13	AABB	Destination address of first block Addr[31:16]
14	CCDD	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
15	XXXX	First word of the first block in the source being loaded
...		...
...		Data for this section.
		...
.	XXXX	Last word of the first block of the source being loaded
.	MMNN	Block size of the 2nd block to load = 0xMMNN words
.	AABB	Destination address of second block Addr[31:16]
.	CCDD	Destination address of second block Addr[15:0]
.	XXXX	First word of the second block in the source being loaded
.		…
n	XXXX	Last word of the last block of the source being loaded (More sections if required)
n+1	0000	Block size of 0000h - indicates end of the source program

Table 12. XINTF Parallel Boot 8-Bit Data Stream

Bytes		XD[7:0] (Byte 1 of 2)	XD[7:0] (Byte 2 of 2)	Description
1	2	AA	08	0x08AA (KeyValue for memory width = 16bits)
3	4	BB	AA	PLLCR register = 0xAABB
5	6	0B	00	PLLSTS[DIVSEL] bits = 0xB
7	8	BB	AA	XTIMING6[31:16]
9	10	DD	CC	XTIMING6[15:0] (XTIMING6 = 0xAABBCCDD)
11	12	FF	EE	XINTCNF2[31:16]
13	14	HH	GG	XINTCNF2[15:0] (XINTCNF2 = 0xEEFFGGHH)
15	16	00	00	reserved
17	18	00	00	reserved
19	20	BB	00	Entry point PC[22:16]
21	22	DD	CC	Entry point PC[15:0] (PC = 0x00BBCCDD)
23	24	NN	MM	Block size of the first block of data to load = 0xMMNN words
25	26	BB	AA	Destination address of first block Addr[31:16]
27	28	DD	CC	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
29	30	BB	AA	First word of the first block in the source being loaded = 0xAABB
...				...
...				Data for this section.
				...
.		BB	AA	Last word of the first block of the source being loaded = 0xAABB
.		NN	MM	Block size of the 2nd block to load = 0xMMNN words
.		BB	AA	Destination address of second block Addr[31:16]
.		DD	CC	Destination address of second block Addr[15:0]
.		BB	AA	First word of the second block in the source being loaded
.				…
n	n+1	BB	AA	Last word of the last block of the source being loaded (More sections if required)
n+2	n+3	00	00	Block size of 0000h - indicates end of the source program

Figure 22 shows an overview of the Parallel XINTF bootloader flow.

Figure 22. XINTF_Parallel Boot Loader Handshake Protocol



1. The 28x device indicates it is ready to start receiving data by pulling the GPIO12 pin low.
2. The bootloader waits until the host puts data on XD[15:0]. The host signals to the 28x device that data is ready by pulling the GPIO13 pin low.
3. The 28x device reads the data and signals the host that the read is complete by pulling GPIO12 high.
4. The bootloader waits until the host acknowledges the 28x by pulling GPIO13 high.
5. The 28x device again indicates it is ready for more data by pulling the GPIO12 pin low.

This process is repeated for each data value to be sent.

Figure 17 shows an overview of the XINTF Parallel bootloader flow.

Figure 23. XINTF Parallel Mode Overview

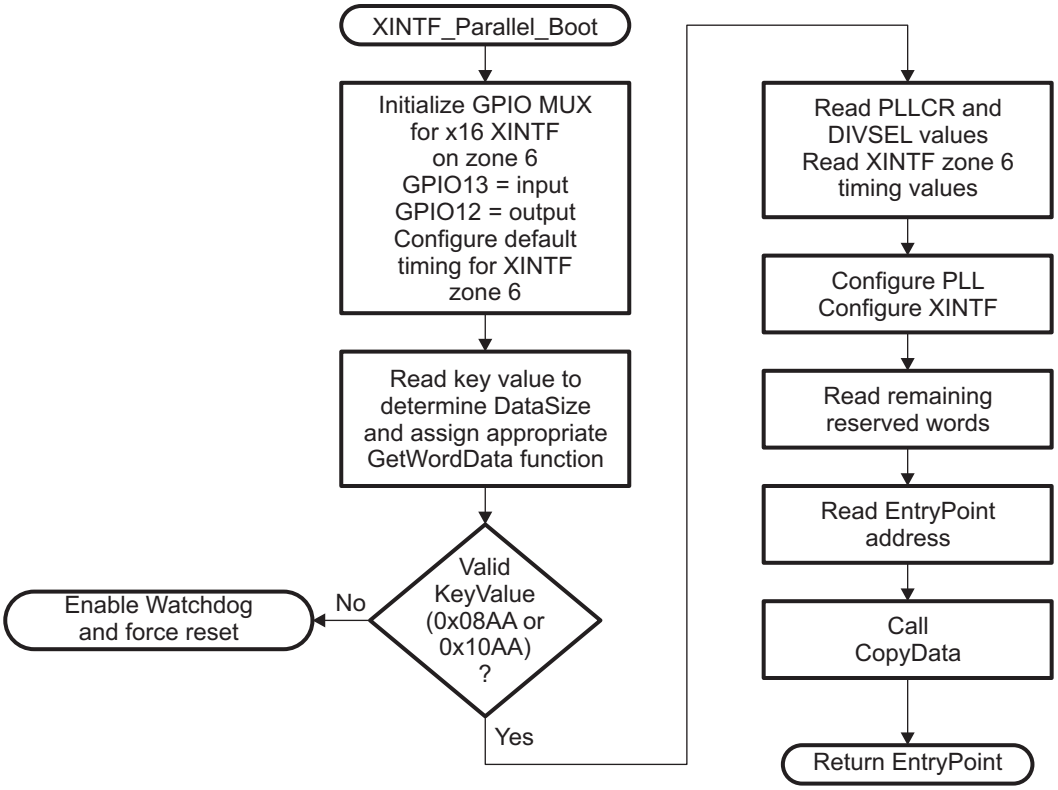


Figure 18 shows the transfer flow from the host side. The operating speed of the CPU and host are not critical in this mode as the host will wait for the 28x and the 28x will in turn wait for the host. In this manner the protocol will work with both a host running faster and a host running slower then the 28x device.

Figure 24. XINTF Parallel Mode - Host Transfer Flow

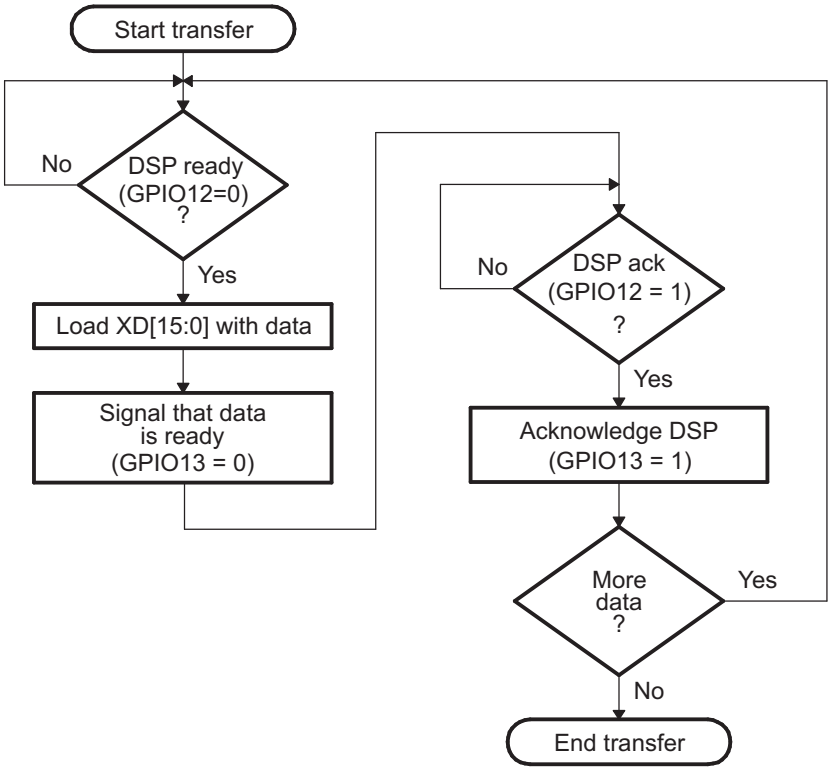


Figure 19 and Figure 20 show the flow used to read a single word of data from the parallel port. The loader uses the method shown in Figure 8 to read the key value and to determine if the incoming data stream width is 8-bit or 16-bit. A different GetWordData function is used by the parallel loader depending on the data size of the incoming data stream.

- **16-bit data stream**

  For an 16-bit data stream, the function XINTF_Parallel_GetWordData16bit is used. This function reads all 16-bits at a time. The flow of this function is shown in Figure 19.

- **8-bit data stream**

  For an 8-bit data stream, the function XINTF_Parallel_GetWordData8bit is used. The 8-bit routine, shown in Figure 20, discards the upper 8 bits of the first read from the port and treats the lower 8 bits as the least significant byte (LSB) of the word to be fetched. The routine will then perform a second read to fetch the most significant byte (MSB). It then combines the MSB and LSB into a single 16-bit value to be passed back to the calling routine.

Figure 25. 16-Bit Parallel GetWord Function

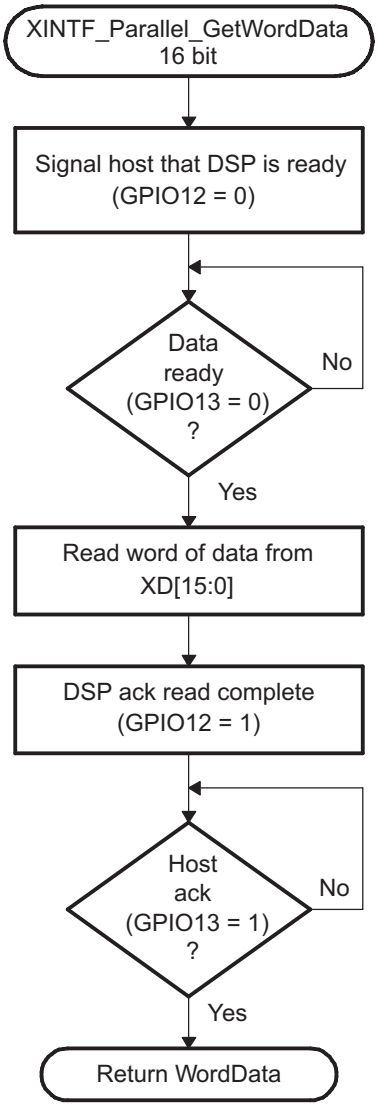
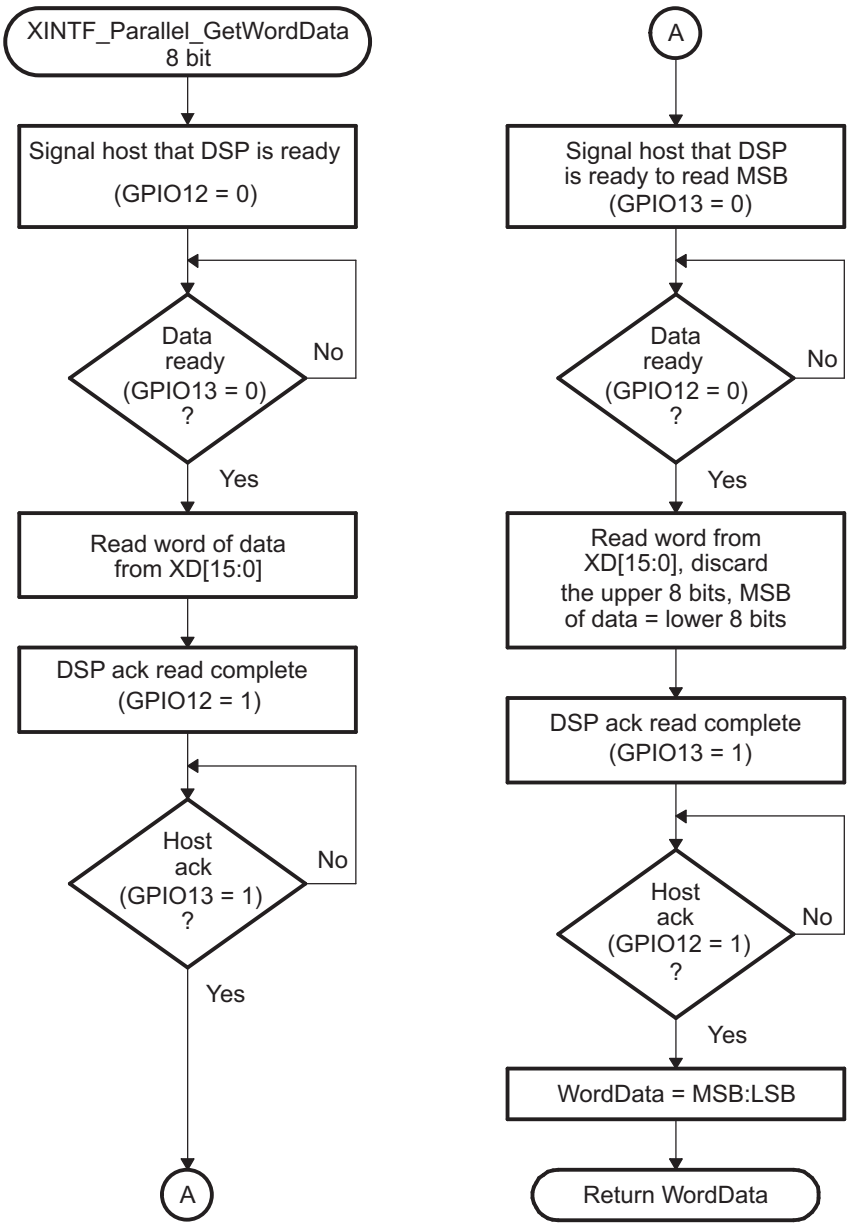


Figure 26. 8-Bit Parallel GetWord Function



2.19 SPI_Boot Function

The SPI loader expects an SPI-compatible 16-bit or 24-bit addressable serial EEPROM or serial flash device to be present on the SPI-A pins as indicated in Figure 27. The SPI bootloader supports an 8-bit data stream. It does not support a 16-bit data stream.

Figure 27. SPI Loader

The SPI-A loader uses following pins:

- SPISIMOA on GPIO16
- SPISOMIA on GPIO17
- SPICLKA on GPIO18
- SPISTEA on GPIO19

The SPI boot ROM loader initializes the SPI module to interface to a serial SPI EEPROM or flash. Devices of this type include, but are not limited to, the Xicor X25320 (4Kx8) and Xicor X25256 (32Kx8) SPI serial SPI EEPROMs and the Atmel AT25F1024A serial flash.

The SPI boot ROM loader initializes the SPI with the following settings: FIFO enabled, 8-bit character, internal SPICLK master mode and talk mode, clock phase = 1, polarity = 0, using the slowest baud rate.

If the download is to be performed from an SPI port on another device, then that device must be setup to operate in the slave mode and mimic a serial SPI EEPROM. Immediately after entering the SPI_Boot function, the pin functions for the SPI pins are set to primary and the SPI is initialized. The initialization is done at the slowest speed possible. Once the SPI is initialized and the key value read, you could specify a change in baud rate or low speed peripheral clock.

Table 13. SPI 8-Bit Data Stream

Byte	Contents
1	LSB: AA (KeyValue for memory width = 8-bits)
2	MSB: 08h (KeyValue for memory width = 8-bits)
3	LSB: LOSPCP
4	MSB: SPIBRR
5	LSB: reserved for future use
6	MSB: reserved for future use
...	...
...	Data for this section. ...
17	LSB: reserved for future use
18	MSB: reserved for future use
19	LSB: Upper half (MSW) of Entry point PC[23:16]
20	MSB: Upper half (MSW) of Entry point PC[31:24] (Note: Always 0x00)
21	LSB: Lower half (LSW) of Entry point PC[7:0]
22	MSB: Lower half (LSW) of Entry point PC[15:8]
...	....
...	Data for this section. ...
...	Blocks of data in the format size/destination address/data as shown in the generic data stream description

Table 13. SPI 8-Bit Data Stream (continued)

Byte	Contents
...	...
...	Data for this section.
	...
n	LSB: 00h
n+1	MSB: 00h - indicates the end of the source

The data transfer is done in "burst" mode from the serial SPI EEPROM. The transfer is carried out entirely in byte mode (SPI at 8 bits/character). A step-by-step description of the sequence follows:

Step 1. The SPI-A port is initialized

Step 2. The GPIO19 (SPISTE) pin is used as a chip-select for the serial SPI EEPROM or flash

Step 3. The SPI-A outputs a read command for the serial SPI EEPROM or flash

Step 4. The SPI-A sends the serial SPI EEPROM an address 0x0000; that is, the host requires that the EEPROM or flash must have the downloadable packet starting at address 0x0000 in the EEPROM or flash. The loader is compatible with both 16-bit addresses and 24-bit addresses.

Step 5. The next word fetched must match the key value for an 8-bit data stream (0x08AA). The least significant byte of this word is the byte read first and the most significant byte is the next byte fetched. This is true of all word transfers on the SPI. If the key value does not match, then the load is aborted and boot loader will enable the watchdog and force a device reset through software.

Step 6. The next 2 bytes fetched can be used to change the value of the low speed peripheral clock register (LOSPCP) and the SPI baud rate register (SPIBRR). The first byte read is the LOSPCP value and the second byte read is the SPIBRR value. The next 7 words are reserved for future enhancements. The SPI bootloader reads these 7 words and discards them.

Step 7. The next 2 words makeup the 32-bit entry point address where execution will continue after the boot load process is complete. This is typically the entry point for the program being downloaded through the SPI port.

Step 8. Multiple blocks of code and data are then copied into memory from the external serial SPI EEPROM through the SPI port. The blocks of code are organized in the standard data stream structure presented earlier. This is done until a block size of 0x0000 is encountered. At that point in time the entry point address is returned to the calling routine that then exits the bootloader and resumes execution at the address specified.

Figure 28. Data Transfer From EEPROM Flow

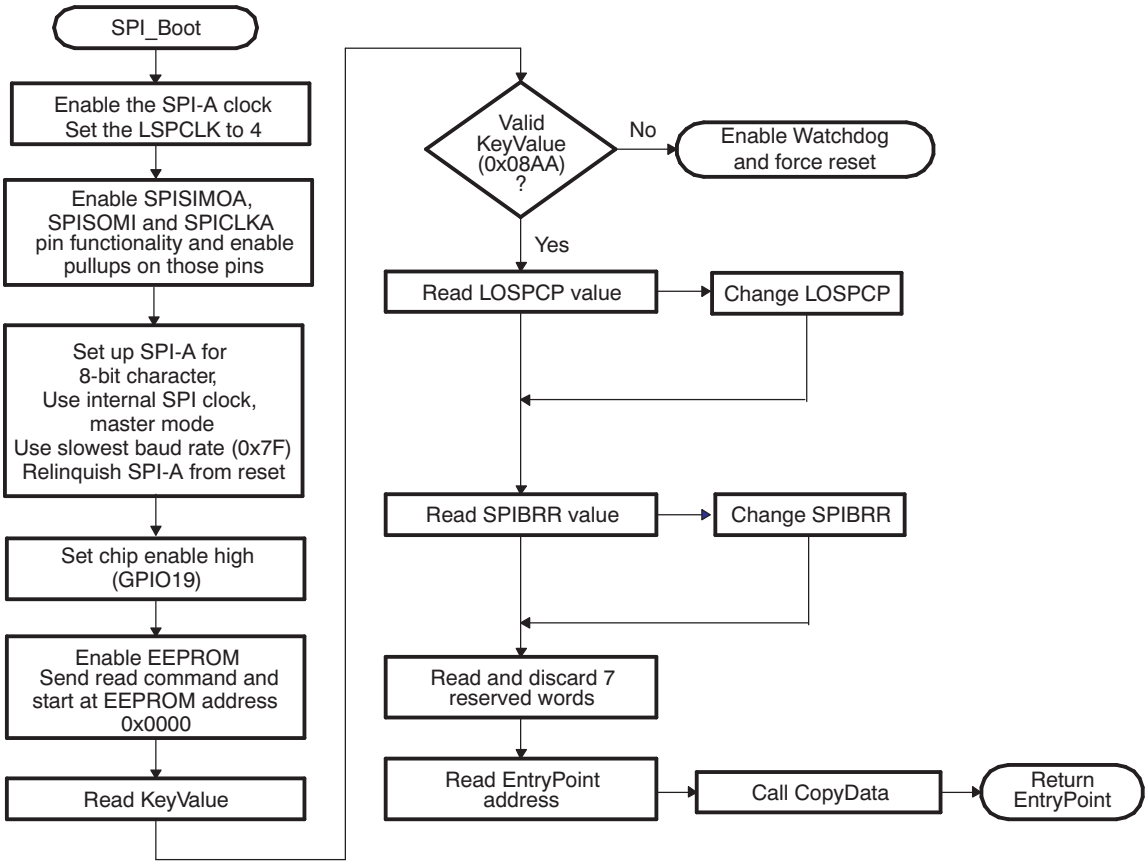
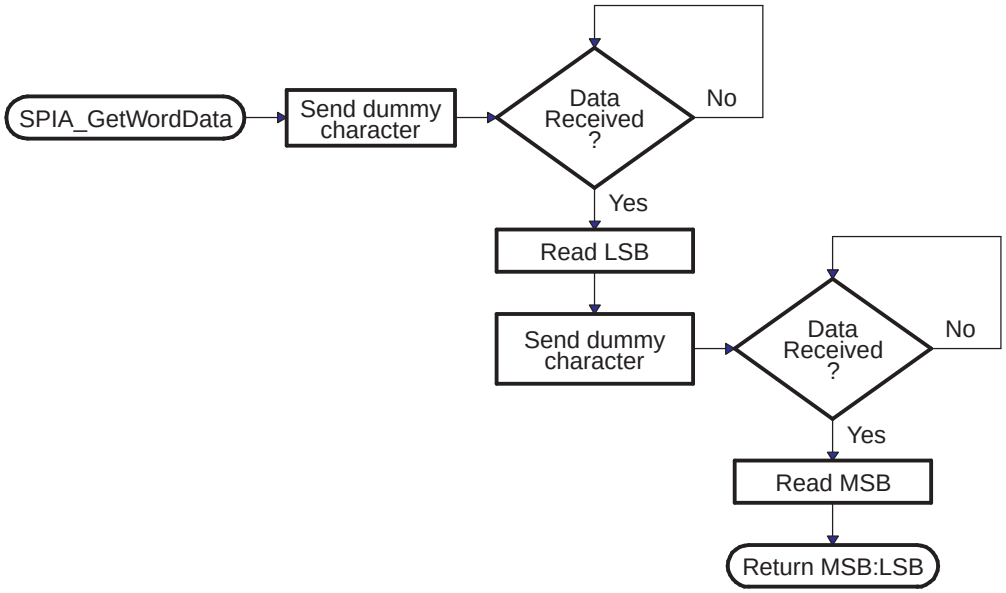


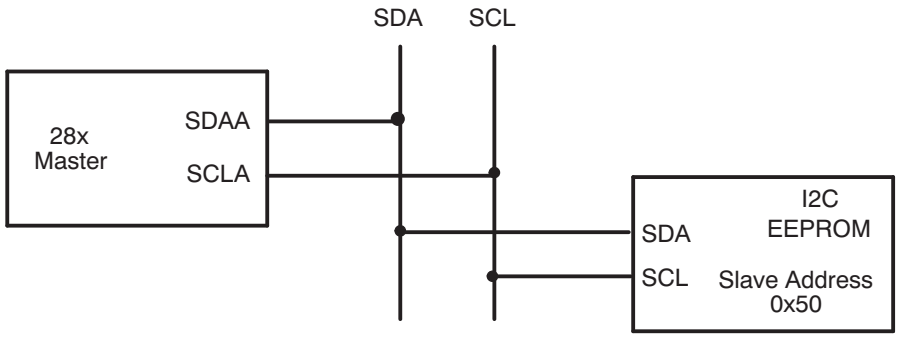
Figure 29. Overview of SPIA_GetWordData Function



2.20 I2C Boot Function

The I2C bootloader expects an 8-bit wide I2C-compatible EEPROM device to be present at address 0x50 on the I2C-A bus as indicated in Figure 30. The EEPROM must adhere to conventional I2C EEPROM protocol, as described in this section, with a 16-bit base address architecture.

Figure 30. EEPROM Device at Address 0x50



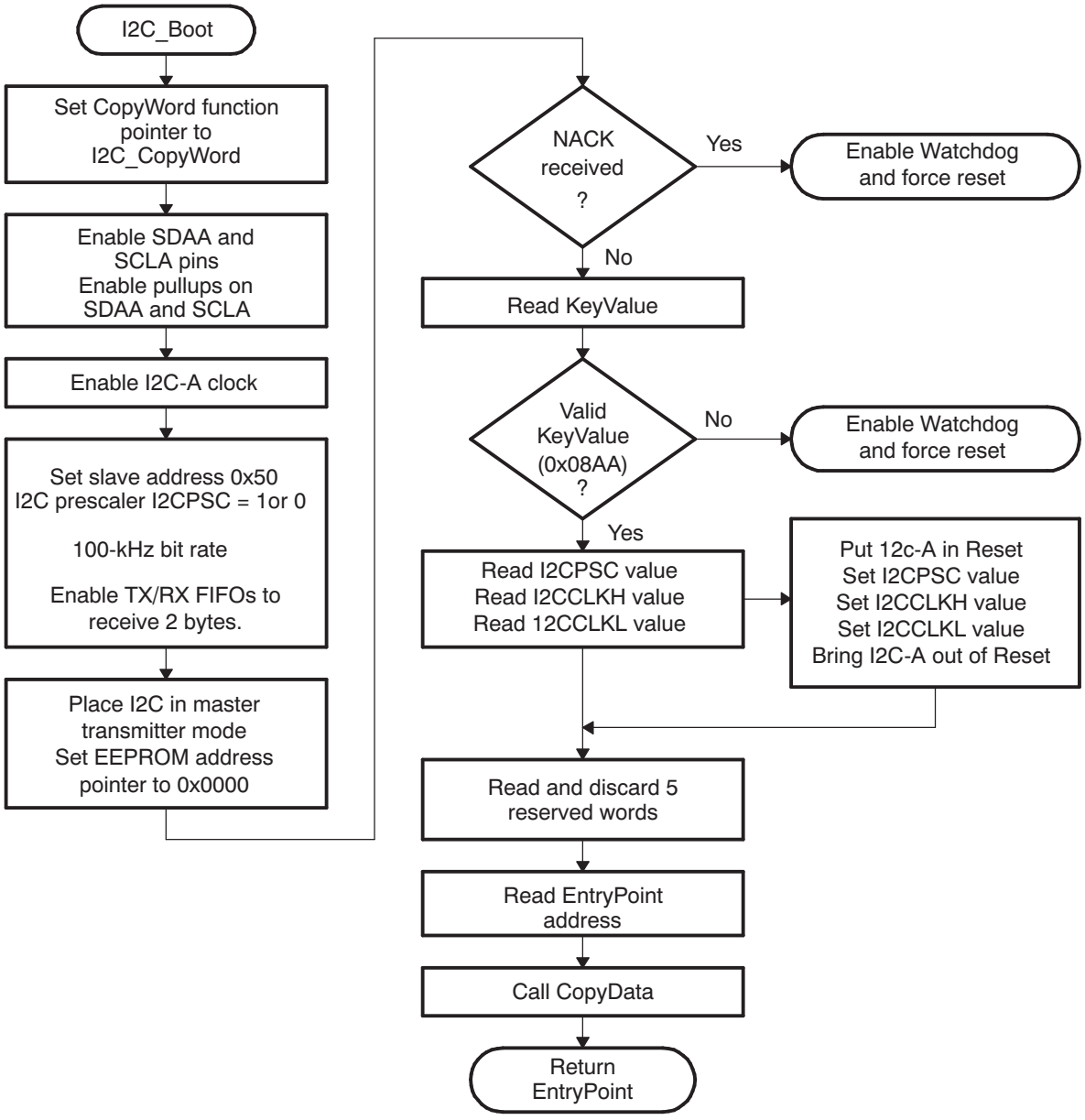
The I2C loader uses following pins:

- SDAA on GPIO 32
- SCLA on GPIO 33

If the download is to be performed from a device other than an EEPROM, then that device must be set up to operate in the slave mode and mimic the I2C EEPROM. Immediately after entering the I2C boot function, the GPIO pins are configured for I2C-A operation and the I2C is initialized. The following requirements must be met when booting from the I2C module:

- The input frequency to the device must be in the appropriate range.
- The EEPROM must be at slave address 0x50.

Figure 31. Overview of I2C_Boot Function



There are two timing options for the I2C-A bootloader as shown in Table 3. Depending on the input clock to the system, choose the appropriate loader. The timing differences are:

- **I2C TIMING1 loader (boot mode C)**

  When using TIMING1, the input frequency to the device must be between 28 MHz and 48 MHz. In this case, the bootloader will set the I2CPSC prescale value to 1 so that the I2C clock will be divided down from SYSCLKOUT to create a 7 MHz to 12 MHz system clock.
- **I2C TIMING2 loader (boot mode 2)**

  When using TIMING2, the input frequency to the device must be between 14 MHz and 24 MHz. In this case the bootloader will set the I2CPSC prescale value to 0 so that the I2C clock is between 7 MHz to 12 MHz system clock.

The bit-period prescalers (I2CCLKH and I2CCLKL) are configured by the bootloader to run the I2C at a 50 percent duty cycle at 100-kHz bit rate (standard I2C mode) when the system clock is 12 MHz. These registers can be modified after receiving the first few bytes from the EEPROM. This allows the communication to be increased up to a 400-kHz bit rate (fast I2C mode) during the remaining data reads.

Arbitration, bus busy, and slave signals are not checked. Therefore, no other master is allowed to control the bus during this initialization phase. If the application requires another master during I2C boot mode, that master must be configured to hold off sending any I2C messages until the application software signals that it is past the bootloader portion of initialization.

The nonacknowledgment bit is checked only during the first message sent to initialize the EEPROM base address. This is to make sure that an EEPROM is present at address 0x50 before continuing. If an EEPROM is not present, code will enable the watchdog and force a device reset through software. The nonacknowledgment bit is not checked during the address phase of the data read messages (I2C_Get Word). If a non acknowledgment is received during the data read messages, the I2C bus will hang. Table 14 shows the 8-bit data stream used by the I2C.

Table 14. I2C 8-Bit Data Stream

Byte	Contents
1	LSB: AA (KeyValue for memory width = 8 bits)
2	MSB: 08h (KeyValue for memory width = 8 bits)
3	LSB: I2CPSC[7:0]
4	reserved
5	LSB: I2CCLKH[7:0]
6	MSB: I2CCLKH[15:8]
7	LSB: I2CCLKL[7:0]
8	MSB: I2CCLKL[15:8]
...	...
...	Data for this section. ...
17	LSB: Reserved for future use
18	MSB: Reserved for future use
19	LSB: Upper half of entry point PC
20	MSB: Upper half of entry point PC[22:16] (Note: Always 0x00)
21	LSB: Lower half of entry point PC[15:8]
22	MSB: Lower half of entry point PC[7:0]
...	...
...	Data for this section. ...
	Blocks of data in the format size/destination address/data as shown in the generic data stream description.
...	...
...	Data for this section. ...
n	LSB: 00h
n+1	MSB: 00h - indicates the end of the source

The I2C EEPROM protocol required by the I2C bootloader is shown in Figure 32 and Figure 33. The first communication, which sets the EEPROM address pointer to 0x0000 and reads the KeyValue (0x08AA) from it, is shown in Figure 32. All subsequent reads are shown in Figure 33 and are read two bytes at a time.

Figure 32. Random Read

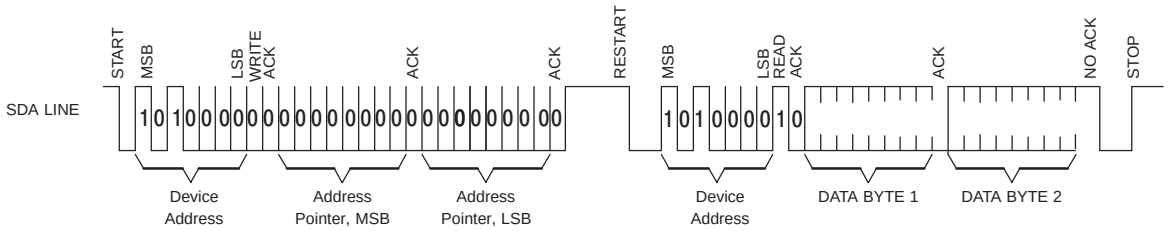
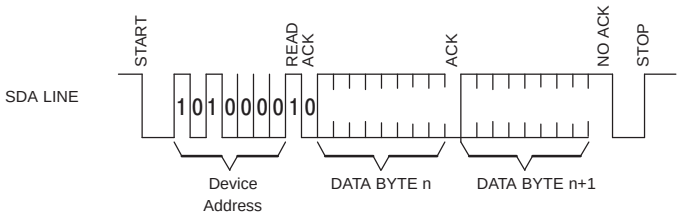


Figure 33. Sequential Read



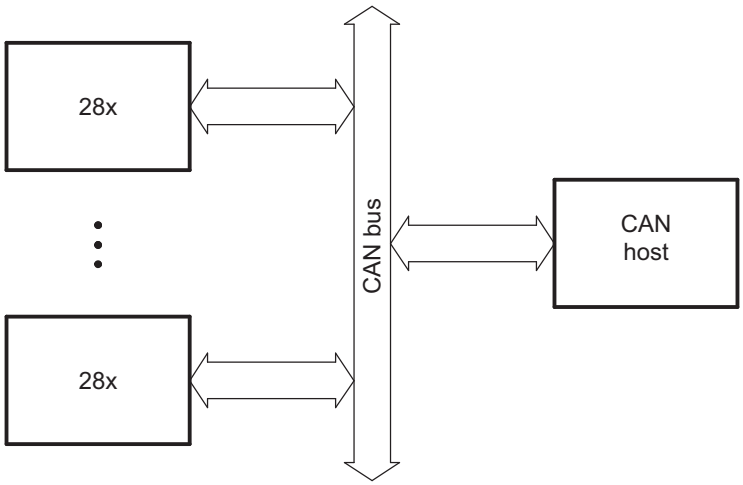
2.21 eCAN Boot Function

The eCAN bootloader asynchronously transfers code from eCAN-A to internal memory. The host can be any CAN node. The communication is first done with 11-bit standard identifiers (with a MSGID of 0x1) using two bytes per data frame. The host can download a kernel to reconfigure the eCAN if higher data throughput is desired.

The eCAN-A loader uses following pins:

- CANRXA on GPIO30
- CANTXA on GPIO31

Figure 34. Overview of eCAN-A bootloader Operation



There are two timing options for the eCAN bootloader as shown in Table 3 Depending on the input clock to the system, choose the appropriate loader. Table 15 shows what the bit rate will be depending on the input clock.

Table 15. Bit-Rate Values for Different XCLKIN Values

Boot Mode	Bit Time	XCLKIN	SYSCLKOUT (1)	CAN Clock	Bit Rate
Timing 1	15	30 MHz	30 MHz	7.5 MHz	500 kbps
Timing 2	10	20 MHz	10 MHz	2.5 MHz	250 kbps

(1) For timing 1, the bootloader sets the input clock divider to /1. For timing 2, the bootloader sets the input clock divider to /2.

The SYSCLKOUT values shown are the reset values with the default PLL setting. The BRP is hard coded to 1 for both timing 1 and timing 2 modes. The bit-time values are hard coded to 15 for timing 1 mode and 10 for timing 2 mode respectively.

Mailbox 1 is programmed with a standard MSGID of 0x1 for boot-loader communication. The CAN host should transmit only 2 bytes at a time, LSB first and MSB next. For example, to transmit the word 0x08AA to the device, transmit AA first, followed by 08. The program flow of the CAN bootloader is identical to the SCI bootloader. The data sequence for the CAN bootloader is shown in Table 16:

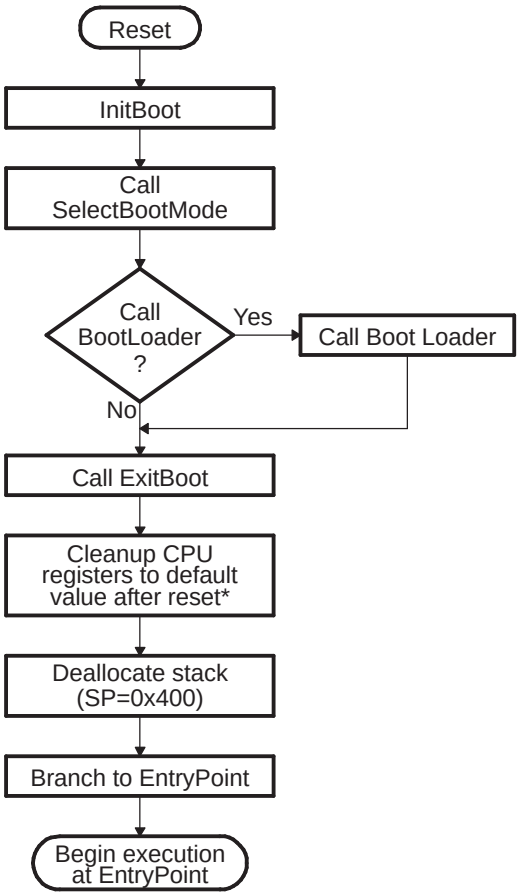
Table 16. eCAN 8-Bit Data Stream

Bytes		Byte 1 of 2	Byte 2 of 2	Description
1	2	AA	08	0x08AA (KeyValue for memory width = 16bits)
3	4	00	00	reserved
5	6	00	00	reserved
7	8	00	00	reserved
9	10	00	00	reserved
11	12	00	00	reserved
13	14	00	00	reserved
15	16	00	00	reserved
17	18	00	00	reserved
19	20	BB	00	Entry point PC[22:16]
21	22	DD	CC	Entry point PC[15:0] (PC = 0xAABBCCDD)
23	24	NN	MM	Block size of the first block of data to load = 0xMMNN words
25	26	BB	AA	Destination address of first block Addr[31:16]
27	28	DD	CC	Destination address of first block Addr[15:0] (Addr = 0xAABBCCDD)
29	30	BB	AA	First word of the first block in the source being loaded = 0xAABB
...				....
...				Data for this section.
				...
.		BB	AA	Last word of the first block of the source being loaded = 0xAABB
.		NN	MM	Block size of the 2nd block to load = 0xMMNN words
.		BB	AA	Destination address of second block Addr[31:16]
.		DD	CC	Destination address of second block Addr[15:0]
.		BB	AA	First word of the second block in the source being loaded
.				…
n	n+1	BB	AA	Last word of the last block of the source being loaded (More sections if required)
n+2	n+3	00	00	Block size of 0000h - indicates end of the source program

2.22 ExitBoot Assembly Routine

The Boot ROM includes an ExitBoot routine that restores the CPU registers to their default state at reset. This is performed on all registers with one exception. The OBJMODE bit in ST1 is left set so that the device remains configured for C28x operation. This flow is detailed in the following diagram:

Figure 35. ExitBoot Procedure Flow



The following CPU registers are restored to their default values:

- ACC = 0x0000 0000
- RPC = 0x0000 0000
- P = 0x0000 0000
- XT = 0x0000 0000
- ST0 = 0x0000
- ST1 = 0x0A0B
- XAR0 = XAR7 = 0x0000 0000

After the ExitBoot routine completes and the program flow is redirected to the entry point address, the CPU registers will have the following values:

Table 17. CPU Register Restored Values

Register	Value			Register	Value		
ACC	0x0000 0000			P	0x0000 0000		
XT	0x0000 0000			RPC	0x00 0000		
XAR0-XAR7	0x0000 0000			DP	0x0000		
ST0	0x0000	15:10	OVC = 0	ST1	0x0A0B	15:13	ARP = 0
		9:7	PM = 0			12	XF = 0
		6	V = 0			11	M0M1MAP = 1
		5	N = 0			10	reserved
		4	Z = 0			9	OBJMODE = 1
		3	C = 0			8	AMODE = 0
		2	TC = 0			7	IDLESTAT = 0
		1	OVM = 0			6	EALLOW = 0
		0	SXM = 0			5	LOOP = 0
						4	SPA = 0
						3	VMAP = 1
						2	PAGE0 = 0
						1	DBGM = 1
						0	INTM = 1

3 Building the Boot Table

This chapter explains how to generate the data stream and boot table required for the bootloader.

3.1 The C2000 Hex Utility

To use the features of the bootloader, you must generate a data stream and boot table as described in Section 2.10. The hex conversion utility tool, included with the 28x code generation tools, can generate the required data stream including the required boot table. This section describes the hex2000 utility. An example of a file conversion performed by hex2000 is described in Section 3.2.

The hex utility supports creation of the boot table required for the SCI, SPI, I2C, eCAN, and parallel I/O loaders. That is, the hex utility adds the required information to the file such as the key value, reserved bits, entry point, address, block start address, block length and terminating value. The contents of the boot table vary slightly depending on the boot mode and the options selected when running the hex conversion utility. The actual file format required by the host (ASCII, binary, hex, etc.) will differ from one specific application to another and some additional conversion may be required.

To build the boot table, follow these steps:

1. **Assemble or compile the code.**

   This creates the object files that will then be used by the linker to create a single output file.

2. **Link the file.**

   The linker combines all of the object files into a single output file in common object file format (COFF). The specified linker command file is used by the linker to allocate the code sections to different memory blocks. Each block of the boot table data corresponds to an initialized section in the COFF file. Uninitialized sections are not converted by the hex conversion utility. The following options may be useful:

   The linker -m option can be used to generate a map file. This map file will show all of the sections that were created, their location in memory and their length. It can be useful to check this file to make sure that the initialized sections are where you expect them to be.

   The linker -w option is also very useful. This option will tell you if the linker has assigned a section to a memory region on its own. For example, if you have a section in your code called ramfuncs.

3. **Run the hex conversion utility.**

   Choose the appropriate options for the desired boot mode and run the hex conversion utility to convert the COFF file produced by the linker to a boot table.

See the *TMS320C28x Assembly Language Tools User's Guide* (SPRU513) and the *TMS320C28x Optimizing C/C++ Compiler User's Guide* (SPRU514) for more information on the compiling and linking process.

Table 18 summarizes the hex conversion utility options available for the bootloader. See the *TMS320C28x Assembly Language Tools User's Guide* (SPRU513) for a detailed description of the hex2000 operations used to generate a boot table. Updates will be made to support the I2C boot. See the Codegen release notes for the latest information.

Table 18. Boot Loader Options

Option	Description
-boot	Convert all sections into bootable form (use instead of a SECTIONS directive)
-sci8	Specify the source of the bootloader table as the SCI-A port, 8-bit mode
-spi8	Specify the source of the bootloader table as the SPI-A port, 8-bit mode
-gpio8	Specify the source of the bootloader table as the GPIO port, 8-bit mode
-gpio16	Specify the source of the bootloader table as the GPIO port, 16-bit mode
-bootorg value	Specify the source address of the bootloader table
-lospcp value	Specify the initial value for the LOSPCP register. This value is used only for the spi8 boot table format and ignored for all other formats. If the value is greater than 0x7F, the value is truncated to 0x7F.
-spibrr value	Specify the initial value for the SPIBRR register. This value is used only for the spi8 boot table format and ignored for all other formats. If the value is greater than 0x7F, the value is truncated to 0x7F.
-e value	Specify the entry point at which to begin execution after boot loading. The value can be an address or a global symbol. This value is optional. The entry point can be defined at compile time using the linker -e option to assign the entry point to a global symbol. The entry point for a C program is normally _c_int00 unless defined otherwise by the -e linker option.
-i2c8	Specify the source of the bootloader table as the I2C-A port, 8-bit
-i2cpsc value	Specify the value for the I2CPSC register. This value will be loaded and take effect after all I2C options are loaded, prior to reading data from the EEPROM. This value will be truncated to the least significant eight bits and should be set to maintain an I2C module clock of 7-12 MHz.
-i2cclkh value	Specify the value for the I2CCLKH register. This value will be loaded and take effect after all I2C options are loaded, prior to reading data from the EEPROM.
-i2cclkl value	Specify the value for the I2CCLKL register. This value will be loaded and take effect after all I2C options are loaded, prior to reading data from the EEPROM.

3.2 Example: Preparing a COFF File For eCAN Bootloading

This section shows how to convert a COFF file into a format suitable for CAN based bootloading. This example assumes that the host sending the data stream is capable of reading an ASCII hex format file. An example COFF file named GPIO34TOG.out has been used for the conversion.

Build the project and link using the -m linker option to generate a map file. Examine the .map file produced by the linker. The information shown in Example 5 has been copied from the example map file (GPIO34TOG.map). This shows the section allocation map for the code. The map file includes the following information:

- **Output Section**
  This is the name of the output section specified with the SECTIONS directive in the linker command file.
- **Origin**
  The first origin listed for each output section is the starting address of that entire output section. The following origin values are the starting address of that portion of the output section.
- **Length**
  The first length listed for each output section is the length for that entire output section. The following length values are the lengths associated with that portion of the output section.
- **Attributes/input sections**
  This lists the input files that are part of the section or any value associated with an output section.

See the *TMS320C28x Assembly Language Tools User's Guide* (SPRU513) for detailed information on generating a linker command file and a memory map.

All sections shown in Example 5 that are initialized need to be loaded into the DSP in order for the code to execute properly. In this case, the codestart, ramfuncs, .cinit, myreset and .text sections need to be loaded. The other sections are uninitialized and will not be included in the loading process. The map file also indicates the size of each section and the starting address. For example, the .text section has 0x155 words and starts at 0x3FA000.

Example 5. GPIO34TOG Map File

```
output                                            attributes/
section         page     origin      length       input sections
--------        ----   ----------  ----------  ----------------
codestart
                  0     00000000     00000002
                        00000000     00000002   DSP280x_CodeStartBranch.obj (codestart)
.pinit            0     00000002     00000000

.switch           0     00000002     00000000   UNINITIALIZED

ramfuncs          0     00000002     00000016
                        00000002     00000016   DSP280x_SysCtrl.obj (ramfuncs)

.cinit            0     00000018     00000019
                        00000018     0000000e   rts2800_ml.lib : exit.obj (.cinit)
                        00000026     0000000a                  : _lock.obj (.cinit)
                        00000030     00000001   --HOLE-- [fill = 0]

myreset           0     00000032     00000002
                        00000032     00000002   DSP280x_CodeStartBranch.obj (myreset)

IQmath            0     003fa000     00000000   UNINITIALIZED

.text             0     003fa000     00000155
                        003fa000     00000046   rts2800_ml.lib : boot.obj (.text)
```

To load the code using the CAN bootloader, the host must send the data in the format that the bootloader understands. That is, the data must be sent as blocks of data with a size, starting address followed by the data. A block size of 0 indicates the end of the data. The HEX2000.exe utility can be used to convert the COFF file into a format that includes this boot information. The following command syntax has been used to convert the application into an ASCII hex format file that includes all of the required information for the bootloader:

Example 6. HEX2000.exe Command Syntax

```
C: HEX2000 GPIO34TOG.OUT -boot -gpio8 -a


Where:
- boot   Convert all sections into bootable form.
- gpio8  Use the GPIO in 8-bit mode data format. The eCAN
         uses the same data format as the GPIO in 8-bit mode.
- a      Select ASCII-Hex as the output format.
```

The command line shown in Example 6 will generate an ASCII-Hex output file called GPIO34TOG.a00, whose contents are explained in Example 7. This example assumes that the host will be able to read an ASCII hex format file. The format may differ for your application. . Each section of data loaded can be tied back to the map file described in Example 5. After the data stream is loaded, the boot ROM will jump to the Entrypoint address that was read as part of the data stream. In this case, execution will begin at 0x3FA0000.

Example 7. GPIO34TOG Data Stream

```
AA 08                                       ;Keyvalue
00 00 00 00 00 00 00 00                     ;8 reserved words
00 00 00 00 00 00 00 00
3F 00 00 A0                                 ;Entrypoint 0x003FA000
02 00                                       ;Load 2 words - codestart section
00 00 00 00                                 ;Load block starting at 0x000000
7F 00 9A A0                                 ;Data block 0x007F, 0xA09A
16 00                                       ;Load 0x0016 words - ramfuncs section
00 00 02 00                                 ;Load block starting at 0x000002
22 76 1F 76 2A 00 00 1A 01 00 06 CC F0      ;Data = 0x7522, 0x761F etc...
FF 05 50 06 96 06 CC FF F0 A9 1A 00 05
06 96 04 1A FF 00 05 1A FF 00 1A 76 07
F6 00 77 06 00
55 01                                       ;Load 0x0155 words - .text section
3F 00 00 A0                                 ;Load block starting at 0x003FA000
AD 28 00 04 69 FF 1F 56 16 56 1A 56 40      ;Data = 0x28AD, 0x4000 etc...
29 1F 76 00 00 02 29 1B 76 22 76 A9 28
18 00 A8 28 00 00 01 09 1D 61 C0 76 18
00 04 29 0F 6F 00 9B A9 24 01 DF 04 6C
04 29 A8 24 01 DF A6 1E A1 F7 86 24 A7
06 .. ..
.. .. ..
.. .. ..
FC 63 E6 6F
19 00 ;Load 0x0019 words - .cinit section
00 00 18 00                                 ;Load block starting at 0x000018
FF FF 00 B0 3F 00 00 00 FE FF 02 B0 3F      ;Data = 0xFFFF, 0xB000 etc...
00 00 00 00 00 FE FF 04 B0 3F 00 00 00
00 00 FE FF .. .. ..
.. .. ..
3F 00 00 00
02 00                                       ;Load 0x0002 words - myreset section
00 00 32 00                                 ;Load block starting at 0x000032
00 00 00 00                                 ;Data = 0x0000, 0x0000
00 00                                       ;Block size of 0 - end of data
```

4 Bootloader Code Overview

This chapter contains information on the Boot ROM version, checksum, and code.

4.1 Boot ROM Version and Checksum Information

The boot ROM contains its own version number located at address 0x3F FFBA. This version number starts at 1 and will be incremented any time the boot ROM code is modified. The next address, 0x3F FFBB contains the month and year (MM/YY in decimal) that the boot code was released. The next four memory locations contain a checksum value for the boot ROM. Taking a 64-bit summation of all addresses within the ROM, except for the checksum locations, generates this checksum.

Table 19. Bootloader Revision and Checksum Information

Address	Contents
0x3F FFB9	Reserved
0x3F FFBA	Boot ROM Version Number
0x3F FFBB	MM/YY of release (in decimal)
0x3F FFBC	Least significant word of checksum
0x3F FFBD	. . .
0x3F FFBE	. . .
0x3F FFBF	Most significant word of checksum

Table 20 shows the boot ROM revision per device. A revision history and code listing for the latest boot ROM code can be found in Section 4. In addition, a .zip file with each revision of the boot ROM code can be downloaded at http://www-s.ti.com/sc/techlit/sprufn5.zip

Table 20. Bootloader Revision Per Device

Device(s)	Silicon REVID (Address 0x883)	Boot ROM Revision
2834x	0 (First silicon)	Version 1b

4.2 Bootloader Code Revision History

The associated boot ROM source code can be downloaded at http://www-s.ti.com/sc/techlit/sprufn5.zip.

- **Version: 1b, Released: June 2008:**
  The initial release of the boot ROM.
  Known issues:
  - Boot to XINTF x32
    This mode has been removed with no plans to implement it at a later date. The boot ROM incorrectly configures GPBMUX2 for peripheral operation instead of XD[31:16].
- **Version: 1, 1a, Not Released**
  TI internal testing only.

Appendix A Revision History

This doc has been revised to include the following technical change(s).

Table 21. Additions, Deletions, and Changes

Location	Description
Table 3	Changed TI test only to Secure boot and added the note.

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 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless-apps

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