Application Report
SWRA500–January 2016

MSP430™ Firmware Updates Over I²C Using Linux®

Timothy Logan

ABSTRACT

In many embedded systems, an MSP430™ microcontroller is connected to an application processor running a version of Linux® such as Android™ or Debian®. The MSP430 manages low-level peripherals such as power devices or sensors and communicates relevant information to the application processor over I²C. In these applications, the Linux-based host must be able to update the firmware running on the MSP430. This document provides a portable software base to update an MSP430 device with an I²C bootloader (BSL) by using standard Linux I²C calls.

Contents

List of Figures

List of Tables

1 Introduction

Applications such as consumer electronics often have a setup where an application processor running a Linux-based operating system is controlling an MSP430 attached by I²C. The MSP430 device might be managing a sensor or performing other low-power centric tasks for the Linux host. In applications like this, the Linux host must be able to update the firmware running on the MSP430 through I²C. This document provides example code and implementation examples to provide users a reference for how to update on an MSP430 device with an I²C BSL over Linux.

This document assumes the user understands how the MSP430 I²C BSL operates. This operation is explained in detail in the varying BSL documents for each MSP430 platform (see Section 7 for details). This document also assumes that the user understands how to use Linux and how to compile code on Linux using the GCC tool chain.

No special software libraries are required to compile the provided code, but having a formal GCC development environment capable of compiling C code is required. An MSP430 with an I²C BSL is required for the provided code to function as expected. For testing, the sample code uses an MSP430FR59691 with an MSP-TS430RGZ48C target board. For the host side, a TI BeagleBone Black is used running the Debian distribution of Linux. To determine which version of the BSL that your device is using, see the data sheet of the device.

For the software for this application report, see http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/i2c_bsl_linux_tools/latest/index_FDS.html.

2 BSL Commands and Firmware Parsers

TI designed this application to segregate the core I²C communication code and the user implementation. This segregation was done to foster a device agnostic architecture and offer the opportunity to expand the functionality to different host architectures. The i2cbsl.c source file contains all BSL command APIs and low-level I²C physical communication. The external functions in this file contain the APIs that directly correspond to the specific BSL commands such as *mass erase* and *program segment*. Low-level I²C communication functions are included in this file such as *send data* and *read data*. Table 1 lists the supported BSL commands in this implementation and a brief description of each command. Figure 1 shows a high-level block diagram of the code organization.

Table 1. Supported BSL Commands

BSL Command	Description
MSP430BSL_sendData	Sends data to be programmed to the memory of the MSP430 device
MSP430BSL_readData	Returns X bytes of data at the specified address and prints them on screen. This command requires the BSL password to be provided through the MSP430BSL_unlockDevice command.
MSP430BSL_unlockDevice	Provides a 32-byte BSL password to unlock the device for memory reads or CRC calculations.
MSP430BSL_massErase	Erases nonprotected memory of the MSP430 device
MSP430BSL_checkCRC	Returns the CRC of the provided memory range. This command requires the BSL password to be provided through the MSP430BSL_unlockDevice command.
MSP430BSL_invokeBSL	Sends the provided software invoke sequence to the MSP430 device
MSP430BSL_setProgramCounter	Sets the program counter of the MSP430 device. This function can reset the device after the firmware update completes.

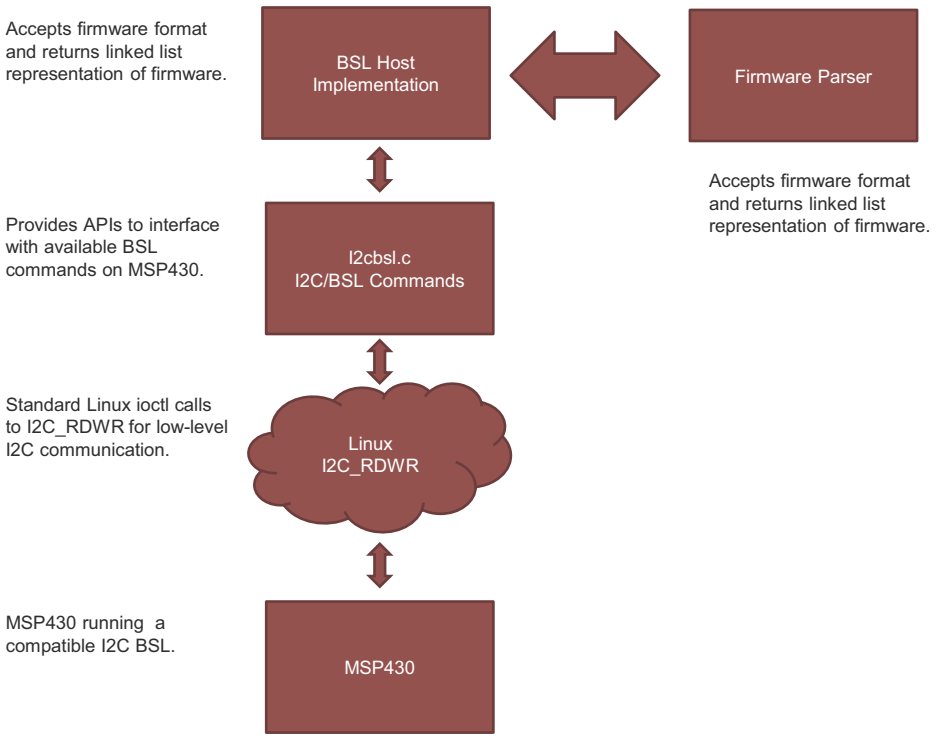

Figure 1. Overall I²C BSL Update Block Diagram

In addition to the core I²C communication APIs, a set of *firmware parser* functions are also provided. These functions take a variety of different firmware formats and parse them into a linked list structure that can be passed into the core I²C communication functions. Figure 2 shows the linked list structure that represents the firmware.

Figure 2 shows a standard linked list structure that represents a series of noncontiguous segments of memory in the MSP430.

```
typedef struct sMSPMemorySegment
{
    uint32_t            ui32MemoryStartAddr;
    uint32_t            ui32MemoryLength;
    uint8_t*            ui8Buffer;
    void*               pNextSegment;
} tMSPMemorySegment;
```

Figure 2. Firmware Linked List Structure

The *ui32MemoryStartAddr* variable is the start address of the segment, the *ui32MemoryLength* is the length of the current segment, the *ui8Buffer* variable is a pointer to the array containing the memory contents, and the *pNextSegment* pointer is a link to the next memory segment in the linked list. For the last memory segment the *pNextSegment* is a value of *NULL*. TI designed this structure to be generic and easily sourced from a variety of firmware formats. TI provides the following example parsers with this document: a TI-TXT parser and a SRecord Array parser. The following sections describe these parsers.

3 Terminal Program

TI created the terminal program as an interactive way for users to completely customize the method by which the firmware update is handled. The *msp430-i2cbsl-tool* folder includes this implementation. By using the terminal program, users can specify BSL options such as BSL password and payload size. Users can also perform special BSL commands such as reading data or calculating CRC through the terminal program. For the firmware format, the terminal program accepts a standard TI-TXT file. This file is automatically parsed and converted into the linked list structure defined in Section 2. Figure 3 shows a printout of the terminal options.

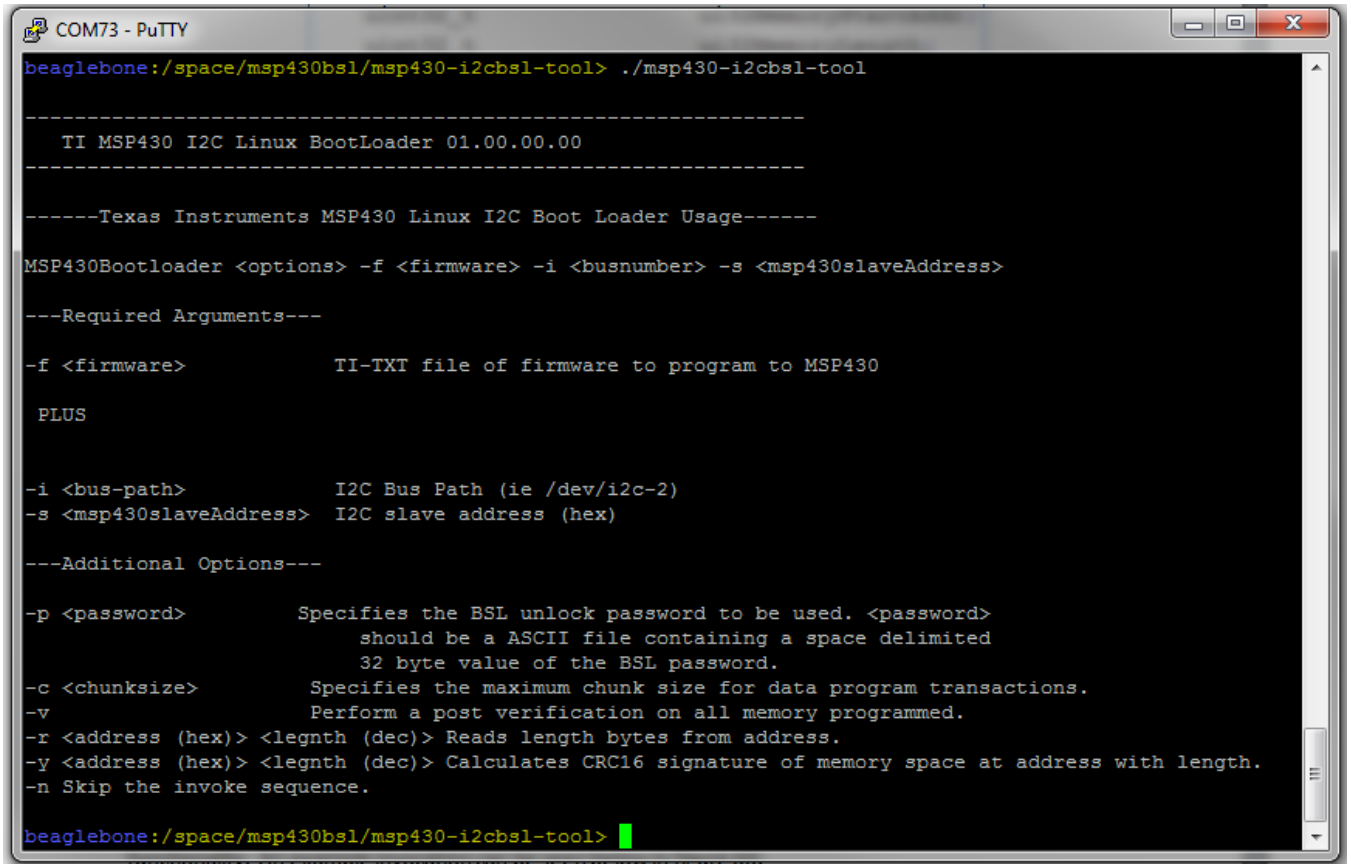

Figure 3. Terminal Application Options

The I²C path, slave address, and firmware file must be provided for basic BSL functionality. Figure 4 shows an example invocation.

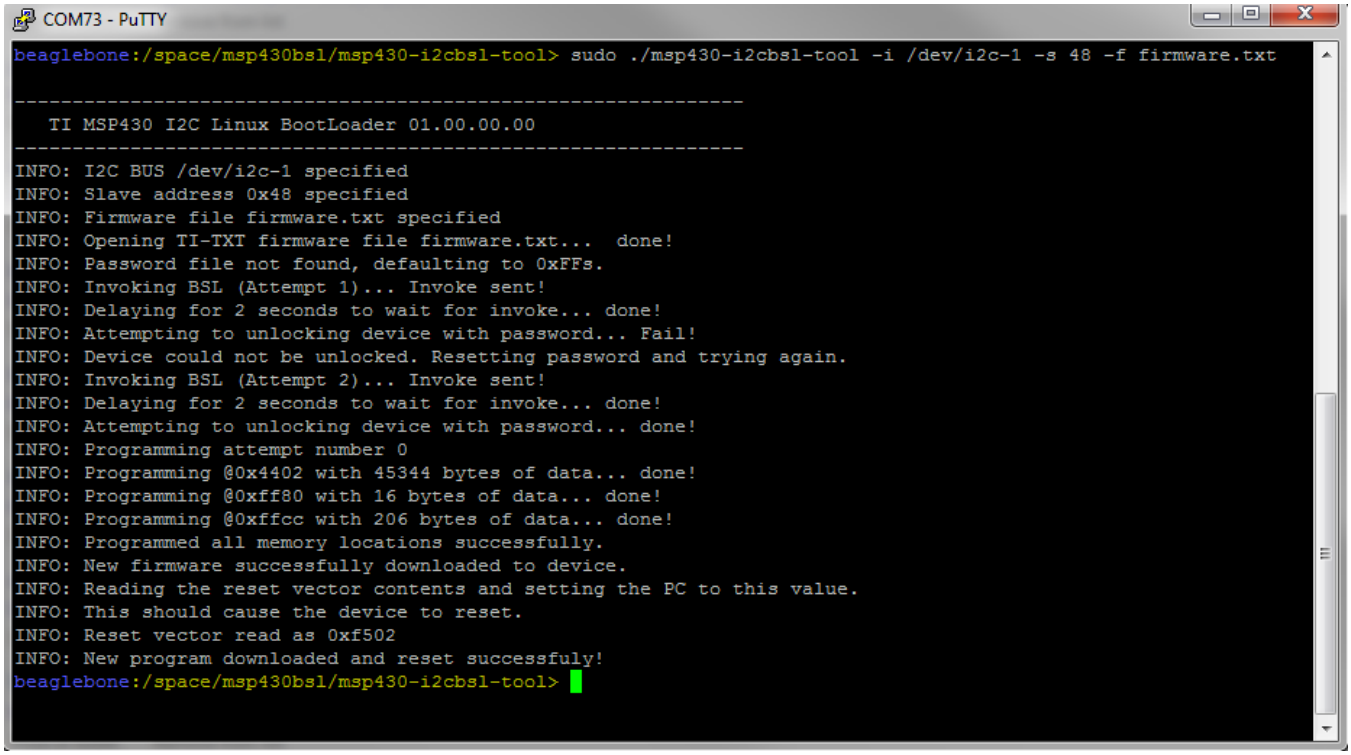

Figure 4. I²C BSL Update Through Terminal Application

In this example, the I²C path is /dev/i2c-1, the slave address is 0x48, and the firmware file to program is firmware.txt. In this instance, the password file is omitted. If the password file is omitted, the device performs a mass erase (resetting the password to all 0xFFs) and the device is unlocked. If a password is provided, it must be specified as a space delimited ASCII file similar to the TI-TXT file. The password file must be provided with no addresses and just the hex values of the BSL password (usually the contents in memory from 0xFFE0 to 0xFFFF). Figure 5 shows an example of the password file.

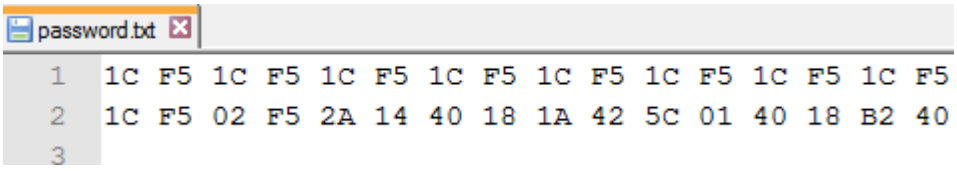

Figure 5. Example BSL Password File

In addition to updating the firmware, the terminal application can read and return the memory contents of the device or calculate the CRC value of a specified range of memory. These commands are available by the use of the –r and –y flags appropriately. These functions require a password to prevent a mass erase. If an invalid password or no password is provided, a mass erase occurs and memory is reset to 0xFF (making a read or CRC calculation meaningless). The *chunk size* of the payload can also be specified by using the –c flag. The –c flag is an advanced setting that changes the maximum payload size when sending data to program memory on the MSP430 device. By default, the maximum payload size is set to 16 bytes but can be increased to the maximum supported buffer size of the device. The maximum supported buffer size varies from device to device. To find the optimal value, see the device-specific BSL guide.

Another important design requirement of performing a BSL update is having a method to invoke the BSL mode on the MSP430 device. For MSP430, the following are ways to invoke the BSL: hardware invocation and software invocation. Hardware invocation requires the user to apply a specific timing pulse to the TEST and RESET pins. Software invocation requires the user to have a custom I^2C command handler that changes the program counter to enter BSL mode. For specifics on software and hardware BSL invocations, see the specific BSL design guide (see *MSP430FR57xx, MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Bootloader (BSL) User's Guide* [SLAU550]). For hardware invocation, the timing pulse must be applied on the TEST and RESET pins before calling the terminal application. If the hardware invocation is used, specify the –n option to omit the software invocation. If the –n flag is not specified before any BSL command is issued, the terminal program sends an I^2C write transaction of the bytes specified in the *invokeString* array of the main.c file. By default, this invoke string is represented by a character sequence of {0xCA, 0xFE, 0xDE, 0xAD, 0xBE, 0xEF, 0xBA, 0xBE}.

4 Simplified Package Program

In addition to a fully functional terminal application, this document also provides a simplified *package* implementation that is fully contained and requires no external input. The source code in this program contains the I^2C slave information and a SRecord C-array representation of the firmware. This implementation is useful for automated system updates for consumer electronics such as tablets or smart phones where user input is inconvenient. The source for this implementation is included within the *msp430-i2cbsl-package* folder. The firmware in this implementation is represented in a C-Array format, which is exported by the SRecord tool (see *SRecord Firmware Parser Tool*, http://srecord.sourceforge.net/). SRecord is an open source tool to convert and manage various formats of embedded firmware. This tool can convert a TI-TXT file to a standard C-Array implementation. To do this conversion, the SRecord program must be invoked with the options shown in Figure 6.

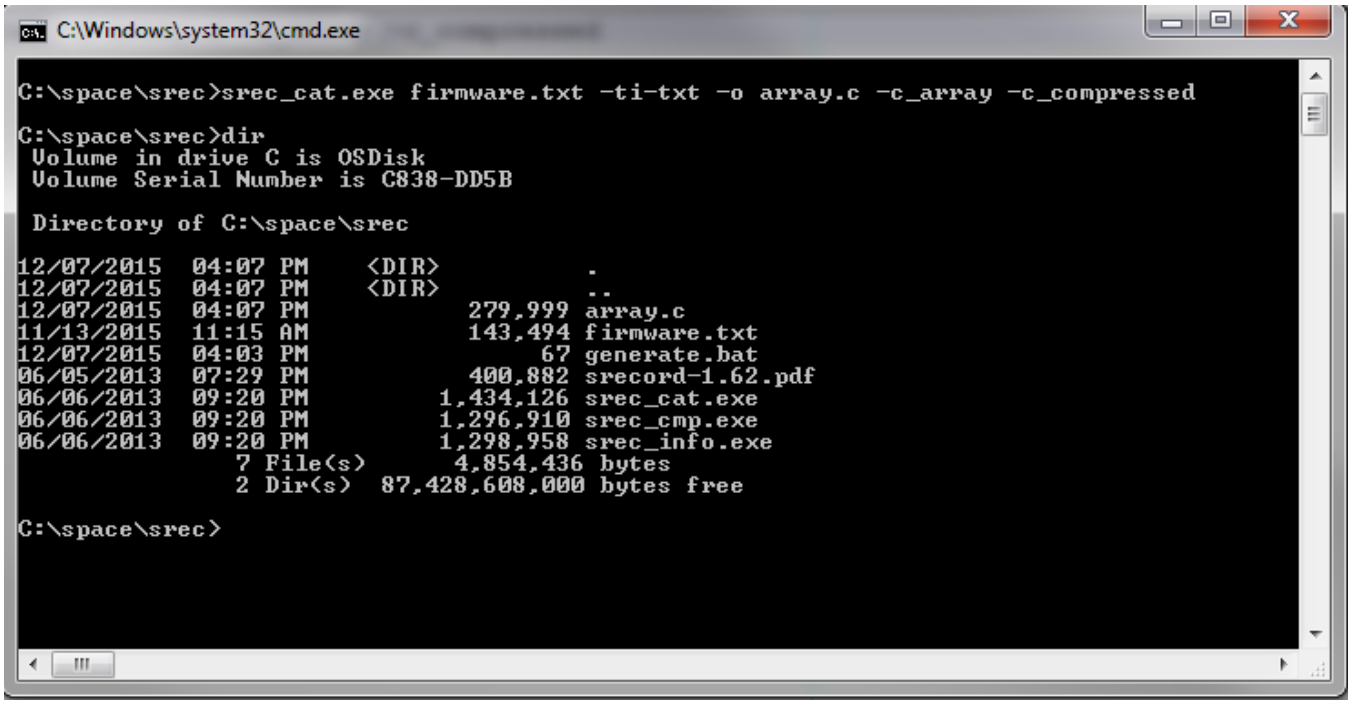

Figure 6. SRecord Invocation

In this example, SRecord accepts the firmware.txt file in TI-TXT format as a parameter and outputs a standard C file with a constant character array representation of the firmware. Information about the size of the firmware, number of segments, and size of each segment is also generated. Figure 7 shows an example output of this firmware information.

```
const unsigned long eprom_address[] =
{
0x00004400, 0x0000FF80, 0x0000FFCC,
};
const unsigned long eprom_length_of_sections[] =
{
0x0000B1E6, 0x00000010, 0x00000160,
};
const unsigned long eprom_sections    = 0x00000003;
const unsigned long eprom_termination = 0x00000000;
const unsigned long eprom_start       = 0x00004400;
const unsigned long eprom_finish      = 0x0001012C;
const unsigned long eprom_length      = 0x0000BD2C;

#define EPROM_TERMINATION 0x00000000
#define EPROM_START       0x00004400
#define EPROM_FINISH      0x0001012C
#define EPROM_LENGTH      0x0000BD2C
#define EPROM_SECTIONS    0x00000003
```

Figure 7. Output of SRecord

The *MSP430BSL_parseSRecordArray* function in the *firmware_parser.c* file accepts each of these parameters (including the firmware array) and generates the memory segment linked list structure. This linked list can then be parsed into any of the standard BSL functions provided in the i2cbsl.c file. Having the firmware integrated into a compiled C file is ideal for this package implementation because it does not require any user input or external file manipulation. The I^2C password is also integrated into the main.c file of this solution in the bslPassword array.

5 Testing on BeagleBone Black

For testing, a TI BeagleBone Black running the stock Debian distribution of Linux was used. Included in the root *src/* directory of the software package is a *Makefile*. This *Makefile* is recursive and compiles the individual components of the project when the make command is invoked. The hardware setup of the BeagleBone Black is ideal because the external pullup resistors required for I^2C communication are integrated on the BeagleBone hardware. For the device testing, an MSP430FR59691 connected to an MSP-TS430RGZ48C target board was used. Figure 8 shows this setup.

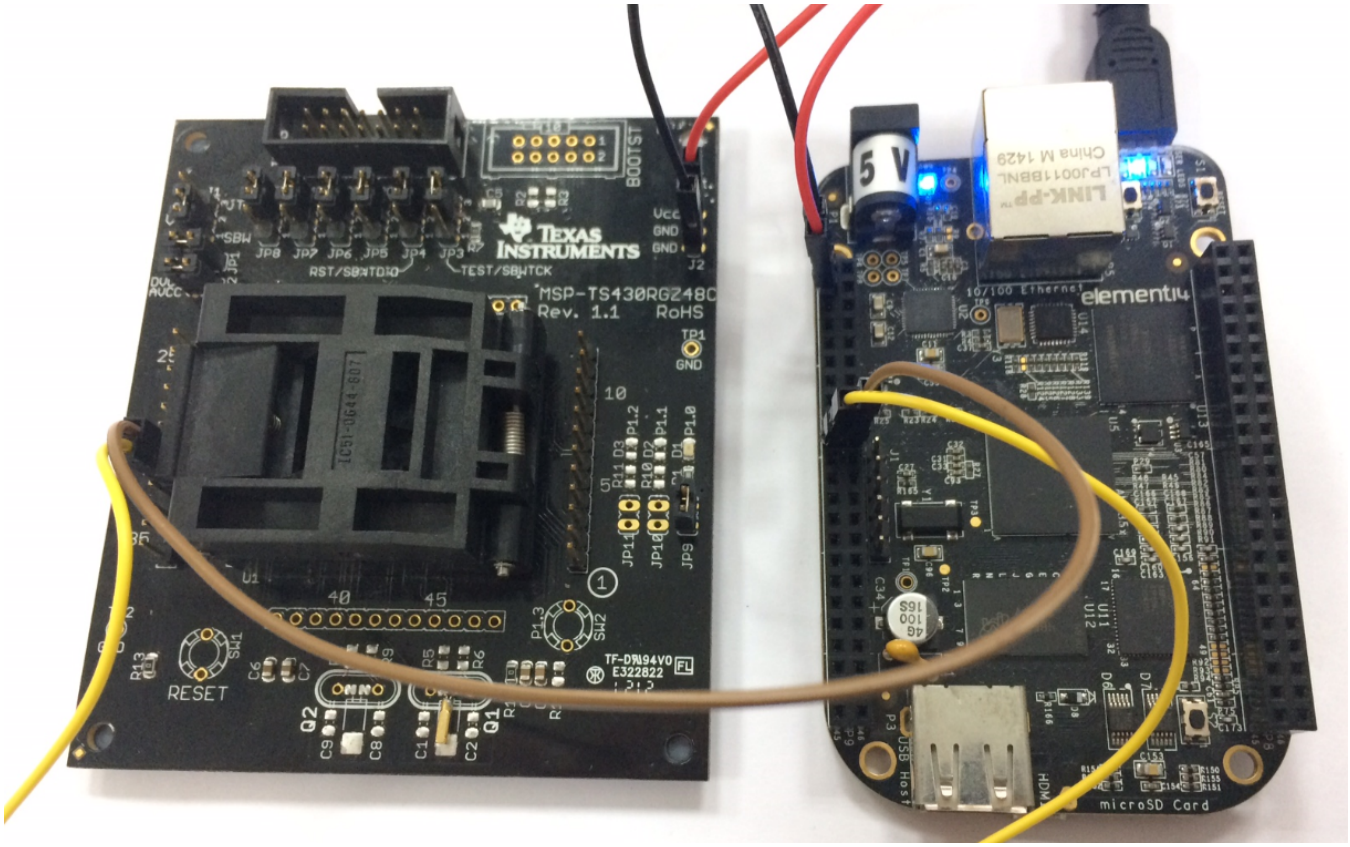

Figure 8. BeagleBone Black and MSP430 Test Setup

The I²C BSL command set varies between MSP430 devices, and certain commands that are available on one variant of the I²C BSL may not be available on a different variant. While functionality was fully tested and verified on MSP430FR59691, a certain level of integration might be required for another MSP430 device. For information on the variations of command structures and command availability between MSP430 devices, see the device-specific BSL documentation. For Linux distributions, any standard distribution of Linux works with the BSL tools without issue. No special external software libraries are used other than standard C libraries coupled with Linux ioctl calls. Figure 9 shows compilation through the GCC compiler on the BeagleBone Black.

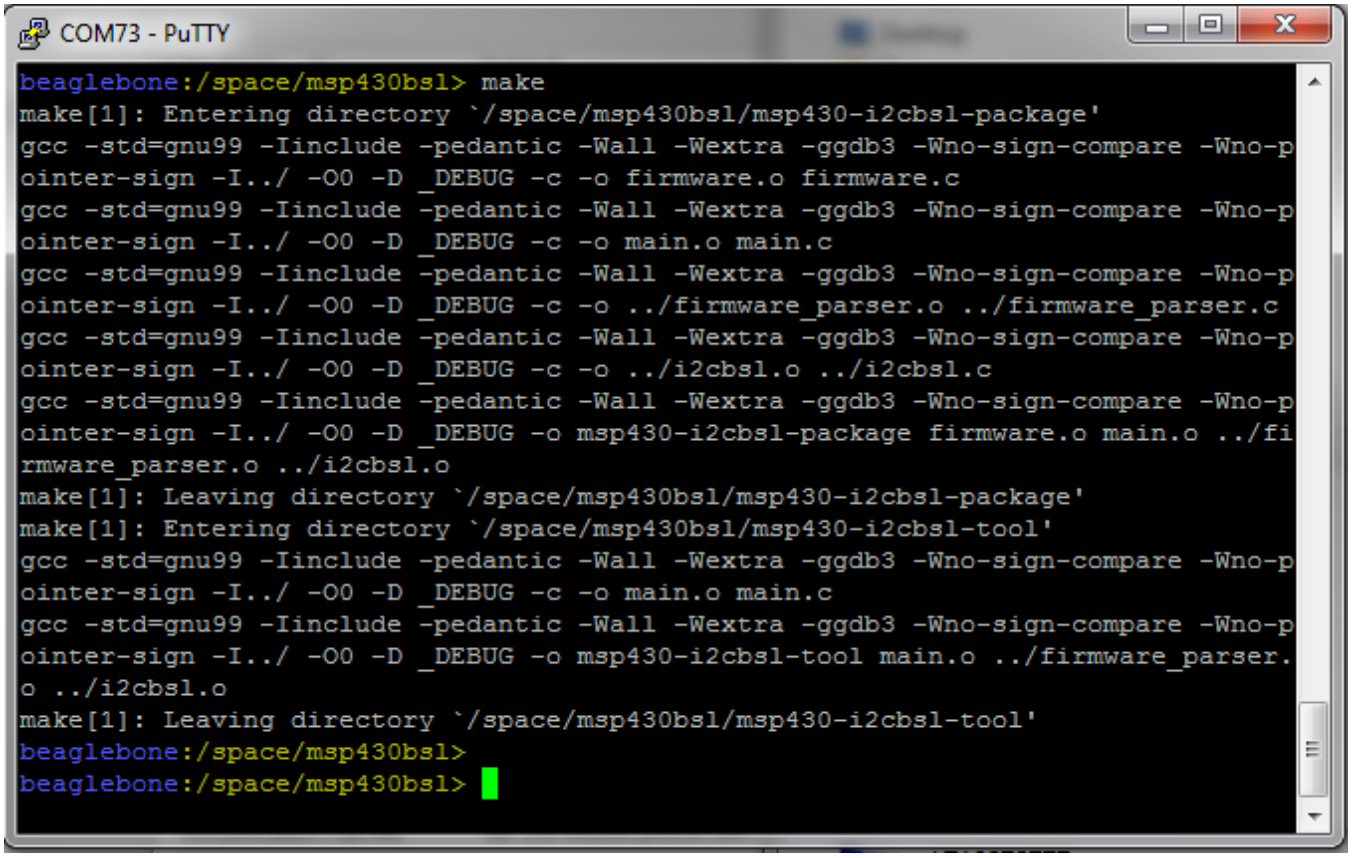

Figure 9. Compilation Through GCC

6 Porting to Other Platforms

TI designed the overall structure of this code to be portable to any embedded host system that can compile standard C code. Only standard C libraries are used and there are no references to nonstandard or proprietary libraries. When porting to other platforms, the only code that must be modified is the core I²C communication code included in the *MSP430BSL_I2CWriteRead* and *MSP430BSL_I2CWrite* functions of the i2cbsl.c file. These functions contain the low-level I²C calls that send or read bytes over the I²C line. This code is platform specific and must be changed according to the I²C communication method of the host. In this implementation, simple ioctl calls to the I2C_RDWR functionality of Linux are used to perform I²C transactions.

This code is focused on user space Linux calls. For porting to Linux kernel space, the *simplified package program* example implementation is the best reference. This implementation is beneficial for kernel-level modules because there is no user interaction and all update parameters and BSL parameters can be contained in a single encapsulated program. For a kernel space implementation, the low-level I²C ioctl calls must be changed to the i2c_transfer implementation. If a user wants to port the I²C code over to a Windows® or UEFI implementation, the low-level I²C communication code in the i2cbsl.c file must be changed to match the I²C APIs of the new platform.

7 References

- *MSP430FR57xx, MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Bootloader (BSL) User's Guide* (SLAU550)
- *Creating a Custom Flash-Based Bootstrap Loader (BSL)* (SLAA450)
- *MSP430FR59691 Product Folder*, http://www.ti.com/product/msp430fr59691
- *BeagleBone Black Homepage*, http://beagleboard.org/BLACK
- *SRecord Firmware Parser Tool*, http://srecord.sourceforge.net/

IMPORTANT NOTICE

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

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

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

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

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

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

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

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

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

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

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

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

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