RDK-BDC24 Firmware Development Package

USER'S GUIDE

SW-RDK-BDC24-UG-6075

Copyright

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

Revision Information

This is version 6075 of this document, last updated on June 04, 2010.

Table of Contents

1 Introduction

The Brushed DC Motor Reference Design Kit (RDK-BDC24) contains a brushed DC motor controller module (the MDL-BDC24) and cable that allows an MDL-BDC24 to be connected to standard UART interface. The RDK-BDC24 is pre-loaded with the same firmware as the bare modules.

With this kit, the capabilities and performance of the MDL-BDC24 can be evaluated. The RDK-BDC has the example code for the LM3S2965 evaluation kit that it shipped with which can be used as reference for a custom application to control the MDL-BDC24 over the CAN network.

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

2 Example Applications

The boot loader (boot_can) and quickstart (qs-bdc24) are programmed onto the MDL-BDC24.

There is an IAR workspace file (`rdk-bdc24.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 a Keil multi-project workspace file (`rdk-bdc24.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-bdc24` subdirectory of the firmware development package source distribution.

2.1 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.

2.2 24 Volt Brushed DC Motor Controller (qs-bdc24)

This application controls a brushed DC motor. Three communication methods are supported; a hobby servo-style input for basic voltage control, a standard UART interface or a CAN interface are also available for more advanced control (the inputs are mutually exclusive).

When using any of the communication methods, a basic voltage control mode is available. In this mode, the external controller directly specifies the desired output voltage. When using CAN or UART, 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 or UART communication interfaces. 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 or UART 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 or UART interfaces. 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.

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.

RDK-BDC24 Communication Program

Usage

```
bdc-comm [OPTION]...
```

Description

The BDC-COMM application is a command line or graphical user interface (GUI) application designed to help you control and monitor the MDL-BDC24. Using your computerŠs COM port and a special cable, the application interfaces to the MDL-BDC24 using simple serial/UART communication. With an MDL-BDC24 as the main interface, the application allows connectivity to a network of MDL-BDC24 and legacy MDL-BDC devices that are connected using the built-in CAN interface of the boards.

Arguments

-h displays usage information.

-c NUM specifies the host COM port number to use.

System Requirements

BDC-COMM is built using the Fast Light Tool Kit (FLTK) cross-platform GUI development software allowing the application to run on both Windows and Linux systems.

NOTE: Because the BDC-COMM application sends out a periodic heartbeat to keep the board alive, slower systems may have trouble keeping up. The heartbeat is sent out every 50 ms, so if you have an older or slow system, the heartbeats may not always get out in time. Missing a heartbeat makes the LED on the MDL-BDC24/MDL-BDC flash instead of remain solid. It also causes the motor to stop spinning if it is currently running.

Hardware Requirements

To use the application, connect the MDL-BDC24 to a PC. The MDL-BDC24 must be the first board in a chain of motor controllers. The legacy MDL-BDC does not have the hardware support needed to communicate with the PC over the serial/UART link. Spotting the difference between a MDL-BDC24 and MDL-BDC is easy. The newer MDL-BDC24 comes in black plastic with a red Texas Instruments logo on the fan grill. The legacy MDL-BDC comes in grey plastic with an orange Luminary Micro logo on the fan grill.

Running BDC-COMM in GUI Mode

To launch the application in GUI mode, simply double-click on the bdc-comm.exe file. When running in Windows, a console window opens briefly before the GUI appears, which is typical behavior for Windows executables. The console window disappears on its own. Launch the GUI from the command line by typing "bdc-comm.exe" in the console window. No additional input arguments are needed.

See the MDL-BDC24_BDC_COMM.pdf document for more details on using bdc-comm in GUI mode.

Running BDC-COMM in Command Line Mode

The BDC-COMM application defaults to command line mode when input arguments are passed to it when launching from the console window. For example, typing the following tells the application to open using COM1 and launches the application in command line mode:

```
bdc-comm.exe -c 1
#
```

When the hash symbol appears, you can begin typing commands. If you need a list of the available commands, type "help".

Configuration Commands

The following commands are used to configure the motor controller. There are several configuration subcommands available:

config lines [<number>]

The number of lines in the optional encoder that is attached to the motor. Providing the optional **<number>** argument will set the number of encoder lines; omitting it will query the current setting. This setting depends upon the presence of an encoder and ranges in value from 0 to 65535.

config turns [<number>]

The number of turns in the optional potentiometer that is attached to the motor. Providing the optional **<number>** argument will set the number of potentiometer turns; omitting it will query the current setting. This setting depends upon the presence of a potentiometer and ranges in value from 0 to 65535.

config brake [<mode>]

The brake/coast configuration of the motor controller. Providing the optional **<mode>** argument will set the brake/coast mode; omitting it will query the current setting. The valid values for brake/coast mode are **jumper** for jumper configuration, **brake** for brake mode, and **coast** for coast mode.

config limit [<mode>]

The state of the soft limit switches. Providing the optional **<mode>** argument will set the soft limit switch mode; omitting it will query the current setting. The soft limit switches can be either on (**on**) or off (**off**).

config fwd [<pos> <cmp>]

The configuration of the forward soft limit switch. Providing the optional **<pos> <cmp>** arguments will set the configuration of the forward limit switch; omitting them will query the current setting. The position (in **<pos>**) is a 16.16 fixed-point value that ranges from 32767.999 to -32768.999 (specified as 2147483647 and -2147483648 respectively); this value represents the number of rotations of the motor. The comparison (in **<cmp>**) is either greater-than (**gt**) or less-than (**lt**). For example, if the configuration is **65536 lt**, the position must be less than 65536 (1.0) in order for the motor to turn in the forward direction.

config rev [<pos> <cmp>]

The configuration of the reverse soft limit switch. Providing the optional **<pos> <cmp>** arguments will set the configuration of the reverse limit switch; omitting them will query the current setting. The position (in **<pos>**) is a 16.16 fixed-point value that ranges from 32767.999 to -32768.000 (specified as 2147483647 and -2147483648 respectively); this value represents the number of rotations of the motor. The comparison (in **<cmp>**) is either greater-than (**gt**) or less-than (**lt**). For example, if the configuration is **-65536 gt**, the position must be greater than -65536 (-1.0) in order for the motor to turn in the reverse direction.

config maxvout [<voltage>]

The maximum output voltage that can be produced by the motor controller. Providing the optional **<voltage>** argument will set the maximum output voltage; omitting it will query the current setting. The maximum output voltage is specified as a voltage in 8.8 fixed-point format, and is used as the output voltage produced when driving in full forward. This can be used to safely drive 7.2V motors (for example) from a 12V battery.

Current Commands

The following are commands to operate the motor controller in current control mode. There are several subcommands available:

cur en

Enables current control mode, implicitly disabling all other control modes. This command must be issued before the motor can be run in current control mode.

cur dis

Disables current control mode. If current control mode is enabled, this will revert back to voltage control mode. Otherwise, this command has no affect on the motor controller.

cur set [<current> [<group>]]

The set point for the motor current. Providing the optional **<current>** argument will set the current set point; omitting it will query the current set point. The current is specified as a 8.8 fixed-point value that ranges from 127.999 to -128.000 (specified as 32767 and -32768 respectively); this value represents the motor current in Amperes. Since the motor controller will only run at +/- 40A for extended periods of time (above 40A, the current protection features start taking affect), values outside the range of 40.000 to -40.000 (specified as 10240 and -10240 respectively) are not advised. Also providing the optional **<group>** argument will defer the update of the current set point until a **system sync** command is sent; the group is a bit field of eight groups, of which this deferred update can be a member of any or all of the groups.

cur p [<coeff>]

The P coefficient for the PID controller used for current control. Providing the optional **<coeff>** argument will set the P coefficient; omitting it will query the P coefficient. The P coefficient is specified as a 16.16 fixed-point value that ranges from 65535.999 to -65536.000 (specified as 2147483647 and -2147483648 respectively).

cur i [<coeff>]

The I coefficient for the PID controller used for current control. Providing the optional **<coeff>** argument will set the I coefficient; omitting it will query the I coefficient. The I coefficient is specified as a 16.16 fixed-point value that ranges from 65535.999 to -65536.000 (specified as 2147483647 and -2147483648 respectively).

cur d [<coeff>]

The D coefficient for the PID controller used for current control. Providing the optional **<coeff>** argument will set the D coefficient; omitting it will query the D coefficient. The D coefficient is specified as a 16.16 fixed-point value that ranges from 65535.999 to -65536.000 (specified as 2147483647 and -2147483648 respectively).

Heartbeat Command

This command toggles the generation of heartbeat messages. Without a periodic heartbeat message, the motor controller will go to neutral as a safety measure (the presence of heartbeat messages indicating that a valid control link is still present). By default, heartbeat messages are generated.

Help Command

This command provides summary help information about the available commands. Either **help**, **h** or **?** will display the help options.

ID Command

id <id number>

This command sets the ID of the motor controller that is to be controlled. Providing the optional **<id>** argument will set the ID and omitting it will query the ID.

Position Control Commands

Commands to operate the motor controller in position control mode. There are several sub-commands available:

pos en <start>

Enables position control mode, implicitly disabling all other control modes. This command must be issued before the motor can be run in position control mode. The required **<start>** argument specifies the value for the current position of the motor (this does not cause the motor to start turning). This value is specified as a 2147483647 to -2147483648); this value represents the number of motor revolutions.

pos dis

Disables position control mode. If position control mode is enabled, this will revert back to voltage control mode. Otherwise, this command has no affect on the motor controller.

pos set [<pos> [<group>]]

The set point for the motor position. Providing the optional **<pos>** argument will set the position set point; omitting it will query the position set point. The position is specified as a 16.16 fixed-point value that ranges from 65535.999 to -65536.000 (specified as 2147483647 to -2147483648 respectively); this value represents the motor position in revolutions. Also providing the optional **<group>** argument will defer the update of the position set point until a **system sync** command is sent; the group is a bit field of eight groups, of which this deferred update can be a member of any or all of the groups.

pos p [<coeff>]

The P coefficient for the PID controller used for position control. Providing the optional **<coeff>** argument will set the P coefficient; omitting it will query the P coefficient. The P coefficient is specified as a 16.16 fixed-point value that ranges from 65535.999 to -65536.000 (specified as 2147483647 and -2147483648 respectively).

pos i [<coeff>]

The I coefficient for the PID controller used for position control. Providing the optional **<coeff>** argument will set the I coefficient; omitting it will query the I coefficient. The I coefficient is specified as a 16.16 fixed-point value that ranges from 65535.999 to -65536.000 (specified as 2147483647 and -2147483648 respectively).

pos d [<coeff>]

The D coefficient for the PID controller used for position control. Providing the optional **<coeff>** argument will set the D coefficient; omitting it will query the D coefficient. The D coefficient is specified as a 16.16 fixed-point value that ranges from 65535.999 to -65536.000 (specified as 2147483647 and -2147483648 respectively).

pos ref [<ref>]

The reference used to determine the motor position. Providing the optional **<ref>** argument will set the position reference; omitting it will query the position reference. There are two position references available; “0” specifies that the encoder should be used and “1” specifies that the potentiometer should be used. The position reference must be set in order to use position control mode.

Quit Command

This command exists the program. Either **qui**, **q** or **exit** will exit the program.

Speed Commands

The following commands are used to operate the motor controller in speed control mode. There are several subcommands available:

speed en
Enables speed control mode, implicitly disabling all other control modes. This command must be issued before the motor can be run in speed control mode.

speed dis
Disables speed control mode. If speed control mode is enabled, this will revert back to voltage control mode. Otherwise, this command has no affect on the motor controller.

speed set [<speed> [<group>]]
The set point for the motor speed. Providing the optional **<speed>** argument will set the speed set point; omitting it will query the speed set point. The speed is specified as a 16.16 fixed-point value that ranges from 65535.999 to -65536.000 (specified as 2147483647 to -2147483648); this value represents the motor speed in RPMs. The practical range of values that can be used is dependent upon the maximum speed of the attached motor. Also providing the optional **<group>** argument will defer the update of the speed set point until a **system sync** command is sent; the group is a bit field of eight groups, of which this deferred update can be a member of any or all of the groups.

speed p [<coeff>]
The P coefficient for the PID controller used for speed control. Providing the optional **<coeff>** argument will set the P coefficient; omitting it will query the P coefficient. The P coefficient is specified as a 16.16 fixed-point value that ranges from 65535.999 to -65536.000 (specified as 2147483647 and -2147483648 respectively).

speed i [<coeff>]
The I coefficient for the PID controller used for speed control. Providing the optional **<coeff>** argument will set the I coefficient; omitting it will query the I coefficient. The I coefficient is specified as a 16.16 fixed-point value that ranges from 65535.999 to -65536.000 (specified as 2147483647 and -2147483648 respectively).

speed d [<coeff>]
The D coefficient for the PID controller used for speed control. Providing the optional **<coeff>** argument will set the D coefficient; omitting it will query the D coefficient. The D coefficient is specified as a 16.16 fixed-point value that ranges from 65535.999 to -65536.000 (specified as 2147483647 and -2147483648 respectively).

speed ref [<ref>]
The reference used to determine the motor speed. Providing the optional **<ref>** argument will set the speed reference; omitting it will query the speed reference. The only speed reference available at this time is the encoder, which is specified as “0”. The speed reference must be set in order to use speed control mode.

Status Commands

Commands to query the status from the motor controller. There are several subcommands available:

stat vout
Queries the output voltage of the motor controller. The value returned is specified as a percentage of the input battery voltage, where 32767 is forward will full input voltage, 0 is neutral, and -32768 is reverse with full input voltage.

stat vbus
Queries the input battery voltage. The value returned is specified as a 8.8 fixed-point value that specifies the voltage.

stat fault
Queries the set of currently active fault conditions, if any. The value returned is a set of flags that indicate the individual fault conditions; bit 0 indicates an over-current fault,

bit 1 indicates an over-temperature fault, bit 2 indicates an under-voltage fault, and bit 3 indicates a gate driver fault.

stat cur
Queries the current flowing through the motor. The value returned is specified as a 8.8 fixed-point value that specifies the motor current in Amperes.

stat temp
Queries the ambient temperature inside the case of the motor controller. The value returned is specified as a 8.8 fixed-point value that specifies the ambient temperature in degrees Celcius.

stat pos
Queries the position of the motor. The value returned is a 16.16 fixed-point value that specifies the position of the motor in revolutions.

stat speed
Queries the speed of the motor. The value returned is a 16.16 fixed-point value that specifies the speed of the motor in RPM.

stat limit
Queries the state of the limit switches (both the hard and soft limit switches). The value returned is a set of flags that indicate the state of the limit switches; bit 0 indicates the state of the forward limit switch (where the bit being set means that the limit switch is open) and bit 1 indicates the state of the reverse limit switch.

stat power [<reset>]
Queries the state of the power flag. This flag is set every time the motor controller is power cycled and can be manually cleared and then queried in order to detect when the motor controller has experienced an unexpected power cycle. Providing the optional **<reset>** parameter (with any value) will clear the power flag; omitting it will query the state of the power flag.

System Commands

Commands to provide system-level control of the motor controller(s). There are several sub-commands available:

system halt
Immediately halts all motor controllers in the network, forcing them to neutral. Once halted, the motor controllers will not operate again until they have received either a **system resume** or a **system reset** command. This command does nothing if the motor controllers are already halted.

system resume
Resumes normal operation of all motor controllers in the network. The motor controllers will remain in neutral until commanded to start driving the motor (in other words, the previous set points are lost when the **system halt** is received). This command does nothing if the motor controllers are not halted.

system reset
Immediately resets all motor controllers in the network. After a reset, all previously set parameters and set points are reset to their defaults.

system enum
Enumerates the motor controllers in the network. A list of the IDs of the accessible motor controllers is provided in response.

system assign <id>
Performs a device ID assignment request. The required **<id>** argument specifies the ID to be assigned. All motor controllers in the network will go into assignment mode for five seconds; if the push button on one of the motor controllers is pressed during this time, that motor controller will change its ID to the provided ID. If a motor controller sees

an assignment request for its current ID and its button is not pressed, it will reset its ID (leaving it with no ID); this is done since it has no way of knowing if the button on another motor controller has been pressed, and having more than one motor controller with the same ID on the network would cause adverse affects on communciations.

system query
Queries information about the motor controller with the current ID. The device type and manufacturer are returned.

system sync <group>
Performs a synchronous update of the set points on several motor controllers on the network. The **<group>** argument specifies a bit field of eight groups that can be synchronized; a one bit in any of these bit positions will result in any pending set point updates with the corresponding bit set in its group to be updated immediately.

system version
Queries the version of the firmware on the motor controller with the current ID.

Update Command

update <filename>

This command updates the firmware in the motor controller. The required **<filename>** argument specifies the name of the file that contains the new firmware image; this filename will be **qs-bdc24.bin** for the firmware images provided by Texas Instruments (the path to that image must be supplied as well if it is not in the current directory). After the firmware is updated, the motor controller is automatically reset and the new firmware starts running.

Voltage Mode Commands

The following commands are used to operate the motor controller in voltage control mode. There are several subcommands available:

volt en
Enables voltage control mode, implicitly disabling all other control modes.

volt dis
Disables voltage control mode. This command doesn't really do anything useful since it is ignored if a control mode other than voltage control mode is enabled, and if voltage control mode is enabled it can not be disabled since it is the default mode anyway. This command is provided solely for orthogonality and completeness.

volt set [<voltage> [<group>]]
The voltage to be supplied to the motor. Providing the optional **<voltage>** argument will set the motor voltage; omitting it will query the motor voltage. The voltage is specified as a percentage of the input battery voltage, where 32767 is forward will full input voltage, 0 is neutral, and -32768 is reverse with full input voltage. Also providing the optional **<group>** argument will defer the update of the voltage set point until a **system sync** command is sent; the group is a bit field of eight groups, of which this deferred update can be a member of any or all of the groups.

volt ramp [<rate>]
The rate that the voltage is changed when a new voltage command is received. Providing the optional **<rate>** argument will set the ramp rate; omitting it will query the ramp rate. The ramp rate is specified as the number of steps per millisecond; 0 will disable ramping, 1 provides the slowest ramping, and 65535 provides the fastest ramping.

This utility is contained in `tools/bin/bdc-comm`.

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 Stellaris 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-BDC24 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	Texas Instruments (Stellaris)
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	Reserved
3	Position Control
4	Current Control
5	Status
6	Reserved
7	Configuration
8	Acknowledge
9-63	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.

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.

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.

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
6-15	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 signed number that specifies the output voltage as a scalar value and is used as a multiplier on the possible ouput voltage. The output voltage is caculated as follows:

```
Output Voltage = (Max output Voltage * Output Voltage Setting)/32768;
```

Thus a value of 32767 is full forward and a value of -32768 is full reverse. 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.

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 a 16-bit unsigned 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 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.

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.

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.

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.

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.

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.

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. The first parameters is a 32-bit 16.16 signed fixed-point value that specifies the starting motor position in revolutions. 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 is a 32-bit 16.16 signed fixed-point value that specifies the motor position in revolutions. The second param-

eter 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.

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.

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.

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.

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.

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-bit 8.8 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.

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.

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.

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.

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	Power
9	Control Mode
10-15	Reserved

Output Voltage

This command requests the current output voltage of the motor controller. The output voltage is specified as a 16-bit signed 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 16.16 signed fixed-point 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

Power

This command requests the power status for the motor controller. The power status is a flag that is set when the motor controller is first powered on, can be cleared by writing to the power status, and the remains clear until power is removed. The power status can be used to detect an unexpected power cycle on the motor controller (such as an external self-resetting thermal fuse tripping and then resetting).

The power status is contained in the LSB of a single byte in the command payload. If the payload is present and the LSB is set, the power status flag is cleared. If the payload is not present, the power status is returned.

Control Mode

This command requests the current control mode for the motor controller. The control mode is an 8-bit number that indicates if the motor controller is in Voltage, Current, Position or Speed control mode.

Table 4.13: Control Mode Status.

Value	Description
0	Voltage Mode
1	Current Mode
2	Speed Mode
3	Position Mode

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 4.14: 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	Fault Time
9-15	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.

Encoder Lines

This is a 16-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.

Potentiometer Turns

This is a 16-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.

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.

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.

Forward Soft Limit Switch

This setting takes two values. The first is 16.16 signed 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.

Reverse Soft Limit Switch

This setting takes two values. The first is a 16.16 signed 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.

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.

Fault Time

This settings specifies the time that the motor controller remains in a fault condition once detected. The time is provided as a 16-bit unsigned value specifiing the number of milliseconds, where the default is 3000 milliseconds (3 seconds), and the minimum allowable value is 500 milliseconds (1/2 second).

If this command is sent with no data, the motor controller will respond by sending this same message to report the current fault time.

5 UART Interface

The RDK-BDC24 has the ability to use a UART for configuration, diagnosis, and real time control of the motor controller. All capabilities that are available via the CAN interface can also be accessed via the UART interface. Additionally, the UART interface can be used to generate traffic on the CAN bus, providing a bridge between the UART and the CAN bus.

The UART interface is run at 115,200 baud with eight data bits, no parity, and one stop bit (8-N-1). This is a standard UART rate and data configuration.

If the commands received via the UART interface have a device ID that matches the ID of the RDK-BDC24, the command is processed locally. Otherwise, the message is re-transmitted on the CAN bus and the response from the CAN bus is re-transmitted on the UART interface. The only downside to this approach is that the UART interface has a lower bandwidth than the CAN bus (115.2 Kbaud full-duplex on the UART interface versus 1 Mbit half-duplex on the CAN bus). The lower bandwidth of the UART interface must be taken into account by the system design if using the UART to CAN bridging capability.

The messages on the UART interface are simply an encapsulated version of the CAN bus messages. On the CAN bus, there is a 29-bit identifier followed by up to eight optional data bytes. On the UART interface, the 29-bit identifier is placed into four bytes (with 3 bits of padding) and the message is packetized. The packet is arranged as follows:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	...
SOF	Size	ID0	ID1	ID2	ID3	Data	
		...size...					

The first byte is a start of frame byte, which has a value of `0xff`. The only place a `0xff` byte will appear is in this location, so this is used as a resynchronization point for the serial link.

The next byte is the size of the raw packet, covering bytes 2 through N (prior to encoding, which is described below). This will range from four to twelve; four if the message has zero data bytes and twelve if it has the maximum of eight data bytes. The start of frame and size bytes are not included in the packet size.

The next four bytes are the CAN identifier, transmitted in little endian format. For example, sending a message to motor controller at ID 5 to set the output voltage uses CAN identifier `0x02020085`; this is sent over the UART as `0x85 0x00 0x02 0x02`, in that order.

The remaining bytes are the optional data bytes associated with the message. Some messages have no optional data bytes, some have up to eight. These data bytes are also sent in little endian format; for example the set output voltage command takes a signed short value (sixteen bits) and is sent with the lower eight bits in the first byte and the upper eight bits in the second byte.

All bytes in the packet (other than the start of frame byte) are subject to encoding. Since `0xff` is the start of frame byte, it can not appear anywhere else in the packet and must be encoded. In order to transmit `0xff` as a data byte (either in the ID or in the optional data), the two-byte sequence `0xfe 0xfe` is used. In order to transmit `0xfe`, the two-byte sequence `0xfe 0xfd` is used. All other bytes are transmitted as is. This doubling of the size of `0xff` and `0xfe` does not affect the value sent as the size of the packet; the size is still the number of bytes after the encoding has been removed from the transmitted data.

Some examples to clarify the packet format (all sent to ID 5):

- To set the motor voltage to `0x0800`:

```
0xff 0x06 0x85 0x00 0x02 0x02 0x00 0x08
```

The motor controller will respond with the following:

```
0xff 0x04 0x05 0x20 0x02 0x02
```

- To set the motor voltage to `0xffff`:

  ```
  0xff 0x06 0x85 0x00 0x02 0x02 0xfe 0xfe 0xfe 0xfe
  ```

  The motor controller will respond with the following:

  ```
  0xff 0x04 0x05 0x20 0x02 0x02
  ```

- To query the motor voltage output:

  ```
  0xff 0x04 0x85 0x00 0x02 0x02
  ```

  The motor controller will respond with the following (assuming that the output voltage is 0x0800):

  ```
  0xff 0x06 0x85 0x00 0x02 0x02 0x00 0x08
  ```

The higher level command protocol is exactly the same as the protocol used over CAN. The same commands and responses are packetized as described above and transmitted over the UART link.

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