Using the Stellaris® Serial Flash Loader

Application Note

AN01242 -03 / SPMA029

Copyright

Texas Instruments.
108 Wild Basin, Suite 350
Austin, TX 78746
Main: +1-512-279-8800
Fax: +1-512-279-8879
http://www.luminarymicro.com

Table of Contents

Introduction

This application note describes how to communicate with the Stellaris® serial flash loader application. The serial flash loader is a small piece of code that allows programming of the flash without the need for a debugger interface. Two serial interfaces are supported for download to the device, UART and SSI. The serial flash loader uses a packet type interface to provide reliable synchronous communications. Two different UART download utilities are also supplied that can be used to program the flash using the serial flash loader.

The serial flash loader application is factory-programmed into each device. Since the flash loader allows programming of a device using basic serial interface, it can be used to program a device during production when a debugger interface is not available. It can also be used to provide a method to download applications to the on-chip flash and run the application without the use of any debugger interface. In the event that the serial flash loader is erased, a downloadable binary (flashloader.bin) to reprogram the flash loader into a device is also provided. The sample code described in this application note can be downloaded from http://www.ti.com/lit/zip/SPMA029.

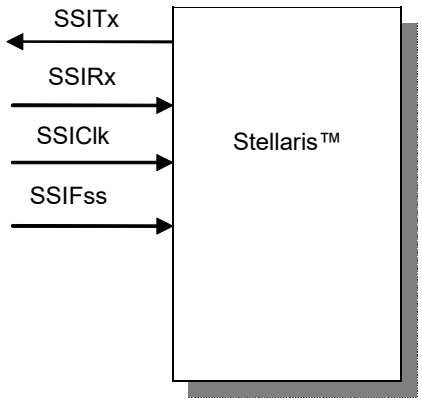
Connections

There are two supported connection options available for the serial flash loader. They are the SSI and UART0 ports which are available on all Stellaris devices. The SSI port has the advantage of supporting higher and more flexible data rates, but it also requires more connections to the device. The UART has the disadvantage of having slightly lower and possibly less flexible rates. However, the UART requires fewer pins and can be easily implemented with any standard UART connection.

SSI

The SSI port uses four wires for SSI communications, which are connected to the following pins: SSITx, SSIRx, SSIClk, and SSIFss as shown in Figure 1. The device communicating with the flash loader is responsible for driving the SSIRx, SSIClk, and SSIFss pins, while the Stellaris device drives the SSITx pin. The format used for SSI communications is the Motorola format with SPH set to 1 and SPO set to 1 (see the Stellaris Family data sheet for more information on this format). The SSI interface has a hardware requirement that limits the maximum rate of the SSI clock. This limit for the SSI clock requires that the SSI clock can be at most 1/12 the crystal frequency of the board running the flash loader.

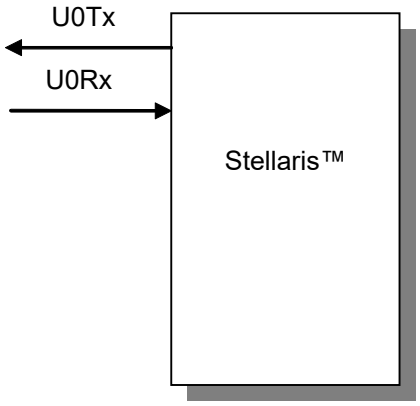
Figure 1. SSI Connections



UART

The UART port uses two wires for UART communications, which are connected to the following pins shown in Figure 2: U0Tx and U0Rx. The device communicating with the flash loader is responsible for driving U0Rx, while the Stellaris device drives the U0Tx pin. The baud rate is flexible, however, the serial format is fixed at 8 data bits, no parity, and one stop bit. The baud rate used for communication is auto-detected by the flash loader and can be any valid baud rate supported by the device communicating with the serial flash loader. The only requirement on baud rate is that the auto-detection sequence requires that the baud rate should be no more than 1/32 the crystal frequency of the board that is running the serial flash loader. This is the same as the hardware requirement for the maximum baud rate for any UART on a Stellaris device.

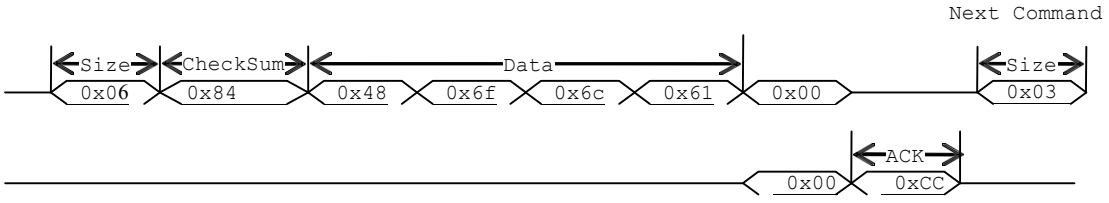
Figure 2. UART Connections



Packet Handling

All communications, with the exception of the UART auto-baud, are done via defined packets that are acknowledged (ACK) or not acknowledged (NAK) by the devices. The packets use the same format for receiving and sending packets. This includes the method used to acknowledge successful or unsuccessful reception of a packet. The basic packet format is shown in Figure 3. While the actual signaling on the serial ports is different, the packet format remains the same for UART or SSI. The top line shows the device that is transmitting data while the bottom line is the response from the other device. In this case, a six-byte packet is sent with the data shown below. This data results in a checksum of 0x48+0x6f+0x6c+0x61 which, when truncated to 8 bits, is 0x84. The next bytes to go out are the 4 data bytes in this packet. The transmitter is allowed to send zeros until a non-zero response is received, which is necessary for SSI and allowed by the UART. The receiver is allowed to return zeros until it is ready to ACK or NAK the packet that is being sent. Neither device should transfer a non-zero byte until it has received a response after transmitting a packet.

Figure 3. Serial Bus Packet Format



SSI

Figure 4 shows the transmission of a simple ping request to the flash loader using the SSI port. The signal names in the figure are relative to the Stellaris device. The first byte transferred is the size of the packet, followed by the checksum of the command, and finally followed by the ping command itself. Since the SSI port must transfer to receive any bytes, there is a series of zeros transferred to the flash loader device until a non-zero value is transmitted by the flash loader. The device transmitting to the flash loader must send only zero values while waiting for a response from the flash loader. Instead of constantly sending zeros, the device can also transfer zero data periodically, to see if the receiving device has acknowledged the packet. In most cases, it takes a few microseconds to respond to any packet.

Figure 4. SSI ping Command

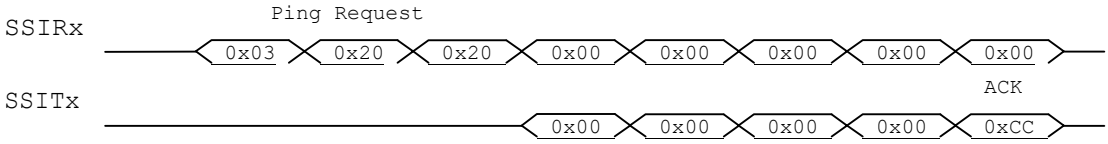
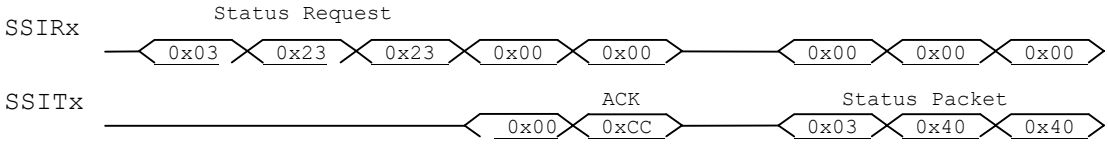


Figure 5 shows an example of the Get Status (0x23) command, which provides data back to the host device. The first non-zero data received from the flash loader after the ACK (0xCC) is the first byte of the returned packet. In this case, the returned packet is a status packet consisting of the size, checksum, and data. The actual data returned is the single byte 0x40, which indicates that the last command issued completed successfully. Any value other than 0x40 indicates that the previous command failed for the reason indicated by the data value returned. For a complete list of return codes, see “COMMAND_GET_STATUS (0x23)” on page 8.

Figure 5. SSI Status Request



UART

In order to determine the baud rate, the serial flash loader needs to determine the relationship between its own crystal frequency and the baud rate. This is enough information for the flash loader to configure its UART to the same baud rate as the device communicating with the flash loader. The method used to perform this automatic synchronization relies on the device sending the flash loader two bytes that are both 0x55 as shown in Figure 6. This generates a series of pulses to the flash loader that it can use to calculate the ratios needed to program the UART clocks to match the host's baud rate. After the host sends the pattern, it should attempt to read back data from the UART. The host should wait for a byte with the value of 0xCC from the device. If this byte is not received after at least twice the time required to transfer the two bytes, the host can resend another synchronization pattern and wait for the 0xCC byte again until the flash loader acknowledges that it has received a synchronization pattern correctly. As an example, the time to wait for data back from the flash loader should be calculated as at least 2*(20(bits/sync)/baud rate (bits/sec)). For a baud rate of 115200, this time is calculated as 2*(20/115200) or 0.35 ms.

Figure 6. UART Auto Baud Rate

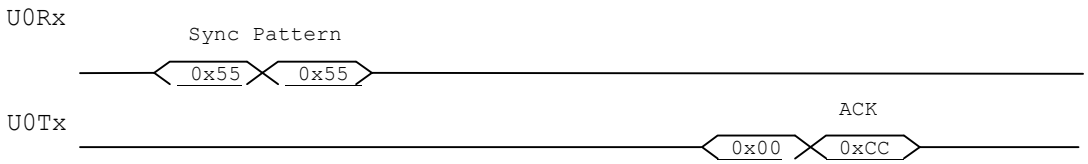


Figure 7 shows the transmission of a simple ping request to the flash loader using the UART port. The first byte transferred is the size of the packet, followed by the checksum and finally followed by the ping command itself. The host device should wait for the first non-zero data before looking for the ACK (0xCC) byte. Typically, it takes a few microseconds to respond to the command.

Figure 7. UART Ping Command

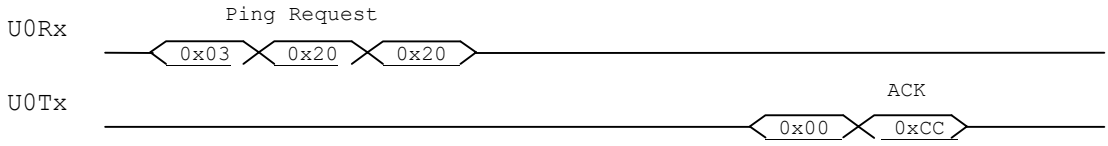
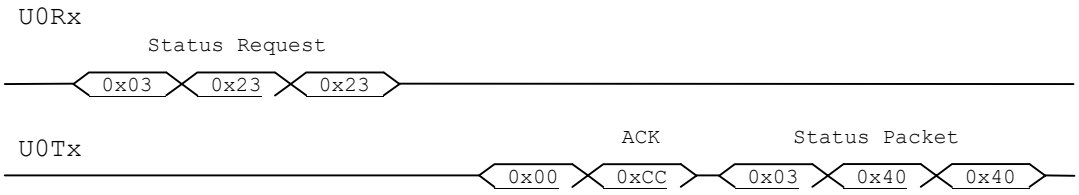


Figure 8 shows the Get Status (0x23) command, which provides data back to the host device. The first non-zero data received from the flash loader after the ACK (0xCC) is the first byte of the returned data. Like the SSI case, the returned data is a status packet consisting of the size, checksum, and data. Since the data returned in this case is 0x40, this indicates that the last command issued completed successfully. Any value other than 0x40 indicates that the previous command failed for the reason indicated by the data value returned. For a complete list of return codes, see “COMMAND_GET_STATUS (0x23)” on page 8.

Figure 8. UART Get Status Request



Command Definitions

This section defines the list of commands that can be sent to the flash loader as well as other commands used when communicating with the flash loader. The first byte of the data in a packet should always be one of the defined commands, followed by data or parameters as determined by the command that is sent.

The following two definitions are the values used to indicate successful or unsuccessful transmission of a packet.

```
#define COMMAND_ACK                0xcc
#define COMMAND_NAK                0x33
```

COMMAND_PING (0x20)

The COMMAND_PING command simply accepts the command and sets the global status to success. The format of the packet is as follows:

```
#define COMMAND_PING                0x20

Byte[0] = 0x03;
Byte[1] = checksum(Byte[2]);
Byte[2] = COMMAND_PING;
```

The ping command requires the standard 2-byte header followed by the value COMMAND_PING. Since there is only one byte in the command, the checksum is simply equal to COMMAND_PING. Since the ping command has no real return status, the receipt of an ACK can be interpreted as a successful ping to the flash loader.

COMMAND_GET_STATUS (0x23)

The COMMAND_GET_STATUS command returns the status of the last command that was issued. Typically, this command should be sent after every command to ensure that the previous command was successful or, if unsuccessful, to properly respond to a failure. The command requires one byte in the data of the packet and should be followed by reading a packet with one byte of data that contains a status code. The last step is to ACK or NAK the received data so the flash loader knows that the data has been read.

```
#define COMMAND_GET_STATUS          0x23

Byte[0] = 0x03
Byte[1] = checksum(Byte[2])
Byte[2] = COMMAND_GET_STATUS
```

The following list shows the definitions for the possible status values that are returned from the serial flash loader when a COMMAND_GET_STATUS command is sent to the device.

```
#define    COMMAND_RET_SUCCESS       0x40
#define   COMMAND_RET_UNKNOWN_CMD   0x41
#define   COMMAND_RET_INVALID_CMD   0x42
#define COMMAND_RET_INVALID_ADDR    0x43
#define COMMAND_RET_FLASH_FAIL      0x44
```

COMMAND_DOWNLOAD (0x21)

The COMMAND_DOWNLOAD command is sent to the flash loader to indicate where to store data and how many bytes to send with the COMMAND_SEND_DATA commands that follow. The command consists of two 32-bit values that are both transferred most significant bit (MSB) first. The first 32-bit value is the address to start programming data into, while the second is the 32-bit size of the data to send. This command also triggers an erase of the full area to be programmed, so this command takes longer than other commands. This results in a longer time to receive the ACK/NAK back from the board. This command should be followed by a COMMAND_GET_STATUS to ensure that the Program Address and Program Size parameters are valid for the device running the flash loader.

The format of the packet to send this command is a follows:

```
#define COMMAND_DOWNLOAD            0x21

Byte[0] = 11
Byte[1] = checksum(Bytes[2:10])
Byte[2] = COMMAND_DOWNLOAD
Byte[3] = Program Address [31:24]
Byte[4] = Program Address [23:16]
Byte[5] = Program Address [15:8]
Byte[6] = Program Address [7:0]
Byte[7] = Program Size [31:24]
Byte[8] = Program Size [23:16]
Byte[9] = Program Size [15:8]
Byte[10] = Program Size [7:0]
```

COMMAND_SEND_DATA (0x24)

The COMMAND_SEND_DATA command should only follow a COMMAND_DOWNLOAD command or another COMMAND_SEND_DATA command if more data is needed. Consecutive COMMAND_SEND_DATA commands automatically increment address and continue programming from the previous location. The caller should limit transfers of data to a maximum of 8 bytes of packet data to allow the flash to program successfully and not overflow input buffers of the serial interfaces. The command terminates programming once the number of bytes indicated by the COMMAND_DOWNLOAD command has been received. Each time this function is called it should be followed by a COMMAND_GET_STATUS command to ensure that the data was successfully programmed into the flash. If the flash loader sends a NAK to this command, the flash loader does not increment the current address to allow retransmission of the previous data.

```
#define COMMAND_SEND_DATA           0x24

Byte[0] = 11
Byte[1] = checksum(Bytes[2:10])
Byte[2] = COMMAND_SEND_DATA
Byte[3] = Data[0]
Byte[4] = Data[1]
Byte[5] = Data[2]
Byte[6] = Data[3]
Byte[7] = Data[4]
Byte[8] = Data[5]
Byte[9] = Data[6]
Byte[10] = Data[7]
```

COMMAND_RUN (0x22)

The COMMAND_RUN command tells the flash loader to execute from the address passed as the parameter in this command. This command consists of a single 32-bit value that is interpreted as the address to execute. The 32-bit value is transmitted MSB first and the flash loader sends an ACK signal back to the host device before actually executing the code at the given address. This allows the host to know that the command was received successfully and the code is now running.

```
#define COMMAND_RUN                 0x22

Byte[0] = 7
Byte[1] = checksum(Bytes[2:6])
```

```
Byte[2] = COMMAND_RUN
Byte[3] = Execute Address[31:24]
Byte[4] = Execute Address[23:16]
Byte[5] = Execute Address[15:8]
Byte[6] = Execute Address[7:0]
```

COMMAND_RESET (0x25)

The COMMAND_RESET command tells the flash loader device to reset. This is useful when downloading a new image that overwrote the flash loader. This is useful when a new image has overwritten the flash loader and wants to start from a full reset. Unlike the run command, this allows the initial stack pointer to be read by the hardware and set up for the new code. It can also be used to reset the flash loader if a critical error occurs and the host device wants to restart communication with the flash loader.

```
#define COMMAND_RESET               0x25

Byte[0] = 3
Byte[1] = checksum(Byte[2])
Byte[2] = COMMAND_RESET
```

The flash loader sends an ACK signal back to the host device before actually executing the software reset to the device running the flash loader. This lets the host know that the command was received successfully and the part will be reset.

User Applications

There are two methods that a download program can use to run an application. The first is to completely overwrite the flash loader with a new application. The second method involves programming the application to an address not used by the flash loader, and then executing this application by issuing a “run” command. To allow the flash loader to be overwritten in flash, the serial flash loader copies itself into RAM before allowing any downloading to occur. The flash memory footprint of the serial flash loader in the flash is less than 2048 bytes. This means that the first location in flash that can safely be programmed, without erasing any part of the flash loader, is at address 0x0800. Because the called program cannot return to the flash loader, all RAM is usable after execution of the user program has begun.

In order to overwrite the flash loader in the flash, the downloaded program must start at address 0x0000 to prevent any part of the flash loader running. Unless the program configures its own stack space, the downloaded program should be followed by a reset command or a power cycle and not a run command, as this will leave the stack in an unknown state.

To use the serial flash loader to download and run a program, the downloaded address must start at a minimum of 0x0800. Once the download is complete, the download program should issue a “run” command to allow the program to execute.

Note: This method of downloading does not allow the application to run from a power cycle or reset.

If this method is used to download and run an application, the program must configure its own stack. This involves setting the ARM stack pointer register to the top of the stack memory provided by the program. The example below demonstrates how to configure the stack in software. This code must

be run during the application's reset vector as this is the only place where the stack has not already been used.

Example 1. Stack Configuration

```
#define STACK_SIZE 64
unsigned long g_Stack[STACK_SIZE];
unsigned long const g_ulStackTop =
    (unsigned long)g_Stack + STACK_SIZE;
void
ResetVector(void)
{
    //
    // Set the stack pointer to the "top" of the stack.
    //
    __asm("ldr   sp, =g_ulStackTop");

    //
    // Other initialization before calling main...
    //
    ...
}
```

In order to exit the application and run a new application, the flash loader must be reset via a software reset, hardware reset, or power-on reset. Once communication with the flash loader has been reestablished, either a new application can be downloaded or the previous application can be run again via a “run” command (the same run command as before). Some care should be taken when resetting the device and to use the flash loader. If the application was communicating with the flash loader via the SSI or UART, care must be taken to be sure that the device communicating with the flash loader has completely finished transmitting all data. If there are remaining bytes being transferred to the flash loader, it may detect these bytes and not correctly restart communications.

Download Utilities

Two different download utilities that can be used to program the flash using the serial flash loader are also supplied, LM Flash Programmer and sflash.exe. Both of these utilities interface to the serial flash loader using the UART port. The utilities can both be downloaded from the Software Updates page at www.luminarymicro.com.

The LM Flash Programmer utility can be run from a Graphical User Interface (GUI) or a command line prompt. This utility is intended to be used to overwrite the serial flash loader in flash. Support for downloading and running an application using the “run” command is not supported with this utility. LM Flash Programmer can also be used to program the flash using the debug port. Details on how to use LM Flash Programmer are provided by clicking the “Help” button on the GUI or typing “lmflash --help” at the command line prompt.

The sflash.exe utility is command-line only. This utility makes full use of all commands supported by the serial flash loader and can be used to overwrite the serial flash loader in flash as well as download and run an application using the “run” command. Details on how to use sflash.exe are provided by typing “sflash --help” at the command line prompt. Both the executable and source code for this utility are provided. The source code is provided as a reference if creating a custom utility is desired.

Conclusion

Either serial interface provides a simple method to program the Stellaris device without the need to have debugger hardware (such as JTAG) available at the time programming is required. The serial flash loader does not protect itself from being overwritten, so it does not reserve any space in the internal flash, leaving the full flash available for the downloaded application. Since the serial flash loader auto-detects the serial interface being used to communicate with it, the decision on which serial interface is left to the host device.

References

The following are available for download at www.luminarymicro.com:

- Stellaris microcontroller data sheet, Publication Number DS-LM3S*nnn* (where *nnn* is the part number for that specific Stellaris family device)
- LM Flash Programmer utility (LMFlashProgrammer.msi)
- Sample download utility (sflash.exe)
- Downloadable binary (flashloader.bin) to reprogram the flash loader into a device

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	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

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

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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