C2000 Gang Programmer (C2000-GANG)

User's Guide

Literature Number: SPRUHS0C
February 2014–Revised March 2016

Contents

Copyright © 2014–2016, Texas Instruments Incorporated

Copyright © 2014–2016, Texas Instruments Incorporated

List of Figures

Copyright © 2014–2016, Texas Instruments Incorporated

List of Tables

Copyright © 2014–2016, Texas Instruments Incorporated

Read This First

If You Need Assistance

If you have any feedback or questions, support for the C2000™ devices and the C2000 Gang Programmer is provided by the Texas Instruments Product Information Center (PIC) and the TI E2E Forum (http://e2e.ti.com/support/microcontrollers/c2000/default.aspx). Contact information for the PIC can be found on the TI web site at support.ti.com. Additional device-specific information is on the C2000 web site at www.ti.com/c2000.

Related Documentation from Texas Instruments

The primary sources of C2000 information are the device-specific data sheets and user's guides. The most current information is on the C2000 web site at www.ti.com/c2000.

Information specific to the C2000 Gang Programmer is at www.ti.com/c2000-gang.

Copyright © 2014–2016, Texas Instruments Incorporated

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference and (2) this device must accept any interference received, including interference that may cause undesired operation.

NOTE: ***This equipment has been tested and found to comply with the limits for a Class B digital devices, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one of more of the following measures:***

- ***Reorient or relocate the receiving antenna***
- ***Increase the separation between the equipment and receiver***
- ***Connect the equipment into an outlet on a circuit different from that to which the receiver is connected***
- ***Consult the dealer or an experienced radio/TV technician for help.***

Warning: ***Changes or modifications not expressly approved by Texas Instruments Inc. could void the user's authority to operate the equipment.***

This Class B digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appereil numerique de la classe B respecte toutes les exigences du Reglement sur le material brouilleur du Canada.

Copyright © 2014–2016, Texas Instruments Incorporated

Introduction

The C2000 Gang Programmer is a C2000 device programmer that can program up to eight identical C2000 devices at the same time. The C2000 Gang Programmer connects to a host PC using a standard RS-232 or USB connection and provides flexible programming options that allow the user to fully customize the process. A top-level view of the C2000 Gang Programmer can be seen in Figure 1-1.

The C2000 Gang Programmer is not a gang programmer in the traditional sense, in that there are not eight sockets provided to program target devices. Instead, the C2000 Gang Programmer is designed to connect to target devices in-circuit (that is, target devices are mounted in the final circuit or system). The C2000 Gang Programmer accesses target devices using connectors that use JTAG signals.

The C2000 Gang Programmer is provided with an expansion board, called the Gang Splitter, that implements the interconnections between the C2000 Gang Programmer and multiple target devices. Eight cables are provided that connect the expansion board to eight target devices (via JTAG connectors).

Chapter 2 describes in detail how to use the C2000 Gang Programmer to program target devices. Various modes of operation are described, and they allow the user to choose the most convenient method of programming. In addition, this chapter describes the various windows that are used to configure the programming procedure for a specific target device.

Chapter 3 describes firmware commands that can be used to control the programming process at fine granularity. Firmware commands can be received over an RS-232 or USB port and correspond to specific actions that the programmer can perform. Take great care in using these commands, because they must often be used in groups for proper behavior, and the order in which they are executed affects the result.

Chapter 5 contains an I/O schematic that shows how signals from the C2000 Gang Programmer can be brought out to each of the target devices via a C2000-standard JTAG connector. The user can easily modify the circuit to connect the signals to the target device pins directly (via a socket) if a traditional gang programmer setup is desired.

Copyright © 2014–2016, Texas Instruments Incorporated

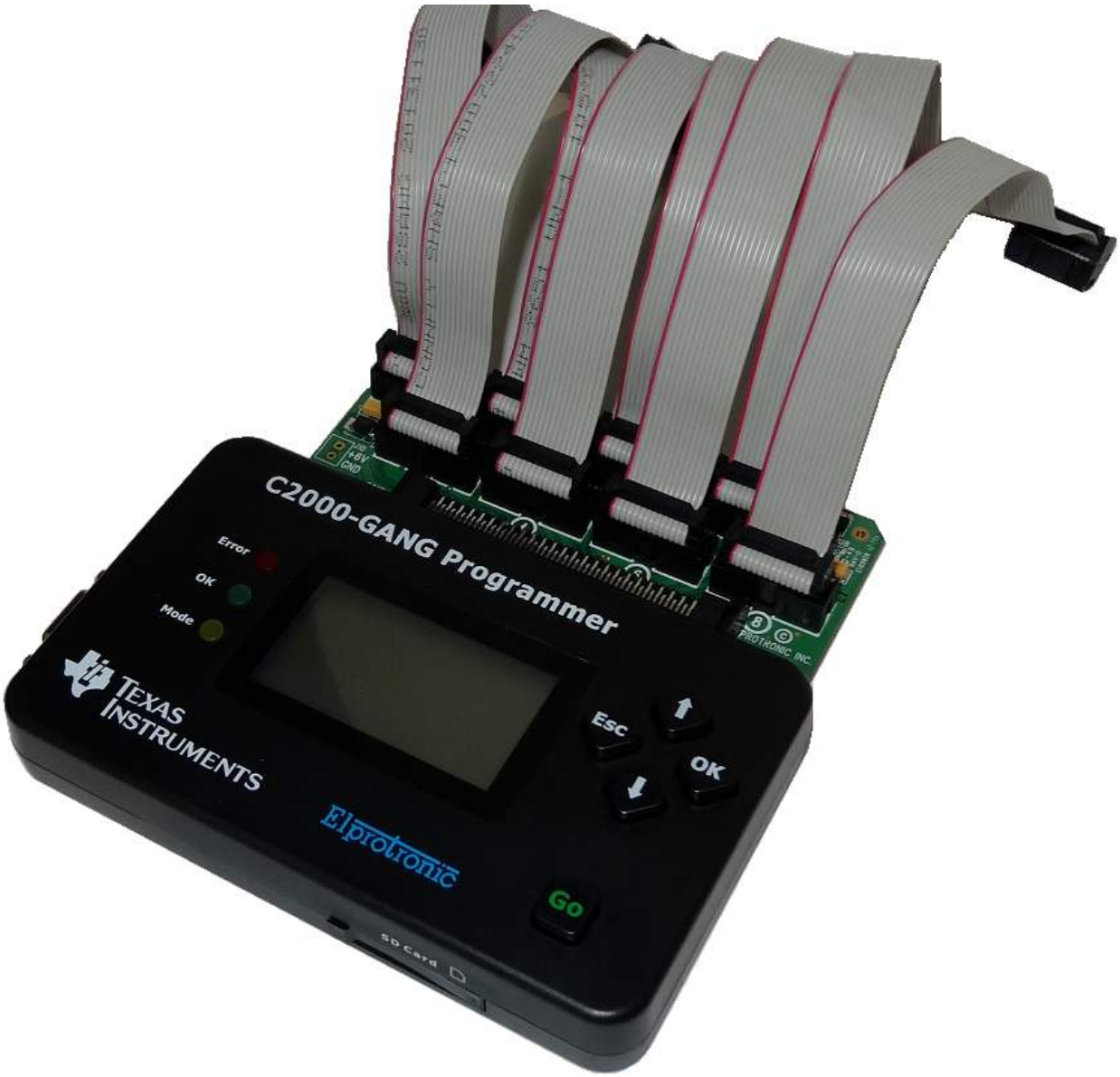


Figure 1-1. Top View of the C2000 Gang Programmer

1.1 Software Installation

TI recommends that you use the latest software version, which can be downloaded from the C2000 website at www.ti.com/tool/c2000-gang.

To install C2000 Gang Programmer software:

1. Download C2000-GANG software from TI website - see link above. Unzip file and run setup.exe file.
2. Follow the instructions in the installation process.
3. When the setup program is complete, C2000 Gang Programmer icons are available in the selected folder. Click the C2000 Gang Programmer Read Me First icon to obtain important information about the C2000 Gang Programmer.
4. The setup program also adds a program group and icons to the Windows desktop.
5. To start the C2000 Gang Programmer software, click the newly created icon.

Copyright © 2014–2016, Texas Instruments Incorporated

1.2 Driver Installation

To install the required drivers:

1. Connect the C2000 Gang Programmer to a USB port on the PC. When the Windows Wizard starts, follow instructions provided by wizard. When the wizard asks for the USB driver location, browse to the CD-ROM drive. Drivers are located in the main CD-ROM directory location and also in the following directory:

   C:\Program Files\Texas Instruments\C2000-GANG\Driver
2. If the RS-232 interface is used for communication with C2000 Gang Programmer, then the additional driver is not required. Check the Device Manager for the COM port number to use with communication via RS-232.

1.3 Hardware Installation

To install the C2000 Gang Programmer hardware:

1. Attach the expansion board (Gang Splitter) to the 100-pin connector on the C2000 Gang Programmer.

   The expansion board provides connectivity for up to eight targets using the included 14-pin cables. The target C2000 flash devices can be in standalone sockets or can be on an application's PCB. These devices can be accessed by JTAG signals.

   NOTE: The C2000-GANG Programmer rev 1.04 should use software rev. 1.5 or higher and splitter rev.1, while lower revisions of C2000-GANG Programmer can use any software revision, but should continue to use the older C2000-GANG Splitter rev.0
2. Connect the C2000 Gang Programmer hardware to a computer's USB port using a USB A-B cable.

   The C2000 Gang Programmer's internals can be supplied from the computer's USB port (5 V, 0.5 A), but an external power supply is needed for programming target devices. Target devices can be powered independently or from the C2000 Gang Programmer's 6-10V power supply connector. In this case the external power supply must provide a voltage between 6 V and 10 V DC and must be capable of providing a minimum current of 1.5 A. The center post of the power supply connector on the C2000 Gang Programmer is the positive-voltage terminal. The programmer indicates the status of the power supply connection by using system LEDs and the LCD back light. The programmer can also be connected to a serial port (COM1 to COM255) using a 9-pin Sub-D connector, if the computer does not have a USB port.
3. An external power supply is required to power the C2000 Gang Programmer if it is not connected via USB port.

   NOTE: Maximum Signal Path Length: 50 cm

   The maximum length of a signal path between the 14-pin JTAG connector on the Gang Splitter and the target device is 50 cm.
4. The C2000 Gang Programmer can supply power at 3.3 V, 160_mA to each target device separately (pin 5 on each 14-pin JTAG cable), provided that the external power supply is attached to the C2000 Gang Programmer. The C2000 Gang Programmer consumes 150 mA by itself.
5. The C2000 Gang Programmer can be supplied from an external power supply connected to the dc connector or via a gang splitter (not populated J10 connector). Because the J10 and dc connectors are connected in parallel, make sure that only one connector provides an external power supply to the C2000 Gang Programmer.

Copyright © 2014–2016, Texas Instruments Incorporated

Operation

This chapter describes how to use the C2000 Gang Programmer to program target devices. Various modes of operation, which allow the user to choose the most convenient method of programming, are described. In addition, this chapter describes the various windows that are used to configure the programming procedure for a specific target device. The explanations in this chapter assume that the user has properly installed the C2000 Gang Programmer hardware and software as described in Chapter 1.

2.1 Programming C2000 Flash Devices Using the C2000 Gang Programmer

The C2000 Gang Programmer is capable of quickly and reliably programming C2000 flash devices using an RS-232 or USB interface. There are four ways to use the programmer to achieve this task and these include:

- Interactive
- From Image
- From Script
- Standalone

The Interactive mode is selected by default, and is the easiest to get started with, because it requires the least amount of preparation. After the user has mastered the Interactive mode it can be used to create images and script files, which can then be used with the From Image and From Script modes, respectively. Images and scripts are ready-to-go setups than can run with minimal user input. They are very useful for repetitive programming, for example in a production environment, because they ensure consistency (because of the re-use of images or scripts, we highly encourage the user to thoroughly test their images or scripts for correctness before committing them to production). The C2000 Gang Programmer can also be run in Standalone mode to program target devices without a PC. To do this, first create an image to use for programming, and then save it to internal memory of the C2000 Gang Programmer. Creating images is described in Section 2.1.6.

The following sections describe how to use these modes of operation.

Copyright © 2014–2016, Texas Instruments Incorporated

2.1.1 Programming Using Interactive Mode

Use the following sequence to start the C2000 Gang Programmer GUI and program C2000 Flash Devices using the Interactive Mode:

1. Click on the C2000 Gang Programmer icon located in the program group that was specified during installation.Figure 2-1 shows the C2000 Gang Programmer GUI in the Interactive Mode (see the Mode group in the top left corner). This window is used to select the target microcontroller, code file used for programming, power supply options, communication interface, and more. This window also shows the result of programming and any errors, if they occur.

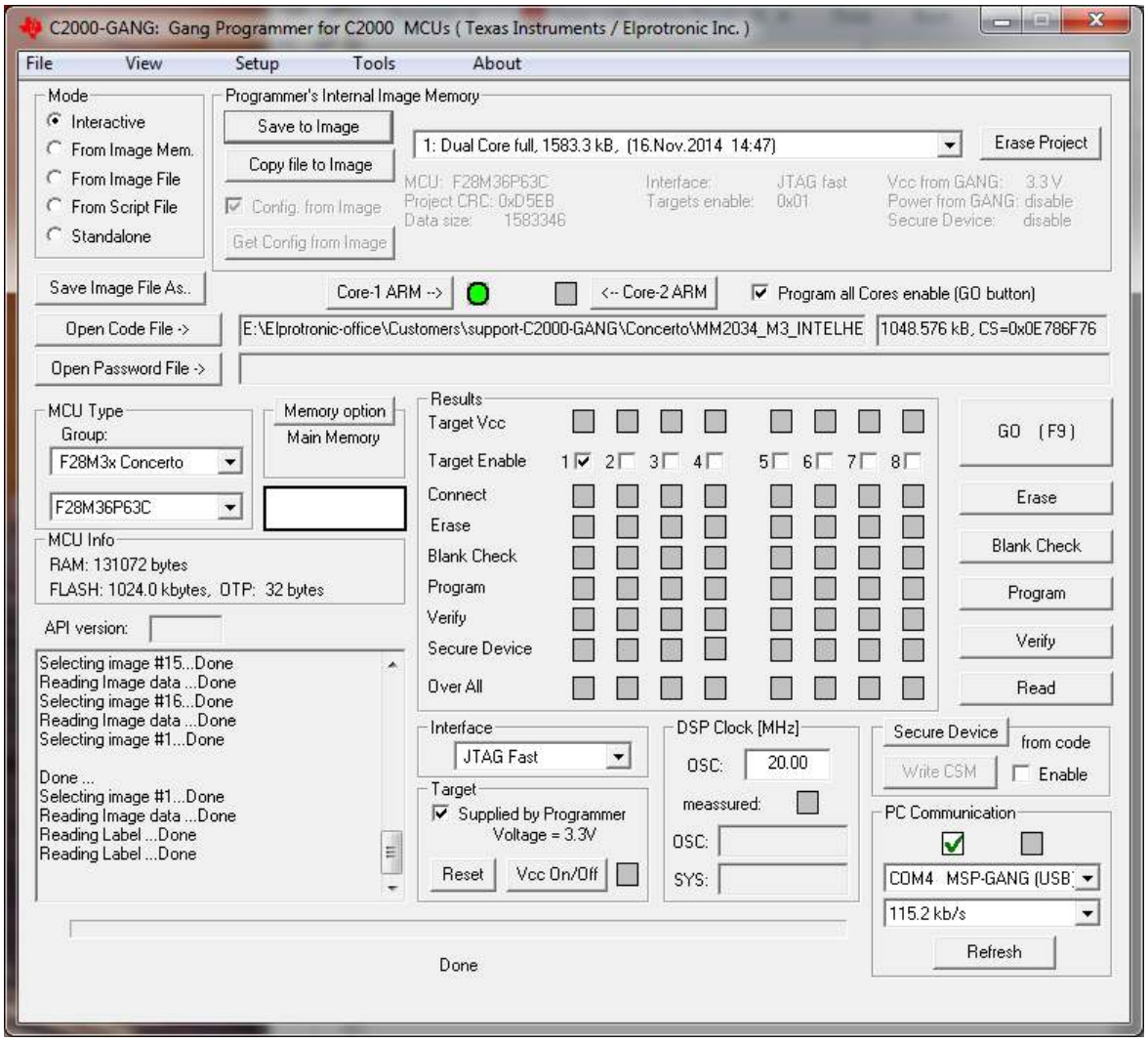


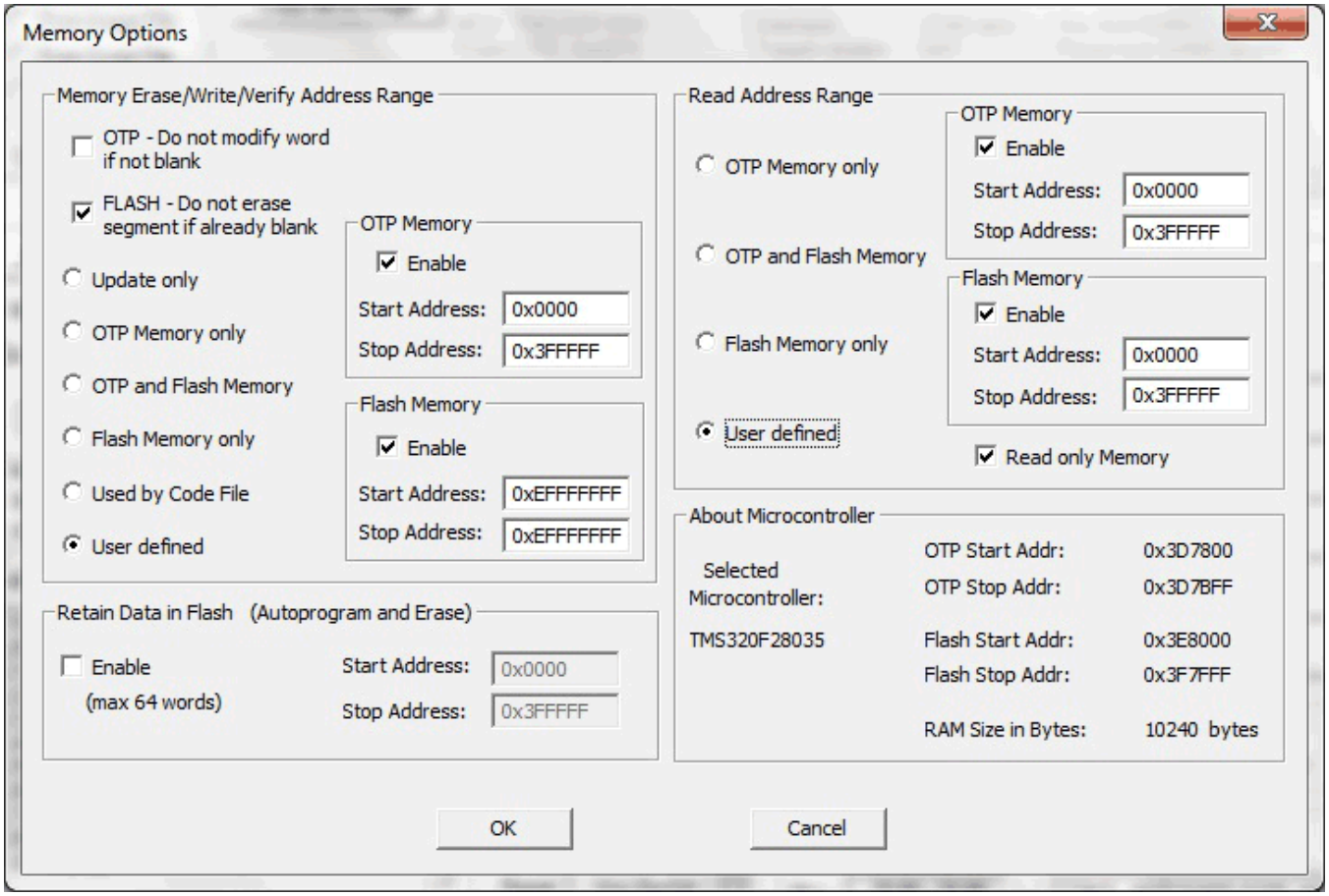
Figure 2-1. Main C2000 Gang Programmer Dialog GUI, Interactive Mode

2. Select a target device using the MCU Type menu (select MCU group and then desired MCU type).
3. Select the code file to be programmed into the devices using the Open Code File button or pulldown menu: File→Open Code File. The formats supported for the code file are TI (.txt), Intel (.hex) and Motorola (.s19, .s28, .s37). Code size and checksum appear on the right side (for details on how the checksum is calculated, seeSection 2.1.10).
4. MCUs provide a method of disabling JTAG by programming a password to flash memory. The password should be specified as data to be programmed to MCU The code file must contain password contents if you intend to lock JTAG using the password feature after programming. If the MCU is already locked using a previously programmed code file, then you must provide the password section (or entire old code file) using the Open Password File button if and only if the password section is different. Functionally, if the MCU is locked by password, the code file's password section is first used

Copyright © 2014–2016, Texas Instruments Incorporated

to attempt to unlock the MCU. If that fails, then the password file's contents are used to attempt to unlock the MCU. If both attempts fail, the MCU remains locked and JTAG access fails. Password file contents are not used to program the MCU.

5. In the Target power group, select if the target is supplied from the C2000 Gang Programmer or from an external power supply. The voltage or current is not customizable because of strict requirements for flash programming.
6. In the Results group, select desired target devices to be programmed. After programming has concluded, a green checkmark or lights appear for successful operations for each target.
7. In the Interface selector, choose the desired communication speed for JTAG (fast, medium or slow).
8. In the Memory Options dialog (pulldown menu: Setup→Memory options) shown in Figure 2-2, select desired memory space to be programmed. By default, the selected option is OTP and Flash Memory and it is correct for most programming tasks (Section 2.1.5 describes how to use the memory configuration window).

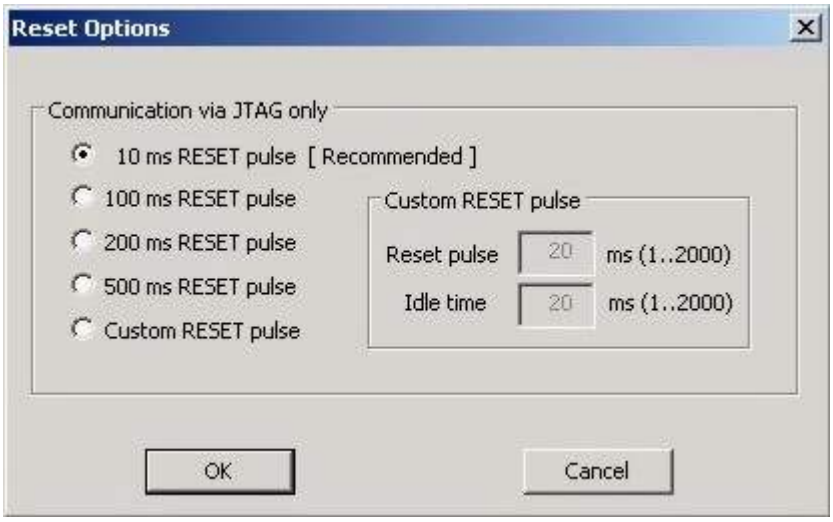


NOTE: The user can select which segments of memory are written to or read from.

Figure 2-2. Memory Options

Copyright © 2014–2016, Texas Instruments Incorporated

9. In the Reset Options dialog (pulldown menu: Setup→Device Reset) shown in Figure 2-3, select the duration of the reset pulse and the delay after reset. By default it is 10 ms, but other options are available if required by the hardware.



NOTE: This window lets the user specify the duration of the reset pulse coming from the C2000 Gang Programmer to the target device. Depending on the hardware implementation, a longer reset pulse might be required.

Figure 2-3. Reset Options

10. If selecting dual-core MCUs (that is, Concerto™, Delfino™), select Core 1 and Core 2 (press Core 1 or Core 2 button) separately to choose configuration options and code file for each core. To enable programming of both cores using the GO procedure, check-in the Program all Cores enable (GO button) checkbox

Following these steps creates a working setup that can program target devices using the C2000 Gang Programmer. Click the Save Project As button to save this configuration settings. These settings can be loaded again later and modified, if necessary (one project holds one configuration). After saving the project, use the buttons described in the following sections to perform the desired actions.

2.1.1.1 GO

Click the GO button in the Main Dialog GUI (or F9 key on the keyboard) to start programming. The progress and completion of the operation are displayed in the Results group. The result is shown as one of the following:

- Idle status
- Test in progress. For power on or off, dc voltage is correct.
- Access enabled
- Access denied (for example, the fuse is blown)
- Device action has been finished successfully
- Device action has been finished, but result failed

2.1.1.2 Erase

Click the Erase button in the Main Dialog GUI to erase a segment of memory (sets each word to 0xFFFF). Use the Memory Options configuration screen shown in Figure 2-2 to specify which addresses should be erased (Section 2.1.5 describes in detail how to use the memory configuration window). This action succeeds after the programmer has attempted to erase the specified memory segment. Use the Blank Check function to verify that this segment has been properly erased.

Copyright © 2014–2016, Texas Instruments Incorporated

2.1.1.3 Blank Check

Click the Blank Check button in the Main Dialog GUI to check that the contents of specified memory have been properly erased. This function is best used after erasing the same segment of memory using the button described in Section 2.1.1.2. Use the same Memory Options configuration screen shown in Figure 2-2 to specify which addresses should be erased (Section 2.1.5 describes in detail how to use the memory configuration window). This function succeeds when the specified memory segments are set to 0xFFFF, and fails otherwise.

2.1.1.4 Program

Click the Program button in the Main Dialog GUI to write the contents of a code files to flash memory on the target device. Addresses specified in the code files are used to determine where the program is written. Make sure that the regions of memory corresponding to the addresses in the code file are enabled for writing in the Memory Options configuration screen shown in Figure 2-2 (Section 2.1.5 describes in detail how to use the memory configuration window).

Configuration conflicts may arise during programming. It is possible that the code the user has chosen is too big to fit in the flash memory of the target MCU, or the appropriate memory segments have not been enabled in the Memory Options configuration screen. If this is the case, a warning message appears to notify the user of insufficient memory; however, the user is still allowed to proceed. If the user proceeds despite the warning, only the portion of code that fits within the MCU's enabled flash memory is written. This function succeeds after the programmer has attempted to write code to the specified memory addresses. Use the Verify function to ensure that the code has been correctly copied to flash on the target MCU.

2.1.1.5 Verify

Click the Verify button in the Main Dialog GUI to verify that the contents of the target MCU's flash memory have been properly programmed. This function is best used after programming the same segment of memory, as performed using the button described above. Make sure that the same memory segments are enabled in the Memory Options configuration window shown in Figure 2-2, as during programming described above, to ensure all programmed segments are verified (Section 2.1.5 describes in detail how to use the memory configuration window).

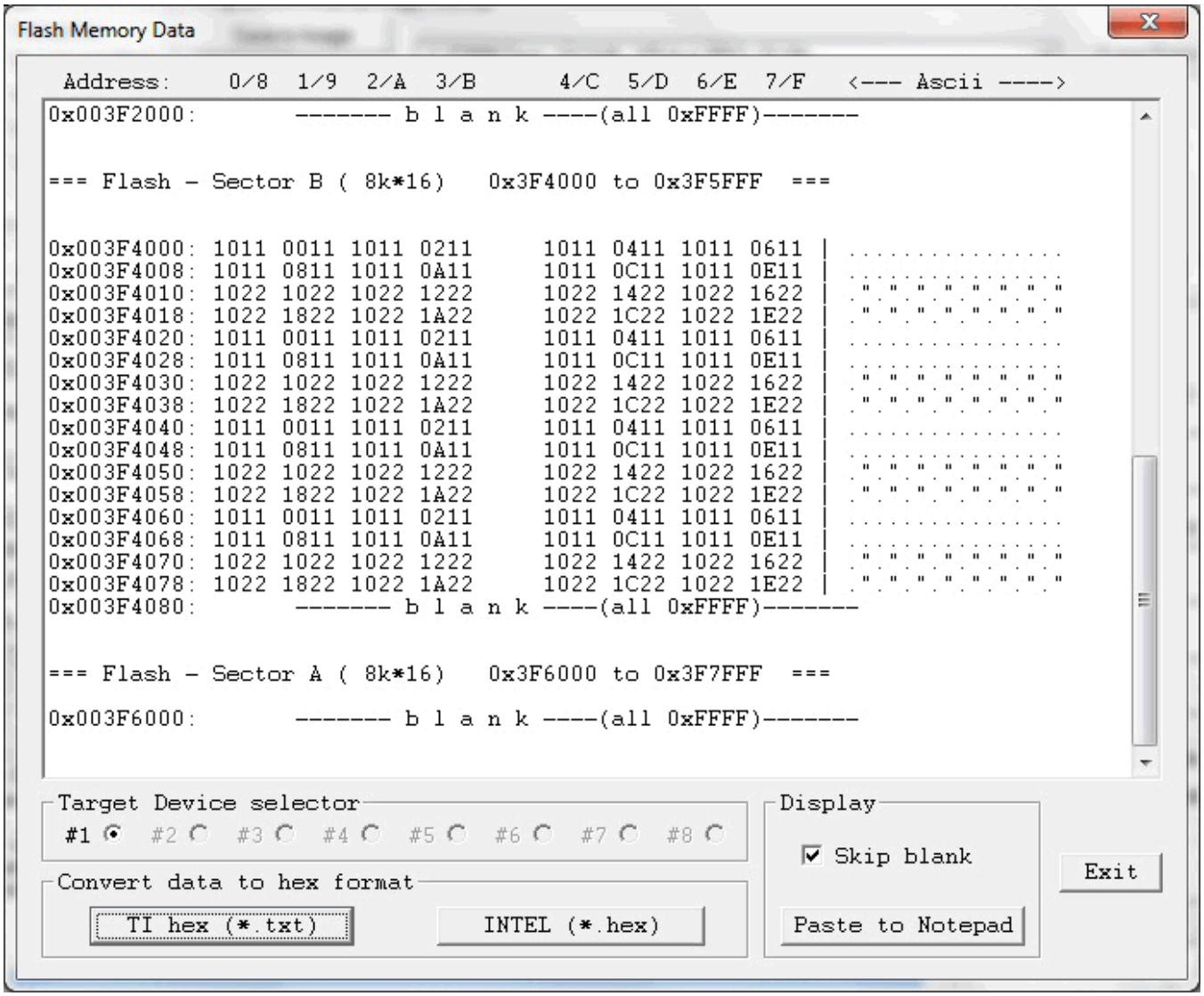
If configuration conflicts arose during programming that indicated that the MCU did not contain sufficient memory for the code to be programmed (either enabled segments or total memory was too small), then the Verify function verifies only the code that was programmed and ignores the code that could not fit in memory. This function succeeds if the code in flash matches the code file, and fail otherwise.

2.1.1.6 Read

Click the Read button in the Main Dialog GUI to read the contents of the target MCU's flash memory. Use the Memory Options configuration screen shown in Figure 2-2 to specify which addresses should be read (Section 2.1.5 describes in detail how to use the memory configuration window).

Once used, data is displayed in the Flash Memory Data window as shown in Figure 2-4. This window can be selected in the View→Flash Memory Data pulldown menu. The Flash Memory Data viewer, shown in Figure 2-4, displays the code address on the left side, data in hex format in the central column, and the same data in ASCII format in the right column. The contents of the code viewer can be converted to TI (*.txt) or Intel (*.hex) file format by clicking on the "TI hex" or "INTEL" button.

Copyright © 2014–2016, Texas Instruments Incorporated



NOTE: This window displays the code addresses on the left side, data in hex format in the center column, and the same data in ASCII format in the right column.

Figure 2-4. Flash Memory Data

2.1.2 Programming >From Image

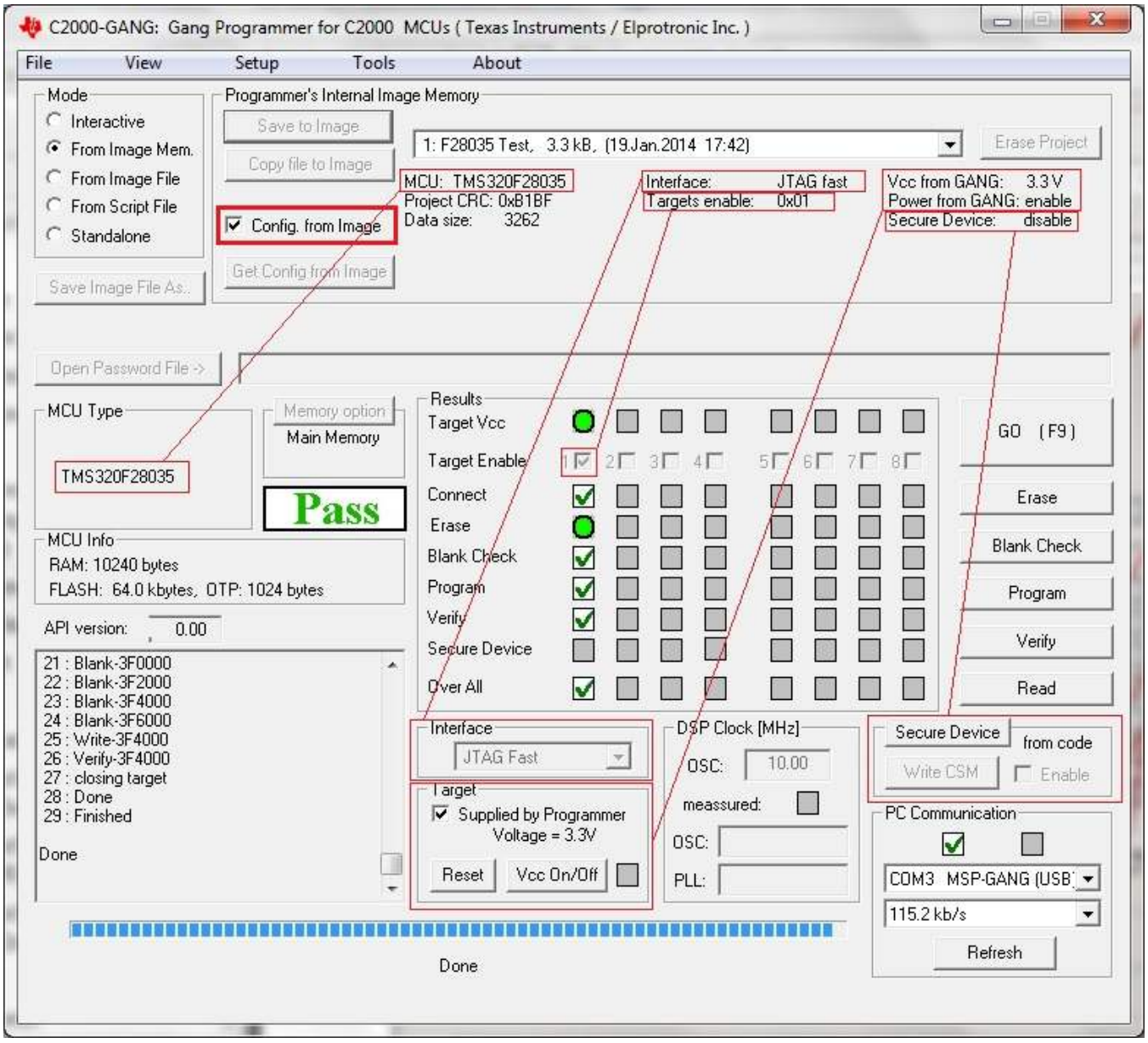
A programming configuration like the one created in Section 2.1.1 can be stored in the form of an image. The advantage of an image is that it contains both the configuration options necessary for programming as well as the code files that are flashed to target devices. Moreover, only images can be saved to internal C2000 Gang Programmer memory and used in Standalone mode, in which the programmer can operate without being connected to a PC. Using the From Image mode allows the user to test images with full GUI support before committing them to production.

After an image has been created, it can be used to greatly simplify programming by using the procedure described in Section 2.1.6. Figure 2-5 shows the main dialog GUI where the From Image option is selected for programming (top left corner). Here the user can load an image from C2000 Gang Programmer internal memory. An image can be created in Interactive Mode and saved to the programmer. One of 16 different images can be selected from internal memory, or one image from each external SD-Card can be used.

Copyright © 2014–2016, Texas Instruments Incorporated

NOTE: C2000 Gang Programmer internal memory and SD-Card are mutually exclusive.

To avoid confusion during programming, connecting an SD-Card to the C2000 Gang Programmer disables its internal memory used for other images. Therefore, when an SD-Card is connected to the programmer only the image on the SD-Card is usable or accessible. If the SD-Card is empty, or contains a corrupted image, then it must be disconnected before C2000 Gang Programmer internal memory can be used.



NOTE: This figure shows the From Image Mem. mode (see the Mode section near the top left corner). The user can load an image from C2000 Gang Programmer internal memory. Saved images contain all configuration necessary for programming and all code files. An image can be created using the Interactive Mode and saved to the programmer. One of 16 different images can be selected from internal memory, or one image from each external SD-Card can be used.

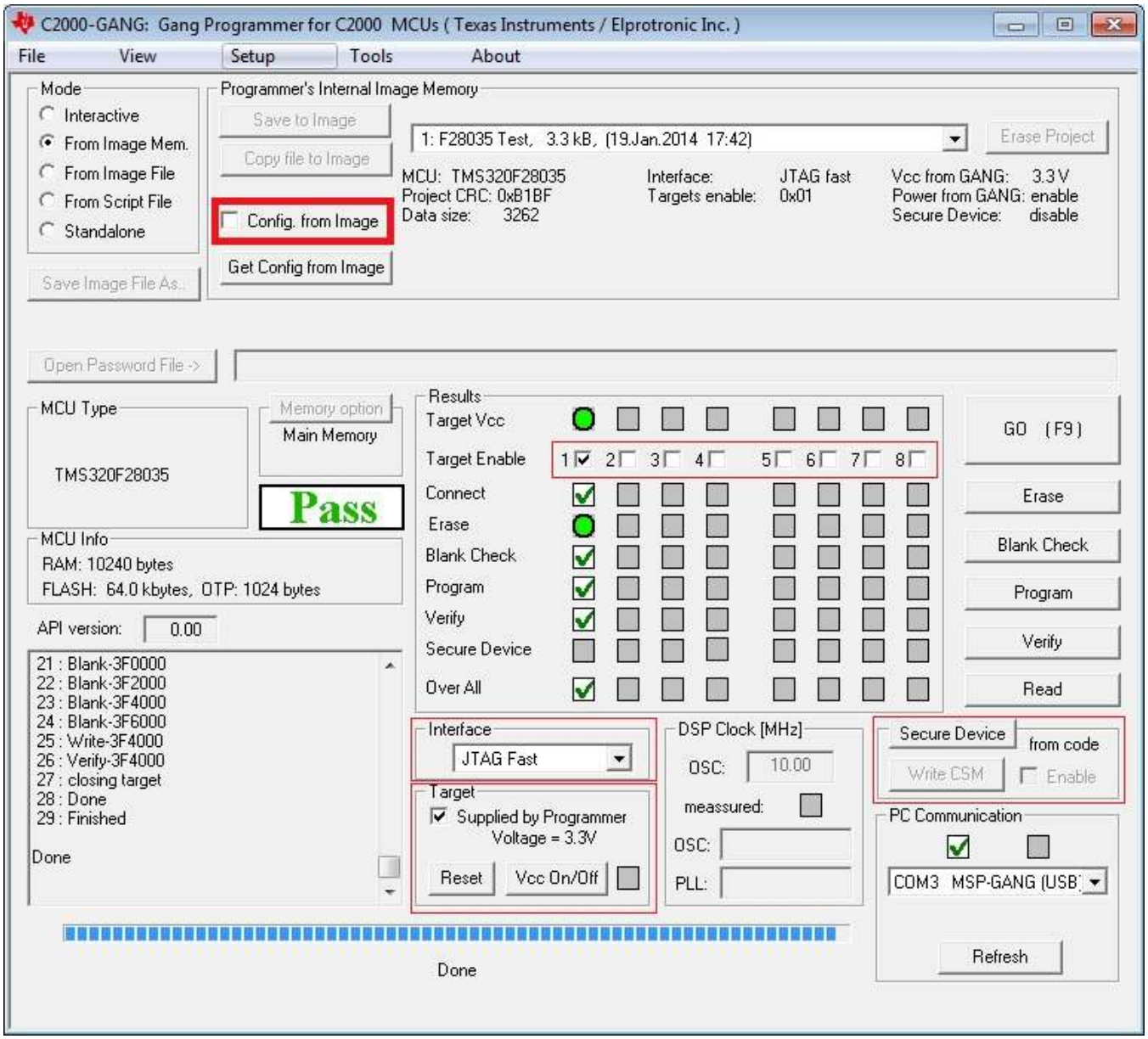
Figure 2-5. Main C2000 Gang Programmer Dialog GUI, From Image Mode

Copyright © 2014–2016, Texas Instruments Incorporated

Figure 2-5 highlights several parts of the GUI. The dropdown menu in the Programmer's Internal Image Memory group (top right) is used to select which image is used for programming, because up to 16 different images might be available. In the same group, the Config. from Image option is enabled, meaning that all configurations options, such as which devices are enabled or power options are being taken from the image.

Sometimes it is useful to use the basic files from an image, such as the MCU type and code files, but also make a few minor modifications to test a different configuration.Figure 2-6 shows the additional configuration options available when the Config. from Image button is disabled. These are highlighted in red and include which devices are enabled for programming, target V_{CC}, interface, communication, and security. However, these changes cannot be committed to the image. If the user wishes to change the current image's configuration or code files then the image needs to be recreated using the original project file and procedure described inSection 2.1.6.

Copyright © 2014–2016, Texas Instruments Incorporated



NOTE: This figure shows the From Image Mem. mode (top left corner). The Config. from Image option is disabled in this example, allowing the user to change various but not all configuration settings from the image. The configuration options that can be changed are highlighted in red. One of the options that cannot be changed, for example, is the target processor type.

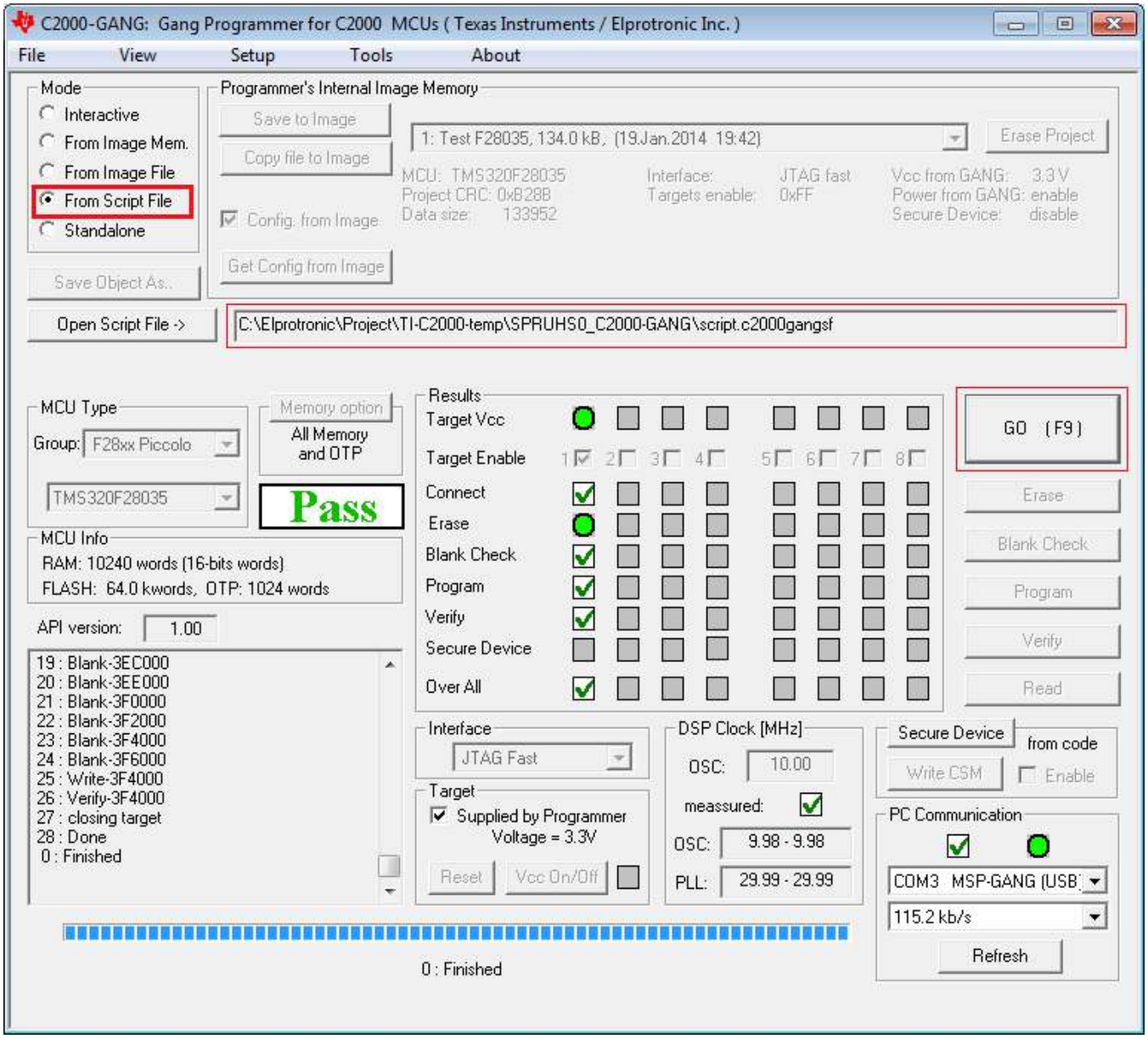
Figure 2-6. Main C2000 Gang Programmer Dialog GUI, From Image Mode and Custom Configuration Enabled

2.1.3 Programming From Script

Use this option to create a script file to automate more complicated programming procedures. Scripts can create functions that open message boxes, target devices, change code files, and any other sequences of reconfigurations up to a total of 1000 commands. Repeated series of instructions can be encompassed into functions for easier programming. The stack supports a call depth of up to 50 CALLs (CALL inside CALL inside CALL, and so on), which is sufficient for most nonrecursive programs.

Copyright © 2014–2016, Texas Instruments Incorporated

Figure 2-7 shows the main dialog GUI where the From Script option is selected for programming (top left corner). A script file is selected using the Open Script File button and it specifies all configuration options, and the code files to be used for programming. A script can be created using any text editor and saved in a simple text file. Follow these guidelines to create a script.



NOTE: This figure shows the From Script mode (see the Mode section near the top left corner). A script file is selected using the Open Script File button and it specifies all configuration options, and the code files to be used for programming. In addition, the script can call individual functions, such as Program or Verify, in the order specified by the programmer.

Figure 2-7. Main C2000 Gang Programmer Dialog GUI, From Script

2.1.3.1 Script Limitations

- Up to a total of 1000 command lines can be used. Empty lines and comments are ignored.
- The stack supports a call depth of up to 50 CALLs (CALL inside CALL inside CALL, and so on).

Copyright © 2014–2016, Texas Instruments Incorporated

2.1.3.2 Command Syntax

- White spaces before instructions, labels, and comments are ignored.
- **;** – Start of a comment. All characters in the same line after the start of a comment are ignored.

NOTE: A comment cannot be placed after a filename.

For example, when specifying a config file to be loaded, a path to a file must be given. This filename cannot be followed by a comment.

- **>** – Start of a label. Place the label name after the character with no spaces in between.

NOTE: A line with a label cannot also contain a command or another label.

For example, this would be illegal:

```
>START VCCOFF
```

2.1.3.3 Instructions

MESSAGE – Message declaration. Contents must be placed between quotes below a message declaration. Maximum of 50 content lines. Example:

```
MESSAGE
"Hello."
"This is my script."
```

GUIMSGBOX setting – Enable or disable pop-up message boxes in the GUI (warning and errors). Setting can be either ENABLE or DISABLE.

IFGUIMSGBOXPRESS option – Apply the option when a message box created by GUI is generated. Option can be OK or CANCEL.

MESSAGEBOX type – Create a pop-up message box with buttons. Contents must be placed between quotes below message declaration. Maximum of 50 content lines. Message box types are:

- OK – One button: OK.
- OKCANCEL – Two buttons: OK and CANCEL
- YESNO – Two buttons: YES and NO
- YESNOCANCEL – Three buttons: YES, NO, and CANCEL

  Example:

```
MESSAGE YESNOCANCEL
"You have three choices:"
"Press yes, no, or cancel."
```

GOTO label – Jump to instruction immediately following the label.

SLEEP number – Pause a number of milliseconds, between 1 and 100000.

F_FROMIMAGEMODE – Switch to Image mode.

CALL label – Call procedure starting at the instruction immediately following the label. Stack saves return address.

RETURN – Return from CALL.

Copyright © 2014–2016, Texas Instruments Incorporated

IF condition operation – Test condition and if true then perform operation. The condition can be one of the following:

- BUTTONOK – OK button is pressed in the message box.
- BUTTONYES – YES button is pressed in the message box.
- BUTTONNO – NO button is pressed in the message box.
- BUTTONCANCEL – CANCEL button is pressed in the message box.
- DONE – Previous process (for example, GO or Read File) finished successfully.
- FAILED – Previous process (for example, GO or Read File) failed.

The operation can be one of the following:

- GOTO label
- CALL label SLEEP number – Pause a number of milliseconds, between 1 and 100000.

F_LOADCFGFILE filename – Load configuration file. Provide a full path and filename.

F_LOADCODEFILE filename – Load code file. Provide a full path and filename.

F_APPENDCODEFILE filename – Append code file. Provide a full path and file name.

F_VCCOFF – Turn V_{CC} OFF from programming adapter to target device.

F_VCCON – Turn V_{CC} ON from programming adapter to target device.

NOTE: V_{CC} from FPA must be enabled first using configuration file.

F_RESET – Perform RESET function from main dialogue screen.

F_GO – Perform GO function from main dialogue screen.

F_ERASEFLASH – Perform ERASE FLASH function from main dialogue screen.

F_BLANKCHECK – Perform BLANK CHECK function from main dialogue screen.

F_WRITEFLASH – Perform WRITE FLASH function from main dialogue screen.

F_VERIFYFLASH – Perform VERIFY FLASH function from main dialogue screen.

F_SECURE– Perform Secure Device function from main dialogue screen.

NOTE: Blows fuse regardless of enable option.

If the F_SECURE command is used, then security is enabled even if the Write CSM enable option is disabled.

F_SETIMAGENUMBER number – Choose image number between 1 and 16 from C2000 Gang Programmer internal memory.

Copyright © 2014–2016, Texas Instruments Incorporated

F_INTERACTIVEMODE – Switch to Interactive mode.

F_NEWRESULTFILENAME – Provide a full path and name of the result file.

F_APPENDRESULTFILENAME – Provide a full path and name of the file where the result should be appended.

F_COMMENTTOFILE – Add a comment at the beginning of the result stream.

F_RESULTTOFILE – Save result to the result file specified by F_NEWRESULTFILENAME or F_APPENDRESULTFILENAME. The following data is saved:

```
Finished task mask: HHHH (16 bits task mask)
Cumulative target mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Requested target mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Connected target mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);

Erased target mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Blank Check target mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Programmed target mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Verified target mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Secured target mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
error_no: error number

VTIO in mV: VTio in mV
Vcc
Error target mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Vcc Cumulative Err mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);

JTAG Init target mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Already Secured mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);

Wrong MCU ID mask: HH (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
```

TRACEOFF – Disable tracing.

TRACEON – Enable tracing and log to the Trace-Scr.txt file in the current working directory. This option is useful for debugging. The trace file contains the sequence of all executed commands from the script file annotated with line numbers. Line numbers are counted without empty lines and without lines containing only comments.

END – End of script.

Copyright © 2014–2016, Texas Instruments Incorporated

The following example script executes this sequence of commands:

1. Label START is created.
2. V_{CC} from programmer to target device is turned OFF.
3. Message box notifies the user of V_{CC} setting and asks for permission to proceed with buttons OK and CANCEL. The program halts here until a button is pressed.
4. If CANCEL was pressed then GOTO finish label (ends the script).
5. If CANCEL was not pressed (in this case this implies that OK was pressed) then load configuration file test-A.c2000cfg to the C2000 Gang Programmer. Configuration file test-A.c2000cfg should be prepared before running this script using Interactive mode.
6. Message box asks the user to proceed. The program halts until OK is pressed.
7. The C2000 Gang Programmer programs the target device using the GO function.
8. Message box asks the user if the test succeeded giving a YES or NO choice.
9. If NO was pressed then GOTO START label (start of script).
10. If NO was not pressed (in this case this implies that YES was pressed) then load configuration file finalcode.c2000cfg to the C2000 Gang Programmer.
11. The C2000 Gang Programmer programs the target device using the GO function. The new configuration changes the code file.
12. Script jumps to the beginning using GOTO START. This can be used to wait for the next target device to be connected.
13. Label finish is created.
14. Script ends.

```
;=====================================================
; Script file - demo program
;-----------------------------------------------------
>START
  VCCOFF
  MESSAGEBOX OKCANCEL
    "VCC if OFF now. Connect the test board."
    "When ready press the button:"
    " "
    "OK - to test the board"
    "CANCEL - to exit from program"
  IF BUTTONCANCEL GOTO finish
  F_LOADCFGFILE C:\Elprotronic\Project\Cpp-Net\C2000\test-A.c2000cfg
  MESSAGEBOX OK
    "Press OK to download the test program."
  F_GO
  MESSAGEBOX YESNO
    "Press YES when the test finished successfully."
    "Press NO when the test failed."
  IF BUTTONNO GOTO START
  F_LOADCFGFILE C:\Elprotronic\Project\Cpp-Net\C2000\finalcode.c2000cfg
  F_GO
  GOTO START
>finish
END
;=====================================================
```


Copyright © 2014–2016, Texas Instruments Incorporated

2.1.3.4 Commands Combined With the Executable File

Programming executable file can be opened with the following commands:

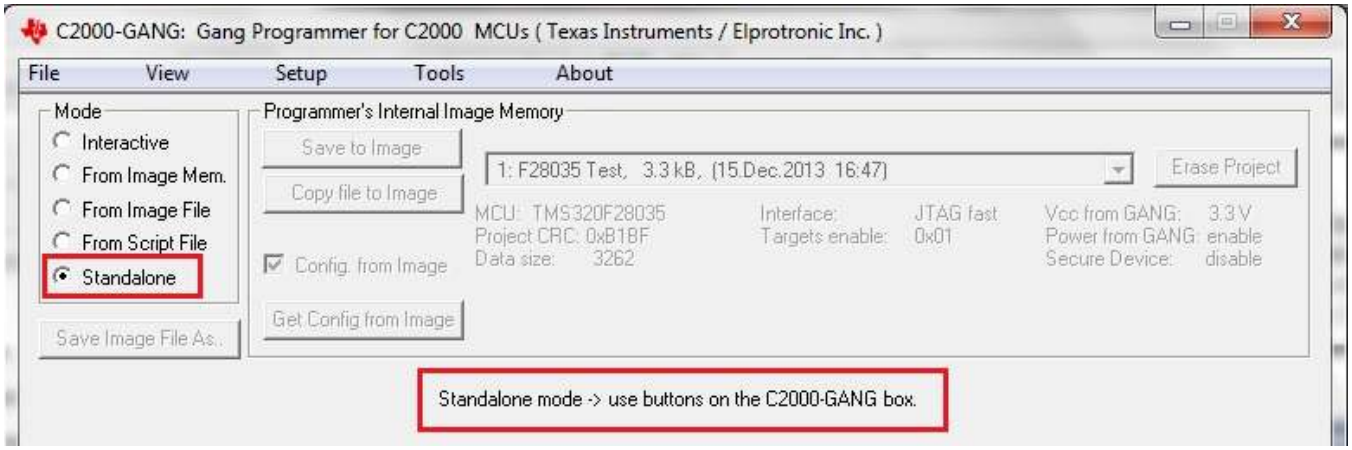
```
-prj project file with file name or full path and name.

-sf script file with file name or full path and name.-
prj project file with file name or full path and name.
For example:
C2000-GANG.exe -sf test.c2000gangsf
or
C2000-GANG.exe -prj test1.c2000gangproj
or
C2000-GANG.exe -prj test1.c2000gangproj -sf test.c2000gangsf
```


Copyright © 2014–2016, Texas Instruments Incorporated

2.1.4 Programming in Standalone Mode

The C2000 Gang Programmer supports a Standalone mode of programming target devices. In this mode the C2000 Gang Programmer can only use images for programming because they contain a complete configuration and code files necessary for the procedure. If the user has not already created an image then follow the procedure outlined in Section 2.1.6. When viewed from the GUI, Figure 2-8 shows that all GUI options are disabled and the C2000 Gang Programmer hardware buttons have to be used for programming.



NOTE: This figure uses the Standalone mode (see the Mode section near the top-left corner). All GUI options are disabled; the C2000 Gang Programmer can only be operated using physical controls on the programmer itself. Standalone mode allows the user to program a target device using an image either from internal memory (up to 16 different images), or an external SD-Card, without the use of a desktop or laptop computer.

Figure 2-8. Main C2000 Gang Programmer Dialog GUI, Standalone Mode

After images have been downloaded to the internal memory or after an SD card with a valid image is connected to the C2000 Gang Programmer, proceed with programming in Standalone mode. In Standalone mode, control the programmer using the arrows and buttons on the actual C2000 Gang programmer, not the GUI on the PC. Use the arrow buttons (up and down) and the enter button to select a desired image for programming. A description of the selected image is displayed on the bottom line, and it is the same description that was created in the GUI when the Save Image button was pressed (see Figure 2-9).

Copyright © 2014–2016, Texas Instruments Incorporated

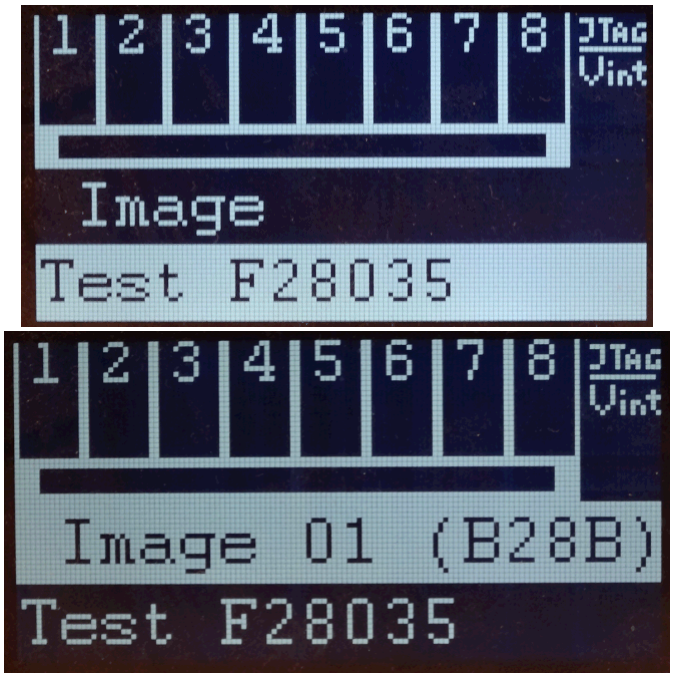


Figure 2-9. Image Option

After the desired image has been selected, press the GO button on the C2000 Gang Programmer hardware to start programming. This button operates the same way as the GO button on the GUI. Progress of the operation in Standalone mode is indicated by a flashing yellow LED and displayed on the LCD display. The result status is represented by green and red LEDs on the C2000 Gang Programmer and details are displayed on the LCD display. If a green LED is ON only, then all targets have been programmed successfully. If only the red LED is displaying, that all results failed. If red and green LEDs are on, then result details should be checked on top of the LCD display. The LCD display shows target numbers 1 to 8 and marks to indicate failure or success: X for failure and V for success. When an error is reported, the bottom line repeatedly displays an error number followed by a short description with time intervals of approximately two seconds.

The selected image contains all necessary configuration options and code files required for programming; however, the user can change the number of target devices being programmed using onboard buttons. On the main display of the C2000 Gang Programmer (see Figure 2-10), use the up or down arrow buttons to find the Target En/Dis option. Press the OK button to enter this menu. A sliding cursor appears below the numbers representing each device at the top of the main display. Use the arrow buttons to underline the device to enable or disable. Press OK to toggle the devices; press Esc to exit to the main menu. Press GO to use the selected image to program the selected devices. If another image is selected or the current image is selected again, the Enable and Disable options reset to what has been configured in the image.

Copyright © 2014–2016, Texas Instruments Incorporated

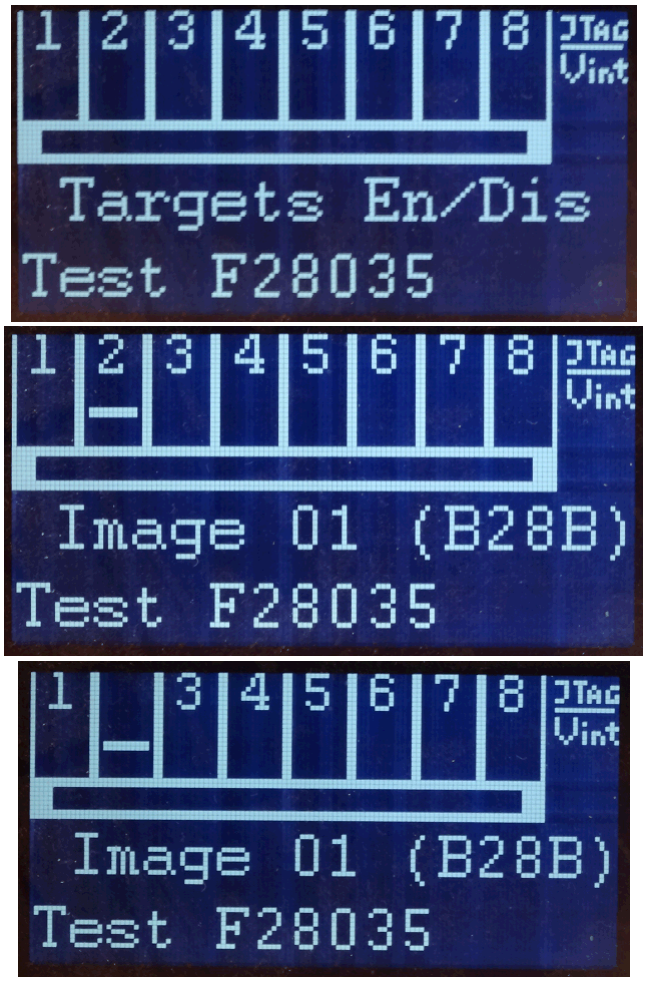


Figure 2-10. Target Enable or Disable Option

In addition to the these options that control programming, the contrast of the LCD display can be changed. Select the Contrast option in the main menu, and press OK. Then use the up and down arrow buttons to adjust the screen contrast. Changes to contrast reset after power down, unless the contrast setting has been set via the GUI on the host computer.

Copyright © 2014–2016, Texas Instruments Incorporated

2.1.5 Memory Setup for GO, Erase, Program, Verify, and Read

The GO, Erase, Program, Verify, and Read operations shown in Figure 2-1 use addresses specified in the Memory Options dialog screen shown in Figure 2-2. The memory setup used by these operations has five main options:

1. Update only – When this option is selected, the GO operation does not erase memory contents. Instead contents of code data taken from the code file are downloaded to flash memory. This option is useful when a relatively small amount of data, such as calibration data, needs to be added to flash memory. Other address ranges should not be included in the code file, meaning that the code file should contain ONLY the data which is to be programmed to flash memory. For example, if the code file contains data as shown in TI format:

   ```
   @3F0000
   25CA 8040 39E3 F802
   @3F4000
   4835 5972 ACB8
   q
   ```

   Then four words of data are written starting at location 0x3F0000 and three words of data starting at location 0x3F4000. The specified addresses should be blank before writing (contain a value of 0xFF). Before the writing operation is actually performed, the C2000 Gang Programmer automatically verifies if this part of memory is blank and proceeds to program the device only if verification is successful.
2. OTP and Flash Memory – This is the most frequently used option during programming. All flash memory is erased before programming, and all contents from the code file are downloaded to the target microcontroller's flash memory. When the microcontroller contains an OTP (One-Time Programmable) segment (for example TMS320F2801, address range 0x3D7800 to 0x3F7BFF), then OTP is modified if data for it was provided in the code file. If the code file does not contain data addressed to the OPT segment, then OTP memory is not modified. Once programmed, OTP memory cannot be erased. Flash memory can be reprogrammed as long as the MCU is not locked.
3. Flash Memory only – Only Flash memory is programmed, OTP segments are not touched. Contents of OTP memory from the code file are ignored.
4. OTP Memory only – Only OTP segments are programmed, Flash memory is not touched. Contents of Flash memory from the code file are ignored.
5. Used by Code File – This option allows main memory segments and OTP memory segments to be modified when specified by the code file. Other flash memory segments are not touched. This option is useful if only some data, like calibration data, needs to be replaced.
6. User defined – This option is functionally similar to options described before, but memory segments are explicitly chosen by the user. When this option is selected, then on the right side of the memory group, the OTP Memory and Flash Memory dialog screens are enabled. The check boxes allow the user to select OTP and flash memory segments to be enabled (erased (not OTP), programmed, verified). Edit lines allow the user to specify the address range (start and stop addresses) for each type of memory. The start address should specify the first word in the segment, and the stop address should specify the last word in the segment (last word is programmed). Depending on the segment size in the chosen MCU, for example 0x1000, the start address should be a multiple of 0x1000; for example, 0x3F0000 or 0x3F1000. The stop address should specify the last word of the segment to be written. Therefore, it should be greater than the start address and point to a word that immediately precedes a memory segment boundary; for example, 0x3F0FFF or 0x3F1FFF.

2.1.6 Creating and Using Images

An image contains the code files and the configuration options necessary for programming of a target device. Images can be stored as a binary file (".c2000gangbin") in internal C2000 Gang Programmer memory (or SD card), or as an image file (".c2000gangimage") on disk for redistribution. Image files intended for redistribution can be encrypted with additional security features described later in this section.

Creating an image is done in Interactive Mode by following the same steps described in Section 2.1.4 followed by pressing the "Save Image File As…" or "Save to Image" buttons. The first button saves the code files and configuration options as a binary file and image file locally on disk, and the second button saves this information directly to the C2000 Gang Programmer internal memory. Note that to use the C2000 Gang Programmer in Standalone mode, you need to program at least one image to internal memory or read a binary file from an SD card (via the SD card connector on the C2000 Gang

Copyright © 2014–2016, Texas Instruments Incorporated

Programmer). If you intend to modify the contents of an image at a later date, it is advisable to save the configuration options as a project. Because an image is read-only, reading a project file is the only way to recreate images easily without reentering the configuration options from scratch. After the project is loaded, a change can be made and a new image with the same name can be created to overwrite the previous one.

In total, 16 different images can be saved internally in the C2000 Gang Programmer or one image can be saved on an SD card. Each image can be selected at any time to program the target devices. The C2000 Gang Programmer also allows the image to be saved in a file, either to be saved on an SD card or to be sent to a customer. In order for the image file to be usable from the SD card, copy only the binary file (".spgangbin") to the SD card and preserve the proper extension (Note that binary files are not encrypted). For redistribution to a customer, the image file can be sent and encrypted with additional security features.

When a new image is saved to a file or to a C2000 Gang Programmer internal memory, an image configuration screen appears (see Figure 2-11). Enter any name up to 16 characters. This name is displayed in the GUI image selector (see Figure 2-1) on the bottom line of the C2000 Gang Programmer LCD screen when the corresponding image is selected. Press OK when the name is entered.

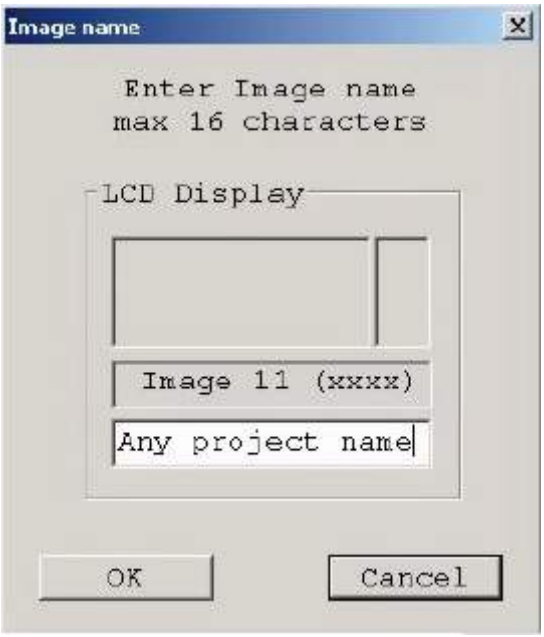
After you have created a programming setup using the steps mentioned above, it is useful to store it in the form of an image. The advantage of an image is that it contains both the configuration options necessary for programming as well as the code files that are flashed to target devices. Moreover, only images can be saved to internal C2000 Gang Programmer memory and used in Standalone mode, where the programmer can operate without being connected to a PC.

Before the user proceeds to making images; however, it is advisable to save the C2000 Gang Programmer setup as a project first. This is recommended because images cannot be modified after they are created; they can only be overwritten. Therefore, if the user wants to change an image that has already been created without recreating the whole configuration from scratch, then it is necessary to load the corresponding project file. After the project is loaded, a change can be made, and a new image with the same name can be created to overwrite the old one.

Images can be saved to the programmer's internal memory or on an external SD-Card. A total of 16 different images can be saved internally, or one image can be saved on an SD-Card. Each image can be selected at any time to program the target devices. The C2000 Gang Programmer also allows the image to be saved in a file, either to be saved on an SD-Card or to be sent to a customer. When the code file and configuration are ready to be saved, press the Save Image button to save to C2000 Gang Programmer internal memory, or the Save Image to file button to save to a file.

Whether the new image being created is saved to a file or to C2000 Gang Programmer internal memory, an image configuration screen appears (see Figure 2-11). Enter any name up to 16 characters. This name is displayed in the GUI image selector (see Figure 2-1), and it is displayed on the bottom line of the C2000 Gang Programmer LCD screen when the corresponding image is selected. Press OK after entering the name.

Copyright © 2014–2016, Texas Instruments Incorporated

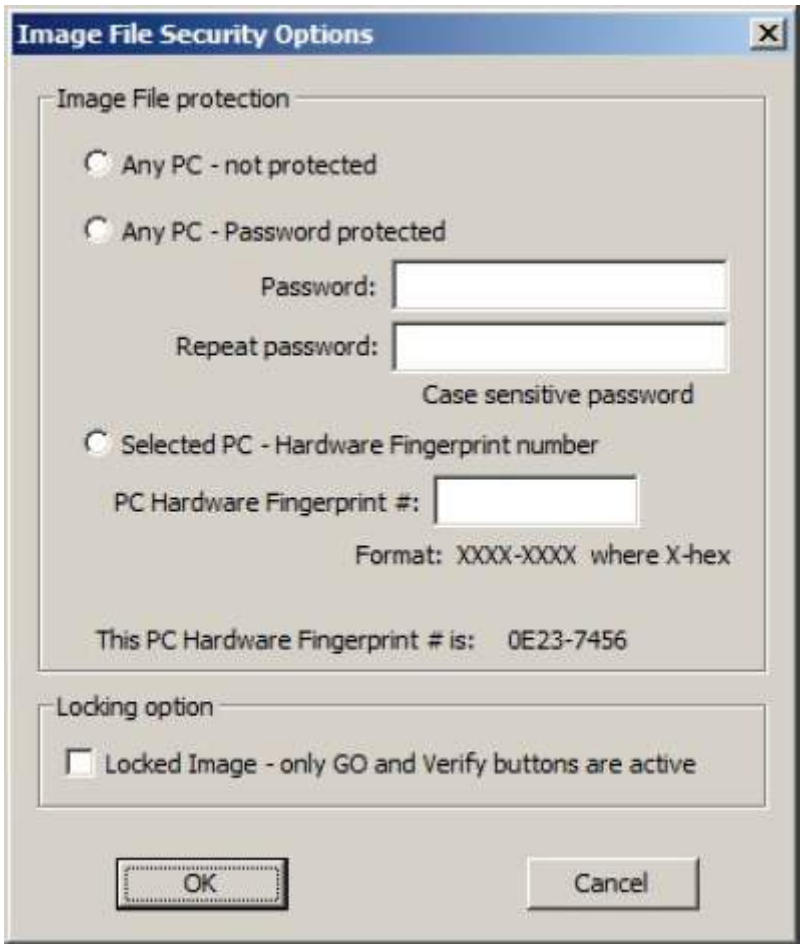


NOTE: The image name is limited to 16 characters. This name is shown on the LCD display of the C2000 Gang Programmer and Image pulldown menu in the GUI.

Figure 2-11. Image Name Configuration Screen

The screen shown in Figure 2-12 allows the user configure what type of security is used to protect the image file. Three options are available; however, for all three options, the contents of the code file are always encrypted and cannot be read.

Copyright © 2014–2016, Texas Instruments Incorporated

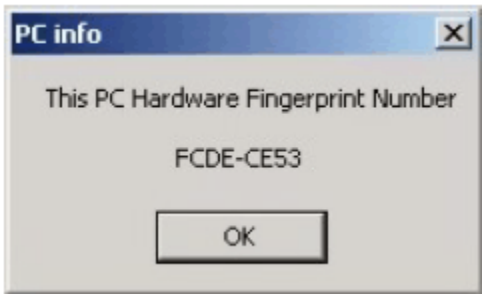


NOTE: During project creation, the user can select to protect project information using various methods.

Figure 2-12. Image File Security Options

1. **Any PC** – Configuration can be opened on any computer using C2000 Gang Programmer software. It can be used for programming only.
2. **Any PC - Password protected** – Configuration can be opened on any computer using the C2000 Gang Programmer software, but only after the desired password has been entered.
3. **Selected PC - Hardware Fingerprint number** – Image can be opened only on the dedicated computer with the same hardware fingerprint number as the number entered in the edited line above. Figure 2-13 shows a window with the hardware fingerprint number. An example usage scenario would involve calling an intended user to provide the hardware fingerprint number of their computer and entering it within this configuration window. This restricts opening this image to only the dedicated computer running C2000 Gang Programmer software.

Copyright © 2014–2016, Texas Instruments Incorporated



NOTE: The fingerprint can be used to secure the project where, for example, only a computer with a matching hardware fingerprint can be used to view and edit the project.

Figure 2-13. Hardware Fingerprint of Computer in Use

The image file can be copied to internal C2000 Gang Programmer memory and used for programming target devices. Select the desired image number in the GUI and press the Load Image from File button (see Figure 2-1). This selected image is subsequently be used for programming target devices.

2.1.7 Programming >From Image File

An image file can be used to program target devices from a self-contained read-only file that has all the necessary configuration options and code files already included. By selecting the "From Image File" Mode, you can use an image file created using the steps described in Section 2.1.6. If the image is password protected, you are prompted to enter the password before you can use the image. Alternatively, if the image is restricted to be used on a specific PC, you are unable to use the image unless your PC matches the hardware fingerprint (for instructions on how to use images from C2000 Gang Programmer internal memory see Section 2.1.2).

Copyright © 2014–2016, Texas Instruments Incorporated

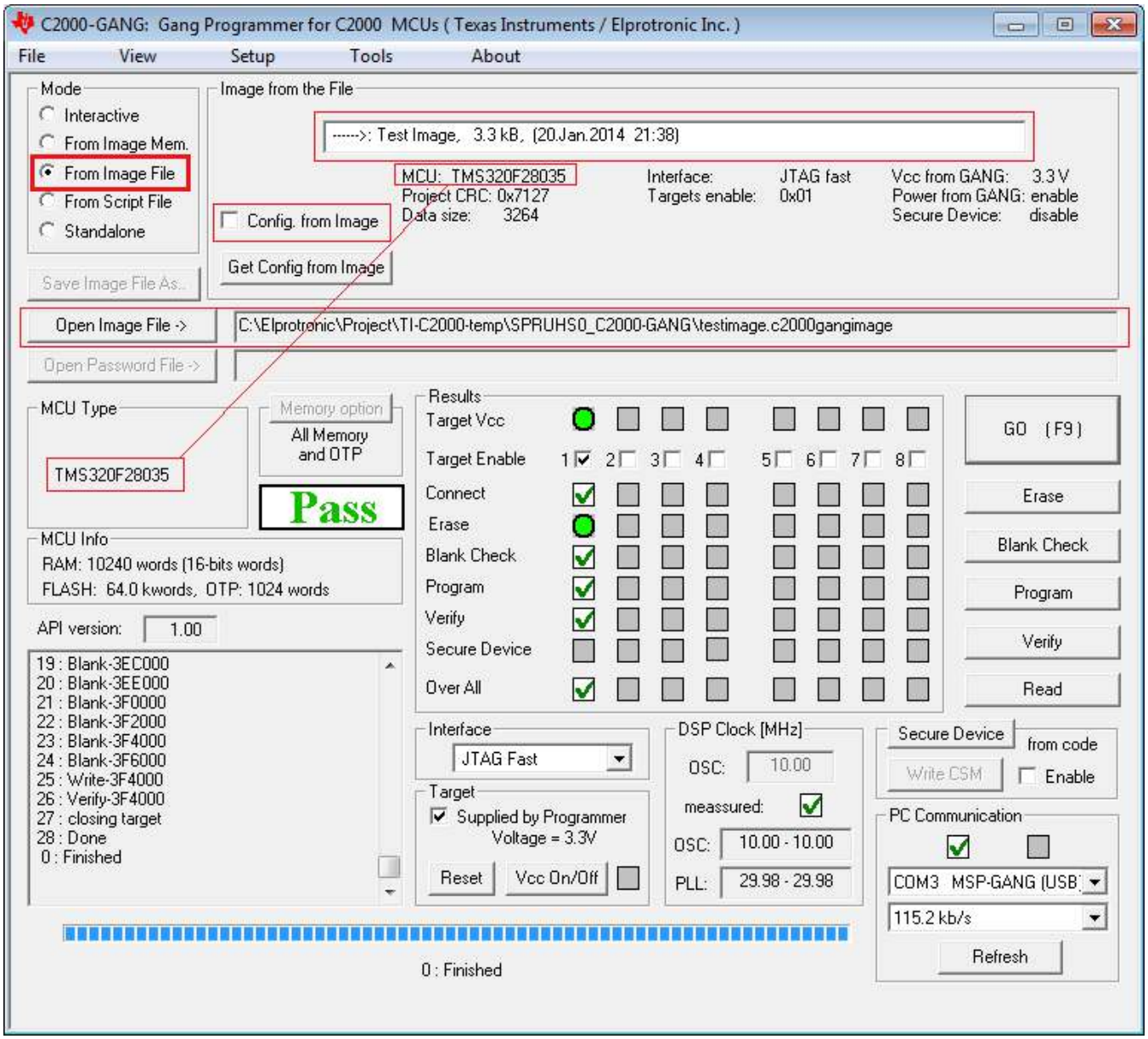


Figure 2-14. Programming From Image File

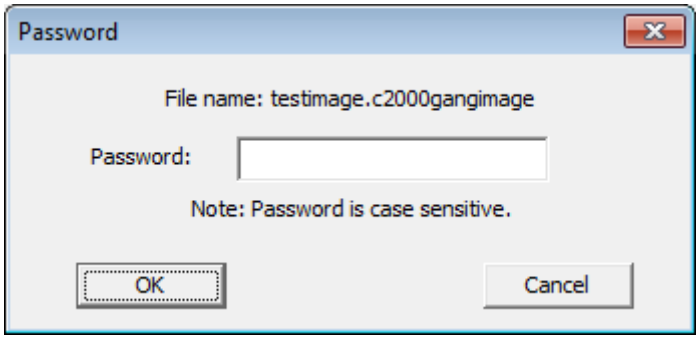


Figure 2-15. Password for Image File

Copyright © 2014–2016, Texas Instruments Incorporated

2.1.8 Programming >From SD Card

The C2000 Gang Programmer can program target devices with an image loaded from an external SD card. To program from an external SD card, copy a binary file (".c2000gangbin") created using steps described in Section 2.1.6 to the root directory of the SD card (preserve the original extension ".c2000gangbin"). If multiple binary files are present in the root directory of the SD card, the first one found is used (the first one found is not necessarily the first one alphabetically). To ensure that the desired binary file is used, verify that only one binary file with the proper extension .c2000gangbin is present in the root directory. The name of the selected file is displayed on the LCD screen of the C2000 Gang Programmer.

When the SD card is connected to the C2000 Gang Programmer, internal memory is disabled, and an image can only be read from the SD card. This mechanism has been deliberately implemented to aid in production, because inserting an SD card to the C2000 Gang Programmer leaves the user with only one option for programming a target device and, therefore, less possibility for misconfiguration errors.

2.1.9 File Extensions

C2000 Gang Programmer software accepts the following file extensions:

Code hex files

*.txt	Texas Instruments
.s19,.s28,*.s37	Motorola
*.hex	Intel

Image files

*.c2000gangbin	binary file, used for saving data in SD card
*.c2000gangimage	image file, can be password protected for distribution

Script files

*.c2000gangsf	script file

Project configuration files

*.c2000gangproj	keep all configuration, file names and data for used project

2.1.10 Checksum Calculation

The checksum (CS) that is displayed on the side of the code file name is used for internal verification. The CS is calculated as the 32-bit arithmetic sum of the 16-bit unsigned words in the code file, without considering the flash memory size or location.

The following formula is used.

```
DWORD CS;
DWORD XL, XH;

  CS = 0;
  for( addr = 0; addr < ADDR_MAX; addr = addr + 2 )
  {
    if(( valid_code[ addr ] ) || ( valid_code[ addr+1 ]))
    {

     if( valid_code[ addr ] )
         XL = (DWORD) code[ addr ];
     else
         XL = 0xFF;
```


Copyright © 2014–2016, Texas Instruments Incorporated

```
    if( valid_code[ addr+1 ] )
        XH = ((DWORD) code[ addr+1 ])<<8;
    else
        XH = 0xFF00;

    CS = CS + XH + XL;
   }
  }
```

As an example, refer to the code file below, which is in the TI hex file (*.txt format).

```
----------------------------------------
@3E8000
40F2

@3E8090
0228 9268 3BDB 8038 0005 FF58
q
----------------------------------------
```

The CS is calculated as shown below:

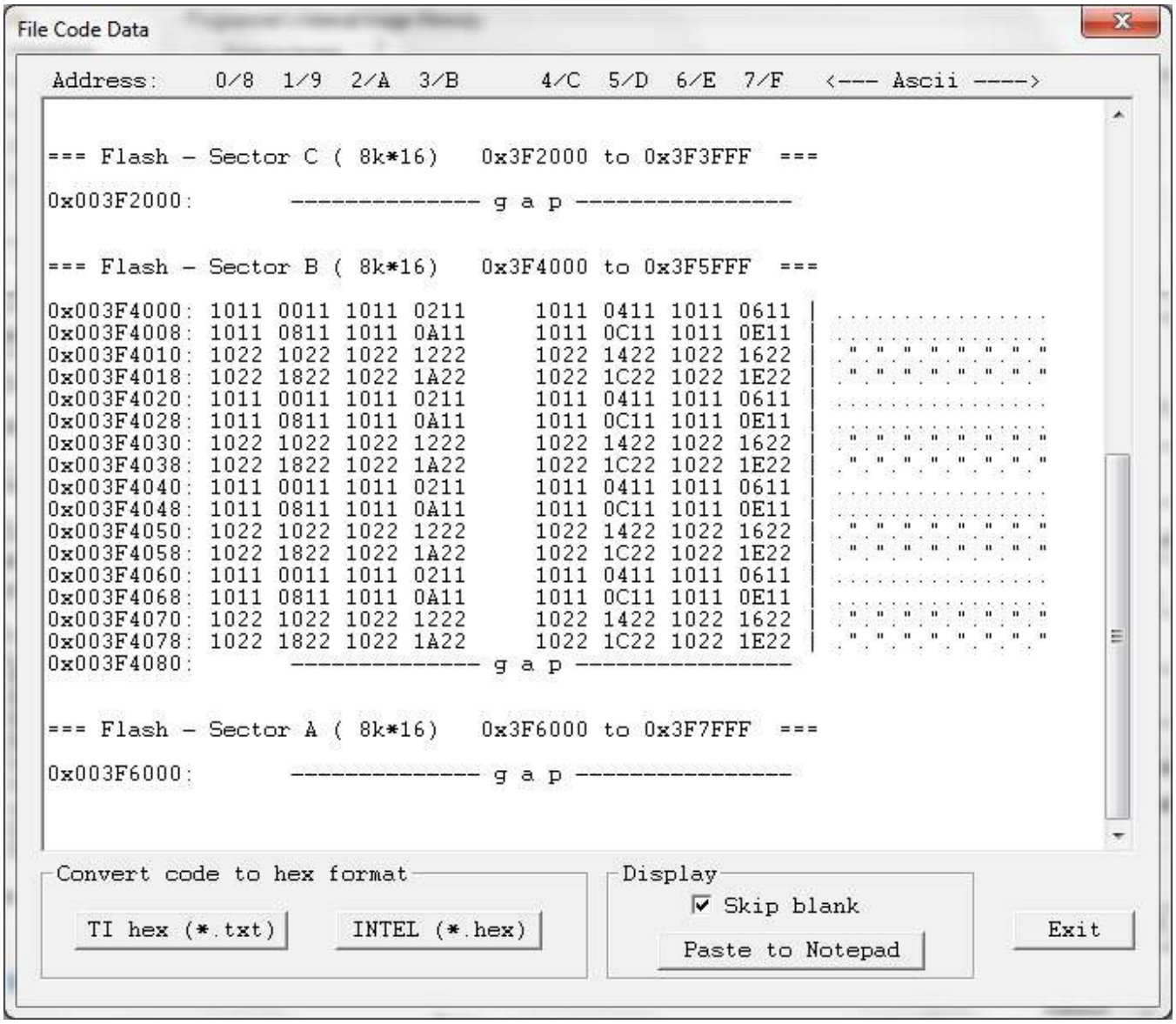
```
CS = 0x40F2 + 0x0228 + 0x9268 + 0x3BDB + 0x8038 + 0x0005 + 0xFF58 = 0x000290F2
```

2.2 Data Viewers

Data from code files and from flash memory can be viewed and compared in data viewers. Contents of the selected file can be viewed by selecting the View→Code File Data option from the dropdown menu. The Code data viewer, shown in Figure 2-16, displays the code address on the left side, data in hex format in the central column, the same data in ASCII format in the right column. Data in hex format is displayed from 0x0000 to 0xFFFF for addresses corresponding to the code file. Data from other addresses is displayed as double dots (..). If code size exceeds flash memory size in the selected microcontroller, this warning message is displayed first.

Data out of the Flash Memory Space of the selected C2000.

Copyright © 2014–2016, Texas Instruments Incorporated



NOTE: The selected option on the bottom ignores all words that have the value of 0xFFFF , which represents empty words.

Figure 2-16. Code File Data

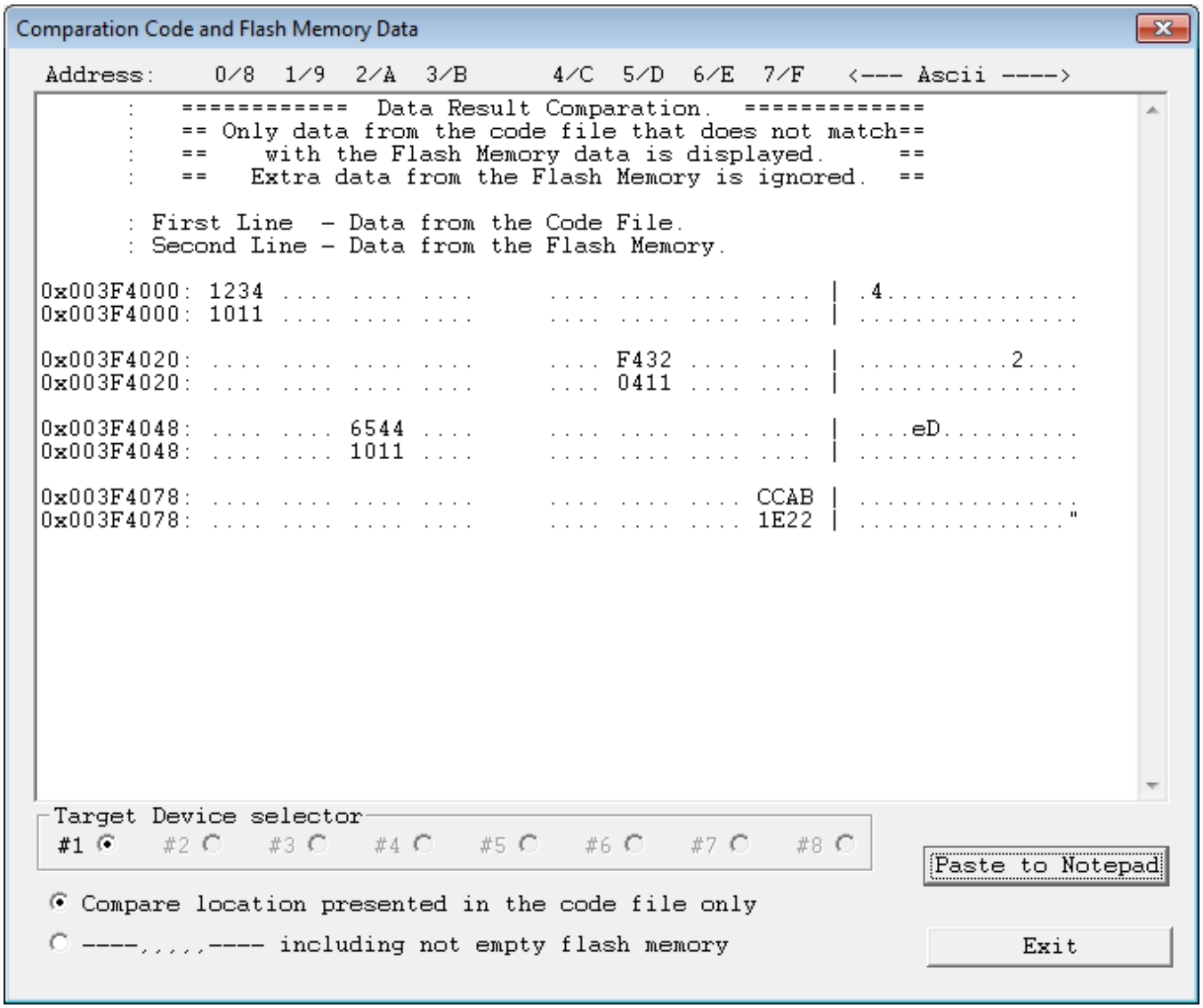
The contents of the code viewer can be converted to TI (*.txt) or Intel (*.hex) file format by clicking on the TI hex or INTEL button.

Contents of flash memory data can be viewed by selecting the View→Flash Memory Data option from the dropdown menu. To be able to see flash memory contents, the Read button must be used first (as described in Section 2.1.1). The Flash Memory Data viewer displays the memory addresses, data in hex and ASCII format in the same way as the Code data viewer shown in Figure 2-16.

Contents of the code file and flash memory can be compared and differences can be displayed in a the viewer by selecting the View→Compare Code & Flash Data options from the dropdown menu. Only data that are not the same in the code file and the flash memory are displayed. The first line displays code file data, and the second line displays flash memory data as shown in Figure 2-17.

Copyright © 2014–2016, Texas Instruments Incorporated

The Compare location presented in the code file only option is chosen by default. This option allows the user to view differences between Code file data and corresponding flash contents (compared by address). Additional data in the flash like DCO calibration and personal data is not compared but can be displayed if desired. If all the aforementioned data are identical, then a "No difference found" message is displayed on the screen.



NOTE: Only words that differ are shown. The selected option on the bottom of the figure specifies that only memory segments corresponding to the code file should be compared. The second option, if selected, performs the comparison and shows any remaining contents of flash memory that do not correspond to the code file.

Figure 2-17. Comparison of Code and Flash Memory Data of the Target Microcontroller

Copyright © 2014–2016, Texas Instruments Incorporated

2.3 Status Messages

The current status is always displayed at the bottom of the progress bar, as shown in Figure 2-1, and previous status and error messages are shown in the history window in the bottom left corner. are displayed in the report window.

All procedures in the C2000 Gang Programmer are divided into small tasks to be executed in series. When first task is finished successfully, then the next task is started. Each task has its own consecutive number assigned by the task manager when the image is created. The tasks are listed below:

- Open Target Device
- Close Target Device
- Erase
  - Segment
  - Main memory
  - OTP memory
- Blank check
- Program
- Gang Program (program unique data to each target)
- Write RAM
- Write GANG RAM (write unique data to each target)
- Verify
- Read memory
- Secure device
- Unlock Target
- Download Firmware
- Start Firmware
- Start DSP Firmware
- Set PLL Frequency
- Get PLL Frequency Status
- Get Initialization Data
- Reset
- Toggle Vcc
- Display Results
- Read Retain Data
- Restore Retain Data
- Finish

For example, the operations Erase, Program, and Verify execute the following tasks:

- Open Target Device
- Erase
- Blank check
- Program
- Verify
- Close Target Device
- Finish

These tasks execute the easiest programming process in small MCU devices. The aforementioned tasks can be divided into smaller tasks that only erase one segment, or erase one block of the main memory. For that reason, many more tasks are displayed in the report window than are described above. For example, when programming the C2000 F28035 the following information would be displayed in the report window:

Copyright © 2014–2016, Texas Instruments Incorporated

```
Executing Main Process...
=== F28035 Test ===
.............
  2 : init target
  3 : unlock CSM
  4 : Download FW
  5 : Start API
  6 : Set PLL freq.
  7 : Get Init Data
  8 : Get PLL status
  9 : Erase-3E8000
 10 : Erase-3EA000
 11 : Erase-3EC000
 12 : Erase-3EE000
 13 : Erase-3F0000
 14 : Erase-3F2000
 15 : Erase-3F4000
 16 : Erase-3F6000
 17 : Blank-3E8000
 18 : Blank-3EA000
 19 : Blank-3EC000
 20 : Blank-3EE000
 21 : Blank-3F0000
 22 : Blank-3F2000
 23 : Blank-3F4000
 24 : Blank-3F6000
 25 : Write-3F4000
 26 : Verify-3F4000
 27 : closing target
 28 : Done
 29 : Finished
Done
```

This report indicates that the main memory block has been erased (tasks 9 to 16), blank checked (tasks 17 to 24), programmed (task 25), and verified (task 26). Finally, access to target devices is closed, and the programming process is finished. Length of task description (including consecutive task number) is limited to 16 characters to be able display this information on the third line of the C2000 Gang Programmer LCD display.

The C2000 Gang Programmer can process up to 1000 tasks per one image saved in internal memory. Having that number of available tasks and one or more code files saved in internal memory (total memory footprint of up to 512 kbytes (256 kwords) in one image), the C2000 Gang Programmer gives the user significant flexibility to perform custom programming procedures. If for any reason the code files and task scripts require more than 512 kbytes of memory, then the next image memory can be combined with the first one for one larger image block (1Mbyte or more). The C2000 Gang Programmer has internal flash memory of 8Mbyte that can, if desired, all be used to form one image with a memory footprint of 8Mbytes.

Error messages are displayed similarly to status messages; however, programming is terminated if the error is related to all target devices. Subsequently, if the problem is resolved or the faulty target device is disabled, then the programming procedure can be restarted to complete the programming process. The result for all devices is reported in the results section (green or red icons). When the global status is reported as FAIL, see the result section for details. Similarly, the C2000 Gang Programmer uses red and green LEDs to indicate the result of its operations (red indicates failure) and details are displayed on the LCD display. Below is the list of errors reported in the C2000 Gang Programmer.

```
ERR_NONE,                   "Operation successful.",

// errors reported by GANGC2000 adapter

ERR_NO_FIRMWARE,            "BOOT Firmware only is in the C2000-
GANG! The API Firmware should be downloaded.",
ERR_FW_NO_CRC,              "API Firmware CRC is not present! The API Firmware should be
reloaded.",
ERR_FW_CRC_ERR,             "API Firmware CRC error! The API Firmware should be reloaded.",
ERR_BOOT_CRC_ERR,           "BOOT CRC error in the C2000-GANG!",
```


Copyright © 2014–2016, Texas Instruments Incorporated

```
ERR_ACCESS_KEY_CRC,          "CRC Access key. Key corrupted. Access to programmer is blocked.",
ERR_INVALID_ACCESS_KEY,      "Invalid programmer's access key. Access to programmer is blocked.",
ERR_UNKNOWN_INTERFACE,       "Unknown interface",
ERR_VCC_TOO_LOW,             "Vcc is too low  ",
ERR_VCC_TOO_HIGH,            "Vcc is too high ",
ERR_VTIO_TOO_LOW,            "VtIO is too low ",
ERR_VTIO_TOO_HIGH,           "VtIO is too high",
ERR_HEADER_PSA,              "Header CRC      ",
ERR_SCRIPT_PSA,              "Script CRC      ",
ERR_EXCEED_SCRIPT_NO,        "Exceed script no",
ERR_UNKNOWN_SCRIPT_CMD,      "Script command ?",
ERR_TARGET_DEV_INIT_ERR,     "MCU device init.",
ERR_RAM_FW_DOWNLOAD,         "RAM FW download ",
ERR_BLANK_CHECK,             "Blank check err ",
ERR_RD_VERIFY,               "Read verify err ",
ERR_FLASH_WRITE,             "Flash write err ",
ERR_FLASH_WR_INIT,           "Image FL WR init",
ERR_FLASH_BP_LOCKED,         "Image Flash lock",
ERR_INVALID_SCRIPT_TYPE,     "Invalid Script T",
ERR_SIZE_TOO_HIGH,           "Size too high   ",
ERR_TARGET_DEV_ID_ERR,       "Used wrong MCU  ",
ERR_TARGET_IR_INTERRUPTED,   "IR Interrupted  ",
ERR_WRONG_INFO_PAGE,         "Page Info number out of range",
ERR_ADDR_TOO_HIGH,           "Address too high",
ERR_INVALID_TARGET_NO,       "Target number out of range",
ERR_GANG_FLASH_WRITE,        "Gang Flash write error",
ERR_SD_READ_INVALID_RESPONSE, "SD Card - Read Response Error",
ERR_SD_BOUNDARY_ADDRESS,     "SD Card - Boundary Address Error",
ERR_SD_INIT_TIMEOUT,         "SD Card - Initialization timeout",
ERR_SD_READ_TIMEOUT,         "SD Card - Read timeout",
ERR_SD_INIT,                 "SD Card - Initialization Error",
ERR_SD_CRC7,                 "SD Card - CRC7 Error",
ERR_SD_CRC16,                "SD Card - CRC16 Error",
ERR_SD_WRITE_CRC,            "SD Card - Write CRC Error",
ERR_SD_DATA_WRITE,           "SD Card - Data Write Error",
ERR_SD_WRITE_TIMEOUT,        "SD Card - Write Timeout",
ERR_SD_READ_MBR,             "SD Card - MBR Sector Error",
ERR_SD_VOLUME,               "SD Card - Volume Error",
ERR_ADDR_IN_FILE,            "SD Card - Address in File Error",
ERR_READ_FILE,               "SD Card - Read File Error",
ERR_FILE_NOT_FOUND,          "SD Card - File not found",
ERR_VERIFY,                  "Verification Error",
ERR_INTERACTIVE_RX_DATA,     "Rx data error   ",
ERR_SECURE_KEY,              "Secure Key Error",
ERR_SECURE_DEVICE,           "Secure Device Er",
ERR_TARGET_NOT_OPEN,         "Target not open ",
ERR_SIZE_ERROR,              "Size err        ",
ERR_CSM_UNLOCK,              "CSM unlock error",
ERR_DSP_FW_OPCODE,           "DSP firmware opcode error",
ERR_DSP_FW_TIMEOUT,          "DSP firmware timeout",
ERR_DSP_FW_START,            "DSP firmware start",
ERR_DSP_PLL_FREQ,            "DSP CLK frequency error",
ERR_ERASE_TIMEOUT,           "Flash Erase timeout",
TASK_IN_PROGRESS,            "Info 0xB0: Task in progress",

//errors from GANGC2000 DLL

ERR_COMM,                    "Communication - Frame has errors !",
ERR_OPEN_COMM,               "Unable to open COM port - already in use?",
ERR_CLOSE_COMM,              "Unable to close COM port !",
ERR_SET_COMM_STATE,          "Unable to modify COM port state !",
ERR_SYNC,                    "Synchronization failed. Programmer connected?",
ERR_RX_HDR_TIMEOUT,          "Timeout during operation - Correct COM port selected?",
ERR_WRONG_BAUDRATE,          "Wrong baud rate specified !",
ERR_COMM_BAUDRATE_CHANGE,    "Communication Port baud rate change",
ERR_COMM_DIAGNOSTIC_RESPONSE, "Communication port - diagnostic response error",
```


Copyright © 2014–2016, Texas Instruments Incorporated

```
ERR_OPEN_COMM_INVALID_HANDLE, "Open Comm port - invalid handle value",
ERR_OPEN_SETUP_COMM,          "Invalid Comm Port Setup",
ERR_OPEN_COMM_TIMEOUT,        "Open Comm Port timeout",
ERR_GET_COMM_STATE,           "Get Comm Port state error",
ERR_CMD_NOT_COMPLETED,        "Command did not complete correctly !",
ERR_CMD_FAILED,               "Command failed or not defined or Target not accessible !",
ERR_READ_INI,                 "Could not read 'default.c2000gangcfg'! ",
ERR_BAD_RECORD,               "File contains invalid record !",
ERR_FILE_END,                 "Unexpected end of file !",
ERR_FILE_IO,                  "Error during file I/O !",
ERR_FILE_DETECT,              "Selected file is of unrecognizable format !",
ERR_FILE_OPEN,                "Unable to open file !",
ERR_ARGUMENT,                 "Function argument(s) out of range !",
NOTE_BOOT_DOWNLOADED,         "Note: Boot downloaded",
ERR_BUSY_USED_POLLING,        "WARNING: Temporary function blocked, due to used main polling",
ERR_IMAGE_CORRUPTED,          "Image Memory corrupted or erased !  Load Image.",
ERR_TARGET_NOACCESS,          "Target not accessible !",
ERR_VERIFY_FAILED,            "Verification failed !",
ERR_NO_PARMS,                 "Main Process Parameters not yet set !  Load Image.",
ERR_IMAGE_ERASE,              "Could not erase Image Buffer !",
ERR_IMAGE_LOAD,               "Could not load Image Buffer !",
ERR_PARMS_LOAD,               "Could not load Main Process Parameters !",
ERR_SEL_BAUDRATE,             "Could not select Baud Rate !",
ERR_SET_VCC,                  "WARNING: Could not set target voltage -
 Short circuitry or settling time too small?",
ERR_WRONG_CMD,                "Invalid firmware command !",
ERR_POWER_SUPPLY,             "Power supply voltage too low !",
ERR_EXT_VCC_IN,               "WARNING: Sense voltage out of range -
 Check pin C2000_VCC_IN of target connector !",
ERR_WRONG_DEVICE,             "Wrong target device connected ! ",
ERR_NO_DEVICE,                "No target device connected",
ERR_IMAGE_OVERWRITTEN,        "File(s) contains already specified data (code overwritten)",
ERR_IMAGE_NO,                 "Selected Image number out of range",
ERR_CFG_FILE_OPEN_ERR,        "Could not open the configuration file.",
ERR_SCRIPT_HEADER_SIZE_ERR,   "Script Header size error",
ERR_IMAGE_ID,                 "Image ID error. Image ignored, program terminated.",
ERR_IMAGE_CONTENTS,           "Image contents (size, no of tasks) error. Program terminated.",
ERR_IMAGE_VERIFICATION,       "Image CRC error. Program terminated.",
ERR_CODE_OVERWRITTEN,         "WARNING: Code overwritten. Code from the file written to already
used location.",
ERR_CODE_FILE_CONTENTS,       "Code in the file contains invalid data.",
ERR_OPEN_FILE,                "Open File error",
ERR_FILE_NAME,                "Extension or file name error",
ERR_IMAGE_FILE_PASSWORD,      "Wrong password for opening the image file",
ERR_IMAGE_FILE_PCHW,          "Wrong PC hardware fingerprint # for opening the image file",
ERR_IMAGE_FILE_ID,            "Image file ID error or file corrupted",
ERR_IMAGE_FILE_CS,            "Check Sum of the Image file error or file corrupted",
ERR_IMAGE_FILE_HEADER,        "Wrong header in the image file or file corrupted",
ERR_IMAGE_FILE_NOT_C2000GANG,"Image file is not for the C2000-GANG programmer.",
ERR_IMAGE_FILE_CONTENTS,      "Image file contents error or file corrupted.",
ERR_IMAGE_FILE_MODE,          "Unknown protection mode of the image file or file corrupted.",
ERR_IMAGE_FILE_OFFSET,        "Data offset in the image file error or file corrupted.",
ERR_IMAGE_FILE_HEX,           "Hex data conversion in the image file error or file corrupted.",
ERR_IMAGE_FILE_CORRUPTED,     "Image file corrupted.",
ERR_IMAGE_FILE_UNLOCK,        "Image file cannot be unlocked",
ERR_CUSTOMIZED_MCU_LICENSE_FILE_OPEN,  "Customized MCUs license file open error",
ERR_BSL_CODE_WITHIN_BSL_DISABLED,   "WARNING: Code specified for the BSL space location, but
access to the BSL is locked.",
ERR_INFO_PAGE_OUT_OF_RANGE, "Info memory page number is out of range.",
ERR_COM_PORT_SCAN_SIZE,       "COM ports scan number is too low.",
ERR_SELFTEST_SIZE,            "Selftest data size too high.",
ERR_DATA_SIZE_TOO_HIGH,       "Data size too high.",
ERR_GANG_MASK_ZERO,           "Gang mask ZERO. Nothing to do.",
ERR_ADDRESS_DEFINITION,       "Address definition.",
ERR_DATA_SIZE_ZERO,           "Data size is below 2.",
ERR_DCO_NO_OUT_OF_RANGE,      "Invalid DCO number.",
```


Copyright © 2014–2016, Texas Instruments Incorporated

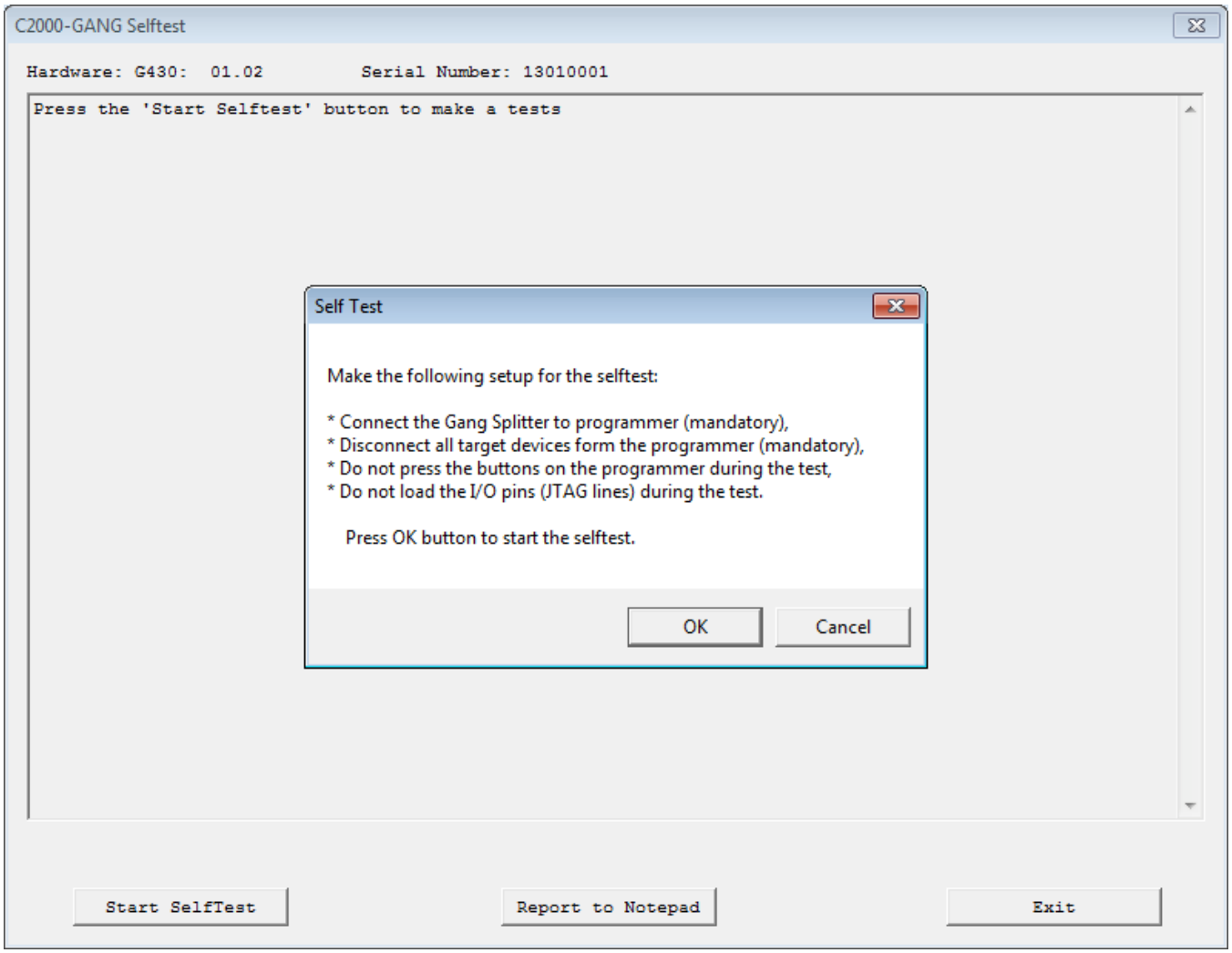
```
ERR_CODE_NOT_IMPLEMENTED,    "Command not implemented.",
ERR_TARGET_NO,               "Wrong Target number",
ERR_CODE_FILE_ERR,           "Code File error.",
ERR_PASSWORD_FILE_ERR,       "Password File error.",
ERR_NOTHING_TO_PROGRAM,      "Nothing to program/verify - empty code in selected memory space.",
ERR_CODE_OUT_OF_RANGE,       "Code out of range of the selected MCU.",
ERR_INVALID_NAME_INDEX,      "Invalid Name Index.",
ERR_IGNORED_PART_OF_THE_CODE,  "Ignored part of the code.",
ERR_SCRIPT_SIZE_TOO_HIGH,    "Script size too high !",
ERR_BOOT_FW_NOT_VALID,       "Boot Firmware not valid",
ERR_DSP_FW_VERIFICATION_ERR,"MCU Firmware verification error.",
ERR_DSP_FW_ERROR,            "MCU Firmware definition error.",
ERR_CSM_INDEX_TOO_HIGH,      "CSM Index Error",
ERR_RETAIN_SIZE_ERROR,       "Retain size error",
ERR_RETAIN_ADDR_ERROR,       "Retain address error",
ERR_RETAIN_DATA_AND_CODE_OVERWRITTEN,  "Retain data and code overwritten",
ERR_INVALID_NUMBER,          "Invalid error number !",
```

2.4 Self Test

The C2000 Gang Programmer Self Test program allows to test most of the hardware for correctness. Connect the programmer to a computer running C2000 Gang Programmer software. The Gang Splitter must be connected to the C2000 Gang Programmer. Disconnect all target devices, because any connected devices can modify the test results and make them invalid.

Activate the Self Test by choosing the Tools→Self Test option from the dropdown menu. Press the Start Self Test button, as shown in Figure 2-18, to begin. If the Self Test reports any problems, then it is advisable to send the test report to TI technical support for assistance.

Copyright © 2014–2016, Texas Instruments Incorporated



NOTE: Use the C2000 Gang Programmer selftest capability to check the integrity of the hardware. Before beginning the test, make sure that no target MCUs are connected to the C2000 Gang Programmer.

Figure 2-18. Self Test

The following is a typical self test report:

```
=== C2000-GANG Self test results ( Saturday, January 18, 2014,  19:08:19 ) ===

Adapter SN ------:  13010001
Hardware --------:  G430:  01.02
Access key ------:  C2000 - Gang Programmer
Silicon Number --:  F701 8846 3100 1300
API Firmware ----:  C2000-G   AC28:  01.00.00.03
BOOT Firmware ---:  G430BOOT   B430:  01.00.02.00
GUI Software ----:  C2000-GANG-GUI   G28x:  01.00.00.02
DLL Software ----:  C2000-GANG-DLL   D280:  01.00.00.02

================ Test results =============


No.       name                  parameter       limits              result         status


  1: Data Bus   (ALL LOW)          0x0000   (0x0000  - 0x0000)  Result:  0x00     ... >>  OK   <<
```


Copyright © 2014–2016, Texas Instruments Incorporated

```
 2: Data Bus    (ALL HIGH)          0x00FF    (0xFFFF  - 0xFFFF)  Result:  0xFFFF    ... >>  OK   <<
 3: Data Bus    (D-0 HI )           0x0001    (0x0101  - 0x0101)  Result:  0x0101    ... >>  OK   <<
 4: Data Bus    (D-1 HI )           0x0002    (0x0202  - 0x0202)  Result:  0x0202    ... >>  OK   <<
 5: Data Bus    (D-2 HI )           0x0004    (0x0404  - 0x0404)  Result:  0x0404    ... >>  OK   <<
 6: Data Bus    (D-3 HI )           0x0008    (0x0808  - 0x0808)  Result:  0x0808    ... >>  OK   <<
 7: Data Bus    (D-4 HI )           0x0010    (0x1010  - 0x1010)  Result:  0x1010    ... >>  OK   <<
 8: Data Bus    (D-5 HI )           0x0020    (0x2020  - 0x2020)  Result:  0x2020    ... >>  OK   <<
 9: Data Bus    (D-6 HI )           0x0040    (0x4040  - 0x4040)  Result:  0x4040    ... >>  OK   <<
10: Data Bus    (D-7 HI )           0x0080    (0x8080  - 0x8080)  Result:  0x8080    ... >>  OK   <<

11: Data-2 Bus  (ALL LOW)           0x0000    (0x0000  - 0x0000)  Result:  0x00      ... >>  OK   <<
12: Data-2 Bus  (ALL HIGH)          0x00FF    (0xFFFF  - 0xFFFF)  Result:  0xFFFF    ... >>  OK   <<
13: Data-2 Bus  (D2-0 HI )          0x0001    (0x0101  - 0x0101)  Result:  0x0101    ... >>  OK   <<
14: Data-2 Bus  (D2-1 HI )          0x0002    (0x0202  - 0x0202)  Result:  0x0202    ... >>  OK   <<
15: Data-2 Bus  (D2-2 HI )          0x0004    (0x0404  - 0x0404)  Result:  0x0404    ... >>  OK   <<
16: Data-2 Bus  (D2-3 HI )          0x0008    (0x0808  - 0x0808)  Result:  0x0808    ... >>  OK   <<
17: Data-2 Bus  (D2-4 HI )          0x0010    (0x1010  - 0x1010)  Result:  0x1010    ... >>  OK   <<
18: Data-2 Bus  (D2-5 HI )          0x0020    (0x2020  - 0x2020)  Result:  0x2020    ... >>  OK   <<
19: Data-2 Bus  (D2-6 HI )          0x0040    (0x4040  - 0x4040)  Result:  0x4040    ... >>  OK   <<
20: Data-2 Bus  (D2-7 HI )          0x0080    (0x8080  - 0x8080)  Result:  0x8080    ... >>  OK   <<

21: Vcc Target-1 (ALL OFF)           0.00 V  (  0.00 to   0.30)  Result:  0.17 V   ... >>  OK   <<
22: Vcc Target-2 (ALL OFF)           0.00 V  (  0.00 to   0.30)  Result:  0.15 V   ... >>  OK   <<
23: Vcc Target-3 (ALL OFF)           0.00 V  (  0.00 to   0.30)  Result:  0.16 V   ... >>  OK   <<
24: Vcc Target-4 (ALL OFF)           0.00 V  (  0.00 to   0.30)  Result:  0.16 V   ... >>  OK   <<
25: Vcc Target-5 (ALL OFF)           0.00 V  (  0.00 to   0.30)  Result:  0.16 V   ... >>  OK   <<
26: Vcc Target-6 (ALL OFF)           0.00 V  (  0.00 to   0.30)  Result:  0.16 V   ... >>  OK   <<
27: Vcc Target-7 (ALL OFF)           0.00 V  (  0.00 to   0.30)  Result:  0.16 V   ... >>  OK   <<
28: Vcc Target-8 (ALL OFF)           0.00 V  (  0.00 to   0.30)  Result:  0.16 V   ... >>  OK   <<

29: Translators VT (OFF)             0.00 V  (  0.00 to   0.50)  Result:  0.21 V   ... >>  OK   <<
30: Translators VT (ON 3.3V)         3.30 V  (  3.10 to   3.50)  Result:  3.30 V   ... >>  OK   <<

31: Vpp Voltage-in                  10.00 V  (  8.00 to  12.00)  Result: 10.59 V   ... >>  OK   <<
32: Vpp Voltage                      7.00 V  (  6.50 to   7.30)  Result:  6.90 V   ... >>  OK   <<
33: Internal Vcc-3.3V                3.30 V  (  3.20 to   3.40)  Result:  3.30 V   ... >>  OK   <<

34: Vcc Target-1 (ALL ON 3.3V)       3.30 V  (  3.10 to   3.50)  Result:  3.34 V   ... >>  OK   <<
35: Vcc Target-2 (ALL ON 3.3V)       3.30 V  (  3.10 to   3.50)  Result:  3.31 V   ... >>  OK   <<
36: Vcc Target-3 (ALL ON 3.3V)       3.30 V  (  3.10 to   3.50)  Result:  3.33 V   ... >>  OK   <<
37: Vcc Target-4 (ALL ON 3.3V)       3.30 V  (  3.10 to   3.50)  Result:  3.31 V   ... >>  OK   <<
38: Vcc Target-5 (ALL ON 3.3V)       3.30 V  (  3.10 to   3.50)  Result:  3.34 V   ... >>  OK   <<
39: Vcc Target-6 (ALL ON 3.3V)       3.30 V  (  3.10 to   3.50)  Result:  3.34 V   ... >>  OK   <<
40: Vcc Target-7 (ALL ON 3.3V)       3.30 V  (  3.10 to   3.50)  Result:  3.34 V   ... >>  OK   <<
41: Vcc Target-8 (ALL ON 3.3V)       3.30 V  (  3.10 to   3.50)  Result:  3.31 V   ... >>  OK   <<

42: Vcc discharge (50ms)Target-1     3.30 V  (  0.30 to   1.50)  Result:  1.01 V   ... >>  OK   <<
43: Vcc discharge (50ms)Target-2     3.30 V  (  0.30 to   1.50)  Result:  0.54 V   ... >>  OK   <<
44: Vcc discharge (50ms)Target-3     3.30 V  (  0.30 to   1.50)  Result:  0.56 V   ... >>  OK   <<
45: Vcc discharge (50ms)Target-4     3.30 V  (  0.30 to   1.50)  Result:  0.57 V   ... >>  OK   <<
46: Vcc discharge (50ms)Target-5     3.30 V  (  0.30 to   1.50)  Result:  0.56 V   ... >>  OK   <<
47: Vcc discharge (50ms)Target-6     3.30 V  (  0.30 to   1.50)  Result:  0.55 V   ... >>  OK   <<
48: Vcc discharge (50ms)Target-7     3.30 V  (  0.30 to   1.50)  Result:  0.55 V   ... >>  OK   <<
49: Vcc discharge (50ms)Target-8     3.30 V  (  0.30 to   1.50)  Result:  0.54 V   ... >>  OK   <<

50: Vcc Target-1 ( #1 ON )           0.00 V  (  3.10 to   3.50)  Result:  3.34 V   ... >>  OK   <<
51: Vcc Target-2 ( #1 ON )           0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
52: Vcc Target-3 ( #1 ON )           0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
53: Vcc Target-4 ( #1 ON )           0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
54: Vcc Target-5 ( #1 ON )           0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
55: Vcc Target-6 ( #1 ON )           0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
56: Vcc Target-7 ( #1 ON )           0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
57: Vcc Target-8 ( #1 ON )           3.30 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
58: Vcc Target-1 ( #2 ON )           0.00 V  (  0.00 to   0.50)  Result:  0.17 V   ... >>  OK   <<
59: Vcc Target-2 ( #2 ON )           0.00 V  (  3.10 to   3.50)  Result:  3.31 V   ... >>  OK   <<
60: Vcc Target-3 ( #2 ON )           0.00 V  (  0.00 to   0.50)  Result:  0.17 V   ... >>  OK   <<
```


Copyright © 2014–2016, Texas Instruments Incorporated

```
 61: Vcc Target-4 ( #2 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 62: Vcc Target-5 ( #2 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 63: Vcc Target-6 ( #2 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 64: Vcc Target-7 ( #2 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 65: Vcc Target-8 ( #2 ON )          3.30 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 66: Vcc Target-1 ( #3 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.17 V   ... >>  OK   <<
 67: Vcc Target-2 ( #3 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.15 V   ... >>  OK   <<
 68: Vcc Target-3 ( #3 ON )          0.00 V  (  3.10 to   3.50)  Result:  3.32 V   ... >>  OK   <<
 69: Vcc Target-4 ( #3 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 70: Vcc Target-5 ( #3 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 71: Vcc Target-6 ( #3 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 72: Vcc Target-7 ( #3 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 73: Vcc Target-8 ( #3 ON )          3.30 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 74: Vcc Target-1 ( #4 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.17 V   ... >>  OK   <<
 75: Vcc Target-2 ( #4 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.15 V   ... >>  OK   <<
 76: Vcc Target-3 ( #4 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 77: Vcc Target-4 ( #4 ON )          0.00 V  (  3.10 to   3.50)  Result:  3.31 V   ... >>  OK   <<
 78: Vcc Target-5 ( #4 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.17 V   ... >>  OK   <<
 79: Vcc Target-6 ( #4 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 80: Vcc Target-7 ( #4 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 81: Vcc Target-8 ( #4 ON )          3.30 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 82: Vcc Target-1 ( #5 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.17 V   ... >>  OK   <<
 83: Vcc Target-2 ( #5 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.15 V   ... >>  OK   <<
 84: Vcc Target-3 ( #5 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 85: Vcc Target-4 ( #5 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 86: Vcc Target-5 ( #5 ON )          0.00 V  (  3.10 to   3.50)  Result:  3.33 V   ... >>  OK   <<
 87: Vcc Target-6 ( #5 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.17 V   ... >>  OK   <<
 88: Vcc Target-7 ( #5 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 89: Vcc Target-8 ( #5 ON )          3.30 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 90: Vcc Target-1 ( #6 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 91: Vcc Target-2 ( #6 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.15 V   ... >>  OK   <<
 92: Vcc Target-3 ( #6 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 93: Vcc Target-4 ( #6 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 94: Vcc Target-5 ( #6 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 95: Vcc Target-6 ( #6 ON )          0.00 V  (  3.10 to   3.50)  Result:  3.34 V   ... >>  OK   <<
 96: Vcc Target-7 ( #6 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 97: Vcc Target-8 ( #6 ON )          3.30 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
 98: Vcc Target-1 ( #7 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.17 V   ... >>  OK   <<
 99: Vcc Target-2 ( #7 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.15 V   ... >>  OK   <<
100: Vcc Target-3 ( #7 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
101: Vcc Target-4 ( #7 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
102: Vcc Target-5 ( #7 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
103: Vcc Target-6 ( #7 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
104: Vcc Target-7 ( #7 ON )          0.00 V  (  3.10 to   3.50)  Result:  3.34 V   ... >>  OK   <<
105: Vcc Target-8 ( #7 ON )          3.30 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
106: Vcc Target-1 ( #8 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.17 V   ... >>  OK   <<
107: Vcc Target-2 ( #8 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.15 V   ... >>  OK   <<
108: Vcc Target-3 ( #8 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
109: Vcc Target-4 ( #8 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
110: Vcc Target-5 ( #8 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
111: Vcc Target-6 ( #8 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
112: Vcc Target-7 ( #8 ON )          0.00 V  (  0.00 to   0.50)  Result:  0.16 V   ... >>  OK   <<
113: Vcc Target-8 ( #8 ON )          3.30 V  (  3.10 to   3.50)  Result:  3.31 V   ... >>  OK   <<

114: SD Power OFF                  0x0000   (0x0000  - 0x0000)  Result:  0x00     ... >>  OK   <<
115: SD Power ON                   0x0001   (0x0001  - 0x0001)  Result:  0x01     ... >>  OK   <<
116: SD Discharge -  delay 2ms     0x0010   (0x0001  - 0x0001)  Result:  0x01     ... >>  OK   <<
117: SD Discharge -  delay 50ms    0x0010   (0x0000  - 0x0000)  Result:  0x00     ... >>  OK   <<

118: BSL RX bus (#1 HIGH)          0x0001   (0x0001  - 0x0001)  Result:  0x01     ... >>  OK   <<
119: BSL RX bus (#2 HIGH)          0x0002   (0x0002  - 0x0002)  Result:  0x02     ... >>  OK   <<
120: BSL RX bus (#3 HIGH)          0x0004   (0x0004  - 0x0004)  Result:  0x04     ... >>  OK   <<
121: BSL RX bus (#4 HIGH)          0x0008   (0x0008  - 0x0008)  Result:  0x08     ... >>  OK   <<
122: BSL RX bus (#5 HIGH)          0x0010   (0x0010  - 0x0010)  Result:  0x10     ... >>  OK   <<
123: BSL RX bus (#6 HIGH)          0x0020   (0x0020  - 0x0020)  Result:  0x20     ... >>  OK   <<
124: BSL RX bus (#7 HIGH)          0x0040   (0x0040  - 0x0040)  Result:  0x40     ... >>  OK   <<
```


Copyright © 2014–2016, Texas Instruments Incorporated

```
125: BSL RX bus (#8 HIGH)          0x0080   (0x0080  - 0x0080)  Result:  0x80     ... >>  OK   <<

126: BSL TX bus (#1 HIGH)          0x0001   (0x0001  - 0x0001)  Result:  0x01     ... >>  OK   <<
127: BSL TX bus (#2 HIGH)          0x0002   (0x0002  - 0x0002)  Result:  0x02     ... >>  OK   <<
128: BSL TX bus (#3 HIGH)          0x0004   (0x0004  - 0x0004)  Result:  0x04     ... >>  OK   <<
129: BSL TX bus (#4 HIGH)          0x0008   (0x0008  - 0x0008)  Result:  0x08     ... >>  OK   <<
130: BSL TX bus (#5 HIGH)          0x0010   (0x0010  - 0x0010)  Result:  0x10     ... >>  OK   <<
131: BSL TX bus (#6 HIGH)          0x0020   (0x0020  - 0x0020)  Result:  0x20     ... >>  OK   <<
132: BSL TX bus (#7 HIGH)          0x0040   (0x0040  - 0x0040)  Result:  0x40     ... >>  OK   <<
133: BSL TX bus (#8 HIGH)          0x0080   (0x0080  - 0x0080)  Result:  0x80     ... >>  OK   <<

134: TDI bus  (#1 HIGH)            0x0001   (0x0001  - 0x0001)  Result:  0x01     ... >>  OK   <<
135: TDI bus  (#2 HIGH)            0x0002   (0x0002  - 0x0002)  Result:  0x02     ... >>  OK   <<
136: TDI bus  (#3 HIGH)            0x0004   (0x0004  - 0x0004)  Result:  0x04     ... >>  OK   <<
137: TDI bus  (#4 HIGH)            0x0008   (0x0008  - 0x0008)  Result:  0x08     ... >>  OK   <<
138: TDI bus  (#5 HIGH)            0x0010   (0x0010  - 0x0010)  Result:  0x10     ... >>  OK   <<
139: TDI bus  (#6 HIGH)            0x0020   (0x0020  - 0x0020)  Result:  0x20     ... >>  OK   <<
140: TDI bus  (#7 HIGH)            0x0040   (0x0040  - 0x0040)  Result:  0x40     ... >>  OK   <<
141: TDI bus  (#8 HIGH)            0x0080   (0x0080  - 0x0080)  Result:  0x80     ... >>  OK   <<

142: TDIO Tx-bus (#1 HIGH)         0x0001   (0x0001  - 0x0001)  Result:  0x01     ... >>  OK   <<
143: TDIO Tx-bus (#2 HIGH)         0x0002   (0x0002  - 0x0002)  Result:  0x02     ... >>  OK   <<
144: TDIO Tx-bus (#3 HIGH)         0x0004   (0x0004  - 0x0004)  Result:  0x04     ... >>  OK   <<
145: TDIO Tx-bus (#4 HIGH)         0x0008   (0x0008  - 0x0008)  Result:  0x08     ... >>  OK   <<
146: TDIO Tx-bus (#5 HIGH)         0x0010   (0x0010  - 0x0010)  Result:  0x10     ... >>  OK   <<
147: TDIO Tx-bus (#6 HIGH)         0x0020   (0x0020  - 0x0020)  Result:  0x20     ... >>  OK   <<
148: TDIO Tx-bus (#7 HIGH)         0x0040   (0x0040  - 0x0040)  Result:  0x40     ... >>  OK   <<
149: TDIO Tx-bus (#8 HIGH)         0x0080   (0x0080  - 0x0080)  Result:  0x80     ... >>  OK   <<

150: TDIO Tx-Rx (#1 HIGH)          0x0001   (0x0001  - 0x0001)  Result:  0x01     ... >>  OK   <<
151: TDIO Tx-Rx (#2 HIGH)          0x0002   (0x0002  - 0x0002)  Result:  0x02     ... >>  OK   <<
152: TDIO Tx-Rx (#3 HIGH)          0x0004   (0x0004  - 0x0004)  Result:  0x04     ... >>  OK   <<
153: TDIO Tx-Rx (#4 HIGH)          0x0008   (0x0008  - 0x0008)  Result:  0x08     ... >>  OK   <<
154: TDIO Tx-Rx (#5 HIGH)          0x0010   (0x0010  - 0x0010)  Result:  0x10     ... >>  OK   <<
155: TDIO Tx-Rx (#6 HIGH)          0x0020   (0x0020  - 0x0020)  Result:  0x20     ... >>  OK   <<
156: TDIO Tx-Rx (#7 HIGH)          0x0040   (0x0040  - 0x0040)  Result:  0x40     ... >>  OK   <<
157: TDIO Tx-Rx (#8 HIGH)          0x0080   (0x0080  - 0x0080)  Result:  0x80     ... >>  OK   <<

158: TDIO Rx-bus (#1 HIGH)         0x0001   (0x0001  - 0x0001)  Result:  0x01     ... >>  OK   <<
159: TDIO Rx-bus (#2 HIGH)         0x0002   (0x0002  - 0x0002)  Result:  0x02     ... >>  OK   <<
160: TDIO Rx-bus (#3 HIGH)         0x0004   (0x0004  - 0x0004)  Result:  0x04     ... >>  OK   <<
161: TDIO Rx-bus (#4 HIGH)         0x0008   (0x0008  - 0x0008)  Result:  0x08     ... >>  OK   <<
162: TDIO Rx-bus (#5 HIGH)         0x0010   (0x0010  - 0x0010)  Result:  0x10     ... >>  OK   <<
163: TDIO Rx-bus (#6 HIGH)         0x0020   (0x0020  - 0x0020)  Result:  0x20     ... >>  OK   <<
164: TDIO Rx-bus (#7 HIGH)         0x0040   (0x0040  - 0x0040)  Result:  0x40     ... >>  OK   <<
165: TDIO Rx-bus (#84 HIGH)        0x0080   (0x0080  - 0x0080)  Result:  0x80     ... >>  OK   <<

166: TMS bus (All HIGH)            0x00FF   (0x00FF  - 0x00FF)  Result:  0xFF     ... >>  OK   <<
167: TMS bus (All LOW)             0x0000   (0x0000  - 0x0000)  Result:  0x00     ... >>  OK   <<

168: TMS bus (#1 HIGH)             0x0001   (0x0001  - 0x0001)  Result:  0x01     ... >>  OK   <<
169: TMS bus (#2 HIGH)             0x0002   (0x0002  - 0x0002)  Result:  0x02     ... >>  OK   <<
170: TMS bus (#3 HIGH)             0x0004   (0x0004  - 0x0004)  Result:  0x04     ... >>  OK   <<
171: TMS bus (#4 HIGH)             0x0008   (0x0008  - 0x0008)  Result:  0x08     ... >>  OK   <<
172: TMS bus (#5 HIGH)             0x0010   (0x0010  - 0x0010)  Result:  0x10     ... >>  OK   <<
173: TMS bus (#6 HIGH)             0x0020   (0x0020  - 0x0020)  Result:  0x20     ... >>  OK   <<
174: TMS bus (#7 HIGH)             0x0040   (0x0040  - 0x0040)  Result:  0x40     ... >>  OK   <<
175: TMS bus (#8 HIGH)             0x0080   (0x0080  - 0x0080)  Result:  0x80     ... >>  OK   <<

176: RST bus (All HIGH)            0x00FF   (0x00FF  - 0x00FF)  Result:  0xFF     ... >>  OK   <<
177: RST bus (All LOW)             0x0000   (0x0000  - 0x0000)  Result:  0x00     ... >>  OK   <<

178: RST bus (#1 HIGH)             0x0001   (0x0001  - 0x0001)  Result:  0x01     ... >>  OK   <<
179: RST bus (#2 HIGH)             0x0002   (0x0002  - 0x0002)  Result:  0x02     ... >>  OK   <<
180: RST bus (#3 HIGH)             0x0004   (0x0004  - 0x0004)  Result:  0x04     ... >>  OK   <<
181: RST bus (#4 HIGH)             0x0008   (0x0008  - 0x0008)  Result:  0x08     ... >>  OK   <<
```


Copyright © 2014–2016, Texas Instruments Incorporated

```
182: RST bus (#5 HIGH)             0x0010   (0x0010  - 0x0010)  Result:  0x10     ... >>  OK   <<
183: RST bus (#6 HIGH)             0x0020   (0x0020  - 0x0020)  Result:  0x20     ... >>  OK   <<
184: RST bus (#7 HIGH)             0x0040   (0x0040  - 0x0040)  Result:  0x40     ... >>  OK   <<
185: RST bus (#8 HIGH)             0x0080   (0x0080  - 0x0080)  Result:  0x80     ... >>  OK   <<

186: Keys buffer (All pull-up)     0x001F   (0x001F  - 0x001F)  Result:  0x1F     ... >>  OK   <<

187: Access to LCD RAM (0xAA)      0x00AA   (0x00AA  - 0x00AA)  Result:  0xAA     ... >>  OK   <<
188: Access to LCD RAM (0x99)      0x0099   (0x0099  - 0x0099)  Result:  0x99     ... >>  OK   <<

189: Image Flash Access (get ID)   0x0002   (0x0001  - 0x0002)  Result:  0x02     ... >>  OK   <<

190: TDI Fuse keys (#1 ON)         0x0001   (0x0001  - 0x0001)  Result:  0x01     ... >>  OK   <<
191: TDI Fuse keys (#2 ON)         0x0002   (0x0002  - 0x0002)  Result:  0x02     ... >>  OK   <<
192: TDI Fuse keys (#3 ON)         0x0004   (0x0004  - 0x0004)  Result:  0x04     ... >>  OK   <<
193: TDI Fuse keys (#4 ON)         0x0008   (0x0008  - 0x0008)  Result:  0x08     ... >>  OK   <<
194: TDI Fuse keys (#5 ON)         0x0010   (0x0010  - 0x0010)  Result:  0x10     ... >>  OK   <<
195: TDI Fuse keys (#6 ON)         0x0020   (0x0020  - 0x0020)  Result:  0x20     ... >>  OK   <<
196: TDI Fuse keys (#7 ON)         0x0040   (0x0040  - 0x0040)  Result:  0x40     ... >>  OK   <<
197: TDI Fuse keys (#8 ON)         0x0080   (0x0080  - 0x0080)  Result:  0x80     ... >>  OK   <<

198: TEST Fuse keys (#1 ON)         1.00    (  2.80 to   3.50)  Result:  3.18     ... >>  OK   <<
199: TEST Fuse keys (#2 ON)         2.00    (  2.80 to   3.50)  Result:  3.16     ... >>  OK   <<
200: TEST Fuse keys (#3 ON)         3.00    (  2.80 to   3.50)  Result:  3.16     ... >>  OK   <<
201: TEST Fuse keys (#4 ON)         4.00    (  2.80 to   3.50)  Result:  3.15     ... >>  OK   <<
202: TEST Fuse keys (#5 ON)         5.00    (  2.80 to   3.50)  Result:  3.17     ... >>  OK   <<
203: TEST Fuse keys (#6 ON)         6.00    (  2.80 to   3.50)  Result:  3.18     ... >>  OK   <<
204: TEST Fuse keys (#7 ON)         7.00    (  2.80 to   3.50)  Result:  3.18     ... >>  OK   <<
205: TEST Fuse keys (#8 ON)         8.00    (  2.80 to   3.50)  Result:  3.15     ... >>  OK   <<

============== Finished ================================
*  Test pass - no errors.
========================================================
```


Copyright © 2014–2016, Texas Instruments Incorporated

2.5 Label

Information and C2000 Gang Programmer software and hardware can be displayed by accessing the About dropdown menu. Select the About→About option to display information similar to that shown in Figure 2-19.

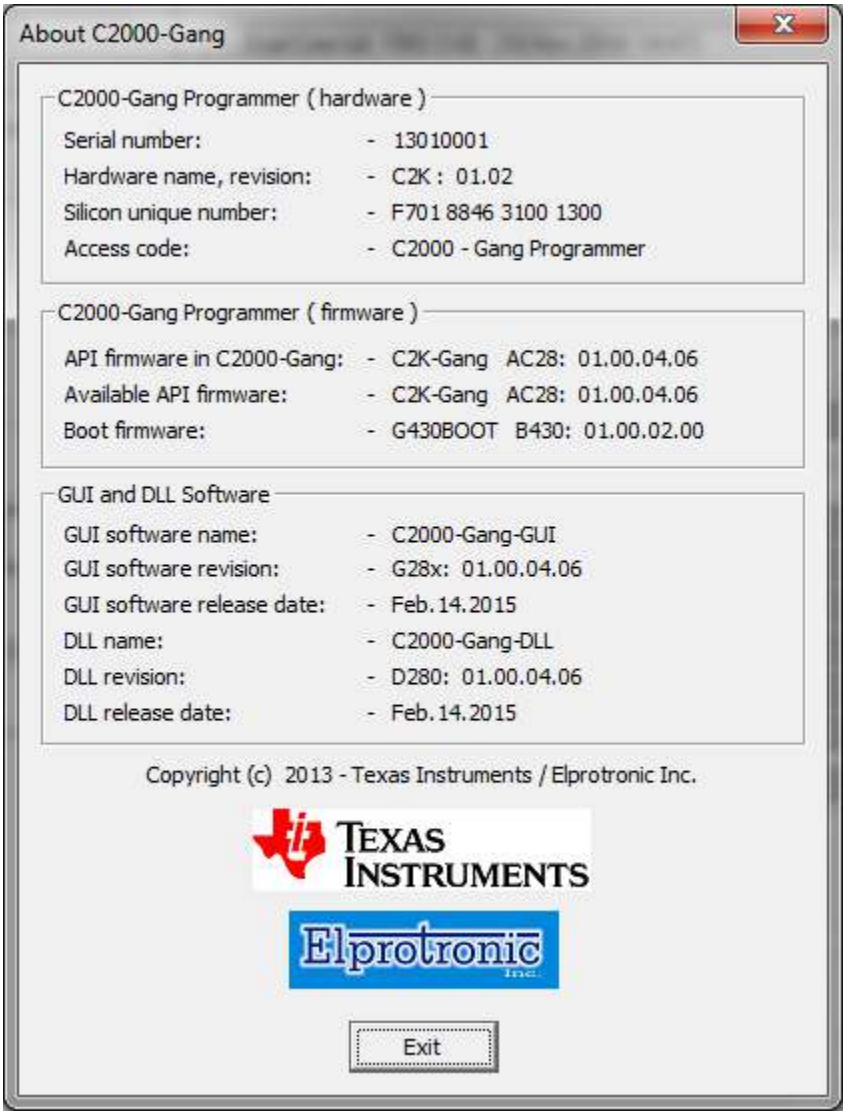


Figure 2-19. Information About the C2000 Gang Programmer

Copyright © 2014–2016, Texas Instruments Incorporated

2.6 Benchmarks

This section shows the results of timing benchmarks used on the C2000 Gang Programmer to measure the programming speed. Table 2-1 shows the result of the benchmark when programming with the JTAG interface. Identical programming speed is seen whether one device is programmed or eight devices are programmed simultaneously, because programming each MCU is done in parallel.

2.6.1 Benchmark for C28035

Table 2-1. Benchmark Results – C28035, 64 kwords (128kB) Code[1]

Mode (via USB)	Interface	Erase, Blank Check, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)	Verify Speed (kB/s)
Interactive Fast	JTAG Fast	20.0	0.3	11.3	430
From Image Memory Fast	JTAG Fast	18.5	0.2	13.0	500

(1) Programming speed and verify speed without startup procedures (initialization takes 1.2 seconds).

Copyright © 2014–2016, Texas Instruments Incorporated

Firmware

3.1 Commands

The C2000 Gang Programmer can be controlled by firmware commands received through USB or its RS-232 serial port. The following firmware commands are supported:

== Commands supported by the BOOT loader ======

- "Hello"
- Boot Commands Disable
- Boot Commands Enable
- Transmit Diagnostics
- Select Baud Rate
- Erase Firmware
- Load Firmware
- Exit Firmware update
- Get Label
- Get Progress Status

== Commands supported by API firmware ======

- Main process
- Interactive process
- Erase Image
- Read Info memory from C2000-GANG
- Write Info memory to C2000-GANG
- Verify Access Key
- Load Image Block
- Verify Image Checksum
- Read Image Header
- Boot update
- Read from Gang Data buffer
- Write to Gang Data buffer
- Disable API Interrupts
- Select Image
- Display Message on the LCD display
- Set temporary configuration
- Get selected status
- Selftest

Copyright © 2014–2016, Texas Instruments Incorporated

3.2 Firmware Interface Protocol

The C2000 Gang Programmer supports a UART communication protocol at baud rates from 9.6 to 115.2 kbaud in half duplex mode. The default baud rate at startup is 9.6 kbaud. This allows for communication between the C2000 Gang Programmer and devices that have a lower communication speed than the maximum 115.2 kbaud. It is recommended that after startup, the communication speed be increased to the common maximum for both devices to enable faster communication. If the control device has a USB interface with a virtual COM port, then it is recommended to use USB for communication between the control device and the C2000 Gang Programmer, because USB is several times faster than RS-232. Communication requires one start bit, eight data bits, even parity bit, and one stop bit. A software handshake is performed by a (not) acknowledge character.

3.3 Synchronization Sequence

To synchronize with the C2000 Gang Programmer the host serial handler transmits a SYNC (CR) character (0x0D) to the C2000 Gang Programmer. The C2000 Gang Programmer acknowledges successful reception of the SYNC character by responding with a DATA ACK character (0x90). If the SYNC is not received correctly, no data is sent back. This sequence is required to establish the communication channel and to react immediately to line faults. The synchronization character is not part of the data frame described in Section 3.4.1. When communication is established, the synchronization character is not required any more, but it can be sent at any time for checking the "alive" status, if required.

The synchronization character is not part of the data frame described in Section 3.4.1.

3.4 Command Messages

The C2000 Gang Programmer has a few type of messages with mandatory responses for each received command.

- Short TX messages with one byte only

  "Hello"

```
Tx ->  0x0d (CR)
Rx ->  0x90 (ACK)
```

  Get Progress Status

```
Tx ->  0xA5
Rx ->  0x80 0x00 <...data...> (without Check Sum)
```

- Standard TX messages with data frame

```
Tx ->  0x3E, CMD, <...data...>, < Check sum >
Rx ->  0x90 (ACK)
   or  0xA0 (NACK)
   or  0xB0 (In Progress) (use Get Progress Status for monitoring)
   or  0x80, 0x00, <...data....>, < Check sum >
```

3.4.1 Frame Structure

The data frame format follows the TI C2000 serial standard protocol (SSP) rules, extended with a preceding synchronization sequence (SS), as described in Section 3.3. The C2000 Gang Programmer is considered the receiver in Table 3-1, which details the data frame for firmware commands. The redundancy of some parameters results from the adaptation of the SSP or to save boot ROM space.

The data frame format of the firmware commands is shown in Table 3-1.

- The first eight bytes (HDR through LH) are mandatory ("–" represents dummy data).
- Data bytes D1 to Dn are optional.
- Two bytes (CKL and CKH) for checksum are mandatory
- Response is mandatory by the C2000 Gang Programmer. Response can be an "Acknowledge" or a full data frame depending on the command.

Copyright © 2014–2016, Texas Instruments Incorporated

The following abbreviations are used in Table 3-1.

CMD	Command identification
R	Do not use this command. Used for internal communication.
T	Target number (1 to 8)
L1, L2	Number of bytes in AL through Dn. The valid values of these bytes are restricted as follows: L1 = L2, L1 < 255, L1 even.
A1, A2, A3	Block start address or erase (check) address or jump address LO or HI byte. The bytes are combined to generate a 24-bit word as follows: Address = A3 × 0x10000 + A2 × 0x100 + A1
LL, LH	Number of pure data bytes (maximum 250) or erase information LO or HI byte or block length of erase check (max is 0xFF).
D1...Dn	Data bytes
CKL, CKH	16-bit checksum LO or HI byte
xx	Can be any data
–	No character (data byte) received or transmitted
ACK	The acknowledge character returned by the C2000 Gang Programmer can be either DATA_ACK = 0x90 (frame was received correctly, command was executed successfully) or DATA_NAK = 0xA0 (frame not valid (for example, wrong checksum, L, L2), command is not defined, is not allowed.
PRS	DATA_IN_PROGRESS = 0xB0 - Tasks in progress. Use Get Progress Status (0xA5) command to get the status and check when task is finished.

Table 3-1. Data Frame for Firmware Commands(1)(2)(3)

C2000-GANG Firmware Command	Prompt	CMD	L1	L2	A1	A2	A3	A4	LL	LH	D1	D2...Dn	CLK	CLH	ACK
"Hello"	0D	-	-	-	-	-	-	-	-		-	-	-	-	**ACK**
Boot Commands Disable	3E	2A	R	R	R	R	R	R	R	R	R	R	CKL	CKH	**ACK**
Boot Commands Enable	3E	2B	R	R	R	R	R	R	R	R	R	R	CKL	CKH	**ACK**
Diagnostic	3E	32	04	04	00	00	-	-	00	00	-	-	CKL	CKH	-
Response-Diagnostic	**80**	**0**	**1E**	**1E**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D08...D1E**	**CKL**	**CKH**	-
Set Baud Rate	3E	38	06	06	D1	00	-	-	00	00	00	00	CKL	CKH	**ACK**
Erase Firmware	3E	39	R	R	R	R	R	R	R	R	R	R	CKL	CKH	**ACK**
Load Firmware	3E	3A	R	R	R	R	R	R	R	R	R	R	CKL	CKH	**ACK**
Exit Firmware Update	3E	3B	R	R	R	R	R	R	R	R	R	R	CKL	CKH	**ACK**
Get Label	3E	40	04	04	00	00	-	-	00	00	-	-	CKL	CKH	-
Response-Get Label	**80**	**00**	**8C**	**8C**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8...D140**	**CKL**	**CKH**	-
Get Progress Status	A5	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Response----,,,----	**80**	**A5**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**	**D9**	**D10...D48**	-	-	-
Main Process	3E	31	04	04	00	00	-	-	00	00	-	-	CKL	CKH	**PRS**
Interactive Task	3E	46	n	n	D1	D2	-	-	D3	D4	D5	D6...Dn	CKL	CKH	-
Response----,,,----	**80**	**0**	**n**	**n**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8...Dn**	**CKL**	**CKH**	-
Erase Image	3E	33	04	04	00	00	-	-	00	00	-	-	CKL	CKH	**PRS**
Get Info C-D	3E	41	04	04	A1	00	-	-	00	00	-	-	CKL	CKH	-
Response-Get Info	**80**	**0**	**80**	**80**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8...D128**	**CKL**	**CKH**	-
Write Info C-D	3E	42	84	84	A1	00	-	-	80	0	D1	D2...D128	CKL	CKH	**ACK**
Verify Access Key	3E	44	04	04	00	00	-	-	00	00	-	-	CKL	CKH	**ACK**
Load Image	3E	43	n	n	A1	A2	A3	00	n-6	00	D1	D2...Dn-6	CKL	CKH	**ACK**
Verify Image CRC	3E	45	08	08	A1	A2	A3	A4	LL	LH	D1	D2	CKL	CKH	**ACK**
Get Image Header	3E	47	06	06	A1	A2	00	00	n	00	-	-	CKL	CKH	-
Response----,,,----	**80**	**0**	**n**	**n**	**D1**	**D2**	-	-	**D3**	**D4**	**D5**	**D6...Dn**	**CKL**	**CKH**	-
Read Gang Buffer	3E	49	4	4	T	0	-	-	n	0	-	-	CKL	CKH	-
Response----,,,----	**80**	**0**	**n**	**n**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8...Dn**	**CKL**	**CKH**	-
Write Gang Buffer	3E	4A	n+4	n+4	T	0	-	-	n	0	D1	D2...Dn	CKL	CKH	**ACK**
Disable API Interrupts	3E	4C	4	4	R	R	-	-	R	R	-	-	CKL	CKH	**ACK**

(1) All numbers are bytes in hexadecimal notation. ACK is sent by the C2000 Gang Programmer.
(2) PROMPT = 0x3E means data frame expected.
(3) Bold bytes represent responses from C2000 Gang Programmer.

Copyright © 2014–2016, Texas Instruments Incorporated

Table 3-1. Data Frame for Firmware Commands[1][2][3] (continued)

C2000-GANG Firmware Command	Prompt	CMD	L1	L2	A1	A2	A3	A4	LL	LH	D1	D2...Dn	CLK	CLH	ACK
Select Image	3E	50	4	4	A1	0	-	-	0	0	-	-	CKL	CKH	**ACK**
Display Message	3E	54	n+4	n+4	A1	A2	-	-	n	00	D1	D2...Dn	CKL	CKH	**ACK**
Set IO State	3E	4E	0C	0C	VL	VH	-	-	08	00	D1	D2...D8	CKL	CKH	**ACK**
Set Temporary Configuration	3E	56	06	06	A1	A2	-	-	2	0	D1	D2	CKL	CKH	**ACK**
Get Gang Status	3E	58	04	04	A1	0	-	-	0	0	-	-	CKL	CKH	-
Response----,,,----	**80**	**0**	**n**	**n**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8...Dn**	**CKL**	**CKH**	-
Remote Selftest	3E	71	n+6	n+6	A1	A2	A3	A4	n	0	D1	D2...Dn	CKL	CKH	-
Response----,,,----	**80**	**0**	**n**	**n**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8...Dn**	**CKL**	**CKH**	-

3.4.2 Checksum

The 16-bit (2-byte) checksum is calculated over all received or transmitted bytes, B1 to Bn, in the data frame except the checksum bytes themselves. The checksum is calculated by XORing words (two consecutive bytes) and bit-wise inverting (~) the result, as shown in the following formulas.

CHECKSUM = INV [(B1 + 256 × B2) XOR (B3 + 256 × B4) XOR...XOR ((Bn – 1) + 256 × Bn)]

or

CKL = INV [B1 XOR B3 XOR...XOR Bn–1]
CKH = INV [B2 XOR B4 XOR...XOR Bn]

An example of a frame for the Execute Self Test command with checksum would appear as:

```
0x3E 0x35 0x06 0x06 0x00 0x00 0x00 0x00 0x00 0x00 0xC7 0xCC
```

3.5 Detailed Description of Commands

3.5.1 General

After the prompt byte (0x3E) and the command identification byte CMD, the frame length bytes L1 and L2 (which must be equal) hold the number of bytes following L2, excluding the checksum bytes CKL and CKH. Bytes A1, A2, A3, A4, LL, LH, and D1 to Dn are command specific. However, the checksum bytes CKL (low byte) and CKH (high byte) are mandatory. If the data frame is received correctly and the command execution is successful, the acknowledge byte ACK (0x90), in progress byte (0xB0) or received message with header byte (0x80) as the first one. Incorrectly received data frames, unsuccessful operations, and commands that are not defined are confirmed with a DATA_NACK = 0xA0.

3.5.2 Commands Supported by the BOOT Loader

3.5.2.1 "Hello"

Short TX messages with one byte only

```
Tx ->  0x0d (CR)
Rx ->  0x90 (ACK)
```

A response is sent only when the <CR> (0x0D byte) has been detected and when it is not the byte used as the part of the data frame. This command can be useful for checking communication with the C2000 Gang Programmer. When there is no response, then the baud rate should be changed. After power-up, the USB interface is used for communication with the C2000 Gang Programmer; however, the RS-232 receiver is also active. To reestablish communication between USB and RS232, the "Hello" command must be sent a minimum of three times via RS232. After this, an ACK (0x90) is transmitted via RS232. This sequence also works in reverse, to reestablish communication between RS232 and USB.

Copyright © 2014–2016, Texas Instruments Incorporated

3.5.2.2 Boot Commands Disable

```
Tx ->  3E   2A ... ... ... CKL CKH
Rx ->  0x90 (ACK)
```

Do not use this command. This command is used during firmware or information memory update. Use the C2000 Gang Programmer executable GUI software for updating firmware or information memory update if required.

3.5.2.3 Boot Commands Enable

```
Tx ->  3E   2B ... ... ... CKL CKH
Rx ->  0x90 (ACK)
```

Do not use this command. This command is used during firmware or information memory update. Use the C2000 Gang Programmer executable GUI software for updating firmware or information memory update if required.

3.5.2.4 Diagnostic

The Diagnostic command retrieves the result of the preceding gang programming command.

```
Tx ->  3E 32 04 04 00 00 00 00 CKL CKH
Rx ->  80 00 1E 1E D1 D2 ... D30 CKL CKH
```

Data bytes D1 to D30 hold the parameters, as follows:

D1-D6: Reserved

D7-D8: Boot revision number: D7 (MSByte), D8 (LSByte)

D9-D10: Hardware version number: D9 (MSByte), D10 (LSByte).

D11 to D12: Firmware version number: D11 (MSByte), D12 (LSByte).

D13 to D20: Character string representing the boot name "C2000BOOT"

D21: Comma (,)

D22 to D30: Zero-terminated application firmware name "C2000-GANG"

When the application is modified or is not present, then bits D11-D12 and D22-D30 are modified and can be used for detection if the application firmware is present, and if present, what type and version of the application firmware is downloaded.

3.5.2.5 Set Baud Rate

```
Tx ->  3E 38 06 06 BR 00 00 00 00 00 CKL CKH
Rx ->  0x90 (ACK)
```

The Set Baud Rate command sets the rate of the serial communications. The default is 9600 baud. Baud rate index 0 to 4, representing the baud rate.

BR → 0 = 9600 baud (default)

BR → 1 = 19200 baud

BR → 2 = 38400 baud

BR → 3 = 57600 baud

BR → 4 = 115200 baud

The Set Baud Rate command takes effect (that is, changes the baud rate) immediately.

3.5.2.6 Erase Firmware

```
Tx ->  3E 39 ... ... ... CKL CKH
Rx ->  0x90 (ACK)
```

Do not use this command. This command is used during firmware or information memory update. Use the C2000 Gang Programmer executable GUI software for updating firmware or information memory update if required.

Copyright © 2014–2016, Texas Instruments Incorporated

3.5.2.7 Load Firmware

```
Tx ->  3E 3A ... ... ... CKL CKH
Rx ->  0x90 (ACK)
```

Do not use this command. This command is used during firmware or information memory update. Use the C2000 Gang Programmer executable GUI software for updating firmware or information memory update if required.

3.5.2.8 Exit Firmware Update Command

```
Tx ->  3E 3B ... ... ... CKL CKH
Rx ->  0x90 (ACK)
```

Do not use this command. This command is used during firmware or information memory update. Use the C2000 Gang Programmer executable GUI software for updating firmware or information memory update if required.

3.5.2.9 Get Label

The Get Label command retrieves all hardware and software information.

```
Tx ->  3E 40 04 04 00 00 00 00 CKL  CKH
Rx ->  80 00 8C 8C D1 D2 ... D140 CKL CKH
```

Data bytes D1 to D140 hold the parameters, as follows:

D1, D2: BOOT software ID ("B430")
D3-D6: BOOT software version (01 00 01 00)
D7, D8: API software ID ("A430")
D9-D12: API software version (01 00 01 09)
D13, D14: Boot revision number: D7 (MSByte), D8 (LSByte)
D15, D16: Hardware version number: D9 (MSByte), D10 (LSByte).
D17, D18: Firmware version number: D11 (MSByte), D12 (LSByte).
D19-D26: Character string representing the boot name "C2000BOOT"
D27: Comma ','
D28-D36: Zero-terminated application firmware name "C2000-GANG"
D37-D44: MCU's Silicon Unique Number
D45-D76: Zero-terminated string of the Programmer description.
D77-D108: Access keys
D109-D116: Programmers serial number YYMMnnnn
D117-D120: MFG ID "ELP "
D121-D124: Hardware ID "C2000"
D125-D126: Hardware revision 0x0101 (rev 1.01)
D127-D140: Spare

3.5.2.10 Get Progress Status

The Get Progress Status command is a low-level command and can be used at any time, even if the C2000 Gang Programmer is busy with other tasks. It replies to the command without interrupting the currently serviced process. Some commands that have long execution times require the Get Progress Status command to monitor the current state.

For example, the Main Process command, which can require a few seconds or more to execute, responds with the character "In Progress 0xB0" as soon as the command has been received and accepted. At this time, the communication link has been released and is ready to use the Get Progress Status command. The current status and progress data can be monitored by polling the Get Progress Status command. The contents of the progress status include the current task number, chunk number, and information about what tasks have been already finished (erase, blank check, program, verify, and more).

Copyright © 2014–2016, Texas Instruments Incorporated

In addition, the comment that is displayed on the LCD display is also available in the progress status message. This makes it possible to mirror the progress status on a PC screen so that the status on the PC screen appears the same as it is in the C2000 Gang Programmer LCD display. In the internal firmware, the progress status buffer is always updated when a new task or new chunk is executed. In cases where the LCD is updated frequently, it might not be possible for the PC screen to exactly mirror it. If polling is done more frequently, then messages on the PC can be updated almost in real time. Polling can be fast, but it is not recommended to send the Get Progress Status command within a 20-ms interval.

The C2000 Gang Programmer has internal 8-level FIFO buffer for progress status (8 internal buffers of 50 bytes each). This allows messages to be retrieved even if status has been changed a few times in the 20-ms interval, as long as the next task is bigger and the status is not updated within the next 100 ms.

One of the bytes (byte 6) in the progress status contains information as to whether the process is still in progress or if it is finished. If the process is finished, then the programmer is ready to receive the next command. If the process is in progress, then only the Get Progress Status command can be used. Do not send any other commands. The next command can also be accepted, but the new command bytes would be collected in the RX buffer until the C2000 Gang Programmer is ready to service it. When the first valid byte of the new command has been received (byte prompt '>' 0x3E), then the receiver cannot get the Get Progress Status command, because the 0xA5 byte, instead of the Get Progress Status command, is treated as a data byte in the data frame.

When the Get Progress Status command is detected (single 0xA5 byte if it is not the frame data contents) then the current status (50 bytes) is transmitted from the C2000 Gang Programmer with following data:

byte	0		0x80
byte	1		0xA5
bytes	2-3	(WORD)	Task counter
bytes	4-5	(WORD)	Chunk counter
byte	6		Status - In Progress, ACK or NACK
byte	7		Ack or nack
bytes	8-9	(WORD)	Finished tasks mask
byte	10		Cumulative gang mask
byte	11		Request gang mask
byte	12		Connected gang mask
byte	13		Erased gang mask
byte	14		Blank check gang mask
byte	15		Programmed gang mask
byte	16		Verified gang mask
byte	17		Secured gang mask
bytes	18-23		Spare
byte	24		Error number
byte	25		Internal VTIO (VTIO = data × 32 mV)
byte	26		VCC gang status mask - A
byte	27		VCC gang status mask - B
byte	28		VCC error mask
byte	29		VCC cumulative error mask
byte	30		JTAG init err mask
byte	31		JTAG Fuse already blown mask
byte	32		Wrong MCU ID mask
byte	33		Progress bar (0 - 100%)
bytes	34-50		Comment text (comment currently displayed on the LCD display)

Where,

Bytes 8-9 are task mask bits:

Copyright © 2014–2016, Texas Instruments Incorporated

CONNECT_TASK_BIT	0x0001
ERASE_TASK_BIT	0x0002
BLANKCHECK_TASK_BIT	0x0004
PROGRAM_TASK_BIT	0x0008
VERIFY_TASK_BIT	0x0010
SECURE_TASK_BIT	0x0020
DCO_CAL_TASK_BIT	0x0040
spare	0x0080 to 0x4000
RST_AND_START_FW_BIT	0x8000

All byte masks (bytes 10 to 17 and 26 to 32) are related to each target device as such:

Target 1	mask 0x01
Target 2	mask 0x02
⋮	⋮
Target 8	mask 0x80

Byte masks 26 to 27 are used in combination as such:

Bits:	B	A	
	0	0	VCC below 0.7 V
	0	1	VCC below VCC min (0.7 V < VCC < VCC min)
	1	0	VCC over VCC min (OK status)
	1	1	VCC over 3.8 V

For example, result 0x83 in connected gang mask (byte 12) means that targets 1, 2, and 8 have been detected and communication with targets successfully established.

Bytes 26 and 27 (VCC status) provide two bits to each target. Bit A for each target and bit B for each target.

3.5.3 Commands Supported by Application Firmware

Commands supported by the application firmware give access to the target device. All of the features that are provided by the C2000 Gang Programmer and available through the GUI and DLL are accessible by these functions. Some of the commands that allow control of the C2000 Gang Programmer are described in the following sections; however, commands that provide data transfer and script information between the C2000 Gang Programmer and DLL are not described here. Users should use the GUI software package (C2000 Gang Programmer executable and DLL) for preparing data for programming, save it in the internal memory or SD card, verify if that works, and then use the commands described in the following sections to control the programming process via RS-232 or USB interface. If it is possible, then it is recommended to use the C2000 Gang Programmer DLL and control the C2000 Gang Programmer via the DLL rather than directly via RS-232 or USB interface using the low-level communication protocol. The C2000 Gang Programmer DLL allows full control of the programmer.

3.5.3.1 Select Image

```
Tx ->  3E 50 4 4 A1 0 0 0 CKL CKH
Rx ->  90 (ACK)
```

The Select Image command sets a number for the current image. After this command, all operations that the C2000 Gang Programmer performs use this image. The C2000 Gang Programmer supports 16 images, 0 through 15. The default image after power on is 0.

A1: holds a number of the image to set (0x00 to 0x0F).

NOTE: When the SD card is inserted to SD slot, then the SD card is selected as the default image, and the Select Image command has no effect.

Copyright © 2014–2016, Texas Instruments Incorporated

3.5.3.2 Main Process

```
Tx ->  3E 31 4 4 0 0 0 0 CKL CKH
Rx ->  B0 (In Progress)
```

The Main Process command begins the gang programming cycle, using the operations defined in the SD or internal image memory. The result of the command execution can be determined using the Get Progress Status command described in Section 3.5.2. It should be noted that the Main Progress commands responds as soon as the command is accepted with byte In Progress (0xB0). When the byte In Progress is received, then the Get Progress Status command should be polled to monitor the progress status. As long as the main process is not finished, byte 6 gives a response of In Progress data (0xB0). When the process is finished, byte 6 changes to ACK (0x90) or NACK (0xA0). When ACK is received, then whole process is finished, and all results are available on bytes 8 to 32. See the Get Progress Status command description for details. During the polling process, it is possible to examine all bytes of the progress status and check the current state; for example, what targets are connected or erased. In the comment bytes (34-50) is the current process, and the same message as is displayed on the LCD display.

3.5.3.3 Set Temporary Configuration

```
Tx ->  3E 56 6 6 A1 0 2 0 DL DH CKL CKH
Rx ->  90 (ACK)
```

By default the Main Process command takes all configuration and setup from the image memory. It is possible to overwrite some of the configuration parameters and execute the Main Process commands with a modified configuration. The following parameters can be modified: Targets VCC, high or low current, external VCC enable or disable, VCC settle time, communication interface (JTAG), enabled target devices and enable process mask (for example, erase or program verify). The Set Temporary Configuration in C2000 Gang Programmer command allows modification of these parameters.

When the Main Process command is finished, then the temporary setups are erased and the configuration from the image memory is restored. When the modified configuration should be used in the next run, then the temporary configuration should be transferred to C2000 Gang Programmer again before starting the Main Process command.

The Set Temporary Configuration in C2000 Gang Programmer command transfers two data: address index (A1) and one 16-bit data [DL (LSB byte) and DH (MSB byte)].

The following address indexes are defined:

CFG_TMP_CLEAR (2)

Data (DH, DH) is irrelevant.

Remove temporary configuration and take it from the image memory.

CFG_TMP_TASK_MASK (4)

Set the execution mask.

By default execution mask is 0xFFFF (execute all procedures).

Data (DH, DL) can be from 0x0000 up to 0xFFFF.

Currently supported bits in the execution mask:

CONNECT_TASK_BIT	0x0001
ERASE_TASK_BIT	0x0002
BLANKCHECK_TASK_BIT	0x0004
PROGRAM_TASK_BIT	0x0008
VERIFY_TASK_BIT	0x0010
SECURE_TASK_BIT	0x0020
DCO_CAL_TASK_BIT	0x0040

Copyright © 2014–2016, Texas Instruments Incorporated

For example, when the target device must be erased, then only the following data should be send (A1, D).

```
4, 0x0003
```

Full command:

```
Tx -> 3E 56 6 6 4 0 2 0 3 0 CKL CKH
```

CFG_TMP_VCC_VALUE (6)

Data - VCC value in mV (range from 1800 to 3600)

CFG_TMP_POWER_VCC_EN (8)

Data	0	Target devices powered from an external power supply
Data	1	Target devices powered from C2000 Gang Programmer

CFG_TMP_INTERFACE (10)

Data	JTAG_FAST	0x0004
Data	JTAG_MED	0x0005
Data	JTAG_SLOW	0x0006
Data	SBW_FAST	0x0008
Data	SBW_MED	0x0009
Data	SBW_SLOW	0x000A

CFG_TMP_GANG_MASK (12)

Sum of target bit masks

Target 1	0x01
Target 2	0x02
Target 3	0x04
⋮	⋮
Target 8	0x80

One target only - Target 1	Data = 0x0001
All targets	Data = 0x00FF

CFG_TMP_VCC_ONOFF (14)

Immediately turn VCC target on of off

Data	0x0001	ON
Data	0x0000	OFF

CFG_TMP_ICC_HI_EN (18)

High (50 mA) current from programmer enable or disable

Data	0x0001	Enable
Data	0x0000	Disable

Copyright © 2014–2016, Texas Instruments Incorporated

CFG_TMP_IO_INTERFACE (20)

Set interface configuration

Data	0x0000	SBW via TDOI line
Data	0x0001	SBW via RST line

CFG_TMP_RESET (22)

Immediately reset target device

Data	0x0001	Reset target device
Data	0x0000	Release Reset line

CFG_TMP_VCC_SETTLE_TIME (26)

Data	0x0000 to 0x00C8	Settle VCC time in step 20 ms

3.5.3.4 Get Gang Status

```
Tx ->3E 58 04 04 A1 0 - - 0 0 - - CKL CKH
Rx ->80 0 n n B0 B1 B2 B3 ... Bn CKL CKH
```

The Get Gang Status command gets the selected status or results from the C2000 Gang Programmer. The following numbers (A1) are available. See the description of the C2000GANG_GetAPIStatus function (Section 4.1.41) for details of the B0...Bn byte contents.

GET_APP_FLAGS	10
GET_LAST_STATUS	12
GET_LAST_ERROR_NO	14

3.5.3.5 Read Gang Buffer

```
Tx -> 3E 49 4 4 T 0 - - n 0 - - CKL CKH
Rx -> 80 0 n n D1 D2 D3 D4 D5 D6 D7 D8...Dn CKL CKH
```

The C2000 Gang Programmer contains a temporary data buffer that can be used for writing data to and reading data from each target device. The buffer size is 64 words for each target device - Buffer[8][64];

T = Target device number, 1 to 8

n = Number of words taken from the Buffer[T-1] [..]

3.5.3.6 Write Gang Buffer

```
Tx -> 3E 4A n+4 n+4 T 0 - - n 0 D1 D2...Dn CKL CKH
Rx -> ACK
```

Write words to selected target's Buffer -> Buffer[8][64]

T = Target device number, 1 to 8

n = Number of bytes written to Buffer[T-1] [..]

Copyright © 2014–2016, Texas Instruments Incorporated

3.5.4 API Firmware Commands That Should Not be Used

3.5.4.1 Interactive Task

```
Tx -> 3E 46 n n D1 ... Dn CKL CKH
Rx -> 80 0 k k D1 ... Dk CKL CKH
```

NOTE: Do not use this command. This command is used by the API-DLL and GUI only.

3.5.4.2 Erase Image

```
Tx -> 3E 33 4 4 0 0 0 0 CKL CKH
Rx -> B0 (In Progress)
```

NOTE: Do not use this command. This command is used by the API-DLL and GUI only.

3.5.4.3 Get Info C-D

```
Tx -> 3E 41 4 4 A1 0 0 0 CKL CKH
Rx -> 80 0 80 80 D1 ... D128 CKL CKH
```

NOTE: Do not use this command. This command is used by the API-DLL and GUI only.

3.5.4.4 Write Info C-D

```
Tx -> 3E 42 84 84 A1 0 80 0 D1 ... D128 CKL CKH
Rx -> ACK
```

NOTE: Do not use this command. This command is used by the API-DLL and GUI only.

3.5.4.5 Verify Access Key

```
Tx -> 3E 44 4 4 0 0 0 0 CKL CKH
Rx -> ACK or NACK
```

NOTE: Do not use this command. This command is used by the API-DLL and GUI only.

3.5.4.6 Load Image

```
Tx -> 3E 43 n n A1 A2 A3 0 n-6 0 D1 ... Dn-6 CKL CKH
Rx -> ACK or NACK
```

The Load Image command loads the data bytes into the image buffer of the C2000 Gang Programmer. Do not use this function in your application. Use C2000-GANG GUI and C2000-GANG DLL for writing data into the internal image buffer.

3.5.4.7 Verify Image CRC

```
Tx -> 3E 45 08 08 A1 A2 A3 0 LL LH D1 D2 CKL CKH
Rx -> ACK or NACK
```

The Verify Image CRC command verifies the image CRC of all written image contents. Do not use this function in your application. Use C2000-GANG GUI and C2000-GANG DLL for writing and verifying data in the internal image buffer.

Copyright © 2014–2016, Texas Instruments Incorporated

3.5.4.8 Get Image Header

```
Tx -> 3E 47 6 6 A1 A2 0 0 n 0 CKL CKH
Rx -> 80 0 n n D1 ... Dn CKL CKH
```

NOTE: Do not use this command. This command is used by the API-DLL and GUI only.

3.5.4.9 Disable API Interrupts

```
Tx -> 3E 4C 4 4 R R R R CKL CKH
Rx -> ACK
```

NOTE: Do not use this command. This command is used by the API-DLL and GUI only.

Copyright © 2014–2016, Texas Instruments Incorporated

3.5.4.10 Display Message

```
Tx -> 3E 54 n+4 n+4 A1 A2 n 0 D1 ... Dn CKL CKH
Rx -> ACK
```

NOTE: Do not use this command. This command is used by the API-DLL and GUI only.

3.5.4.11 Set IO State

```
Tx -> 3E 4E 0C 0C VL VH 08 00 D1 D2 D3 D4 D5 D6 D7 D8 CKL CKH
Rx -> ACK
```

Modify static levels on the I/O pins (JTAG lines).

Vcc - V_{CC} level in mV (V_{CC} = VH × 256 + VL)

D1 - Open destination buffer for output and transferred data for each target
- 0 = none
- 1 = TDI (target1 to target8)
- 2 = TDOI (target1 to target8)
- 3 = TMS (target1 to target8)
- 4 = RST (target1 to target8)
- 5 = BSL-RX (target1 to target8)

D2 - data transferred to the buffer above
- b0 to b7 - target1 to target8

D3 - output enable bits: 0 = high impedance, 1 = output
- b2 (0x04) - common RST - the same state for all eight targets (Note: if the RST buffer above is selected, then this state is ignored)
- b3 (0x08) - common TEST - the same state for all eight targets
- b4 (0x10) - common TCK - the same state for all eight targets
- b5 (0x20) - common TMS - the same state for all eight targets (Note: if the TMS buffer above is selected, then this state is ignored)

D4 - output level on all targets: 0 = LOW, 1 = HIGH
- b2 (0x04) - common RST - the same level for all eight targets (Note: if the RST buffer above is selected, then this state is ignored)
- b3 (0x08) - common TEST - the same level for all eight targets
- b4 (0x10) - common TCK - the same level for all eight targets
- b5 (0x20) - common TMS - the same level for all eight targets (Note: if the TMS buffer above is selected, then this state is ignored)

D5 - V_{CC} enable bits to each targets
- b0 to b7 - target1 to target8

D6 - I_{CC} HI enable: 0 = disable, 1 = enable

D7 - spare

D8 - spare

Example 1

Generate a short RST pulse on target 1 only and force RST level LOW on targets 2 to 5 and RST level HIGH on targets 6 and 7. V_{CC} on targets 1 to 7 is 3.3 V (0x0CE4) and on target 8 is 0 V (disabled).

```
Tx -> 3E 4E 0C 0C E4 0C 08 00 04 60 00 00 7F 00 00 00 CKL CKH
```

then

```
Tx -> 3E 4E 0C 0C E4 0C 08 00 04 61 00 00 7F 00 00 00 CKL CKH
```


Copyright © 2014–2016, Texas Instruments Incorporated

Example 2

Generate a short RST pulse on all targets. V_{CC} on targets 1 to 7 is 3.3 V (0x0CE4) and on target 8 is 0 V (disabled).

```
Tx -> 3E 4E 0C 0C E4 0C 08 00 00 00 04 00 7F 00 00 00 CKL CKH
```

then

```
Tx -> 3E 4E 0C 0C E4 0C 08 00 00 00 04 04 7F 00 00 00 CKL CKH
```

3.5.4.12 Remote Selftest

```
Tx -> 3E 71 n+6 n+6 A1 A2 A3 A4 n 0 D1 D2...Dn CKL CKH
Rx -> 80 0 n n D1 D2 D3 D4 D5 D6 D7 D8...Dn CKL CKH
```

NOTE: Do not use this command. This command is used by the API-DLL and GUI only.

Copyright © 2014–2016, Texas Instruments Incorporated

Dynamic Link Library for C2000 Gang Programmer

4.1 C2000-GANG.dll Description

The C2000-GANG.dll is a Dynamic Link Library (DLL) that provides functions for controlling the C2000 Gang Programmer. The C2000-GANG.dll controls the Gang Programmer via RS 232 or USB (VCP) interface. The C2000-GANG.dll greatly simplifies the control of the C2000 Gang Programmer, because the user is isolated from the complexities of the communication via USB or RS232 interface protocol. Together with the C2000-GANG.dll are provided two more files that should be used during the compilation process.

- C2000-GANG.h: This file is the header file for the C2000-GANG.dll, and provides the function prototypes, typedefs, #defines, and data structures for the functions of the C2000-GANG.dll. This file is normally located in the same directory as the application source file and should be included by the application source files. This file is used during compile time.
- C2000-GANG.lib: This file is the library file for the C2000-GANG.dll and is required to access the DLL functions. This file is normally located in the same directory as the application source file and should be added to the Linker Object, Library Modules list of the application. This file is used during link time.

All C2000 Gang Programmer DLL functions have the same "C2000GANG_" prefix in the function name. It is easy in the application software to determine what functions are used with the C2000-GANG.dll. The following sections describe each function.

Examples of using the new C2000-GANG.dll are provided and can be found in these locations (if the default installation directory was used):

C:\Program Files\Texas Instruments\C2000-GANG\Examples\C_Applications_C2000_DLL
and
C:\Program Files\Texas Instruments\C2000-GANG\Examples\Cpp_Applications_C2000_DLL

These examples show how to configure the C2000 Gang Programmer to the desired target device type, select code, and subsequently program connected devices. In addition, the examples also show how to write a serial number into a custom memory location. To use these examples copy the MSG-Gang.dll into the working directory.

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.1 C2000GANG_GetDataBuffers_ptr

C2000GANG_GetDataBuffers_ptr gives access to the internal data buffers that provide code contents, data to be programmed, and buffers of data that was read from each target device with following structure.

```
#define  DBUFFER_SIZE         0x42000
#define  FLASH_END_ADDR       0x3FFFFF
#define  FLASH_BUF_LEN        DBUFFER_SIZE
#define  GANG_SIZE            8

typedef struct
{
   WORD  SourceCode[DBUFFER_SIZE];        //source code from the file
   WORD  UsedCode[DBUFFER_SIZE];          //combined data (source code, serialization etc)
   WORD  CommonTx[DBUFFER_SIZE];          //data used to writing - the same data to all targets
   WORD  GangTx[DBUFFER_SIZE][GANG_SIZE]; //selective data used to writing
   WORD  GangRx[DBUFFER_SIZE][GANG_SIZE]; //data read from all targets
   WORD  Tmp[DBUFFER_SIZE];
   WORD  CSM_Passsword[NO_OF_CSM_PASSWORDS][CSM_PASSWORD_SIZE];
   BYTE  Flag_ScrCode[DBUFFER_SIZE];         //0 - empty  1-valid data  in  SourceCode[x];
         #define     CODE1_FLAG        1
         #define     CODE2_FLAG        2
         #define     APPEND_CODE_FLAG  4
   BYTE  Flag_UsedCode[DBUFFER_SIZE];     //0 - empty  1-valid data  in  UsedCode[x];
   BYTE  Flag_WrEn[DBUFFER_SIZE];         //0 - none   1-write/verify enable in FlashMem[x]
   BYTE  Flag_EraseEn[DBUFFER_SIZE];         //0 - none   1-erase enable   in FlashMem[x]
   BYTE  Flag_RdEn[DBUFFER_SIZE];         //0 - none   1-read enable in FlashMem[x]
   BYTE  Flag_Sp1[DBUFFER_SIZE];          //spare
   BYTE  Flag_Sp2[DBUFFER_SIZE];          //spare
   BYTE  Flag_Sp3[DBUFFER_SIZE];          //spare
   BYTE  Flag_CSM_Passw[NO_OF_CSM_PASSWORDS][CSM_PASSWORD_SIZE];  //spare
} DATA_BUFFERS;
extern   DATA_BUFFERS  dat;
```

In the application software, the pointer to the dat buffer can be initialized as follows.

```
DATA_BUFFERS *DBuf;
void *temp;
C2000GANG_GetDataBuffers_ptr((&temp));
DBuf = (DATA_BUFFERS *)temp;
```

Syntax

```
LONG C2000GANG_GetDataBuffers_ptr(void ** x)
```


Copyright © 2014–2016, Texas Instruments Incorporated

4.1.2 C2000GANG_SetGangBuffer, C2000GANG_GetGangBuffer

The C2000 Gang Programmer contains a temporary data buffer that can be used for writing and reading data to each target device. Buffer size is 64 words for each target device.

```
Buffer[8][64];
```

C2000GANG_SetGangBuffer writes data to selected Buffer. C2000GANG_GetGangBuffer reads contents from the selected buffer.

Syntax

```
LONG C2000GANG_SetGangBuffer(BYTE target, BYTE size, WORD *data)
LONG C2000GANG_GetGangBuffer(BYTE target, BYTE size, WORD *data)
```

Arguments

BYTE target	Target number (1 to 8)
BYTE size	Size of data (1 to 64)
WORD *data	Pointer to data buffer from where data is taken or to where the data should be saved

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.3 C2000GANG_GetDevice

Reads all specific parameters of a device type from the internal C2000-GANG .DLL table and returns data related to the selected device.

Syntax

```
LONG WINAPI C2000GANG_GetDevice(LPTSTR lpszDeviceName, void **lpData)
```

Arguments

LPTSTR lpszDeviceName	MCU name. The device name; for example, 'C2000F5438A' for desired MCU or (blank) for currently selected MCU
void *lpData	Pointer to internal structure

Result

LONG	Error code

```
typedef struct
{
   long  Group;
   long  RAM_size;
   long  OTP_start_addr;
   long  OTP_end_addr;
   long  MainMem_start_addr;
   long  MainMem_end_addr;
   long  family_index;
   long  Internal_OSC_kHz;
   long  spare[32];
} DEVICE_INFO;
extern   DEVICE_INFO  device;
```

In the application software, the pointer to the device info structure can be initialized as follows.

```
DEVICE_INFO *Device;
void *temp;
     C2000GANG_GetDevice(" ", &temp);
     Device = (DEVICE_INFO *)temp;
```


Copyright © 2014–2016, Texas Instruments Incorporated

4.1.4 C2000GANG_LoadFirmware

Load firmware from C2000-GANG.dll to C2000 Gang Programmer,

NOTE: Do not use this command. This command is used by the API-DLL and GUI only.

Syntax

```
LONG C2000GANG_LoadFirmware(void)
```

4.1.5 C2000GANG_InitCom

C2000GANG_InitCom opens a communications port, sets the baudrate and checks if the C2000 Gang Programmer is present.

Syntax

```
LONG C2000GANG_InitCom(LPTSTR lpszPort, LONG lBaud)
```

Arguments

char * lpszComPort	Name of the port
LONG lBaudRate	Baud rate

Result

LONG	Error code

4.1.6 C2000GANG_ReleaseCom

Release communications port

Syntax

```
LONG C2000GANG_ReleaseCom(void)
```

Arguments

None

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.7 C2000GANG_GetErrorString

Returns the error string for the selected error number (response from any functions that returns error status).

Syntax

```
LPTSTR C2000GANG_GetErrorString(LONG lErrorNumber)
```

Arguments

LONG lErrorNumber	Error number

Result

LPTSTR	Error string

4.1.8 C2000GANG_SelectBaudrate

C2000GANG_SelectBaudrate sets the rate of the serial communications. The default is 9600 baud. Baud rate index 0 to 4, representing the baud rate. The Select Baud Rate command takes effect (that is, changes the baud rate) immediately.

Syntax

```
LONG C2000GANG_SelectBaudrate(LONG lBaud)
```

Arguments

LONG lBaud	Baud rate in bytes per second 0 = 9600 baud (default) 1 = 19200 baud 2 = 38400 baud 3 = 57600 baud 4 = 115200 baud

Result

LONG	Error code

4.1.9 C2000GANG_GetDiagnostic

See the Get Diagnostic command (Section 3.5.2.4) for detailed information about received data contents.

Syntax

```
LONG C2000GANG_GetDiagnostic(void **lpData)
```

Arguments

void ** lpData	Pointer to data buffer

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.10 C2000GANG_MainProcess

C2000GANG_MainProcess starts the execution if all function saved inside image memory (or SD card memory). That includes targets initialization, fuse check, memory erase, blank check, program, verification, and more, if selected (for example, DCO calibration).

Syntax

```
LONG C2000GANG_MainProcess(LONG timeout)
```

Arguments

LONG timeout	In seconds

Result

LONG	Error code

4.1.11 C2000GANG_InteractiveProcess

C2000GANG_InteractiveProcess starts the execution if all function provided in the interactive mode, similar to the C2000GANG_MainProcess function; however, data is taken from the PC, not from the image (or SD) memory.

Syntax

```
LONG C2000GANG_InteractiveProcess(LONG timeout)
```

Arguments

LONG timeout	In seconds

Result

LONG	Error code

4.1.12 C2000GANG_Interactive_Open_Target_Device

C2000GANG_Interactive_Open_Target_Device is used in the interactive mode and in initializing access to target devices (setting Vcc, checking fuse, and initializing JTAG communication with target devices). The argument 'name' is displayed on the LCD display. It can contains no more then 16 characters. Extra characters are ignored.

Syntax

```
LONG C2000GANG_Interactive_Open_Target_Device(LPTSTR name)
```

Arguments

LPTSTR name

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.13 C2000GANG_Interactive_Close_Target_Device

C2000GANG_Interactive_Close_Target_Device is used in the interactive mode and in closing access to target devices.

Syntax

```
LONG C2000GANG_Interactive_Close_Target_Device(void)
```

Result

LONG	Error code

4.1.14 C2000GANG_Interactive_DefReadTargets

Note: The target device must be opened first if not open yet (see C2000GANG_Interactive_Open_Target_Device, Section 4.1.12).

C2000GANG_Interactive_DefReadTargets reads the contents of the selected target devices (one to eight targets) simultaneously from Start_addr to the End_addr and saves it in the internal data buffer (see DATA_BUFFERS dat; structure for details).

Syntax

```
LONG C2000GANG_Interactive_DefReadTargets(BYTE mask, BYTE bar_min, BYTE bar_max, LONG Start_addr,
LONG End_addr)
```

Arguments

BYTE mask	Mask of the target devices that data should be read from
BYTE bar_min	Beginning progress bar value displayed on the LCD display (valid values are 0 to 100).
BYTE bar_max	Ending —,,,---
LONG Start_addr	Data read from Start_addr location
LONG End_addr	Data read up to the End_addr location

Result

LONG	Error code

4.1.15 C2000GANG_Interactive_ReadTargets

Note: The target device must be opened first if not open yet (see C2000GANG_Interactive_Open_Target_Device, Section 4.1.12).

C2000GANG_Interactive_ReadTargets reads the contents of the selected target devices (one to eight targets) simultaneously from the locations specified in the configuration memory (see configuration setup for details) and saves it in the internal data buffer (see DATA_BUFFERS dat; structure for details).

Syntax

```
LONG C2000GANG_Interactive_ReadTargets(BYTE mask)
```

Arguments

BYTE mask	Mask of the target devices that data should be read from

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.16 C2000GANG_Interactive_ReadWords

Note: The target device must be opened first if not open yet (see C2000GANG_Interactive_Open_Target_Device, Section 4.1.12).

C2000GANG_Interactive_ReadWords reads contents from one selected target device and saves it in the desired data buffer.

Syntax

```
LONG WINAPI C2000GANG_Interactive_ReadWords( BYTE target_no, LONG addr, LONG size, WORD * data )
```

Arguments

BYTE target_no	Target number (one to eight) of the desired target device
LONG addr	Start address from read data
LONG size	Number of read words
WORD *data	Pointer to buffer where data would be saved

Result

LONG	Error code

4.1.17 C2000GANG_Interactive_WriteWord_to_RAM

Note: The target device must be opened first if not open yet (see C2000GANG_Interactive_Open_Target_Device, Section 4.1.12).

C2000GANG_Interactive_WriteWord_to_RAM writes one word (16 bits) to any RAM or I/O location. The address must be even.

Syntax

```
LONG C2000GANG_Interactive_WriteWord_to_RAM(LONG addr, LONG data)
```

Arguments

LONG addr	RAM address location
BYTE data	Data (16 bits)

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.18 C2000GANG_Interactive_WriteWords_to_RAM

Note: The target device must be opened first if not open yet (see C2000GANG_Interactive_Open_Target_Device, Section 4.1.12).

C2000GANG_Interactive_WriteWords_to_RAM writes 'size' number of words to any RAM or I/O location. The starting address must be even.

Syntax

```
LONG C2000GANG_Interactive_WriteWords_to_RAM(LONG addr, LONG size, WORD * data)
```

Arguments

LONG addr	RAM address location
LONG size	Number of words to be written
WORD * data	Data block

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.19 C2000GANG_Interactive_WriteWords_to_FLASH

Note: The target device must be opened first if not open yet (see C2000GANG_Interactive_Open_Target_Device, Section 4.1.12).

C2000GANG_Interactive_WriteWords_to_FLASH writes 'size' number of words to any flash location. The starting address must be even.

Syntax

```
LONG C2000GANG_Interactive_WriteWords_to_FLASH(LONG addr, LONG size, WORD * data)
```

Arguments

LONG addr	RAM address location
LONG size	Number of words to be written
WORD * data	Data block

Result

LONG	Error code

4.1.20 C2000GANG_Interactive_Copy_Gang_Buffer_to_RAM

Note: The target device must be opened first if not open yet (see C2000GANG_Interactive_Open_Target_Device, Section 4.1.12).

C2000GANG_Interactive_Copy_Gang_Buffer_to_RAM writes 'size' number of words from the internal Gang_Buffer[8][64] to RAM - simultaneously to all active target devices. Data for each target can be different. Contents from Gang_Buffer[0][n] are written to target 1, contents from Gang_Buffer[1][n] are written to target 2, and contents from Gang_Buffer[7][n] are written to target 8.

Data in the Gang_Buffer should be prepared and sent to C2000 Gang Programmer first. See C2000GANG_GetGangBuffer and C2000GANG_SetGangBuffer functions for details.

Syntax

```
LONG C2000GANG_Interactive_Copy_GANG_Buffer_to_RAM(LONG addr, LONG size)
```

Arguments

LONG addr	RAM address location
LONG size	Number of words to be written (up to 64)

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.21 C2000GANG_Interactive_Copy_Gang_Buffer_to_FLASH

Note: The target device must be opened first if not open yet (see C2000GANG_Interactive_Open_Target_Device, Section 4.1.12).

C2000GANG_Interactive_Copy_Gang_Buffer_to_FLASH writes 'size' number of words from the internal Gang_Buffer[8][64] to FLASH, simultaneously to all active target devices. Data for each target can be different (for example, calibration data or serial numbers). Contents from Gang_Buffer[0][n] are written to target 1, contents from Gang_Buffer[1][n] are written to target 2, and contents from Gang_Buffer[7][n] are written to target 8.

Data in the Gang_Buffer should be prepared and sent to C2000 Gang Programmer first. See C2000GANG_GetGangBuffer and C2000GANG_SetGangBuffer functions for details.

Syntax

```
LONG C2000GANG_Interactive_Copy_GANG_Buffer_to_FLASH(LONG addr, LONG size)
```

Arguments

LONG addr	FLASH address location
LONG size	Number of words to be written

Result

LONG	Error code

4.1.22 C2000GANG_Interactive_EraseSectors

Note: The target device must be opened first if not open yet (see C2000GANG_Interactive_Open_Target_Device, Section 4.1.12).

C2000GANG_Interactive_EraseSectors erases flash sectors starting from the sector with address location StartAddr and ending with the sector with EndAddr location.

Syntax

```
LONG C2000GANG_Interactive_EraseSectors(LONG StartAddr, LONG EndAddr)
```

Arguments

LONG StartAddr	FLASH address location of the first sector to be erased. Address aligned to the sector size.
LONG EndAddr	Address of the last sector to be erased. The address is aligned to the sector size.

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.23 C2000GANG_Interactive_BlankCheck

Note: The target device must be opened first if not open yet (see C2000GANG_Interactive_Open_Target_Device, Section 4.1.12).

C2000GANG_Interactive_BlankCheck verifies all flash contents starting from StartAddr and ending with EndAddr are 0xFF.

Syntax

```
LONG C2000GANG_Interactive_BlankCheck(LONG StartAddr, LONG EndAddr)
```

Arguments

LONG StartAddr LONG EndAddr	Blank check (if 0xFF) from StartAddr location to EndAddr location Start Address must be even, End address must be odd.

Result

LONG	0 = blank !0 = error (not blank or error)

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.24 C2000GANG_SelectImage

C2000GANG_SelectImage sets an active image to work with. C2000 Gang Programmer supports 16 images.

Syntax

```
LONG C2000GANG_SelectImage(LONG lImage)
```

Arguments

LONG lImage	Image number (0 to 15)

Result

LONG	Error code

4.1.25 C2000GANG_EraseImage

C2000GANG_EraseImage clears (presets with 0xFF) active image memory. Use the C2000GANG_SelectImage function to select desired image memory.

Syntax

```
LONG C2000GANG_EraseImage(void)
```

Result

LONG	Error code

4.1.26 C2000GANG_CreateGangImage

C2000GANG_CreateGangImage creates a command script and the data to be written to target devices according to the current C2000 Gang Programmer configuration. After the image data is prepared, then it can be saved in the selected image memory by calling the C2000GANG_LoadImageBlock function.

Syntax

```
LONG C2000GANG_CreateGangImage(LPTSTR name)
```

Arguments

LPTSTR name	Image name; maximum of 16 characters. Image name is displayed on the LCD display.

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.27 C2000GANG_LoadImageBlock

C2000GANG_LoadImageBlock saves the previously prepared image contents into the selected image memory. The selected image memory must be erased first. Use the following sequence for preparing and saving an image into image memory:

```
C2000GANG_CreateGangImage(name);
C2000GANG_SelectImage(lImage);
C2000GANG_EraseImage();
C2000GANG_LoadImageBlock();
C2000GANG_VerifyPSAImageBlock();
```

Syntax

```
LONG C2000GANG_LoadImageBlock(void)
```

Arguments

None

Result

LONG	Error code

4.1.28 C2000GANG_VerifyPSAImageBlock

C2000GANG_VerifyPSAImageBlock verifies the checksum of all blocks used in the selected image memory. The image memory number should be selected first using C2000GANG_SelectImage function.

Syntax

```
LONG C2000GANG_VerifyPSAImageBlock(void)
```

Arguments

None

Result

LONG	Error code

4.1.29 C2000GANG_ReadImageBlock

C2000GANG_ReadImageBlock reads the header from the selected image memory. A maximum of 127 words can be read. Access to the remaining image memory (up to 512 kBytes (256 kWords)) is blocked.

Syntax

```
LONG C2000GANG_ReadImageBlock(LONG addr, LONG size, void *lpData)
```

Arguments

LONG address	
LONG size	
void *lpData	Pointer to word buffer where the result is saved

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.30 C2000GANG_Read_Code_File

C2000GANG_Read_Code_File reads or appends a code file or reads a password file and saves it in its internal buffer. By default, the file is treated as the main code file as long as the setup has not redirected the file to 'Append code' or 'Password code' using the C2000GANG_SetConfig function.

```
C2000GANG_SetConfig(CFG_OPEN_FILE_TYPE, CODE_FILE_INDEX)
C2000GANG_SetConfig(CFG_OPEN_FILE_TYPE, APPEND_FILE_INDEX)
C2000GANG_SetConfig(CFG_OPEN_FILE_TYPE, PASSW_FILE_INDEX)
```

When the C2000GANG_Read_Code_File is executed, the flag set by C2000GANG_SetConfig(CFG_OPEN_FILE_TYPE, CODE_FILE_INDEX) is set to the default value of Read Code File.

Syntax

```
LONG C2000GANG_Read_Code_File(LPTSTR FullPath)
```

Arguments

LPTSTR FullPath	Path to the code file (*.hex,*.txt or *.s19, *.s28, *.s37)

Result

LONG	Error code

4.1.31 C2000GANG_Save_Config, C2000GANG_Load_Config, C2000GANG_Default_Config

The current configuration file can be saved using the C2000GANG_Save_Config function and recalled when required using the C2000GANG_Load_Config function. The current configuration can be erased and the default configuration loaded by calling the C2000GANG_Default_Config function. When the new configuration is loaded, some of the parameters can be modified item-by-item using C2000GANG_SetConfig and can be read from the configuration item-by-item using C2000GANG_GetConfig. The C2000 Gang Programmer configuration can also be created using the C2000 Gang Programmer GUI software (C2000-GANG-exe) by setting the desired programmer setup, verifying it, and then saving the configuration using the Save Setup as... option. The setup created using the GUI can be loaded by the DLL when the above mentioned configuration file is selected using the C2000GANG_Load_Config function.

Syntax

```
LONG C2000GANG_Save_Config(LPTSTR filename)
LONG C2000GANG_Load_Config(LPTSTR filename)
LONG C2000GANG_Default_Config(void)
```

Arguments

LPTSTR filename	Path to the configuration file

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.32 C2000GANG_SetConfig, C2000GANG_GetConfig

Syntax

```
LONG C2000GANG_SetConfig(LONG index, LONG data)
```

Arguments

LONG index	Configuration index. See list below.
LONG data	Configuration data

Result

LONG	Error code

Syntax

```
LONG C2000GANG_GetConfig(LONG index)
```

Arguments

LONG index	Configuration index. See list below.

Result

LONG data	Configuration data

List of Indexes

```
#define     FROMIMAGE_BIT           0x1000

#define     CFG_INTERFACE           0
            #define   INTERFACE_NONE       0
            #define   INTERFACE_JTAG       4
            #define   INTERFACE_SBW        8
            #define   INTERFACE_BSL        0xC
            #define   INTERFACE_TYPE_MAX_INDEX      INTERFACE_BSL

#define     CFG_JTAG_SPEED          1
            #define   INTERFACE_FAST       0
            #define   INTERFACE_MED        1
            #define   INTERFACE_SLOW       2
            #define   INTERFACE_SPEED_MAX_INDEX     INTERFACE_SLOW

#define     CFG_IO_INTERFACE        4
            #define   SBW_VIA_RST_BIT      0x01
            //  0  -  SBW_VIA_TDIO  (pin 1)  and TCK/TEST (pin-7/8)
            //  1  -  SBW_VIA_RST   (pin 11) and TCK/TEST (pin-7/8)

#define     CFG_POWERTARGETEN       6
            // disable     0  (external power supply)
            // enable      1  (targets supplied by C2000-GANG)

#define     CFG_VCCINDEX            7
            // Vcc in mV   1800 - 3600

#define     CFG_BLOWFUSE            9
            // disable     0
            // enable      1

#define     CFG_TARGET_EN_INDEX     10
            // Targets GANG enable mask - 0x00 ...0xFF. Enable all targets -> 0xFF
            #define  TARGET_1_MASK     0x01
            #define  TARGET_2_MASK     0x02
```


Copyright © 2014–2016, Texas Instruments Incorporated

```
                #define  TARGET_3_MASK     0x04
                #define  TARGET_4_MASK     0x08
                #define  TARGET_5_MASK     0x10
                #define  TARGET_6_MASK     0x20
                #define  TARGET_7_MASK     0x40
                #define  TARGET_8_MASK     0x80

        #define     CFG_FLASHERASEMODE      11
                    #define  ERASE_NONE_MEM_INDEX       0
                    #define  ERASE_ALL_MEM_INDEX        1
                    #define  ERASE_PRG_ONLY_MEM_INDEX   2
                    #define  ERASE_INFILE_MEM_INDEX     3
                    #define  ERASE_DEF_CM_INDEX         4
                    #define  WRITE_OTP_MEM_ONLY_INDEX   5
                    #define  ERASE_MAX_INDEX            ERASE_DEF_CM_INDEX



        #define     CFG_ERASESTARTADDR      17
                    // FLASH/FRAM start erase address

        #define     CFG_ERASESTOPADDR       18
                    // FLASH/FRAM end erase address

        #define     CFG_FLASHREADMODE       19
                    #define  READ_ALL_MEM_INDEX         0
                    #define  READ_PRGMEM_ONLY_INDEX     1
                    #define  READ_INFOMEM_ONLY_INDEX    2
                    #define  READ_DEF_MEM_INDEX         3
                    #define  READ_MEM_MAX_INDEX         READ_DEF_MEM_INDEX

        #define     CFG_FINALACTION_MODE    24
                    #define  APPLICATION_NO_RESET            0
                    #define  APPLICATION_TOGGLE_RESET        1
                    #define  APPLICATION_TOGGLE_VCC          2
                    #define  APPLICATION_JTAG_RESET          3
                    #define  APPLICATION_RESET_MAX_INDEX     APPLICATION_JTAG_RESET

        #define     CFG_BEEPMODE            25
                       // sum of following bits
                    #define  BEEP_PCSPK_EN_BIT       1     //Beep via PC Speaker enable
                    #define  BEEP_OK_EN_BIT          2     //Beep when OK enable
                    #define  BEEP_SOUND_EN_BIT       4     //Sound enable

        #define     CFG_DEFERASEMAINEN      26
                    // disable     0
                    // enable      1

        #define     CFG_CUSTOMRESETPULSETIME   27
                    //  time in ms  1.....2000

        #define     CFG_CUSTOMRESETIDLETIME    28
                    //  time in ms  1.....2000

        #define     CFG_RETAIN_CAL_DATA_INDEX  31
                    // disable     0
                    // enable      1

        #define     CFG_FINALACTIONRUNTIME     32
                    // 0 - infinite,
                    //  1...120 time in seconds

        #define     CFG_FINALACTIONVCCOFFTIME  33
                    // Vcc-OFF (then again ON) time after programming when the
                    // APPLICATION_TOGGLE_VCC option is selected.
```


Copyright © 2014–2016, Texas Instruments Incorporated

```
#define     CFG_READMAINMEMEN       39
            // disable     0
            // enable      1

#define     CFG_READDEFSTARTADDR    40
            // Memory READ start address

#define     CFG_READDEFSTOPADDR     41
            // Memory READ end address

#define     CFG_COMPORT_NO          42
            // Communication COM Port number - 0..255

#define     CFG_UART_SPEED          43
            // Baud Rate index
            #define  UART_9600      0
            #define  UART_19200     1
            #define  UART_38400     2
            #define  UART_57600     3
            #define  UART_115200    4

#define     CFG_OPEN_FILE_TYPE      44
            #define     CODE_FILE_INDEX         0
            #define     APPEND_FILE_INDEX       1
            #define     PASSW_FILE_INDEX        2
            #define     SECONDCODE_FILE_INDEX   3
            #define     CODE2_FILE_INDEX        4

#define     CFG_USE_SCRIPT_FILE     45
            // disable     0
            // enable      1

#define     CFG_IMAGE_NO            46
            //image number - 0...15

#define     CFG_RESETTIME           47
            #define  RESET_10MS_INDEX        0
            #define  RESET_100MS_INDEX       1
            #define  RESET_200MS_INDEX       2
            #define  RESET_500MS_INDEX       3
            #define  RESET_CUSTOM_INDEX      4
            #define  RESET_MAX_INDEX         RESET_CUSTOM_INDEX

#define     CFG_PROJECT_SOURCE      48
            #define  INTERACTIVE_MODE           0
            #define  FROM_IMAGE_MEMORY_MODE     1
            #define  STANDALONE_MODE            2
            #define  FROM_IMAGE_FILE_MODE       3
            #define  PROJECT_SOURCE_MAX_INDEX   FROM_IMAGE_FILE_MODE

#define     CFG_COPY_CFG_FROM_MEMORY_EN   49
            // disable     0
            // enable      1

#define     CFG_RUNNING_SCRIPT_MODE    50
            #define     RUNNING_SCRIPT_NONE     0
            #define     RUNNING_SCRIPT_ONLINE   1
            #define     RUNNING_SCRIPT_OFFLINE  2

#define     CFG_VCC_SETTLE_TIME     51
            // Vss settle time in step 20 ms. Range 0...200 ( time 0...4000 ms)

#define     CFG_JTAG_UNLOCK_EN      52
            // disable     0
            // enable      1
```


Copyright © 2014–2016, Texas Instruments Incorporated

```
#define     CFG_SKIP_ERASE_IF_BLANK    53
            // disable     0
            // enable      1

#define     CFG_DO_NOT_OVERWRITE_OTP   54
            // disable     0
            // enable      1

#define     CFG_WRITE_OTP_STARTADDR    55
#define     CFG_WRITE_OTP_ENDADDR      56
#define     CFG_READ_OTP_STARTADDR     57
#define     CFG_READ_OTP_ENDADDR       58

#define     CFG_OTP_WRITE_EN           61
#define     CFG_READ_OTP_EN            62
#define     CFG_READ_MAIN_EN           63
#define     CFG_READ_RDONLY_EN         64

#define     CFG_ERASE_MAIN_EN          65

#define     CFG_RETAIN_DATA_EN         67
#define     CFG_RETAIN_START_ADDR      68
#define     CFG_RETAIN_END_ADDR        69
#define     RETAIN_DATA_MAX_SIZE       0x40

#define     CFG_CSM_PASSW_INDEX        70
#define     CSM_NONE_INDEX             0
#define     CSM_CODE_FILE_INDEX        1
#define     CSM_PASSWORD_FILE_INDEX    2
#define     CSM_DEFINED_INDEX          3
#define     CSM_MAX_INDEX              CSM_DEFINED_INDEX

#define     CFG_DSP_OSC_FREQ_KHZ       71
```


Copyright © 2014–2016, Texas Instruments Incorporated

4.1.33 C2000GANG_GetNameConfig, C2000GANG_SetNameConfig

Set or Get file names for code file, script file, password file or warning sounds.

Syntax

```
LPTSTR C2000GANG_GetNameConfig(LONG index)
```

Arguments

LONG index	See list of indexes below

Result

LPTSTR	File name

Syntax

```
LONG C2000GANG_SetNameConfig(LONG index, LPTSTR name)
```

Arguments

LONG index	See list of indexes below
LPTSTR file_name	

Result

LONG	Error code

```
#define CODEFILE_INDEX           0
#define SCRIPTFILE_INDEX         1
#define PASSWORDFILE_INDEX       2
#define SOUNDERRFILE_INDEX       3
#define SOUNDOKFILE_INDEX        4
#define SOUNDWARNINGFILE_INDEX   5
```


Copyright © 2014–2016, Texas Instruments Incorporated

4.1.34 C2000GANG_SetTmpGANG_Config

See the Set temporary configuration command (Section 3.5.3.3) for details.

Syntax

```
LONG C2000GANG_SetTmpGANG_Config(LONG no, LONG data)
```

Arguments

LONG no	Index list of indexes below
LONG data	

Result

LONG	Error code

```
//----- TMP_CFG_INDEX -----------
#define CFG_TMP_CLEAR 2
#define CFG_TMP_TASK_MASK 4
#define CFG_TMP_VCC_VALUE 6
#define CFG_TMP_POWER_VCC_EN 8
#define CFG_TMP_INTERFACE 10
#define CFG_TMP_GANG_MASK 12
#define CFG_TMP_VCC_ONOFF 14
#define CFG_LCD_CONTRAST 16
#define CFG_TMP_ICC_HI_EN 18
#define CFG_TMP_IO_INTERFACE 20
#define CFG_TMP_RESET 22
#define CFG_TMP_KEYBOARD_EN 24
#define CFG_TMP_VCC_SETTLE_TIME 26
```

4.1.35 C2000GANG_GetLabel

See the Get Label command (Section 3.5.2.9) for detailed LABEL information.

Syntax

```
LONG C2000GANG_GetLabel(BYTE *Data)
```

Arguments

BYTE *Data	Pointer to data buffer where the label is saved

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.36 C2000GANG_GetInfoMemory, C2000GANG_SetInfoMemory

Reads or writes 64 words to the internal Information memory. Information memory contains configuration data such as LCD contrast and USB port configuration, and it is not intended to be modified by the user. Use the GUI software to set the Information memory.

Syntax

```
LONG C2000GANG_GetOTPMemory(BYTE page, WORD *data)
LONG C2000GANG_SetOTPMemory(BYTE page, WORD *data)
```

Arguments

BYTE page	Page info 0 or 1
WORD *data	Pointer to or from data buffer

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.37 C2000GANG_Get_qty_MCU_Family, C2000GANG_Get_MCU_FamilyName, C2000GANG_Check_MCU_Name, C2000GANG_Get_MCU_Name

Set of functions that allows to get names of all supported MCU's, names of MCU groups and subgroups.

Syntax

```
LONG C2000GANG_Get_qty_MCU_Family(void)
LONG C2000GANG_Get_MCU_FamilyName(LONG index, LPTSTR name)
LONG C2000GANG_Check_MCU_Name(LPTSTR name)
LONG C2000GANG_Get_MCU_Name(LONG group_index, LONG index, LPTSTR name)
```

Use these functions in the following order:

```
typedef struct
{
    int no;
    char name[24];
} MCU_FAMILY;

MCU_FAMILY MCU_family_list[30];

typedef struct
{
    int index;
    char name[24];
} MCU_NAME;

MCU_NAME MCU_name_list[100];

n = C2000GANG_Get_qty_MCU_Family(); //get no of MCU groups
  for(k=0; k<n; k++)
  {
    P = C2000GANG_Get_MCU_FamilyName(k, MCU_family_list[k].name);
    If(p == 0) break;
    MCU_family_list[k].no = p;
  }
```

Currently following names and numbers should be received using above functions:

{ F28FIXED, " F28xx Fixed-point" },
{ F28PICCOLO, " F28xx Piccolo" },
{ F28DELFINO, " F28xx Delfino" },

List of the MCU names in selected group can be taken as follows (as an example - list of the MCUs from the F28xx group (group number F28FIXED, defined in DLL header file)):

```
for(n = 0; n< 100; n++) MCU_name_list[n].index = 0;
for(n = 0; n< 100; n++)
{
  p = C2000GANG_Get_MCU_Name(50, n, MCU_name_list[n].name);
  if(p == 0) break;
  MCU_name_list[n].index = n;
}
```


Copyright © 2014–2016, Texas Instruments Incorporated

4.1.38 C2000GANG_Set_MCU_Name

The C2000GANG_Set_MCU_Name allows to select desired target MCU.

Syntax

```
LONG C2000GANG_Set_MCU_Name(LPTSTR name);
```

Arguments

LPTSTR MCU_name	MCU name, the same as it is listed in the GUI software

Result

LONG	Error code

4.1.39 C2000GANG_HW_devices

The C2000GANG_HW_devices function scanning all available COM ports and saving information about these ports in following structure.

```
#define MAX_COM_SIZE 60
#define HW_NAME_SIZE 30
typedef union
{
   unsigned char bytes[HW_NAME_SIZE];
   struct
   {
      unsigned short ComNo;
      char ComName[7];
      char description[HW_NAME_SIZE-2-7];
   }x;
}COM_PORTS_DEF;
COM_PORTS_DEF *AvailableComPorts = NULL;
C2000GANG_HW_devices(MAX_COM_SIZE, (void **) &AvailableComPorts));
```

If detected, USB VCP information is placed at the first location.

Syntax

```
LONG C2000GANG_HW_devices(LONG max, void **AvailableComPorts)
```

Arguments

LONG max	
void **AvailableComPorts	

Result

LONG	Error code

Copyright © 2014–2016, Texas Instruments Incorporated

4.1.40 C2000GANG_GetProgressStatus

C2000GANG_GetProgressStatus gets progress status from C2000 Gang Programmer. The data received contains a Gang Mask of all processes done in the previous function. Each bit in the Gang mask represents one targeted device:

bit 0 → Target 1, bit 1 → Target 2, ... bit 7 → Target 8

For example, when connected_gang_mask is 0x7A, then targets 2, 4, 5, 6, and 7 are detected, and communication with these targets is established. The cumulative mask contains the final result for all targets.

Syntax

```
LONG C2000GANG_GetProgressStatus(void *lpData)
```

Arguments

void *lpData	Pointer to structure below

Result

LONG	Error code

```
#define  SCRIPT_TEXT_SIZE  16
union GANG_PROGRESS_STATUS
{
  BYTE       bytes[PROGRESS_STATUS_SIZE+4];
  struct
  {
   BYTE    header;
   BYTE     ctr;
   WORD    task_ctr;                  //byte offset - 0
   WORD     chunk_ctr;                 //byte offset - 2
   BYTE     run;                     //byte offset - 4
   BYTE     ack;                     //byte offset - 5
   WORD     Finished_tasks_mask;       //byte offset - 6,7
         //---  task mask bits ----
         // CONNECT_TASK_BIT     0x0001
         // ERASE_TASK_BIT       0x0002
         // BLANKCHECK_TASK_BIT  0x0004
         // PROGRAM_TASK_BIT     0x0008
         // VERIFY_TASK_BIT       0x0010
         // SECURE_TASK_BIT      0x0020
         // FREQ_TEST_TASK_BIT   0x0040
         //spare                    0x0080 to 0x4000
         // RST_AND_START_FW_BIT 0x8000

   BYTE     cumulative;                //byte offset - 8
         //target masks
         // TARGET_1_MASK     0x01
         // TARGET_2_MASK     0x02
         // TARGET_3_MASK     0x04
         // TARGET_4_MASK     0x08
         // TARGET_5_MASK     0x10
         // TARGET_6_MASK     0x20
         // TARGET_7_MASK     0x40
         // TARGET_8_MASK     0x80
   BYTE     Rq_gang_mask;              //byte offset - 9
   BYTE     Connected_gang_mask;       //byte offset - 10
   BYTE     Erased_gang_mask;          //byte offset - 11
   BYTE     BlankCheck_gang_mask;         //byte offset - 12
   BYTE     Programmed_gang_mask;         //byte offset - 13
   BYTE     Verified_gang_mask;           //byte offset - 14
   BYTE     Secured_gang_mask;            //byte offset - 15
   BYTE     spare[6];                       //byte offset - 16..21
```


Copyright © 2014–2016, Texas Instruments Incorporated

```
    BYTE     error_no;                       //byte offset - 22
    BYTE     VTIO_32mV;                      //byte offset - 23
    BYTE     VccSt_LOW;                      //byte offset - 24
    BYTE     VccSt_HI;                       //byte offset - 25
          // VccSt_LOW, VccSt_HI provide 2 bits to each target.
          // Bit A for each target and bit B for each target.
          //    Bits    B   A
           //             0 0     Vcc below 0.7V
          //          0   1       Vcc below Vcc min   ( 0.7V < Vcc < Vcc min)
          //          1   0    Vcc over Vcc min (OK status)
          //          1   1    Vcc over 3.8V
    BYTE     VccErr;                         //byte offset - 26
          //current Vcc below min
    BYTE     VccErr_Cumulative;              //byte offset - 27
          //Cumulative (during programing) Vcc below min
    BYTE     JTAG_init_err_mask;           //byte offset - 28
    BYTE     JTAG_Fuse_already_blown_mask; //byte offset - 29
    BYTE     Wrong_MCU_ID_mask;            //byte offset - 30
    BYTE     Progress_bar;              //byte offset - 31
          // 0...100%
    char     comment[SCRIPT_TEXT_SIZE];    //byte offset - 32..47
   }st;
};
```


Copyright © 2014–2016, Texas Instruments Incorporated

4.1.41 C2000GANG_GetAPIStatus

C2000GANG_GetAPIStatus gets the selected status or results from the C2000 Gang Programmer. The following numbers (no) are available:

```
GET_APP_FLAGS      10
GET_LAST_STATUS    12
GET_LAST_ERROR_NO  14
```

Syntax

```
LONG C2000GANG_GetAPIStatus (LONG no, BYTE *data)
```

Arguments

LONG no	Status type
BYTE *data	Pointer to status results. See below.

Result

LONG	Error code

```
no = GET_APP_FLAGS (10)
response:
   Byte-0
       b0 (LSB)  Hardware rev-0
       b1        initialization finished (after power-up)
       b2        access key CRC error
       b3        invalid access key
       b4        running from SD card
       b5        File in SD card found
       b6        target secure device in process
       b7        keypad enabled
   Byte-1
       b0        key pressed
       b1..b7    spare
   Byte-2        spare
   Byte-3        spare

no = GET_LAST_STATUS (12)
response:
   Byte-0        Error number in the last execute transaction
   Byte-1        targets connection mask
   Byte-2        active targets mask
   Byte-3        targets error mask
   Byte-4..7     spare

no = GET_LAST_ERROR_NO (14)
   Byte-0        last error number from C2000 Gang Programmer for any command
                 error numbers 1...255 - see error list numbers
```


Copyright © 2014–2016, Texas Instruments Incorporated

4.1.42 C2000GANG_Set_IO_State

The C2000GANG_Set_IO_State modifies the static levels on the I/O pins (JTAG lines). The JTAG lines can be set to the desired level (low or high) or they can be high impedance. The state and the level can be the same on all outputs. The level on one selected line (RST, TDI, TDOI, TMS or BSL-RX) can be different for each target.

Syntax

```
LONG  C2000GANG_Set_IO_State(long  Vcc_mV,  BYTE * data );
```

Arguments

Vcc_mV	Voltage level in mV on the target's V_{CC}
data[0]	Open destination buffer for output and transferred data for each targets. 0 = None 1 = TDI (target1 to target8) 2 = TDOI (target1 to target8) 3 = TMS (target1 to target8) 4 = RST (target1 to target8) 5 = BSL-RX (target1 to target8)
data[1]	Data transferred to the buffer above. b0 to b7 - target1 to target8
data[2]	Output enable bits: 0 = high impedance, 1 = output b2 (0x04) - common RST - the same state for all 8 targets (Note: if the RST buffer above is selected, then this state is ignored) b3 (0x08) - common TEST - the same state for all 8 targets b4 (0x10) - common TCK - the same state for all 8 targets b5 (0x20) - common TMS - the same state for all 8 targets (Note: if the TMS buffer above is selected, then this state is ignored)
data[3]	Output level on all targets: 0 = LOW, 1 = HIGH b2 (0x04) - common RST - the same level for all eight targets (Note: if the RST buffer above is selected, then this state is ignored) b3 (0x08) - common TEST - the same level for all eight targets b4 (0x10) - common TCK - the same level for all eight targets b5 (0x20) - common TMS - the same level for all eight targets (Note: if the TMS buffer above is selected, then this state is ignored)
data[4]	V_{CC} enable bits to each target b0 to b7 - target1 to target8
data[5]	I_{CC} HI enable: 0 = disable, 1 = enable
data[6]	spare
data[7]	spare

Example 1

Generate a short RST pulse on target 1 only and force RST level LOW on targets 2 to 5 and RST level HIGH on targets 6 and 7. V_{CC} on targets 1 to 7 is 3.3 V (0x0CE4) and on target 8 is 0 V (disabled).

```
WORD data[4] = { 0460 0000 7F00 0000 };
C2000GANG_Set_IO_State( 3300, data );
```

then

```
data[0] = 0x0461;
C2000GANG_Set_IO_State( 3300, data );
```


Copyright © 2014–2016, Texas Instruments Incorporated

Example 2

Generate a short RST pulse on all targets. V_{CC} on targets 1 to 7 is 3.3 V (0x0CE4) and on target 8 is 0 V (disabled).

```
WORD data[4] = { 0000 0400 7F00 0000 };
C2000GANG_Set_IO_State( 3300, data );
```

then

```
data[2] = 0x0400;
C2000GANG_Set_IO_State( 3300, data );
```


Copyright © 2014–2016, Texas Instruments Incorporated

Schematics

The following pages show the schematics for the C2000 Gang Programmer.

Copyright © 2014–2016, Texas Instruments Incorporated

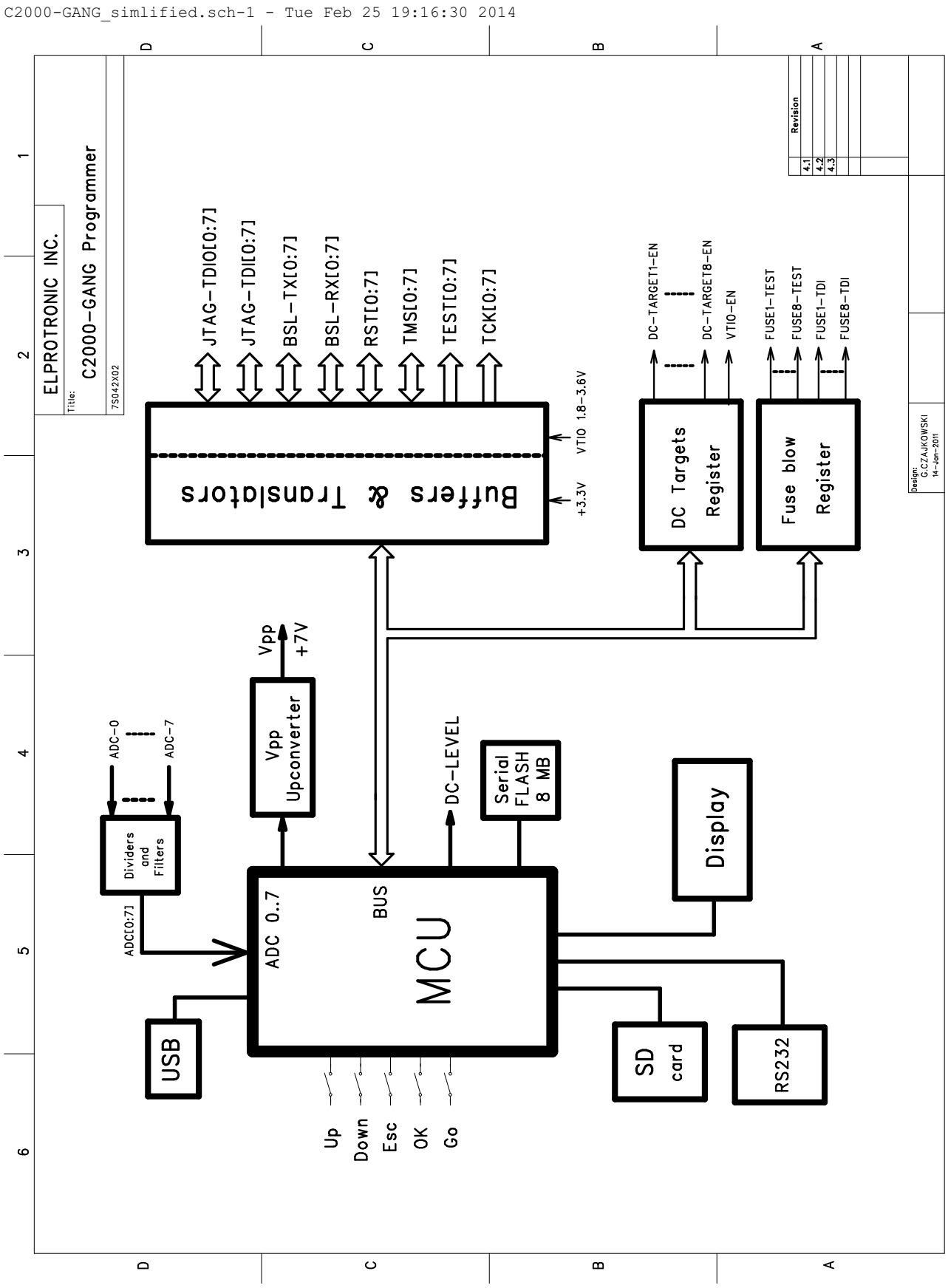


Figure 5-1. C2000 Gang Programmer Simplified Schematic (1 of 3)

Copyright © 2014–2016, Texas Instruments Incorporated

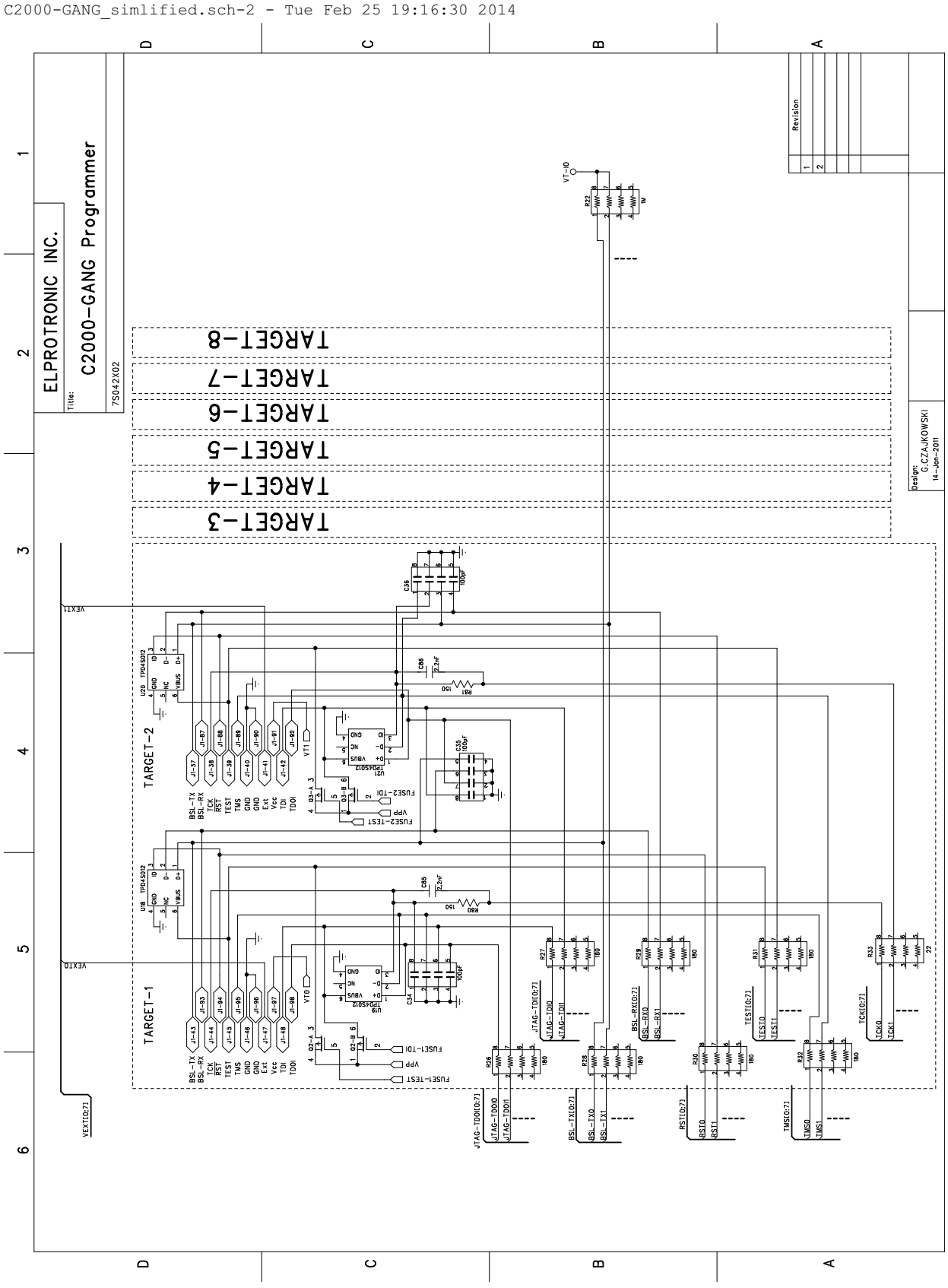


Figure 5-2. C2000 Gang Programmer Simplified Schematic (2 of 3)

Copyright © 2014–2016, Texas Instruments Incorporated

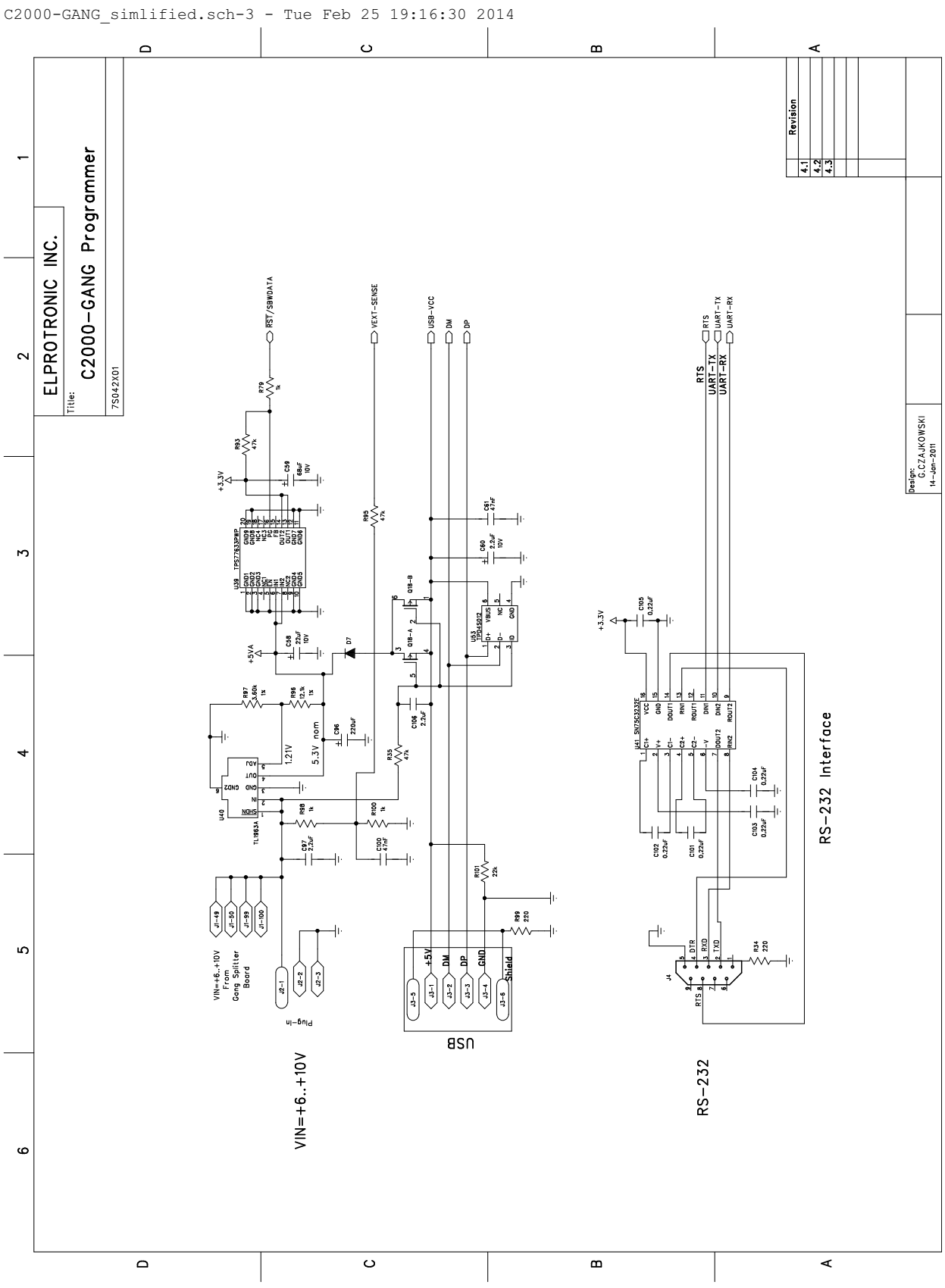


Figure 5-3. C2000 Gang Programmer Simplified Schematic (3 of 3)

Copyright © 2014–2016, Texas Instruments Incorporated

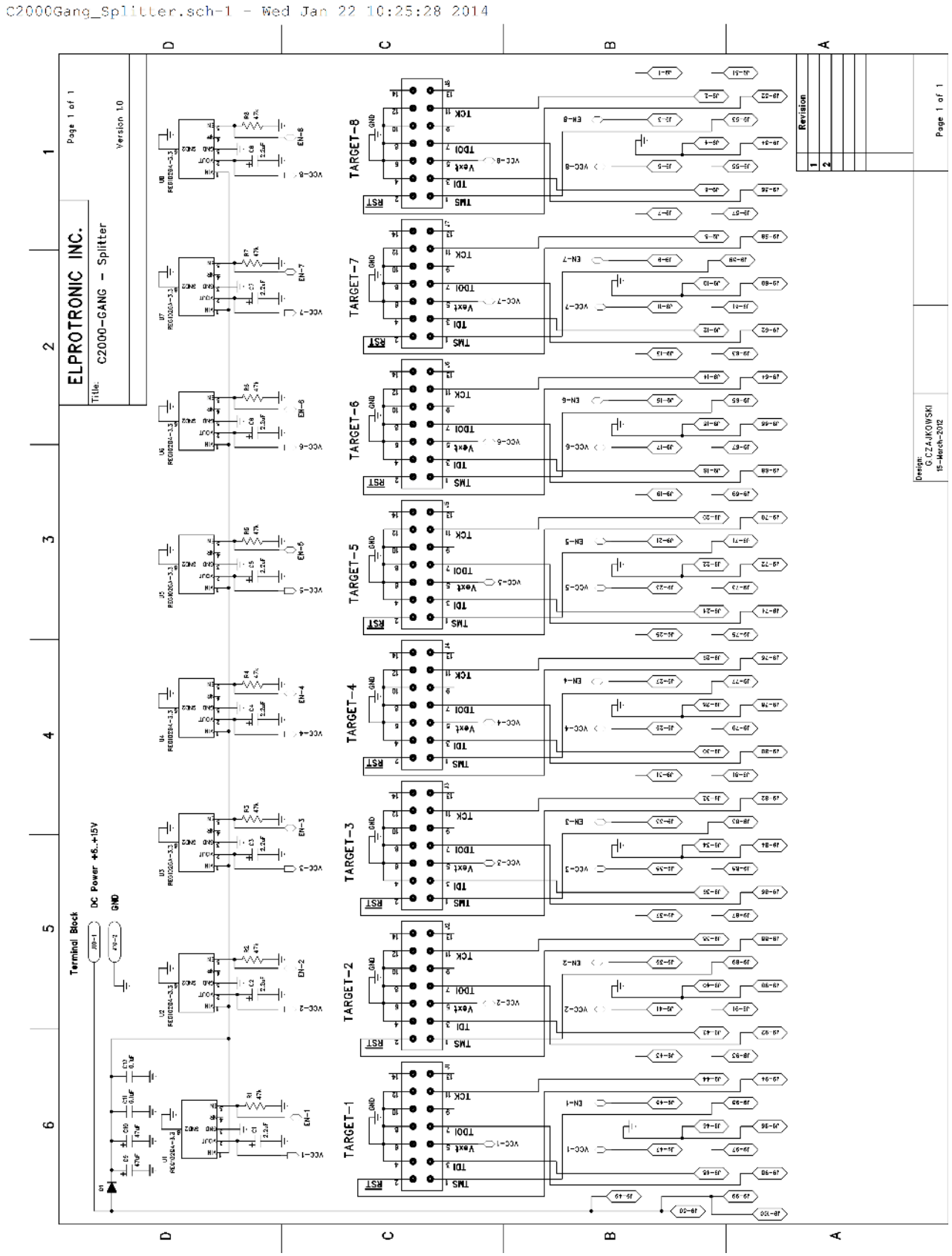


Figure 5-4. C2000 Gang Splitter rev. 0 Schematic

Copyright © 2014–2016, Texas Instruments Incorporated

Table 5-1. Gang Splitter rev. 0 Bill of Materials (BOM)

Item	Name	Drawing and Part Number	Qt.	Description
1	BLANK PC BOARD	C2000-GANG-SP rev-0	1	Blank PC Board
THROUGH HOLE COMPONENTS				
1,2,3,4,5,6,7,8	Connector	35-514-0	8	14-pins Header Connector (Mode Electronics / Electrosonic)
J9	Connector	TX24-100R-LT-H1E	1	100p-Receptical Right Angle Connector (JAE Electronics)
	Bumpers	SJ61A6	3	Bumpon cylindrical 0.312 x 0.215, black
SMT COMPONENTS				
C1,C2,C3,C4,C5,C6,C7,C8	Tantalum Capacitor	TAJA225K016R	8	Cap Tan Chip 2.2uF 16V 10% Size-A (AVX or eq.)
C9,C10	Tantalum Capacitor	TAJC476K016R	2	Cap Tan Chip 47uF 16V 10% Size-C (AVX or eq.)
C11,C12	Capacitor	0603YC224KAT2A	2	Cap Cer Chip 0.22uF X7R 10% 16V 0603 (AVX or eq.)
D1	Diode	MBRS130LT3 or B130B-13-F	1	Schottky Diode 2A/30V SMB (On Semi / Diodes Inc)
R1,R2,R3,R4,R5,R6,R7,R8	Resistor	MCR03 EZP J 473	8	SMD Chip Res 47 kOhm 5% 1/10W 0603
U1,U2,U3,U4,U5,U6,U7,U8	IC	REG102GA-3.3	8	LDO - 3.3V (TI)

Copyright © 2014–2016, Texas Instruments Incorporated

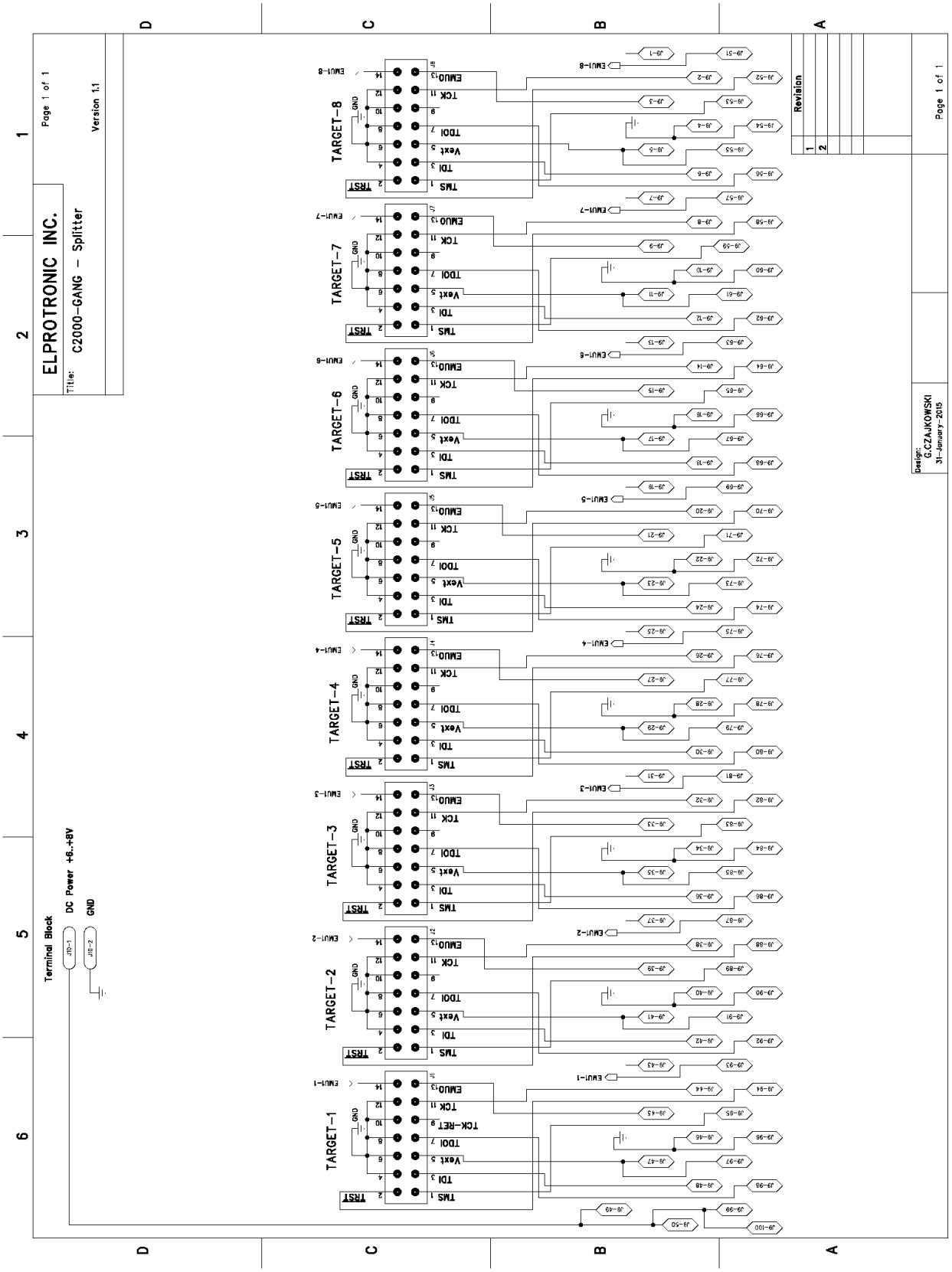


Figure 5-5. C2000 Gang Splitter rev.1 Schematic

Copyright © 2014–2016, Texas Instruments Incorporated

Table 5-2. Gang Splitter rev. 1 Bill of Materials (BOM)

Item	Name	Drawing and Part Number	Qt.	Description
1	BLANK PC BOARD	C2000-GANG-SP rev-1	1	Blank PC Board
1,2,3,4,5,6,7,8	Connector	SBH11-PBPC-D07-ST-BK	8	14-pins Header Connector (Sullins)
J9	Connector	TX24-100R-LT-H1E	1	100p-Receptical Right Angle Connector (JAE Electronics)
	Bumpers	SJ61A6	3	Bumpon cylindrical 0.312 x 0.215, black

Supported MCU List

6.1 F28x Fixed Point MCU

TMS320F2801, TMS320F28015, TMS320F28016, TMS320F2802, TMS320F28044, TMS320F2806, TMS320F2808, TMS320F2809, TMS320F2810, TMS320F2811, TMS320F2812, TMS320F28232, TMS320F28234, TMS320F28235

6.2 Piccolo™ F280x

TMS320F28022, TMS320F28023, TMS320F28024, TMS320F28025, TMS320F28026, TMS320F28027, TMS320F28035, TMS320F28032, TMS320F28033, TMS320F28034, TMS320F28030, TMS320F28031, TMS320F28021, TMS320F28020, TMS320F280200, TMS320F28062, TMS320F28062U, TMS320F28063, TMS320F28063U, TMS320F28064, TMS320F28064U, TMS320F28065, TMS320F28065U, TMS320F28066, TMS320F28066U, TMS320F28067, TMS320F28067U, TMS320F28068, TMS320F28068U, TMS320F28069, TMS320F28069U, TMS320F28050, TMS320F28051, TMS320F28052, TMS320F28052F, TMS320F28052M, TMS320F28053, TMS320F28054, TMS320F28054F, TMS320F28054M, TMS320F28055, TMS320F280220, TMS320F280230, TMS320F280260, TMS320F280270, TMS320F28075, TMS320F28074

6.3 Delfino F283xx

TMS320F28332, TMS320F28334, TMS320F28335, TMS320F28377D, TMS320F28376D, TMS320F28375D, TMS320F28374D, TMS320F28376S, TMS320F28374S

6.4 C28x + ARM®

F28M35H52C, F28M35H22C, F28M35M52C, F28M35M22C, F28M35M20B, F28M35E20B, F28M36P63C, F28M36P53C, F28M36H53C, F28M36H53B, F28M36H33C, F28M36H33B

Copyright © 2014–2016, Texas Instruments Incorporated

Revision History

Changes from B Revision (November 2015) to C Revision **Page**

- Added the text before "TRACEOFF - Disable Tracing". 23
- Added Section 2.1.3.4 *Commands Combined With the Executable File* 25

Copyright © 2014–2016, Texas Instruments Incorporated

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

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

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

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2016, Texas Instruments Incorporated