LUMINARY MICRO®

RDK-BDC Firmware Development Package

USER'S GUIDE

SW-RDK-BDC-UG-4201

Legal Disclaimers and Trademark Information

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

Revision Information

This is version 4201 of this document, last updated on March 03, 2009.

Table of Contents

1 Introduction

The Brushed DC Motor Reference Design Kit (RDK-BDC) contains a brushed DC motor controller module (the MDL-BDC) and a LM3S2965 evaluation kit (the EK-LM3S2965). The MDL-BDC is pre-loaded with the same firmware as the bare modules, and the LM3S2965 evaluation kit is pre-loaded with an application that controls the motor via communications over the CAN bus.

With this kit, the capabilities and performance of the MDL-BDC can be evaluated. Additionally, the code for the LM3S2965 evaluation kit application can be used as reference for a custom application to control the MDL-BDC over the CAN network.

This document describes the CAN protocol and the example applications that are provided for this reference design board.

2 Example Applications

The boot loader (boot_can) and quickstart (qs-bdc) are programmed onto the MDL-BDC, and the user interface (bdc-ui) is programmed onto the LM3S2965 evaluation kit.

There is an IAR workspace file (`rdk-bdc.eww`) that contains the peripheral driver library project, along with the Brushed DC Motor Controller software project, in a single, easy to use workspace for use with Embedded Workbench version 5.

There is also an equivalent IAR workspace file (`rdk-bdc-ewarm4.eww`) for use with Embedded Workbench version 4.42a.

There is a Keil multi-project workspace file (`rdk-bdc.mpw`) that contains the peripheral driver library project, along with the Brushed DC Motor Controller software project, in a single, easy to use workspace for use with uVision.

All of these examples reside in the `boards/rdk-bdc` subdirectory of the firmware development package source distribution.

2.1 Brushed DC User Interface (bdc-ui)

This application provides a simple user interface for the Brushed DC Motor Controller board, running on the EK-LM3S2965 board and communicating over CAN. In addition to running the motor, the motor status can be viewed, the CAN network enumerated, and the motor controller's firmware can be updated.

The direction buttons (left, right, up, and down) on the left side of the EK-LM3S2965 are used to navigate through the user interface, and the select button on the right side of the EK-LM3S2965 is used to select items.

The user interface is divided into several panels; the top line of the display always contains the name of the current panel. By moving the cursor to the top line and pressing select, a menu is displayed which will allow a different panel to be displayed by pressing select again.

Of the control modes available via this application, only voltage control mode is usable with the motor and power supply provided with the RDK-BDC. In order to use current control mode, a larger motor (with an attached load) and power supply are required. In order to use speed control mode, a motor with an encoder is required. In order to use position control mode, a motor with an encoder or a potentiometer is required.

The panels in the user interface will be individually discussed below. At startup, the Voltage Control Mode panel is displayed first.

Voltage Control Mode

The voltage control mode panel allows the motor to be controlled by directly selecting the output voltage. The speed of the motor is directly proportional to the voltage applied, and applying a "negative" voltage (in other words, electronically reversing the power and ground connections) will result in the motor spinning in the opposite direction.

There are three parameters that can be adjusted on this panel: the ID, voltage, and ramp rate. The up and down buttons are used to select the parameter to be modified, and the left and right buttons are used to adjust the parameter's value. The following parameters can be adjusted:

- ID, which selects the motor controller to which commands are sent. If the ID is changed while the motor is running, the motor will be stopped.

If the select button is pressed, a demonstration mode will be enabled or disabled. In demonstration mode, the output voltage is automatically cycled through a sequence of values.

- Voltage, which specifies the output voltage sent from the motor controller to the motor. A positive voltage will result in voltage being applied to the white output terminal and ground being applied to the green output terminal, while a negative voltage will apply voltage to the green output terminal and ground to the white output terminal.

If the select button is pressed, changes to the output voltage will not be sent to the motor controller immediately (allowing the ramp to be used). The text color of the voltage changes from white to black to indicate that a deferred update is active. Pressing select again will send the final output voltage to the motor controller.

- Ramp, which specifies the rate of change of the output voltage. When set to “none”, the output voltage will change immediately. When set to a value, the output voltage is slowly changed from the current to to the target value at the specified rate. This can be used to avoid browning out the power supply or to avoid over-torquing the motor on startup (for example preventing a loss of traction when a wheel is being driven).

The bottom portion of the panel provides the current motor controller status.

Current Control Mode

The current control panel allows the motor to be controlled via closed-loop current control. The torque of the motor is directly proportional to the current applied, and applying a “negative” current will result in the motor spinning in the opposite direction.

There are five parameters that can be adjusted on this panel: the ID, current, and PID parameters. The up and down buttons are used to select the parameter to be modified, and the left and right buttons are used to adjust the parameter’s value. The following parameters can be adjusted:

- ID, which selects the motor controller to which commands are sent. If the ID is changed while the motor is running, the motor will be stopped.

If the select button is pressed, a demonstration mode will be enabled or disabled. In demonstration mode, the output current is automatically cycled through a sequence of values.

- Current, which specifies the output current sent from the motor controller to the motor. A positive current will result in voltage being applied to the white output terminal and ground being applied to the green output terminal, while a negative current will apply voltage to the green output terminal and ground to the white output terminal.

If the select button is pressed, changes to the output current will not be sent to the motor controller immediately (allowing an arbitrary step function to be applied). The text color of the current changes from white to black to indicate that a deferred update is active. Pressing select again will send the final output current to the motor controller.

- P coefficient, which specifies the gain applied to the instantaneous motor current error.

- I coefficient, which specifies the gain applied to the integral of the motor current error.

- D coefficient, which specifies the gain applied to the derivitive of the motor current error.

The bottom portion of the panel provides the current motor controller status.

Speed Control Mode

The speed control panel allows the motor to be controlled via closed-loop speed control. The voltage applied to the motor is varied in order to achieve a desired output speed. Applying a "negative" speed will result in the motor spinning in the opposite direction.

The speed control mode requires that an encoder be attached to the output of the motor, either directly to the motor's output shaft or at some stage within or after the gearbox that is optionally attached to the motor. Examples of encoders that can be used are quadrature encoders and gear tooth sensors. The speed will be regulated based on the measurement point; if the output speed of a gearbox is measured, then the motor will be running faster or slower than the desired speed (based on the gear ratio of the gearbox) in order to make the gearbox output match the set speed.

There are five parameters that can be adjusted on this panel: the ID, speed, and PID parameters. The up and down buttons are used to select the parameter to be modified, and the left and right buttons are used to adjust the parameter's value. The following parameters can be adjusted:

- ID, which selects the motor contorller to which commands are sent. If the ID is changed while the motor is running, the motor will be stopped.

If the select button is pressed, a demonstratino mode will be enabled or disabled. In demonstratino mode, the output speed is automatically cycled through a sequence of values.

- Speed, which specifies the speed that the motor should run. A positive speed will result in voltage being applied to the white output terminal and groud being applied to the green output terminal, while a negative speed will apply voltage to the green output terminal and ground to the white output terminal.

If the select button is pressed, changes to the output speed will not be sent to the motor controller immediately (allowing an arbitrary step function to be applied). The text color of the speed changes from white to black to indicate that a deferred update is active. Pressing select again will send the final output speed to the motor controller.

- P coefficient, which specifies the gain applied to the instantaneous motor speed error.
- I coefficient, which specifies the gain applied to the integral of the motor speed error.
- D coefficient, which specifies the gain applied to the derivitive of the motor speed error.

The bottom portion of the panel provides the current motor controller status.

Position Control Mode

The position control panel allows the motor to be controlled via closed-loop position control. The voltage applied to the motor is varied in order to move the shaft to a desired position. The motor will spin in either direction in order to achieve the requested position.

There are six parameters that can be adjusted on this panel: the ID, position, PID parameters, and position reference. The up and down buttons are used to select the parameter to be modified, and the left and right buttons are used to adjust the parameter's value. The following parameters can be adjusted:

- ID, which selects the motor controller to which commands are sent. If the ID is changed while the motor is running, the motor will be stopped.

If the select button is pressed, a demonstration mode will be enabled or disabled. In demonstratino mode, the motor position is automatically cycled through a sequence of values.

- Position, which specifies the position to which the motor should turn.

If the select button is pressed, changes to the position will not be sent to the motor contorller immediately (allowing an arbitrary step funciton to be applied). The text color of the position changes from white to black to indicate that a deferred update is active. Pressing select again will send the final output position to the motor controller.

- P coefficient, which specifies the gain applied to the instantaneous motor position error.
- I coefficient, which specifies the gain applied to the integral of the motor position error.
- D coefficient, which specifies the gain applied to the derivitive of the motor position error.
- Position reference, which specifies how the motor position is measured. If this is set to “encoder”, an encoder (such as a quadrature encoder) is used to measure the motor position (a gear tooth sensor can not be used with position control mode). If this is set to “potentiometer”, a potentiometer coupled to the motor output shaft (pre- or post-gearbox) is used to measure the motor position.

The bottom portion of the panel provides the current motor controller status.

Configuration

This panel allows general parameters of the motor controller to be configured. There are ten parameters that can be adjusted on this panel: the ID, number of encoder lines, number of potentiometer turns, brake or coast, soft limit switch enable, forward soft limit switch position, forward soft limit switch comparison, reverse soft limit switch position, reverse soft limit switch comparison, and maximum output voltage. The up and down buttons are used to select the parameter to be modified, and the left and right buttons are used to adjust the parameter’s value. The following parameters can be adjusted:

- ID, which selects the motor controller that is to be configured.
- Encoder lines, which specifies the number of lines in the attached encoder. When using a quadrature encoder, this number will match the clocks per revolution (CPR) specified by the encoder manufacturer. When using a gear tooth sensor, this will be twice the number of teeth in the gear that is being measured.
- Potentiometer turns, which specifies the number of full turn in the travel of the potentiometer. Typical potentiometers used for rotational measurement have one, three, five, or ten turns in their travel.
- Brake/coast, which specifies the action to be taken when the motor is stopped. This can be set to “jumper”, which uses the brake/coast jumper on the MDL-BDC to determine whether to brake or coast, “brake” to apply dynamic braking, and “coast” to electrically disconnect the motor windings and allow it to coast to a stop under the affects of friction.
- Soft limit, which specifies whether or not the soft limit switches are enabled. When enabled, the soft limit switches use measured motor position to prevent the motor from running forward or backward. For positioning applications, which require either an encoder or a potentiometer, this allows the use of limit switches (to prevent rotational extremes, thereby protecting the attached assembly) without the need to physically place and wire up real switches.

- Forward limit, which specifies the motor position that corresponds to the position of the forward soft limit switch.
- Forward compare, which specifies the comparison applied to the forward soft limit switch. This will be "lt" if the motor position must be less than the position of the forward limit switch in order to run forward, or "gt" if it must be greater. The "lt" setting will be used for setups where positive voltage applied to the motor results in the measured motor position increasing, and "gt" will be used for setups where positive voltage results in the measured motor position decreasing.
- Reverse limit, which specifies the motor position that corresopnds to the position of the reverse soft limit switch.
- Reverse compare, which specifies the comparison applied to the reverse soft limit switch. This will be "lt" if the motor position must be less than the position of the reverse limit switch in order to run backward, or "gt" if it must be greater. The "gt" setting will be used for setups where positive voltage applied to the motor results in the measured motor position increasing, and "lt" will be used for setups where positive voltage results in the measured motor position decreasing.
- Maximum output voltage, which specifies the maximum voltage that can be safely applied to the attached motor. All voltage commands are scaled such that a "full scale" voltage output matches this value. This can be used to attach 7.2V motor (for example) to the MDL-BDC and avoid applying 12V to them.

Device List

This panel lists the motor controllers that reside on the CAN network. All 63 possible device IDs are listed, with those that are not present shown in a dark gray and those that are present in a bright white. By moving the cursor to a particular ID and pressing the select button, a device ID assignment will be performed. The motor controller(s) will wait for five seconds after an assignment request for its button to be pressed, indicating that it should accept the device ID assignment. So, for example, if there are three motor controllers on a network, the following sequence can be used to give them each unique IDs:

- Move the cursor to number 1 and press select. The LED on all three motor controllers will blink green to indicate that assignment mode is active.
- Press the button on one of the motor controllers. It will blink its LED yellow one time to indicate that it's ID is one.
- Move the cursor to number 2 and press select.
- Press the button on the second motor controller. It will blink its LED yellow two times to indicate that it's ID is two.
- Move the cursor to number 3 and press select.
- Press the button on the third motor controller. It will blink its LED yellow three times to indicate that it's ID is three.

Once complete, this panel will then show that there are devices at IDs 1, 2, and 3.

Firmware Update

This panel allows the firmware on the motor controller to be updated over the CAN network. A firmware image for the motor controller is stored in the flash of the EK-LM3S2965 board and used to update the motor controller. The local copy of the motor controller firmware can be updated using the UART boot loader protocol to transfer the image from a PC. When the UART “update” begins, this panel will automatically be displayed.

The ID of the motor controller to be updated can be changed on this panel. By using the local firmware image, multiple motor controllers can be updated (one at a time) using this panel, without the need to redownload from a PC each time.

When not updating, the firmware version of the currently selected motor controller will be displayed. If there is no motor controller on the CAN network with the current ID, the firmware version will be displayed as “—”.

By pressing the select button when the “Start” button is highlighted, the update of the motor controller firmware will commence.

When the firmware is being transferred (either from the PC using the UART or to the motor controller using the CAN network), the ID, firmware version, and “Start” buttons will all be grayed out. A progress bar will appear below them to indicate what is happening and the how far it is through the process.

Help

This panel displays a condensed version of this application help text. Use the up and down buttons to scroll through the text.

About

This panel simply displays the startup splash screen.

2.2 Boot Loader (boot_can)

The boot loader is a small piece of code that can be programmed at the beginning of flash to act as an application loader as well as an update mechanism for an application running on a Stellaris microcontroller. The capabilities of the boot loader are configured via the bl_config.h include file. For this example, the boot loader uses CAN to load an application.

2.3 Brushed DC Motor Controller (qs-bdc-3330)

This application controls a brushed DC motor. Two communication methods are supported; a hobby servo-style input for basic voltage control or a CAN input for more advanced control (the two inputs are mutually exclusive).

When using either communication methods, a basic voltage control mode is available. In this mode, the external controller directly specifies the desired output voltage. When using CAN, an output voltage slew rate can be specified, which results in the output voltage adjusting in a linear fashion from the current voltage to the new voltage (as opposed to directly jumping to the new voltage if the slew rate is disabled).

Additional advanced control methods are also available when using the CAN communication interface. There are current control mode, speed control mode, and position control mode. Each of these modes is mutually exclusive and operating using a PID controller whose gains are fully programmable via the CAN interface. Each PID controller starts with all of its gains set to zero, so no output voltage will be generated by any of these modes until the PID controller is at least partially configured.

In speed control mode, the speed of the motor is measured using the quadrature encoder input. Only PHA is used for measuring the speed, so it can be used with gear tooth sensors as well (which only provide a single pulse stream, not a quadrature pair as provided by a shaft encoder).

In position control mode, the position of the motor can be measured using the quadrature encoder input or the analog input. When using the analog input, a 10K potentiometer must be coupled to the output shaft of the motor (either before or after gearing) in some manner so that the motor position can be tracked.

The status of the motor controller can also be monitored over the CAN interface. The bus voltage, output voltage, motor current, ambient temperature, speed, position, limit switch values, fault status, power status, and firmware version can all be queried.

This version of the firmware corresponds to the pre-programmed firmware on the MDL-BDC.

2.4 Brushed DC Motor Controller (qs-bdc)

This application controls a brushed DC motor. Two communication methods are supported; a hobby servo-style input for basic voltage control or a CAN input for more advanced control (the two inputs are mutually exclusive).

When using either communication methods, a basic voltage control mode is available. In this mode, the external controller directly specifies the desired output voltage. When using CAN, an output voltage slew rate can be specified, which results in the output voltage adjusting in a linear fashion from the current voltage to the new voltage (as opposed to directly jumping to the new voltage if the slew rate is disabled).

Additional advanced control methods are also available when using the CAN communication interface. There are current control mode, speed control mode, and position control mode. Each of these modes is mutually exclusive and operating using a PID controller whose gains are fully programmable via the CAN interface. Each PID controller starts with all of its gains set to zero, so no output voltage will be generated by any of these modes until the PID controller is at least partially configured.

In speed control mode, the speed of the motor is measured using the quadrature encoder input. Only PHA is used for measuring the speed, so it can be used with gear tooth sensors as well (which only provide a single pulse stream, not a quadrature pair as provided by a shaft encoder).

In position control mode, the position of the motor can be measured using the quadrature encoder input or the analog input. When using the analog input, a 10K potentiometer must be coupled to the output shaft of the motor (either before or after gearing) in some manner so that the motor position can be tracked.

The status of the motor controller can also be monitored over the CAN interface. The bus voltage, output voltage, motor current, ambient temperature, speed, position, limit switch values, fault status, power status, and firmware version can all be queried.

This version of the firmware contains enhancements not present in the pre-programmed firmware on the MDL-BDC. These enhancements are:

- Addition of the System Halt, System Reset, and System Resume commands.

- Addition of the Maximum Output Voltage command to allow motors with lower voltage ratings (such as 7.2V motors) to be used.
- Addition of the ability to read the value of the motor controller's parameters.

3 Development System Utilities

These are tools that run on the development system, not on the embedded target. They are provided to assist in the development of firmware for Stellaris microcontrollers.

These tools reside in the `tools` subdirectory of the firmware development package source distribution.

FreeType Rasterizer

Usage:

`ftrasterize [OPTION]... [INPUT FILE]`

Description:

Uses the FreeType font rendering package to convert a font into the format that is recognized by the graphics library. Any font that is recognized by FreeType can be used, which includes TrueType®, OpenType®, PostScript® Type 1, and Windows® FNT fonts. A complete list of supported font formats can be found on the FreeType web site at `http://www.freetype.org`.

FreeType is used to render the glyphs of a font at a specific size in monochrome, using the result as the bitmap images for the font. These bitmaps are compressed and the results are written as a C source file that provides a tFont structure describing the font.

The source code for this utility is contained in `tools/ftrasterize`, with a pre-built binary contained in `tools/bin`.

Arguments:

-b specifies that this is a bold font. This does not affect the rendering of the font, it only changes the name of the file and the name of the font structure that are produced.

-f FILENAME specifies the base name for this font, which is used to create the output file name and the name of the font structure. The default value is "font" if not specified.

-i specifies that this is an italic font. This does not affect the rendering of the font, it only changes the name of the file and the name of the font structure that are produced.

-s SIZE specifies the size of this font, in points. The default value is 20 if not specified.

INPUT FILE specifies the name of the input font file.

Example:

The following example produces a 24-point font called test from test.ttf:

```
ftrasterize -f test -s 24 test.ttf
```

The result will be written to `fonttest24.c`, and will contain a structure called `g_sFontTest24` that describes the font.

GIMP Script For Luminary Micro Button

Description:

This is a script-fu plugin for GIMP (`http://www.gimp.org`) that produces push button images that can be used by the push button widget. When installed into

`${HOME}/.gimp-2.4/scripts`, this will be available under Xtns->Buttons->LMI Button. When run, a dialog will be displayed allowing the width and height of the button, the radius of the corners, the thickness of the 3D effect, the color of the button, and the pressed state of the button to be selected. Once the desired configuration is selected, pressing OK will create the push button image in a new GIMP image. The image should be saved as a raw PPM file so that it can be converted to a C array by `pnmtoc`.

This script is provided as a convenience to easily produce a particular push button appearance; the push button images can be of any desired appearance.

This script is located in `tools/lmi-button/lmi-button.scm`.

String Table Generator

Usage:

```
mkstringtable [INPUT FILE] [OUTPUT FILE]
```

Description:

Converts a comma separated file (.csv) to a table of strings that can be used by the Luminary Micro Graphics Library. The source .csv file has a simple fixed format that supports multiple strings in multiple languages. A .c and .h file will be created that can be compiled in with an application and used with the graphics library's string table handling functions. The strings will also be compressed in order to reduce the space required to store them.

The format of the input .csv file is simple and easily edited in any plain text editor or a spreadsheet editor capable of reading and editing a .csv file. The .csv file format has a header row where the first entry in the row can be any string as it is ignored. The remaining entries in the row must be one of the GrLang* language definitions defined by the graphics library in `grlib.h` or they must have a `#define` definition that is valid for the application as this text is used directly in the C output file that is produced. Adding additional languages only requires that the value is unique in the table and that the name used is defined by the application.

The strings are specified one per line in the .csv file. The first entry in any line is the value that is used as the actual text for the definition for the given string. The remaining entries should be the strings for each language specified in the header. Single words with no special characters do not require quotations, however any strings with a "," character must be quoted as the "," character is the delimiter for each item in the line. If the string has a quote character """ it must be preceded by another quote character.

The following is an example .csv file containing string in English (US), German, Spanish (SP), and Italian:

```
LanguageIDs,GrLangEnUS,GrLangDE,GrLangEsSP,GrLangIt
STR_CONFIG,Configuration,Konfigurieren,Configuracion,Configurazione
STR_INTRO,Introduction,Einfuhrung,Introduccion,Introduzione
STR_QUOTE,Introduction in ""English"","Einfuhrung, in Deutch",Prueba,Verifica
...
```

In this example, `STR_QUOTE` would result in the following strings in the various languages:

- GrLangEnUs – `Introduction in "English"`
- GrLangDE – `Einfuhrung, in Deutch`
- GrLangEsSP – `Prueba`
- GrLangIt – `Verifica`

The resulting .c file contains the string table that must be included with the application that is using the string table. While the contents of this .c file are readable, the string table itself may be unintelligible due to the compression used on the strings themselves. The .h file that is created has the definition for the string table as well as an enumerated type `enum SCOMP_STR_INDEX` that contains all of the string indexes that were present in the original .csv file.

The code that uses the string table produced by this utility must refer to the strings by their identifier in the original .csv file. In the example above, this means that the value `STR_CONFIG` would refer to the "Configuration" string in English (GrLangEnUS) or "Konfigurieren" in German (GrLangDE).

This utility is contained in `tools/bin`.

Arguments:

INPUT FILE specifies the input .csv file to use to create a string table.

OUTPUT FILE specifies the root name of the output files as `<OUTPUT FILE>.c` and `<OUTPUT FILE>.h`. The value is also used in the naming of the string table variable.

Example:

The following will create a string table in `str.c`, with prototypes in `str.h`, based on the input file `str.csv`:

```
mkstringtable str.csv str
```

In the produced `str.c`, there will be a string table in `g_pucTablestr`.

NetPNM Converter

Usage:

```
pnmtoc [OPTION]... [INPUT FILE]
```

Description:

Converts a NetPBM image file into the format that is recognized by the Luminary Micro Graphics Library. The input image must be in the raw PPM format (in other words, with the `P6` tag). The NetPBM image format can be produced using GIMP, NetPBM (`http://netpbm.sourceforge.net`), ImageMagick (`http://www.imagemagick.org`), or numerous other open source and proprietary image manipulation packages.

The resulting C image array definition is written to standard output; this follows the convention of the NetPBM toolkit after which the application was modeled (both in behavior and naming). The output should be redirected into a file so that it can then be used by the application.

To take a JPEG and convert it for use by the graphics library (using GIMP; a similar technique would be used in other graphics programs):

1. Load the file (File->Open).
2. Convert the image to indexed mode (Image->Mode->Indexed). Select "Generate optimum palette" and select either 2, 16, or 256 as the maximum number of colors (for a 1 BPP, 4 BPP, or 8 BPP image respectively). If the image is already in indexed mode, it can be converted to RGB mode (Image->Mode->RGB) and then back to indexed mode.
3. Save the file as a PNM image (File->Save As). Select raw format when prompted.
4. Use `pnmtoc` to convert the PNM image into a C array.

This sequence will be the same for any source image type (GIF, BMP, TIFF, and so on); once loaded into GIMP, it will treat all image types equally. For some source images, such as a GIF which is naturally an indexed format with 256 colors, the second step could be skipped if an 8 BPP image is desired in the application.

The source code for this utility is contained in `tools/pnmtoc`, with a pre-built binary contained in `tools/bin`.

Arguments:

-c specifies that the image should be compressed. Compression is bypassed if it would result in a larger C array.

Example:

The following will produce a compressed image in foo.c from foo.ppm:

```
pnmtoc -c foo.ppm > foo.c
```

This will result in an array called `g_pucImage` that contains the image data from `foo.ppm`.

Serial Flash Downloader

Usage:

`sflash [OPTION]... [INPUT FILE]`

Description:

Downloads a firmware image to a Stellaris board using a UART connection to the Stellaris Serial Flash Loader or the Stellaris Boot Loader. This has the same capabilities as the serial download portion of the Luminary Micro Flash Programmer.

The source code for this utility is contained in `tools/sflash`, with a pre-built binary contained in `tools/bin`.

Arguments:

-b BAUD specifies the baud rate. If not specified, the default of 115,200 will be used.

-c PORT specifies the COM port. If not specified, the default of COM1 will be used.

-d disables auto-baud.

-h displays usage information.

-l FILENAME specifies the name of the boot loader image file.

-p ADDR specifies the address at which to program the firmware. If not specified, the default of 0 will be used.

-r ADDR specifies the address at which to start processor execution after the firmware has been downloaded. If not specified, the processor will be reset after the firmware has been downloaded.

-s SIZE specifies the size of the data packets used to download the firmware date. This must be a multiple of four between 8 and 252, inclusive. If using the Serial Flash Loader, the maximum value that can be used is 76. If using the Boot Loader, the maximum value that can be used is dependent upon the configuration of the Boot Loader. If not specified, the default of 8 will be used.

INPUT FILE specifies the name of the firmware image file.

Example:

The following will download a firmware image to the board over COM2 without auto-baud support:

```
sflash -c 2 -d image.bin
```

4 CAN Interface

The RDK-BDC has the ability to use CAN for configuration and real time control of the motor controller. The interface allows for connecting up to 63 uniquely identifiable devices on the CAN network. The CAN interface uses a well defined interface for accessing any devices on the network. The basic interfaces provided by the CAN interface are the following:

- Firmware Update
- Allows for Voltage, Current, Speed and Position control modes.
- Allows for configuration of parameters for all control modes.
- System enumeration of devices.
- Motor status information in all modes.

The CAN interface provides a number of commands and divides them into groups based on the type of command. The commands are grouped according to broadcast messages, system level commands, motor control commands based on control type, configuration commands and motor control status information. The interface also provides a method to extend the network protocol to other devices by defining a CAN device encoding that takes into account device type and manufacturer.

4.1 CAN Device Encoding

The CAN interface uses the message object identifier to specify which device as well as the type of command being sent to a device on the CAN network. The CAN interface allows for 63 different nodes on the CAN network numbered from 1 to 63. It also allows the device type, device manufacturer and command type to be included with any access to the CAN interface. The CAN interface uses all of these values to uniquely identify CAN devices on the CAN network. The CAN interface has some pre-defined manufacturer and device types that are shown in the table below.

The CAN interface uses message identifiers that are the 29 bit versions (extended frame format) for communication over the CAN bus. The message identifier field is transmitted in each CAN message and uniquely identifies the purpose of the message and contributes to CAN bus arbitration. As with any CAN communications, the message identifier is used in the arbitration of messages on the CAN bus. This makes the lowest value message identifier the highest priority message. Accordingly, the message identifier's assignment is done in a manner consistent with this CAN bus fundamental mechanism. Individual motor controller devices are also addressable within the identifier so that parameters contained within the data portion of the CAN packet may be provided to specific devices on the CAN network. The 29-bit message identifier is divided into the following fields:

Table 4.1: Message Identifier Fields.

Byte 3								Byte 2								Byte 1								Byte 0							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSVD			Device Type					Manufacturer								API										Device Number					

The Device Type field occupies the most-significant 5 bits of the message identifier. The Device Type encoding of 0 is reserved for system broadcast messages that are intended to reach all devices on the CAN network.

Table 4.2: Device Type Encodings.

Value	Encoding
0	Broadcast Messages
1	Robot Controller
2	Motor Controller
3	Relay Controller
4	Gyro Sensor
5	Accelerometer Sensor
6	Ultrasonic Sensor
7	Gear Tooth Sensor
8-30	Reserved.
31	Firmware Update.

The Manufacturer field is an 8-bit field that identifies the manufacturer of a device or controller. The Manufacturer encoding of 0 is reserved for system messages and is used for broadcast messages.

Table 4.3: Manufacturer Encodings.

Value	Encoding
0	Broadcast Messages
1	National Instruments
2	Luminary Micro
3	DEKA
4-255	Reserved

The APIs are defined as two fields that allow for grouping of the APIs into classes and providing an index to APIs within that class. The first 6 bit field is the API class and the second 4 bit field is the API index which determines which class API to use. The table below contains all of the currently defined API classes.

Table 4.4: API Class.

Value	Encoding
0	Voltage Control
1	Speed Control
2	Position Control
3	Current Control
4	Status
5	Configuration
6	Acknowledge
7-63	Reserved

4.2 System Control Interface

The functions in the system control group are CAN APIs that are implemented for all devices. The system control message are the highest priority CAN messages and are not specific to a given device manufacturer or device type. The table below shows a listing of the system control commands.

Table 4.5: System Control APIs.

Value	API Name
0	System Halt
1	System Reset
2	Device Assignment
3	Device Query
5	Heartbeat
6	Synchronous Update
7	Firmware Update
8	Firmware Version
9	Enumeration
10	System Resume
11-15	Reserved

System Halt

Upon receiving this message the motor controller will stop driving the motor and go to a neutral state. The motor can not be driven again until either a System Reset or System Resume has been received.

This command is not available in the 3330 version of the MDL-BDC firmware.

System Reset

Upon receiving this message the motor controller will stop driving the motor, go to a neutral state, and reset internal settings to their boot settings.

This command is not available in the 3330 version of the MDL-BDC firmware.

Device Assignment

This message is used to assign an identifier to a motor controller. This message is typically sent from the bus controller when it first configures the CAN network. When the motor controller receives this message it will enter the assignment state and should remain in this state for 5 seconds or until it accepts the new device number. The motor controller and the CAN device that issued the device assignment command should not generate any other CAN traffic during this time with the exception of heartbeat commands. If the device number that was sent with this command is zero then all motor controllers will set their device number to zero and leave the assignment state. The two remaining case are the device matched the motor controller's current device number or it did not match. If the number did not match then the motor controller waits for five seconds for a button press and if the button is pressed, it will accept the assignment and configure itself to use the new device number and store the device number so that it is used after the next power cycle. If the device number matched the motor controller's device number then the motor controller will reset its current device number to zero and continue to wait for a button press. If no button press occurs, then the motor controller will have reset its current device number and will need to be reassigned to a new device number.

Device Query

This command indicates that the motor controller should return some basic information about itself. This command uniquely addresses a device and only the addressed device will respond to this message. In response to this message, the motor controller will send back eight bytes of data. The first byte indicating the motor controller's function and the second indicates the manufacturer. The remaining bytes are reserved for future use.

Heartbeat

When this command is received the motor controller will reset its timeout for receiving CAN communications. This message is sent to the controller on a periodic basis to keep the CAN link active. If a CAN message is not received after 100ms, the motor controller will assume that the link is broken and enter a fault state, causing the motor controller to go into neutral.

Synchronous Update

This command allows for up to eight groups of devices to be simultaneously updated with a single command. The one byte of data that is sent with this command serves as a bit mask of the groups that should be updated. If there is a match then the motor controller will apply its pending updates. Because the value is used as a bit mask, the controller can be in 1 or all of the 8 groups. Synchronized updates provide two advantages. First, if the next value is known ahead of time, the next value can be transmitted earlier. Second, synchronized updates can provide finer coordination between motion controllers.

Firmware Update

This command is sent to a specific device to initiate a firmware update. After receiving this command the motor controller will enter the CAN boot loader update and follow that protocol to update the firmware.

Firmware Version

This command is sent to request the current firmware version for the motor controller. This command uniquely addresses a device and only the addressed device will respond to this message. The motor controller will send back four bytes of data that indicate the firmware version of the motor controller.

Enumeration

This command causes the motor controller to send out a response to indicate that device is present on the CAN network. In order to prevent all devices from responding at once, the motor controllers will wait for (device number) $*$ 1ms after the enumerate command before responding. Once enumeration has been started, the CAN device that requested the enumeration sequence should wait at least 80ms before generating any other CAN traffic to avoid affecting the enumeration sequence. After the enumeration sequence is complete, normal CAN activity should resume allowing the motor controllers to keep their CAN links active.

System Resume

Upon receiving this message the motor controller will return to normal operation, cancelling a previous System Halt message.

This command is not available in the 3330 version of the MDL-BDC firmware.

4.3 Voltage Control Interface

This group of commands is used to operate the motor controller with direct control of the motor voltage. The basic commands consist of enable/disable of voltage mode, setting the voltage and the voltage ramp rate.

Table 4.6: Voltage Mode APIs.

Value	API Name
0	Voltage Mode Enable
1	Voltage Mode Disable
2	Voltage Set
3	Voltage Ramp Set
4-15	Reserved

Voltage Mode Enable

This command is used to initialize the operational mode of the motor controller to voltage control mode. In response to this message, the motor controller sets its output voltage to neutral.

Voltage Mode Disable

This command is used to disable voltage control mode if it was in use by the motor controller. In response to this command, the motor controller returns to voltage control mode and sets its output voltage to neutral.

Set Output Voltage

This command is used to set the current voltage of the motor controller in voltage control mode. The first parameter is a 16 bit 8.8 signed fixed point number and is always provided to specify the output voltage. The second parameter is an optional 8 bit value and specifies the synchronization group to be used. If the second parameter is not included or is zero the voltage update is immediate. If a motor controller receives a new voltage command with a synchronization group while an existing update is pending then the existing update is overwritten.

If this command is sent with no data, the motor controller will respond by sending this same message to report the voltage set point. This feature is not available in the 3330 version of the MDL-BDC firmware.

Set Voltage Ramp Rate

This command is used to limit ramp rate of the output voltage over an extended period of time. The first parameter is as a 16 bit 8.8 unsigned fixed point number that indicates the maximum rate of change for the voltage. A value of 0 disables ramping if was previously enabled. Enabling the voltage ramp allows the motor controller to ramp the output voltage is useful to avoid excessive current draw when changing motor speeds rapidly.

If this command is sent with no data, the motor controller will respond by sending this same message to report the voltage ramp rate. This feature is not available in the 3330 version of the MDL-BDC firmware.

4.4 Speed Control Interface

This set of commands is used to configure and operate the motor controller at specific speed settings. The motor control speed commands consist of enable/disable of speed mode, setting the motor speed, setting the PID loop control parameters and setting the encoder source for detecting speed.

Table 4.7: Speed Mode APIs.

Value	API Name
0	Speed Mode Enable
1	Speed Mode Disable
2	Speed Set
3	Speed Proportional Constant
4	Speed Integral Constant
5	Speed Differential Constant
6	Speed Reference
7-15	Reserved

Speed Mode Enable

This command sets the operational mode of the motor controller to speed control. In response to this command, the motor controller sets its output speed neutral.

Speed Mode Disable

This command disables speed control mode of the motor controller and returns the controller to the default control mode. In response to this message, the motor controller sets its output voltage to neutral.

Speed Set

This command sets the motor controller target rotational speed. The first parameter is a 32 bit 16.16 signed fixed point value that specifies the rotational speed in revolutions per seconds. The second parameter is an optional 8 bit value that specifies the synchronization group to be used. If the second parameter is not included or is zero the voltage update is immediate. If a motor controller receives a new speed command with a synchronization group while an existing update is pending then the existing update is overwritten.

If this command is sent with no data, the motor controller will respond by sending this same message to report the speed set point. This feature is not available in the 3330 version of the MDL-BDC firmware.

Speed Proportional Constant

The command sets the proportional constant used in the PID algorithm calculations used for speed control. The proportional constant is a 32 bit 16.16 signed fixed point number.

If this command is sent with no data, the motor controller will respond by sending this same message to report the speed proportional constant. This feature is not available in the 3330 version of the MDL-BDC firmware.

Speed Integral Constant

The command sets the integral constant used in the PID algorithm calculations used for speed control. The integral constant is a 32 bit 16.16 signed fixed point number.

If this command is sent with no data, the motor controller will respond by sending this same message to report the speed integral constant. This feature is not available in the 3330 version of the MDL-BDC firmware.

Speed Differential Constant

The command sets the differential constant used in the PID algorithm calculations used for speed control. The differential constant is a 32 bit 16.16 signed fixed point number.

If this command is sent with no data, the motor controller will respond by sending this same message to report the speed differential constant. This feature is not available in the 3330 version of the MDL-BDC firmware.

Speed Reference

This command sets the speed reference used for measuring the current speed of the motor. The only speed reference at this time is for the motor controller to use an encoder to calculate its current speed.

If this command is sent with no data, the motor controller will respond by sending this same message to report the speed reference. This feature is not available in the 3330 version of the MDL-BDC firmware.

4.5 Position Control Interface

This set of commands is used to configure and operate the motor controller using position control mode. The motor control position commands consist of enable/disable of position mode, setting the motor position, setting the PID loop control parameters and setting the source for detecting position. Position control is managed through the use of an externally attached optical encoder combined with the motor controller's encoder inputs or through the use of a potentiometer.

Table 4.8: Position Mode APIs.

Value	API Name
0	Position Mode Enable
1	Position Mode Disable
2	Position Set
3	Position Proportional Constant
4	Position Integral Constant
5	Position Differential Constant
6	Position Reference
7-15	Reserved

Position Control Mode Enable

This command sets the operational mode of the motor controller to positional control mode. In response to this command, the motor controller sets its position to the 32 bit parameter provided with the position control enable command. This sets the original position of the motor to a known position. This is necessary for systems that may move in either direction on the initial position target.

Position Control Mode Disable

This command disables position control mode of the motor controller and returns the controller to the default control mode. In response to this message, the motor controller sets its output voltage to neutral.

Position Set

This command sets the target shaft position of the attached motor. The first parameter specifies the position set point as a 32 bit value. The second parameter is an optional 8 bit value that specifies the synchronization group to be used. If the second parameter is not included or is zero the position update is immediate. If a motor controller receives a new position command while an existing update is pending then the existing update is overwritten.

If this command is sent with no data, the motor controller will respond by sending this same message to report the position set point. This feature is not available in the 3330 version of the MDL-BDC firmware.

Position Proportional Constant

The command sets the proportional constant used in the PID algorithm calculations used for position control. The proportional constant is a 32 bit 16.16 signed fixed point number.

If this command is sent with no data, the motor controller will respond by sending this same message to report the position proportional constant. This feature is not available in the 3330 version of the MDL-BDC firmware.

Position Integral Constant

The command sets the integral constant used in the PID algorithm calculations used for speed control. The integral constant is a 32 bit 16.16 signed fixed point number.

If this command is sent with no data, the motor controller will respond by sending this same message to report the position integral constant. This feature is not available in the 3330 version of the MDL-BDC firmware.

Position Differential Constant

The command sets the differential constant used in the PID algorithm calculations used for speed control. The differential constant is a 32 bit 16.16 signed fixed point number.

If this command is sent with no data, the motor controller will respond by sending this same message to report the position differential constant. This feature is not available in the 3330 version of the MDL-BDC firmware.

Position Reference

This command sets the position reference used for measuring the current position of the motor. The position references supported at this time are an encoder or a potentiometer.

If this command is sent with no data, the motor controller will respond by sending this same message to report the position reference. This feature is not available in the 3330 version of the MDL-BDC firmware.

4.6 Current Control Interface

This set of commands is used to configure and operate the motor controller using current control mode. The motor control current commands consist of enable/disable of current mode, setting the motor current, setting the PID loop control parameters and setting the source for detecting current. Current control is managed through the use of an externally attached optical encoder combined with the motor controller's encoder inputs.

Table 4.9: Current Mode APIs.

Value	API Name
0	Current Mode Enable
1	Current Mode Disable
2	Current Set
3	Current Proportional Constant
4	Current Integral Constant
5	Current Differential Constant
6-15	Reserved

Current Mode Enable

This command sets the operational mode of the motor controller to current control. In response to this command, the motor controller sets its output current to a neutral setting.

Current Mode Disable

This command is used to disables current control mode if it was in use by the motor controller. In response to this command, the motor controller returns to the default control mode and sets its output to neutral.

Current Set

This command sets the target winding current of the attached motor. The first parameter specifies the current set point as a 16.16 fixed point number. The second parameter is an optional 8 bit value and specifies the synchronization group to be used. If the second parameter is not included or is zero the position update is immediate. If a motor controller receives a new current set command while an existing update is pending then the existing update is overwritten.

If this command is sent with no data, the motor controller will respond by sending this same message to report the current set point. This feature is not available in the 3330 version of the MDL-BDC firmware.

Current Proportional Constant

The command sets the current constant used in the PID algorithm calculations used for current control. The proportional constant is a 32 bit 16.16 signed fixed point number.

If this command is sent with no data, the motor controller will respond by sending this same message to report the current proportional constant. This feature is not available in the 3330 version of the MDL-BDC firmware.

Current Integral Constant

The command sets the integral constant used in the PID algorithm calculations used for current control. The integral constant is a 32 bit 16.16 signed fixed point number.

If this command is sent with no data, the motor controller will respond by sending this same message to report the current integral constant. This feature is not available in the 3330 version of the MDL-BDC firmware.

Current Differential Constant

The command sets the differential constant used in the PID algorithm calculations used for current control. The differential constant is a 32 bit 16.16 signed fixed point number.

If this command is sent with no data, the motor controller will respond by sending this same message to report the current differential constant. This feature is not available in the 3330 version of the MDL-BDC firmware.

4.7 Motor Control Status

This command is used to retrieve status information from the motor controller. The current values of various motor controller outputs and measurements can be obtained through these commands.

Table 4.10: Motor Control Status.

Value	API Name
0	Output Voltage
1	Bus Voltage
2	Current
3	Temperature
4	Position
5	Speed
6	Limit
7	Fault
8-15	Reserved

Output Voltage

This command requests the current output voltage of the motor controller in volts. The output voltage is specified as a 16 bit 8.8 signed fixed point number.

Bus Voltage

This command requests the current input voltage to the motor controller in volts. The bus voltage is specified as a 16 bit 8.8 signed fixed point number.

Current

This command requests the current being used by the motor attached to the motor controller in amperes. The current is specified as a 16 bit 8.8 signed fixed point number.

Temperature

This command requests the current case temperature of the microcontroller in the motor controller in degrees Celsius. The temperature is specified as a 16 bit 8.8 signed fixed point number.

Position

This command requests the current position of the motor. The position is specified as a 32 bit signed number.

Speed

This command requests the current rotational speed of the motor in revolutions per second. The speed is specified as a 32 bit 16.16 signed fixed point number.

Limit

This command requests the status of the current forward and backward limit of the motor. The limit status is an 8 bit number that is a bitmask of the limit values where a bit sit has the meaning listed.

Table 4.11: Limit Status Bits.

Bit	Description
0	Forward Limit Reached
1	Reverse Limit Reached

Fault

This command requests the current fault status for the motor controller. The fault status is a 16 bit number that is a bitmask of the current fault conditions where a bit set indicates the fault is active.

Table 4.12: Fault Status Bits.

Bit	Description
0	Current Fault
1	Temperature Fault
2	Bus Voltage Fault

4.8 Motor Control Configuration

These commands are used to configure the motor controller to specific drive settings. This includes all of the following settings:

Table 4.13: Motor Configuration APIs.

Value	API Name
0	Number of Brushes
1	Number of Encoder Lines
2	Number of Potentiometer Turns
3	Break/Coast Setting
4	Limit Mode
5	Forward Direction Limit
6	Reverse Direction Limit
7	Maxmimum Output Voltage
8-15	Reserved

Brushes

This is an 8 bit count of the number of brushes.

If this command is sent with no data, the motor controller will respond by sending this same message to report the number of brushes. This feature is not available in the 3330 version of the MDL-BDC firmware.

Encoder Lines

This is a 32 bit count of the number of lines in the encoder.

If this command is sent with no data, the motor controller will respond by sending this same message to report the number of encoder lines. This feature is not available in the 3330 version of the MDL-BDC firmware.

Potentiometer Turns

This is a 32 bit count of the number of turns in the potentiometer.

If this command is sent with no data, the motor controller will respond by sending this same message to report the number of potentiometer turns. This feature is not available in the 3330 version of the MDL-BDC firmware.

Break/Coast

This is an 8 bit value with the following values: use the jumper, override the jumper with brake mode or override the jumper with coast mode.

If this command is sent with no data, the motor controller will respond by sending this same message to report the brake/coast setting. This feature is not available in the 3330 version of the MDL-BDC firmware.

Soft Limit Switches

This is an 8 bit value that enables or disables the soft limit switches.

If this command is sent with no data, the motor controller will respond by sending this same message to report the enable state of the soft limit switches. This feature is not available in the 3330 version of the MDL-BDC firmware.

Forward Soft Limit Switch

This setting takes two values. The first is 16.16 fixed-point value that is the position of the forward soft limit switch. The second is an 8 bit value that specifies if the motor position must be greater than or less than the position of the forward soft limit switch. Greater than is encoded as 0 and less than is encoded as 1. Less than should be used if positive voltage results in the position increasing and greater than if positive voltage results in the position decreasing.

If this command is sent with no data, the motor controller will respond by sending this same message to report the position of the forward soft limit switch. This feature is not available in the 3330 version of the MDL-BDC firmware.

Reverse Soft Limit Switch

This setting takes two values. The first is a 16.16 fixed-point value that is the position of the reverse soft limit switch. The second is an 8 bit value that specifies if the motor position must be greater than or less than the position of the reverse soft limit switch. Greater than is encoded as 0 and less than is encoded as 1. Less than should be used if negative voltage results in the position increasing and greater than if negative voltage results in the position increasing.

If this command is sent with no data, the motor controller will respond by sending this same message to report the position of the reverse soft limit switch. This feature is not available in the 3330 version of the MDL-BDC firmware.

Maximum Output Voltage

This setting specifies the maximum output voltage. The voltage commands are scaled such that “full voltage” is this maximum voltage value. The maximum output voltage is specified as a 16 bit

8.8 unsigned fixed point number.

If this command is sent with no data, the motor controller will respond by sending this same message to report the maximum output voltage.

This command is not available in the 3330 version of the MDL-BDC firmware.

Company Information

Founded in 2004, Luminary Micro, Inc. designs, markets, and sells ARM Cortex-M3-based microcontrollers (MCUs). Austin, Texas-based Luminary Micro is the lead partner for the Cortex-M3 processor, delivering the world's first silicon implementation of the Cortex-M3 processor. Luminary Micro's introduction of the Stellaris family of products provides 32-bit performance for the same price as current 8- and 16-bit microcontroller designs. With entry-level pricing at $1.00 for an ARM technology-based MCU, Luminary Micro's Stellaris product line allows for standardization that eliminates future architectural upgrades or software tool changes.

Luminary Micro, Inc.
108 Wild Basin, Suite 350
Austin, TX 78746
Main: +1-512-279-8800
Fax: +1-512-279-8879
http://www.luminarymicro.com
sales@luminarymicro.com

Support Information

For support on Luminary Micro products, contact:

support@luminarymicro.com
+1-512-279-8800, ext 3