

# MSP Gang Programmer (MSP-GANG)

## User's Guide



Literature Number: SLAU358  
September 2011



<b>Preface</b> .....	<b>7</b>
<b>1 Introduction</b> .....	<b>9</b>
1.1 Software Installation .....	10
1.2 Driver Installation .....	11
1.3 Hardware Installation .....	11
<b>2 Operation</b> .....	<b>13</b>
2.1 Programming MSP430 Flash Devices Using the MSP Gang Programmer .....	13
2.1.1 Programming Using Interactive Mode .....	14
2.1.2 Programming From Image .....	18
2.1.3 Programming From Script .....	21
2.1.4 Programming in Standalone Mode .....	25
2.1.5 Memory Setup for GO, Erase, Program, Verify, and Read .....	28
2.1.6 Creating and Using Images .....	29
2.2 Data Viewers .....	32
2.3 Status Messages .....	34
2.4 Self Test .....	37
2.5 Label .....	42
2.6 Benchmark .....	43
<b>3 Firmware</b> .....	<b>45</b>
3.1 Commands .....	45
3.2 Firmware Interface Protocol .....	46
3.3 Synchronization Sequence .....	46
3.4 Command Messages .....	46
3.4.1 Frame Structure .....	46
3.4.2 Checksum .....	48
3.5 Detailed Description of Commands .....	49
3.5.1 General .....	49
3.5.2 Commands Supported by the BOOT Loader .....	49
3.5.3 Commands Supported by Application Firmware .....	52
3.5.4 API Firmware Commands That Should Not be Used .....	56
<b>4 Dynamic Link Library for MSP-GANG Programmer</b> .....	<b>59</b>
4.1 Gang430.dll Wrapper Description .....	59
4.2 MSP-GANG.dll Description .....	59
4.2.1 MSPGANG_GetDataBuffers_ptr .....	60
4.2.2 MSPGANG_SetGangBuffer, MSPGANG_GetGangBuffer .....	61
4.2.3 MSPGANG_GetDevice .....	62
4.2.4 MSPGANG_LoadFirmware .....	63
4.2.5 MSPGANG_InitCom .....	63
4.2.6 MSPGANG_ReleaseCom .....	63
4.2.7 MSPGANG_GetErrorString .....	64
4.2.8 MSPGANG_SelectBaudrate .....	64
4.2.9 MSPGANG_GetDiagnostic .....	64
4.2.10 MSPGANG_MainProcess .....	65
4.2.11 MSPGANG_InteractiveProcess .....	65

4.2.12	MSPGANG_Interactive_Open_Target_Device .....	65
4.2.13	MSPGANG_Interactive_Close_Target_Device .....	66
4.2.14	MSPGANG_Interactive_DefReadTargets .....	66
4.2.15	MSPGANG_Interactive_ReadTargets .....	66
4.2.16	MSPGANG_Interactive_ReadBytes .....	67
4.2.17	MSPGANG_Interactive_WriteWord_to_RAM .....	67
4.2.18	MSPGANG_Interactive_WriteByte_to_RAM .....	68
4.2.19	MSPGANG_Interactive_WriteBytes_to_RAM .....	68
4.2.20	MSPGANG_Interactive_WriteBytes_to_FLASH .....	69
4.2.21	MSPGANG_Interactive_Copy_Gang_Buffer_to_RAM .....	69
4.2.22	MSPGANG_Interactive_Copy_Gang_Buffer_to_FLASH .....	70
4.2.23	MSPGANG_Interactive_EraseSectors .....	70
4.2.24	MSPGANG_Interactive_BlankCheck .....	71
4.2.25	MSPGANG_Interactive_DCO_Test .....	71
4.2.26	MSPGANG_SelectImage .....	72
4.2.27	MSPGANG_EraseImage .....	72
4.2.28	MSPGANG_CreateGangImage .....	72
4.2.29	MSPGANG_LoadImageBlock .....	73
4.2.30	MSPGANG_VerifyPSImageBlock .....	73
4.2.31	MSPGANG_ReadImageBlock .....	73
4.2.32	MSPGANG_Read_Code_File .....	74
4.2.33	MSPGANG_Save_Config, MSPGANG_Load_Config, MSPGANG_Default_Config .....	74
4.2.34	MSPGANG_SetConfig, MSPGANG_GetConfig .....	75
4.2.35	MSPGANG_GetNameConfig, MSPGANG_SetNameConfig .....	78
4.2.36	MSPGANG_SetTmpGANG_Config .....	79
4.2.37	MSPGANG_GetLabel .....	79
4.2.38	MSPGANG_GetInfoMemory, MSPGANG_SetInfoMemory .....	80
4.2.39	MSPGANG_Get_qty_MCU_Family, MSPGANG_Get_MCU_FamilyName, MSPGANG_Check_MCU_Name, MSPGANG_Get_MCU_Name .....	81
4.2.40	MSPGANG_Set_MCU_Name .....	82
4.2.41	MSPGANG_HW_devices .....	82
4.2.42	MSPGANG_GetProgressStatus .....	83
4.2.43	MSPGANG_GetAPIStatus .....	85
<b>5</b>	<b>Schematics .....</b>	<b>87</b>
	<b>Revision History .....</b>	<b>90</b>

## List of Figures

1-1.	Top View of the MSP Gang Programmer .....	10
2-1.	Main MSP Gang Programmer Dialog GUI, Interactive Mode.....	14
2-2.	Memory Options .....	15
2-3.	Reset Options .....	16
2-4.	Flash Memory Data .....	18
2-5.	Main MSP Gang Programmer Dialog GUI, From Image Mode .....	19
2-6.	Main MSP Gang Programmer Dialog GUI, From Image Mode and Custom Configuration Enabled .....	20
2-7.	Main MSP Gang Programmer Dialog GUI, From Script.....	21
2-8.	Main MSP Gang Programmer Dialog GUI, Standalone Mode .....	25
2-9.	Image Option.....	26
2-10.	Target Enable/Disable Option .....	27
2-11.	Memory Options, BSL Sectors Selected .....	29
2-12.	Image Name Configuration Screen .....	30
2-13.	Image File Security Options.....	31
2-14.	Hardware Fingerprint of Computer in Use .....	31
2-15.	Code File Data .....	32
2-16.	Comparison of Code and Flash Memory Data of the Target Processor .....	33
2-17.	Self Test .....	38
2-18.	Information About the MSP Gang Programmer .....	42
5-1.	Gang Splitter Schematic.....	88

## List of Tables

2-1.	Benchmark Results.....	43
3-1.	Data Frame for Firmware Commands .....	47



## ***Read This First***

---

---

---

### **If You Need Assistance**

If you have any feedback or questions, support for the MSP430 devices and the MSP-GANG is provided by the Texas Instruments Product Information Center (PIC) and the TI E2E Forum (<https://community.ti.com/forums/12.aspx>). Contact information for the PIC can be found on the TI web site at [support.ti.com](http://support.ti.com). Additional device-specific information can be found on the MSP430 web site at [www.ti.com/msp430](http://www.ti.com/msp430).

### **Related Documentation from Texas Instruments**

The primary sources of MSP430 information are the device-specific data sheets and user's guides. The most current information is found at [www.ti.com/msp430](http://www.ti.com/msp430).

Information specific to the MSP-GANG can be found at <http://focus.ti.com/docs/toolsw/folders/print/msp-gang.html>.



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



## Introduction

---

---

---



The MSP Gang Programmer is an MSP430 device programmer that can program up to eight identical MSP430 flash or FRAM devices at the same time. The MSP 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 MSP Gang Programmer can be seen in [Figure 1-1](#).

The MSP 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 MSP Gang Programmer is designed to connect to target devices in-circuit (that is, target devices are mounted in the final circuit or system). The MSP Gang Programmer accesses target devices using connectors that use JTAG or Spy-Bi-Wire (SBW) signals.

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

[Chapter 2](#) describes in detail how to use the MSP 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 4](#) describes Gang430.dll and MSP-GANG.dll, including the functions available through them.

[Chapter 5](#) contains an I/O schematic that shows how signals from the MSP Gang Programmer can be brought out to each of the target devices via an MSP430-standard JTAG or Spy-Bi-Wire connector. The circuit can be easily modified to connect signals to the target device pins directly (via a socket) if a traditional gang programmer setup is desired.

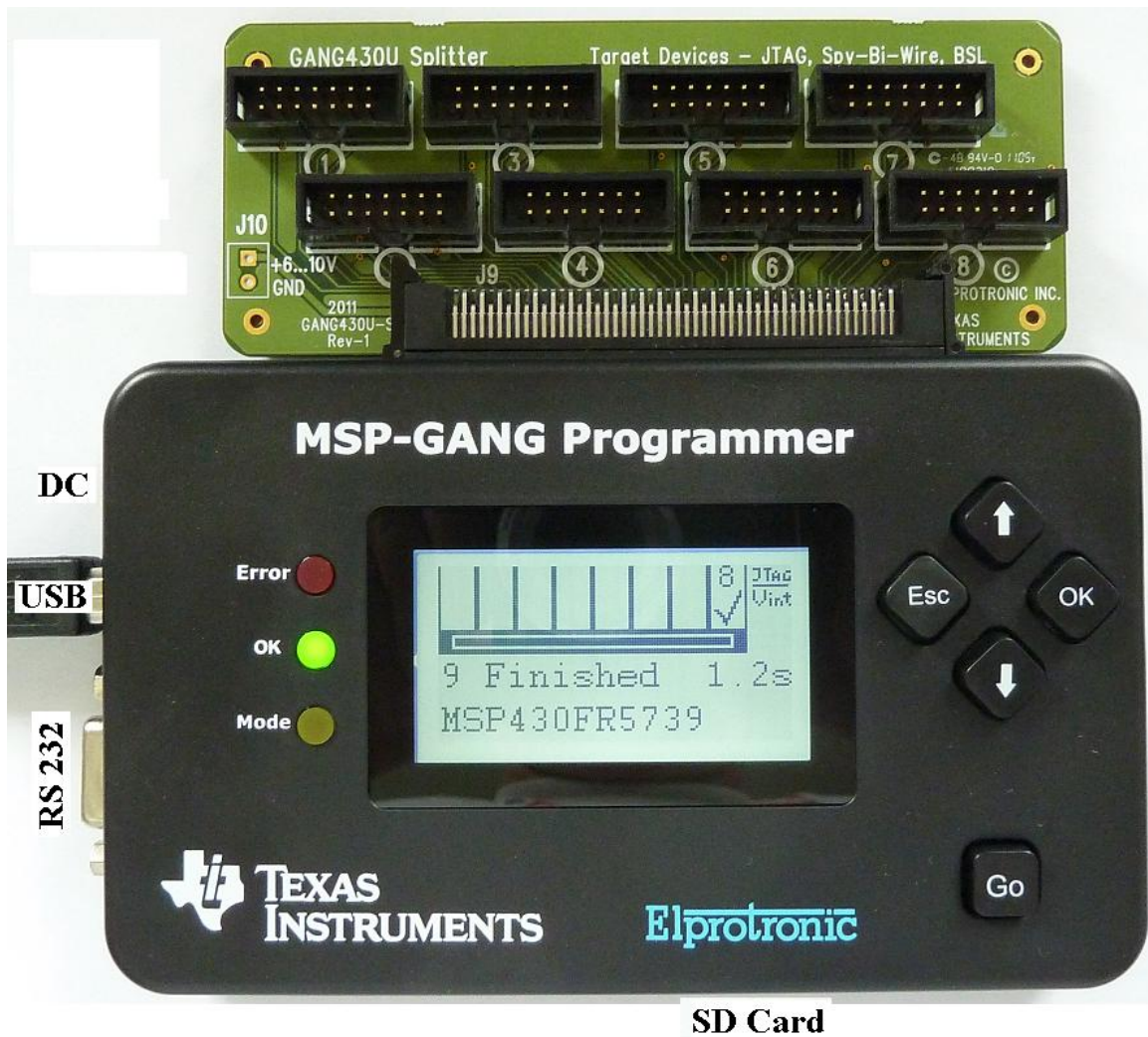


Figure 1-1. Top View of the MSP Gang Programmer

## 1.1 Software Installation

It is highly recommend that you use the latest software version, which can be downloaded from the MSP430 website at [www.ti.com/msp-gang](http://www.ti.com/msp-gang).

To install MSP Gang Programmer software:

1. Insert the MSP430 CD-ROM into the CD-ROM drive of the host computer. Setup automatically opens the default browser and displays the MSP430 start page.  
If the start page does not open automatically, run the file setup.exe located in the root directory of the CD-ROM using a web browser. The MSP430 start page is then displayed in a browser window.
2. Follow the instructions in the installation process.
3. When the setup program is complete, MSP Gang Programmer icons are created in the selected folder. Click on the MSP Gang Programmer Read Me First icon to obtain important information about the MSP Gang Programmer .
4. A program group and icons are added to the Windows desktop.
5. To start the MSP Gang Programmer software, click the newly created icon.

## 1.2 Driver Installation

To install the required drivers:

1. Connect the MSP-GANG programmer to PC USB port. When the Windows Wizard starts then follow instruction provided by wizard. When Wizard ask for the USB driver location then browse the D: location where D: represents the CD-ROM location. Drivers are located in the main CD-ROM directory location and also in the following directory:  
C:\Program Files\Texas Instruments\MSP-GANG\Driver
2. If the RS-232 interface is used for communication with MSP-GANG, then the additional driver is not required. Check in the Device Manager the COM-port number that should be used with communication via RS-232.

## 1.3 Hardware Installation

To install the MSP Gang Programmer hardware:

1. Attach the expansion board (Gang Splitter) to the 100-pin connector on the MSP Gang Programmer.  
The expansion board provides connectivity for up to eight targets using the included 14-pin cables. The target MSP430 flash devices can be in standalone sockets or can be on an application's PCB. These devices can be accessed via JTAG or Spy-Bi-Wire (SBW) signals.
2. Connect the MSP Gang Programmer hardware to a computer's USB port using a USB A-B cable.  
The programmer can be fully supplied from the computer's USB port (5 V, 0.5 A). 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 MSP Gang Programmer if it is not connected via USB port or if the total current consumption of the programmed target devices exceeds 0.3 A.

---

**NOTE: External Power Supply**

If an external power supply is used then it must provide a voltage between 6 V and 10 V dc and must be capable of providing a minimum current of 800 mA. The center post of the power supply connector on the MSP 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.

---

---

**NOTE: Maximum Signal Path Length: 50 cm**

The maximum length of a signal path between the 14-pin JTAG or SBW connector on the Gang Splitter and the target device is 50 cm. If cables between the programmer and targets are longer than 20 cm, serial resistors of 100  $\Omega$  should be inserted on signals TCK and TMS between Gang Splitters and JTAG or SBW flat cables.

---

4. The MSP Gang Programmer can supply power at a specified voltage  $V_{CC}$  to each target device separately (pin 2 on each 14-pin JTAG or SBW cable). The maximum current for each target device is programmable and can be 30 mA or 50 mA. If the higher current limit is selected (50 mA) and eight target devices are connected, then the total current taken by all devices can reach up to 400 mA. In this case, the MSP Gang Programmer must be supplied from an external power supply instead of from the USB port. This is because the total current drawn from the USB port cannot exceed 0.5 A, and 150 mA are consumed by the MSP Gang Programmer itself, leaving 350 mA for the target devices.

**CAUTION**

When an external power supply is used to power target devices, it is important to disconnect  $V_{CC}$  (pin 2 on the JTAG/SBW connector) from the targets to avoid power-supply conflicts that could potentially damage the MSP Gang Programmer and the target devices.

When target devices are powered from an external power supply, the  $V_{CC}$  from the target device should be connected to  $V_{extin}$  (pin 4) on the JTAG or SBW connectors. This voltage is used by the MSP Gang Programmer for sensing the presence of an external power supply.

Also, the same voltage value that is used for powering the target device should be set in the MSP Gang Programmer as the desired  $V_{CC}$  level. This information is mandatory to provide correct I/O levels for the TMS, TCK, TDI, TDO, RST, etc. signals. If the wrong  $V_{CC}$  is provided, then the I/O levels between the programmer and target devices can be too low or too high, and communication can become unreliable.

5. The MSP 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 MSP Gang Programmer.

## Operation

---

---

---

This chapter describes how to use the MSP 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 MSP Gang Programmer hardware and software as described in [Chapter 1](#).

### 2.1 Programming MSP430 Flash Devices Using the MSP Gang Programmer

The MSP Gang Programmer is capable of quickly and reliably programming MSP430 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 that 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 MSP 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 MSP Gang Programmer. Creating images is described in [Section 2.1.6](#).

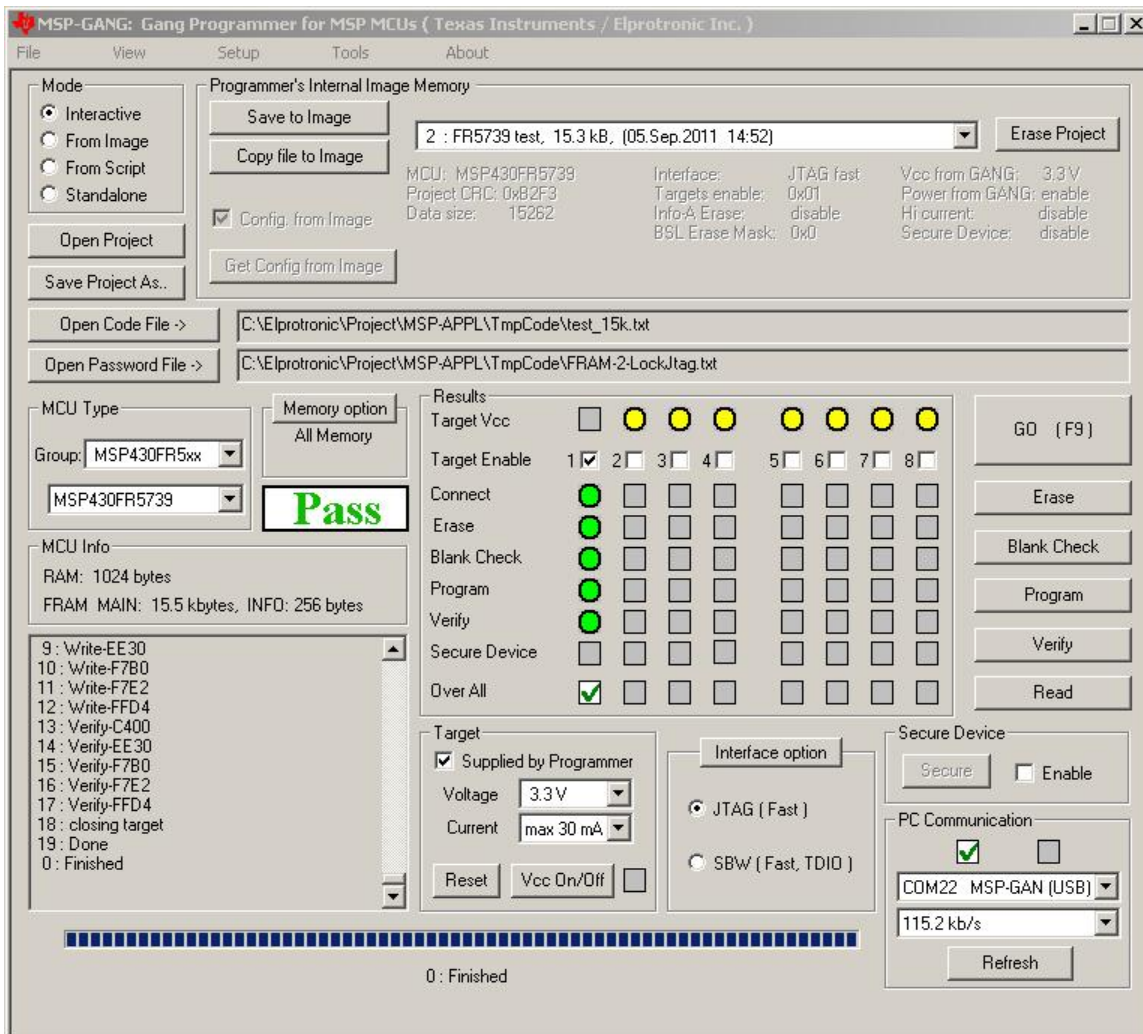
The following subsections describe how to use these modes of operation.



### 2.1.1 Programming Using Interactive Mode

Use the following sequence to start the MSP Gang Programmer GUI and program MSP430 Flash Devices using the Interactive Mode:

1. Click on the MSP Gang Programmer icon located in the program group specified during installation. **Figure 2-1** shows the MSP Gang Programmer GUI in the Interactive Mode (see top left corner). This window is used to select the target processor, 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.

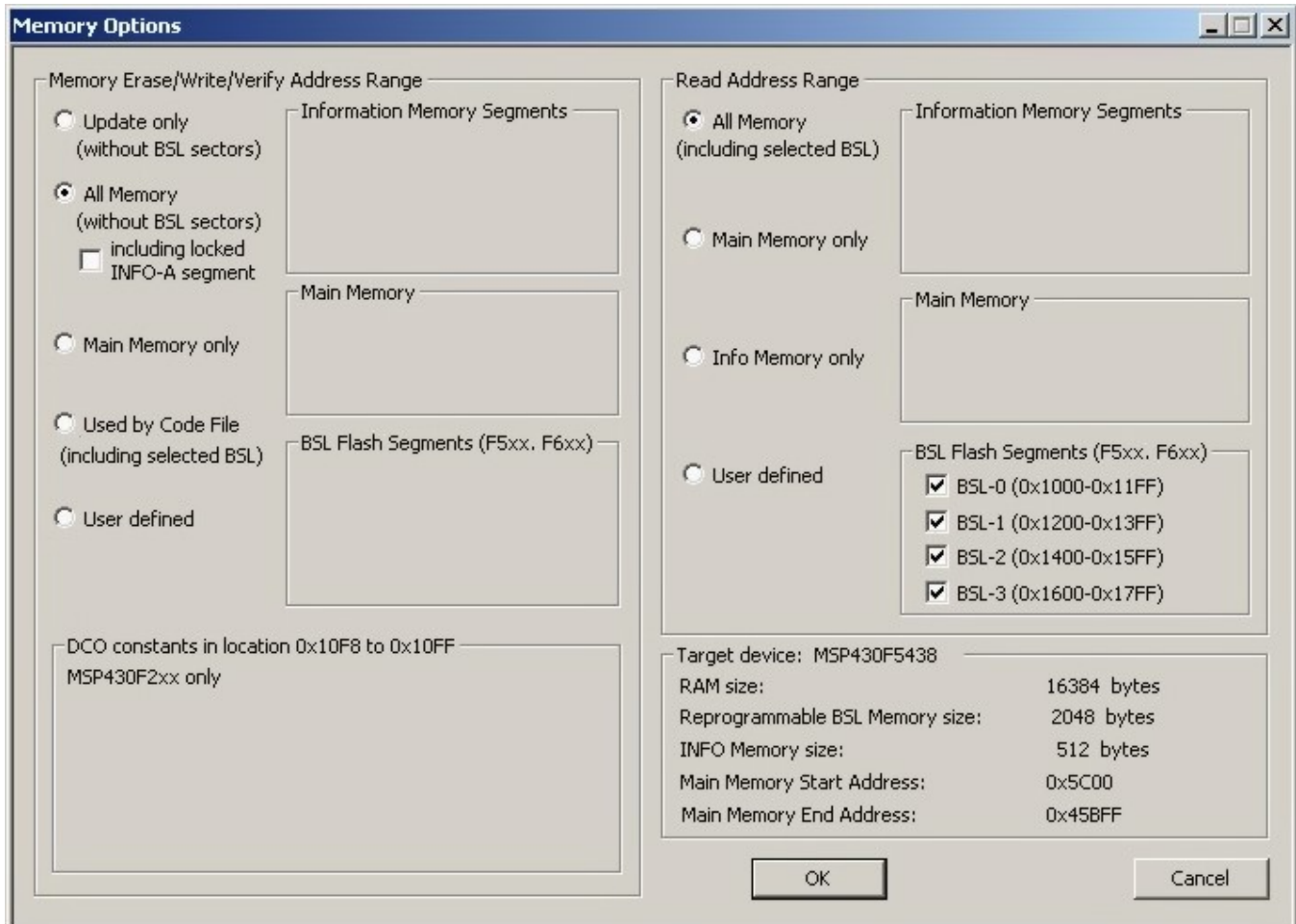


NOTE: This figure shows the Interactive Mode (see the Mode section in the top left corner). The user can select the target processor, source file for programming, power supply options, communication interface, and more. This window also shows the result of programming and any errors, if they occur.

**Figure 2-1. Main MSP 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 device(s) using the Open Code File button or pulldown menu: File→Open Code File. The formats supported for the code file are TI (.txt) and Intel (.hex) and Motorola (.s19, .s28, .s37).
4. In the Target power group, select the desired  $V_{CC}$  voltage and select if the target is supplied from the MSP Gang Programmer or from an external power supply. If targets are supplied by the programmer, then select the maximum current used by each target, 30 mA or 50 mA.
5. 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.

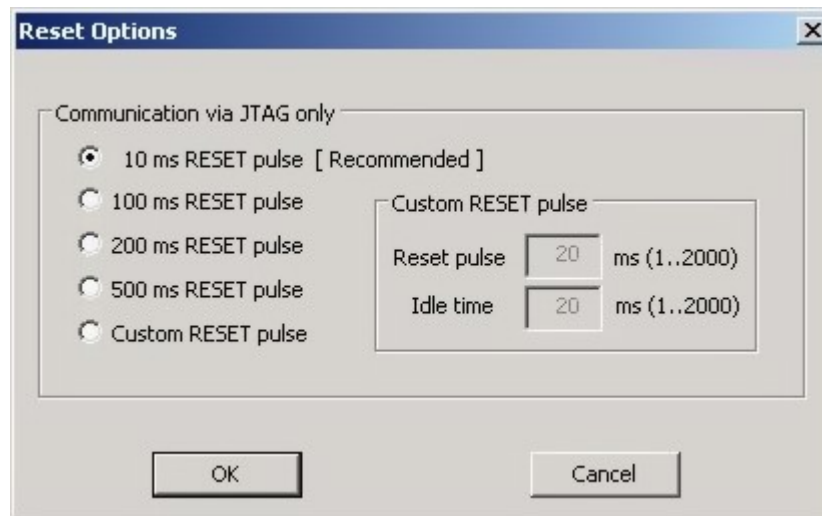
6. In the Interface selector, choose the desired interface (JTAG or Spy-Bi-Wire) and communication speed (fast, medium, or slow).
7. In the Memory Options dialog (pulldown menu: Setup→Memory options ) shown in [Figure 2-2](#), select desired memory space to be programmed. By default, the selected option is All Memory and it is correct for most programming tasks. ([Section 2.1.5](#) describes in detail 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**

8. 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 MSP Gang Programmer to the target device. Depending on the hardware implementation, a longer reset pulse might be required.

**Figure 2-3. Reset Options**

Following these steps creates a working setup that can program target devices using the MSP 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/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 byte to 0xFF). 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.



### 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 above. 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 0xFF, and fails otherwise.

### 2.1.1.4 Program

Click the Program button in the Main Dialog GUI to write the contents of a code file(s) to flash memory on the target device. Addresses specified in the code file(s) 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

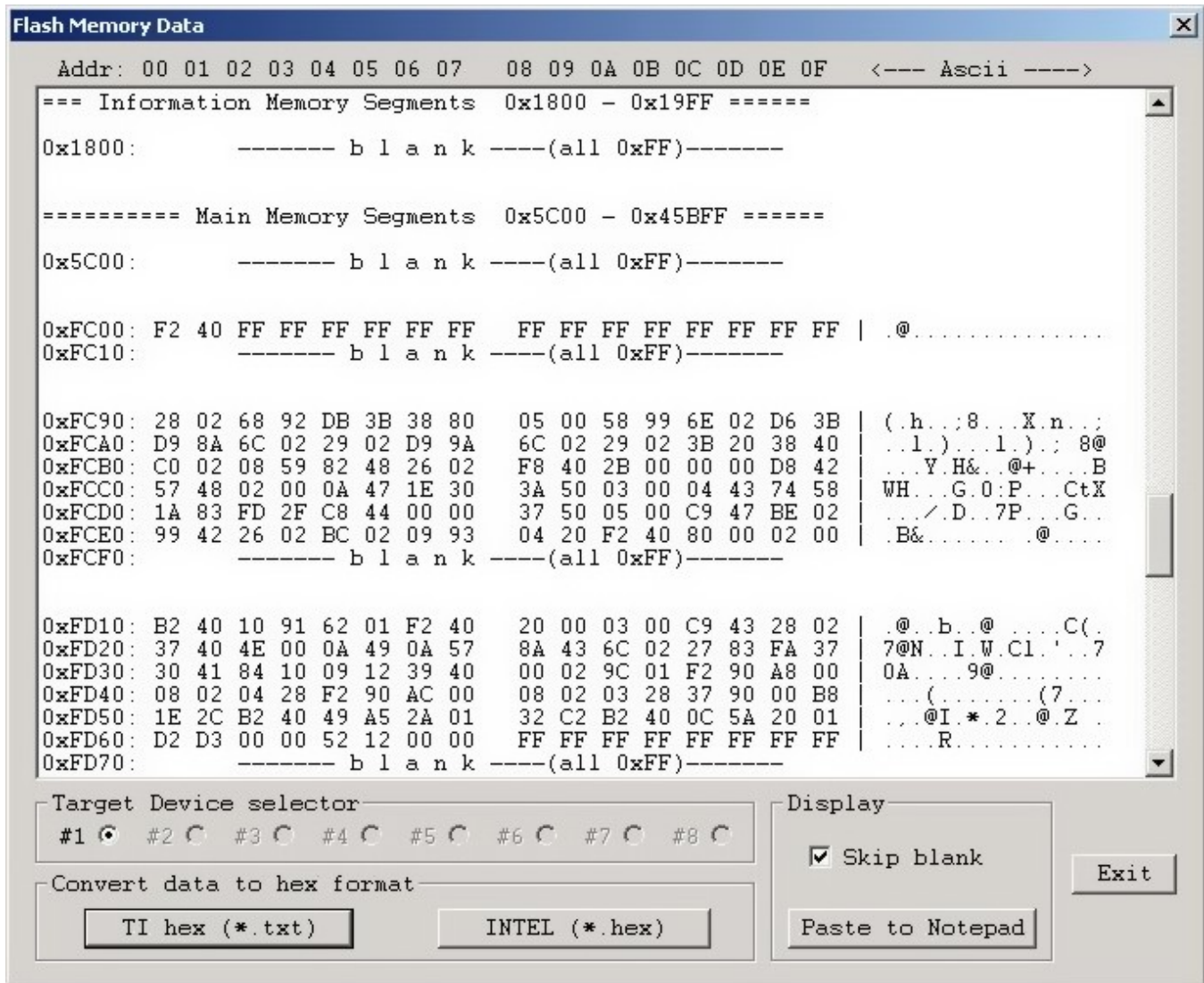
Click the Verify button in the Main Dialog GUI to verify that the contents of the target MCU's flash memory have been properly programmed. This function is best used after programming the same segment of memory, as performed using the button described above. Make sure that the same memory segments are 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.



The screenshot shows a window titled "Flash Memory Data" with a scrollable text area. The text area displays memory addresses from 0x1800 to 0x19FF (Information Memory Segments) and 0x5C00 to 0x45BFF (Main Memory Segments). It shows hex data and ASCII characters. For example, 0x1800 is blank (all 0xFF). 0x5C00 is also blank. 0xFC00 contains hex data: F2 40 FF FF FF FF FF FF FF FF FF FF FF FF FF. 0xFC10 is blank. 0xFC90 contains hex data: 28 02 68 92 DB 3B 38 80 05 00 58 99 6E 02 D6 3B. 0xFCA0 contains hex data: D9 8A 6C 02 29 02 D9 9A 6C 02 29 02 3B 20 38 40. 0xFCB0 contains hex data: C0 02 08 59 82 48 26 02 F8 40 2B 00 00 00 D8 42. 0xFCC0 contains hex data: 57 48 02 00 0A 47 1E 30 3A 50 03 00 04 43 74 58. 0xFCD0 contains hex data: 1A 83 FD 2F C8 44 00 00 37 50 05 00 C9 47 BE 02. 0xFCE0 contains hex data: 99 42 26 02 BC 02 09 93 04 20 F2 40 80 00 02 00. 0xFCF0 is blank. 0xFD10 contains hex data: B2 40 10 91 62 01 F2 40 20 00 03 00 C9 43 28 02. 0xFD20 contains hex data: 37 40 4E 00 0A 49 0A 57 8A 43 6C 02 27 83 FA 37. 0xFD30 contains hex data: 30 41 84 10 09 12 39 40 00 02 9C 01 F2 90 A8 00. 0xFD40 contains hex data: 08 02 04 28 F2 90 AC 00 08 02 03 28 37 90 00 B8. 0xFD50 contains hex data: 1E 2C B2 40 49 A5 2A 01 32 C2 B2 40 0C 5A 20 01. 0xFD60 contains hex data: D2 D3 00 00 52 12 00 00 FF FF FF FF FF FF FF FF. 0xFD70 is blank.

At the bottom of the window, there is a "Target Device selector" with radio buttons #1 through #8. Below it is a "Convert data to hex format" section with buttons for "TI hex (\*.txt)" and "INTEL (\*.hex)". To the right is a "Display" section with a checked "Skip blank" checkbox and a "Paste to Notepad" button. An "Exit" button is also present.

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**

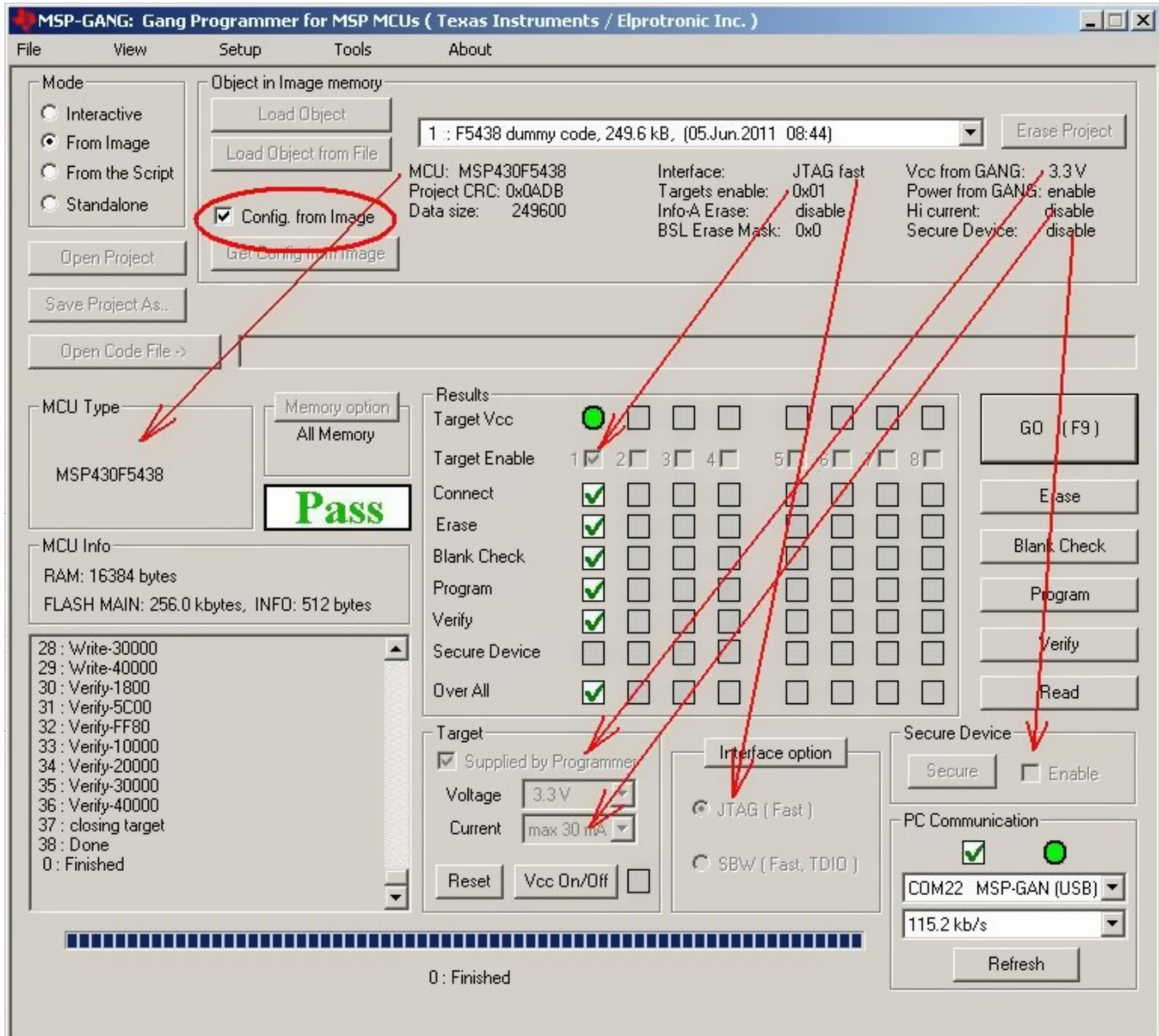
### 2.1.2 Programming From Image

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 MSP 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.

Once 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 MSP 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.

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

To avoid confusion during programming, connecting an SD-Card to the MSP 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 MSP Gang Programmer internal memory can be used.



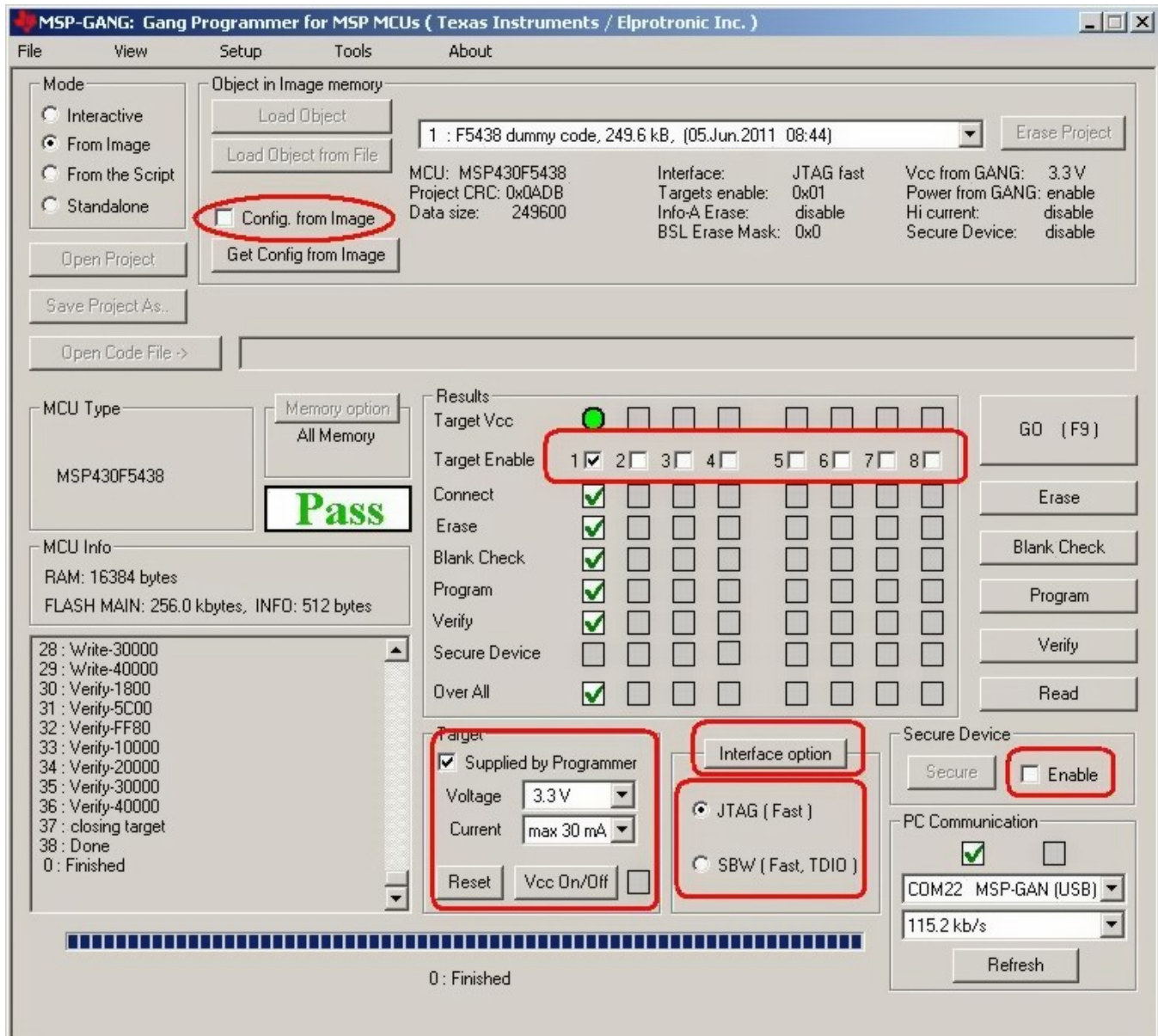
NOTE: This figure uses the From Image Mode (see the Mode section near the top left corner). The user can load an image from MSP Gang Programmer internal memory. Saved images contain all configuration necessary for programming and all code files. An image can be created using the Interactive Mode and 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 MSP Gang Programmer Dialog GUI, From Image Mode**



Figure 2-5 highlights several parts of the GUI. The drop-down menu in the Object in 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 high-lighted in red and include which devices are enabled for programming, target  $V_{CC}$  and current, 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 in Section 2.1.6.



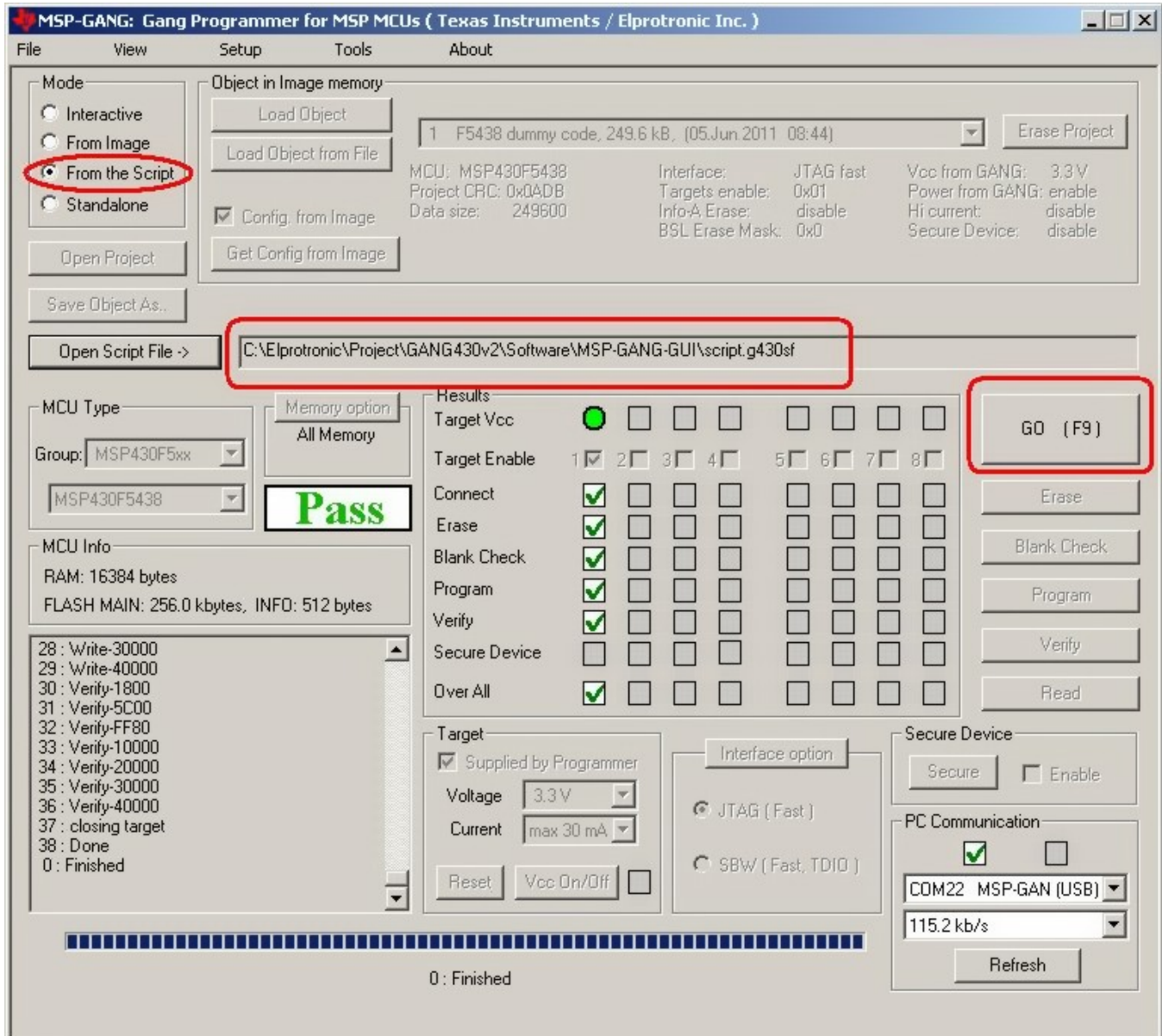
NOTE: This figure uses the From Image Mode (top left corner). The Config. from Image option is disabled in this example, allowing the user to change various but not all configuration settings from the image. The configuration options that can be changed are highlighted in red. One of the options that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

**Figure 2-6. Main MSP 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, adjust voltage, 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, etc.), which is sufficient for most nonrecursive programs.

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 uses 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 MSP 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, etc.).

### 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, errors, etc.). 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.

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

**RETURN** – Return from CALL.

**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, Read File, etc.) finished successfully.
- **FAILED** – Previous process (for example GO, Read File, etc.) failed.

The operation can be one of the following:

- **GOTO** label
- **CALL** label **SLEEP** number – Pause a number of milliseconds, between 1 and 10000.

**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\_VCCINMV** – Set  $V_{CC}$  in mV, between 1800 to 3600 in steps of 100 mV.

**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\_BLOWFUSE** – Perform BLOW FUSE function from main dialogue screen.

---

**NOTE:** **Blows fuse regardless of enable option.**

If the BLOW FUSE command is used, then the security fuse is blown even if the Blow Security Fuse enable option is disabled.

---

**F\_SETIMAGENUMBER number** – Choose image number between 1 and 16 from MSP Gang Programmer internal memory.

**F\_STANDALONEMODE** – Switch to Standalone mode.

**F\_INTERACTIVEMODE** – Switch to Interactive mode.

**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.

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.g430cfg to the MSP Gang Programmer. Configuration file test-A.cfg 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 MSP 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.g430cfg to the MSP Gang Programmer.
11. The MSP 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.

```

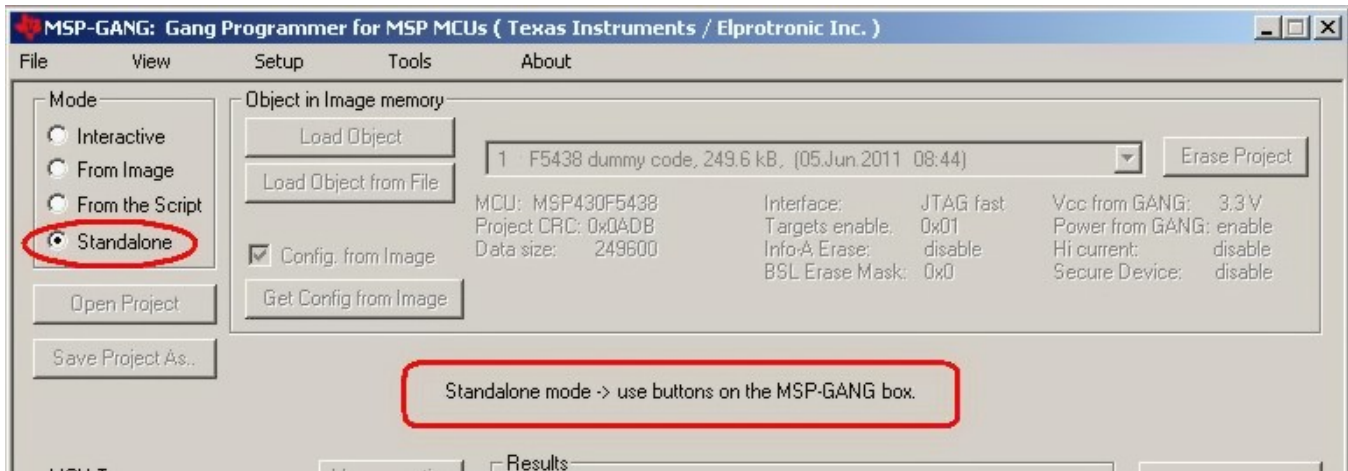
;=====
; Script file - demo program
;-----
>START
VCCOFF
MESSAGEBOX OKCANCEL
"VCC is OFF now. Connect the test board."
"When ready press the button:"
" "
"OK - to test the board"
"CANCEL - to exit from program"
IF BUTTONCANCEL GOTO finish
F_LOADCFGFILE C:\Elprotronic\Project\Cpp-Net\GANG430U\test-A.G430cfg
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\GANG430U\finalcode.G430cfg
F_GO
GOTO START
>finish
END
;=====

```



### 2.1.4 Programming in Standalone Mode

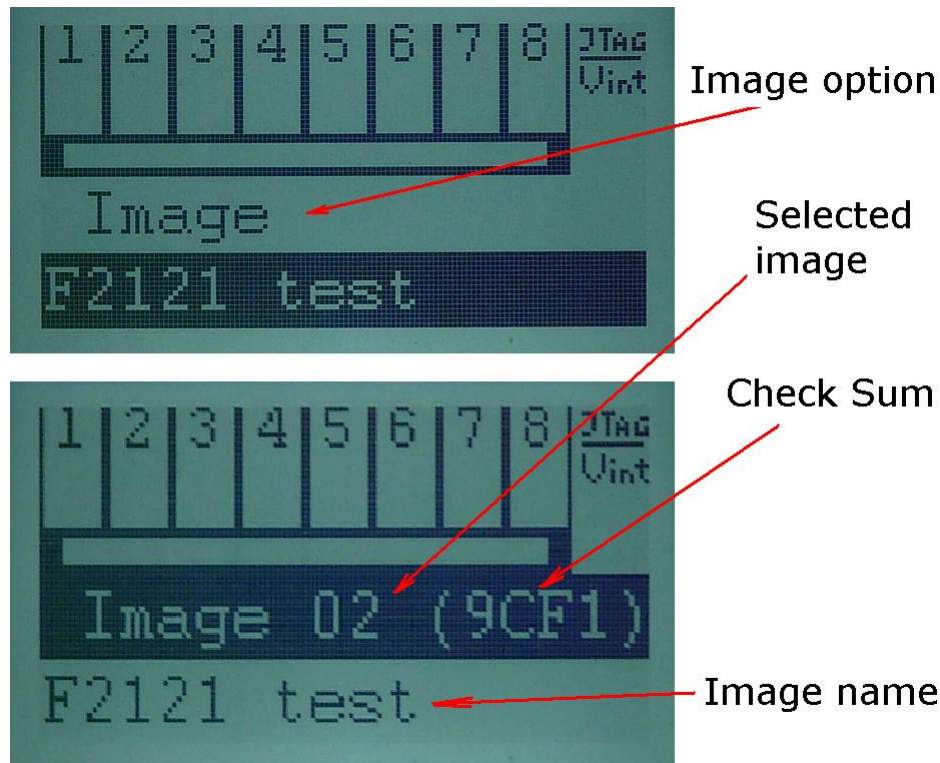
The MSP Gang Programmer supports a Standalone mode of programming target devices. In this mode the MSP Gang Programmer can only use images for programming because they contain a complete configuration and code files necessary for the procedure. If the user has not already created an image then follow 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 MSP 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 MSP 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.

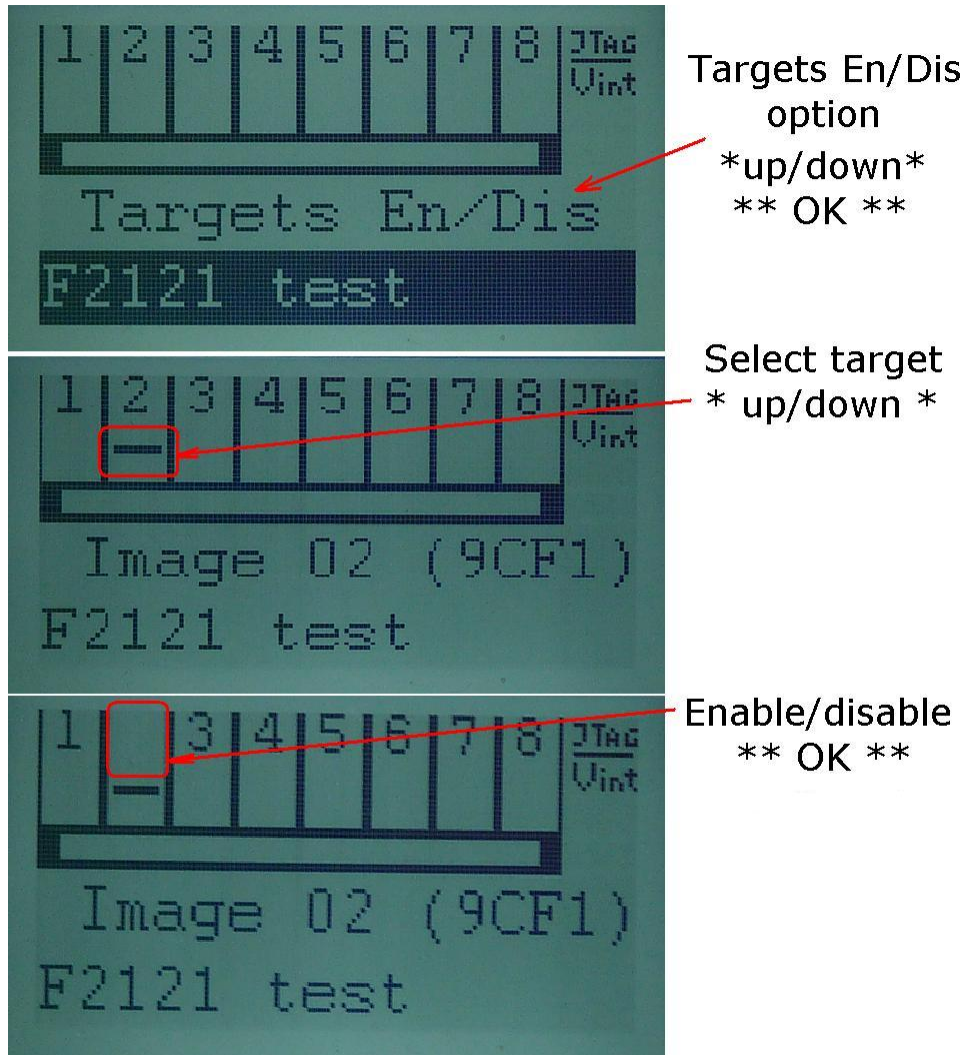
**Figure 2-8. Main MSP Gang Programmer Dialog GUI, Standalone Mode**

After images have been download to the internal memory or after an SD card with a valid image is connected to the MSP Gang Programmer, proceed with programming in Standalone mode. Use the arrow buttons (up/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](#)).


**Figure 2-9. Image Option**

After the desired image has been selected, press the GO button on the MSP 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 MSP 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 MSP-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.



**Figure 2-10. Target Enable/Disable Option**

In addition to 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.

## 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:

```
@1008
25 CA 80 40 39 E3 F8 02
@2200
48 35 59 72 AC B8
q
```

Then 8 bytes of data are written starting at location 0x1008 and 6 bytes of data starting at location 0x2200. The specified addresses should be blank before writing (contain a value of 0xFF). Before the writing operation is actually performed, the MSP Gang Programmer automatically verifies if this part of memory is blank and proceeds to program the device only if verification is successful.

---

### NOTE: Even Number of Bytes

The number of bytes in all data blocks must be even. Words (two bytes) are used for writing and reading data. In case that the code file contains an odd number of bytes, the data segment is appended by a single byte containing a blank value of 0xFF. This value does not overwrite the current memory contents (because Update only is selected), but verification fails if the target device does not contain a blank value of 0xFF at that location.

---

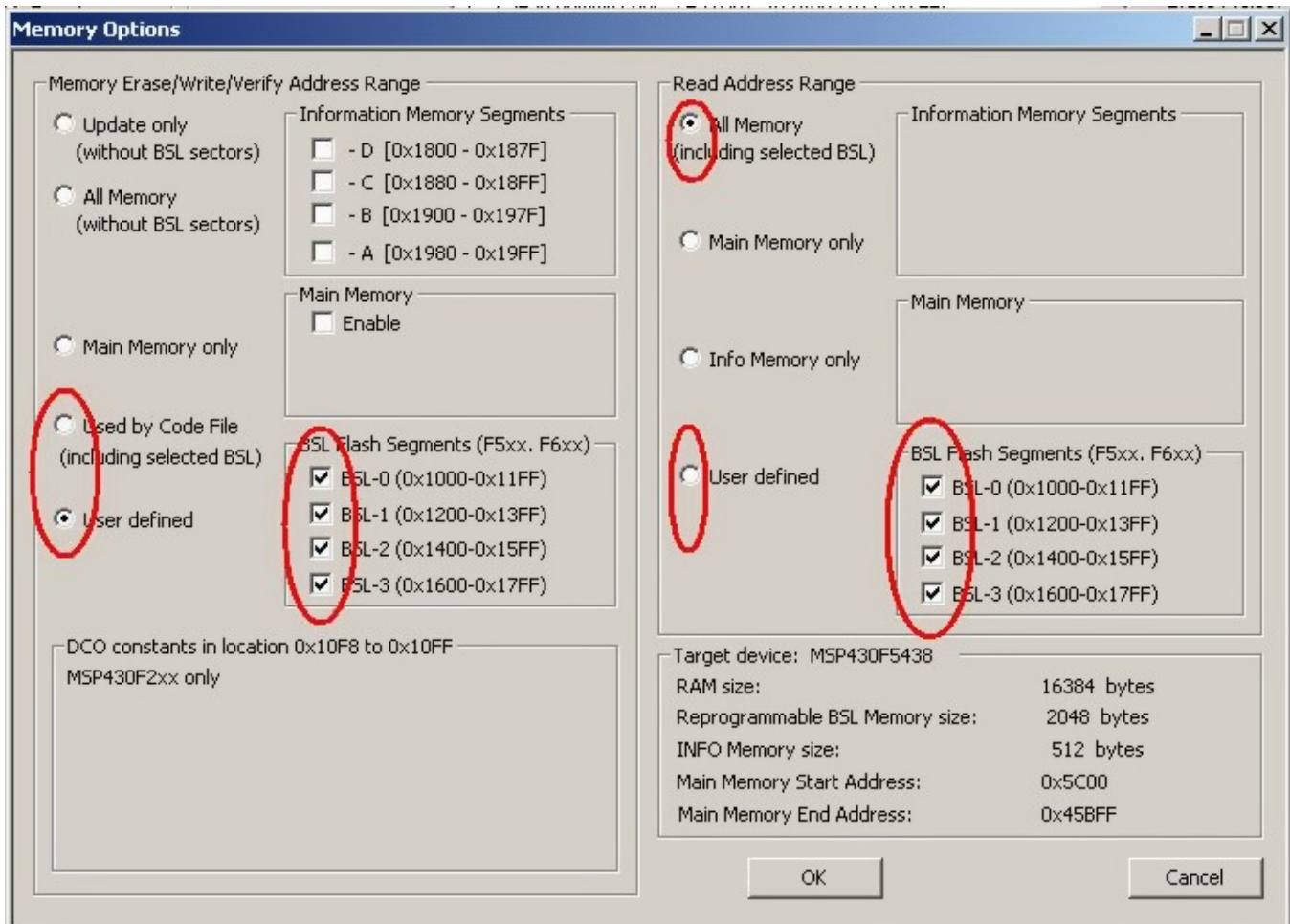
2. All Memory – This is the most frequently used option during programming. All memory is erased before programming, and all contents from the code file are downloaded to the target microcontroller's flash memory. When the microcontroller contains an INFO-A segment that can be locked (for example the MSP430F2xx series contains DCO constants at locations 0x10F8 to 0x10FF), then INFO-A can be erased or left unmodified. The including locked INFO-A segment should be selected or unselected respectively. When INFO-A is not erased, none of the data is saved into INFO-A, even if this data is specified in the code file. In addition, the DCO constants in the Retain Data in Flash group should be selected if the DCO constants should be restored after erasing the INFO-A segment.
3. Main memory only – Flash information memory (segments A and B, C, D) are not modified. Contents of information memory from the code file are ignored.
4. Used by Code File – This option allows main memory segments and information 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.
5. 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, in the Memory Options dialog screen, check boxes and address edit lines are enabled. The check boxes allow the user to select information memory segments to be enabled (erased, programmed, verified). Edit lines in the Main Memory group allow the user to specify the main memory address range (start and stop addresses). The start address should specify the first byte in the segment, and the stop address should specify the last byte in the segment (last byte is programmed). Because the main memory segment size is 0x200, the start address should be a multiple of 0x200; for example, 0x2200. The stop address should specify the last byte of the segment to be written. Therefore, it should be greater than the start address and point to a byte that immediately precedes a memory segment boundary; for example, 0x23FF or 0x55FF.

### 2.1.5.1 Writing and Reading BSL Flash Sectors in the F5xx/F6xx MCUs

The MSP430F5xx and MSP430F6xx microcontrollers have BSL firmware saved in flash memory sectors. By default, access to these sectors (Read or Write) is blocked, however it is possible to modify the BSL firmware if required, which allows the user to upload newer or custom defined BSL firmware. These BSL



sectors are located in memory starting at 0x1000 to 0x17FF. The MSP Gang Programmer software handles modification of these BSL flash sectors using the same method as all other memory sectors. However, to avoid unintentional erasing of BSL sectors, the most commonly used memory option, All Memory, blocks access to these BSL sectors. Access to BSL sectors is unlocked only when the Used by Code File or User defined option is selected and desired selected BSL sectors are enabled, as shown in Figure 2-11. Contents of BSL sectors can be read even when the All Memory option is selected.



NOTE: The user can select which segments of memory are written to or read from. The selected configuration shows how the user can configure the programmer to overwrite segments of memory used by the Bootstrap Loader (BSL).

Figure 2-11. Memory Options, BSL Sectors Selected

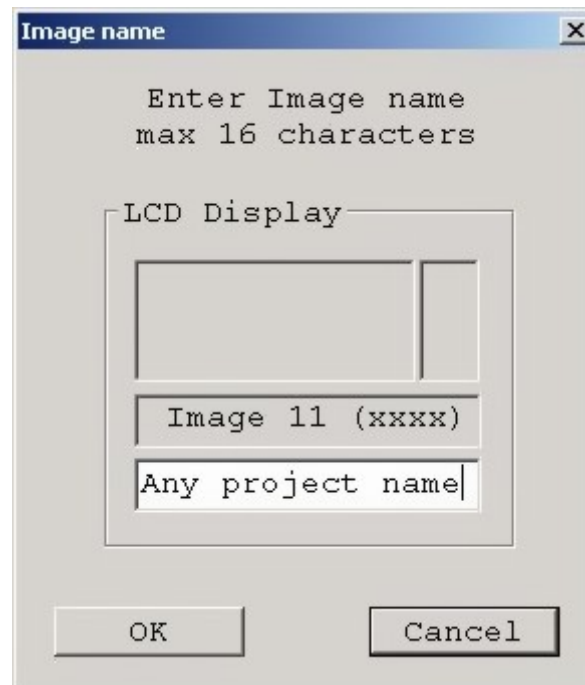
### 2.1.6 Creating and Using Images

Once 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 MSP 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 MSP Gang Programmer setup as a project first. This is recommended because images cannot be modified once created, only 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. Once 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 device(s). The MSP 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 MSP Gang Programmer internal memory, or the Save Image to file button to save to a file.

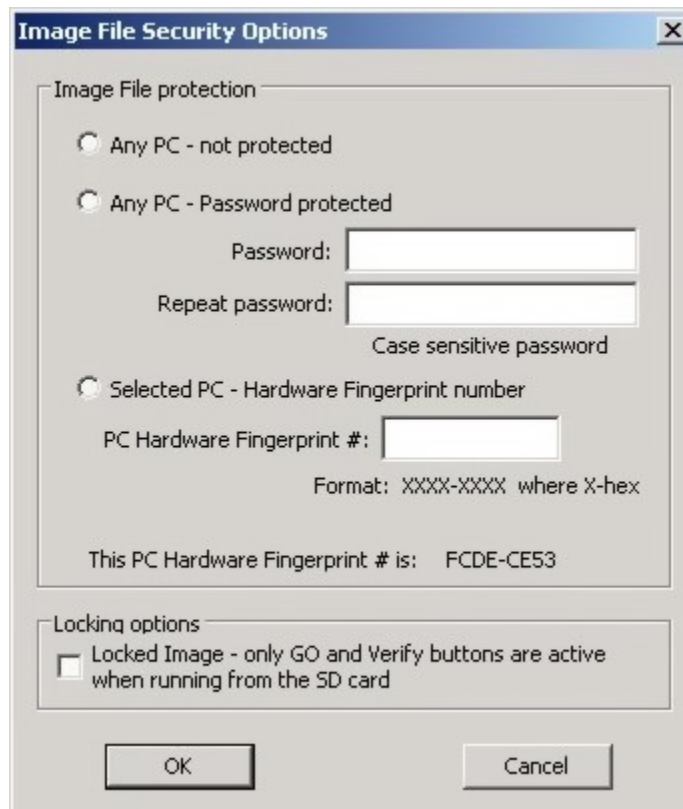
Whether the new image being created is saved to a file or to MSP Gang Programmer internal memory, an image configuration screen appears (see [Figure 2-12](#)). Enter any name up to 16 characters. This name is displayed in the GUI image selector (see [Figure 2-1](#)) and it is displayed on the bottom line of the MSP Gang Programmer LCD screen when the corresponding image is selected. Press OK when the name is entered.



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

**Figure 2-12. Image Name Configuration Screen**

The screen shown in [Figure 2-13](#) 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.



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

**Figure 2-13. Image File Security Options**

1. **Any PC** – Configuration can be opened on any computer using MSP Gang Programmer software. It can be used for programming only.
2. **Any PC - Password protected** – Configuration can be opened on any computer using the MSP 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 2-14](#) 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 MSP Gang Programmer software.



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 2-14. Hardware Fingerprint of Computer in Use**

The image file can be copied to internal MSP 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.

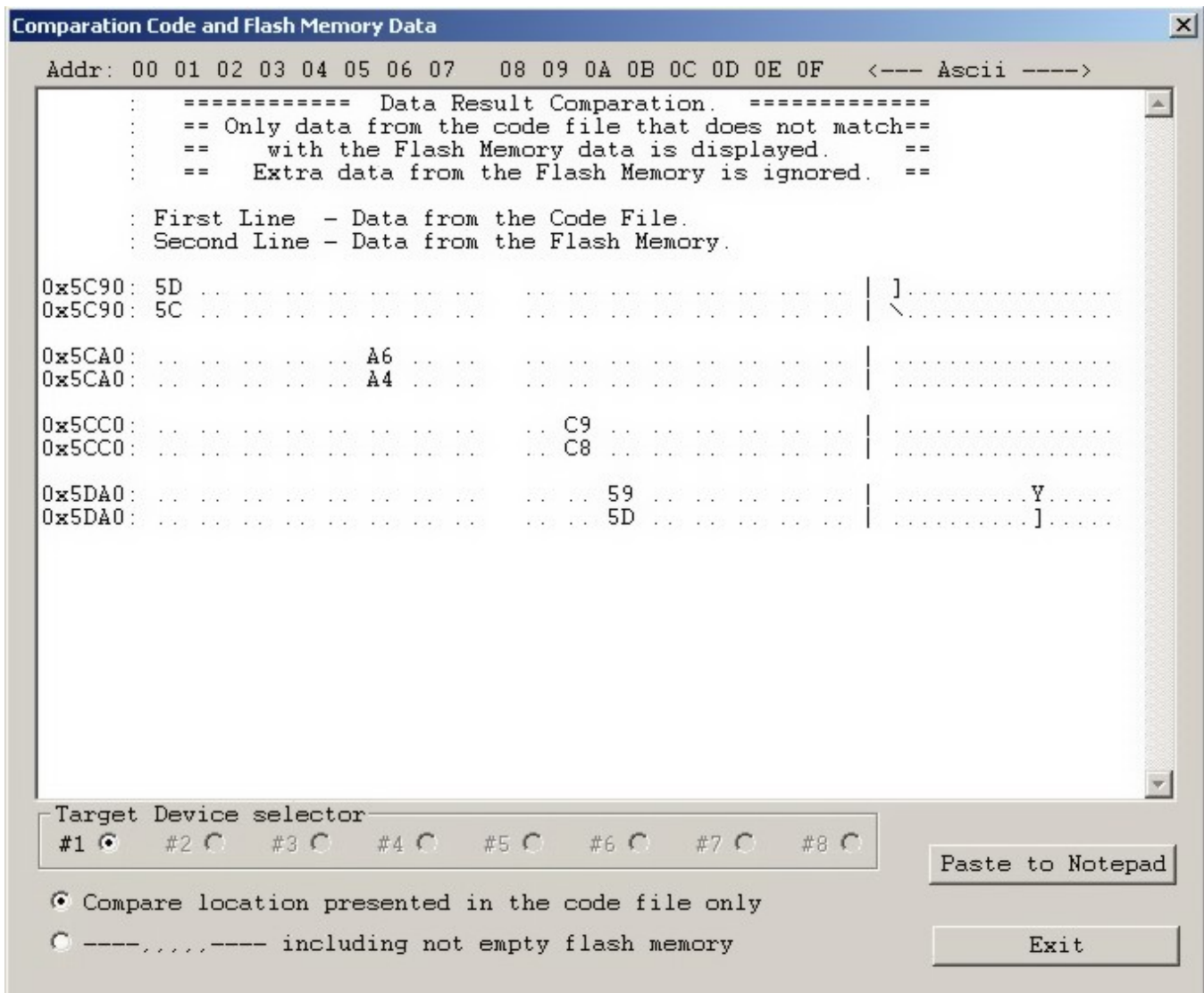




Contents of flash memory data can be viewed by selecting the View→Flash Memory Data option from the drop-down menu. To be able to see flash memory contents, the Read button must be used first (as described in Section 2.1.1). The Flash Memory Data viewer displays the memory addresses, data in hex and ASCII format in the same way as the Code data viewer shown in Figure 2-15.

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 drop-down 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 2-16.

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, personal data, etc. 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 bytes 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 2-16. Comparison of Code and Flash Memory Data of the Target Processor

## 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 MSP 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 most commonly executed tasks are listed below:

- Initialization
- Open Target Device
- Close Target Device
- Erase
  - Segment
  - Main memory
  - Info memory
  - BSL 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
- Save Info-A
- DCO calibration
- Retain Info-A
- SetPC and run
- Capture PC and Stop
- Stop PC
- Secure device
- Finish

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

- Initialization
- Open Target Device
- Erase
- Blank check
- Program
- Verify
- Close target and 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, erase info segment, erase one block of the main memory, etc. For that reason, many more tasks are displayed in the report window than are described above. For example, when programming the MSP430F5438 the following information would be displayed in the report window:

```
Executing Main Process...
=== F5438 Test ===
.....
2 : init target
```

```

3 : erasing-Info
4 : erasing-Info
5 : erasing-Info
6 : erasing-Main
7 : erasing-Main
8 : erasing-Main
9 : erasing-Main
10 : erasing-Main
11 : erasing-Main
12 : erasing-Main
13 : erasing-Main
14 : erasing-Main
15 : BlankCh-Info
16 : BlankCh-Info
17 : BlankCh-Info
18 : BlankCh-Main
19 : BlankCh-Main
20 : BlankCh-Main
21 : BlankCh-Main
22 : BlankCh-Main
23 : Programming
24 : Programming
25 : Programming
26 : Programming
27 : Programming
28 : Programming
29 : Verifying
30 : Verifying
31 : Verifying
32 : Verifying
33 : Verifying
34 : Verifying
35 : closing target
36 : Finished
Done

```

This report indicates that sectors INFO-B,C,D and the main memory block have been erased (tasks 2 to 14) blank checked (tasks 15 to 22), programmed (tasks 23 to 28) and verified (tasks 29 to 34). 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 MSP Gang Programmer LCD display.

The MSP 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 in one image), the MSP 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 taken and combined with the first one for one larger image block (1Mbyte or more). The MSP Gang Programmer has internal flash memory of 8Mbyte that can, if desired, all be used to form one image with a memory footprint of 8Mbyte.

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 MSP 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 MSP Gang Programmer.

```

ERROR 1: BOOT Firmware only is in the \productName! The API Firmware should be downloaded.
ERROR 2: API Firmware CRC is not present! The API Firmware should be reloaded.
ERROR 3: API Firmware CRC error! The API Firmware should be reloaded.
ERROR 4: BOOT CRC error in the \productName!
ERROR 10: CRC Access key. Key corrupted. Access to programmer is blocked
ERROR 11: Invalid programmer's access key. Access to programmer is blocked
ERROR 12: Unknown interface
ERROR 13: Vcc is too low

```

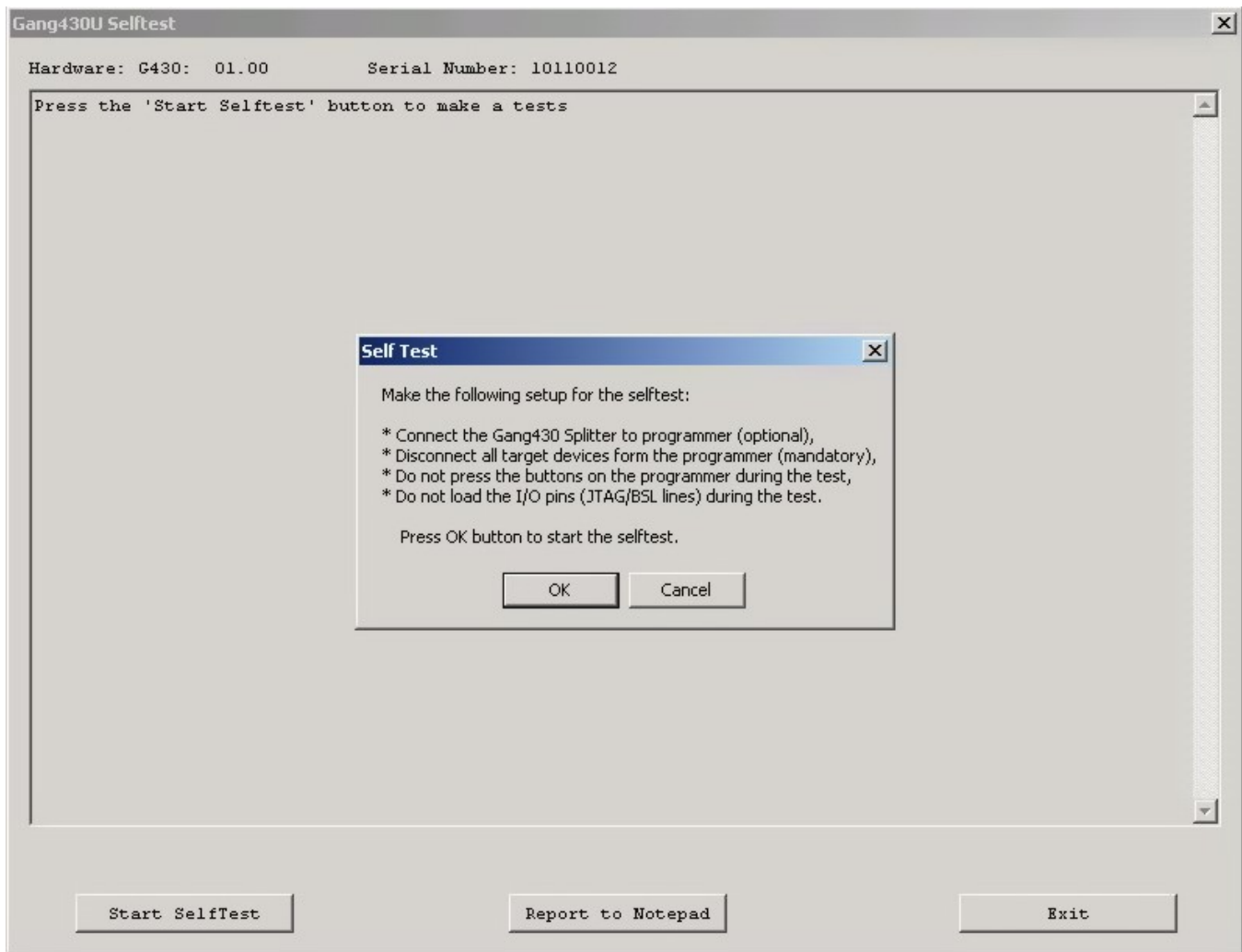
```
ERROR 14: Vcc is too high
ERROR 15: VtIO is too low
ERROR 16: VtIO is too high
ERROR 17: Header CRC
ERROR 18: Script CRC
ERROR 19: Exceed script no
ERROR 20: Script command ?
ERROR 21: MCU Fetch synch
ERROR 22: CPU JTAG synch
ERROR 23: MCU device init
ERROR 24: RAM FW download
ERROR 25: Blank check err
ERROR 26: Read verify err
ERROR 27: Flash write err
ERROR 28: Image FL WR init
ERROR 29: Image Flash lock
ERROR 30: Invalid Script T
ERROR 31: Size too high
ERROR 32: Used wrong MCU
ERROR 33: IR Interrupted
Info 0xB0: Task in progress
ERROR: Communication - Frame has errors !"
ERROR: Unable to open COM port - already in use?
ERROR: Unable to close COM port !
ERROR: Unable to modify COM port state !
ERROR: Synchronization failed. Programmer connected?
ERROR: Timeout during operation - Correct COM port selected?
ERROR: Command did not complete correctly !
ERROR: Command failed or not defined or Target not accessible !
ERROR: Wrong baud rate specified !
ERROR: Could not read "\"\productName.cfg"
ERROR: File contains invalid record !
ERROR: Unexpected end of file !
ERROR: Error during file I/O !
ERROR: Selected file is of unrecognizable format !
ERROR: Unable to open file !
ERROR: Function argument(s) out of range !
WARNING: RAM affected during programming cycle. Verify may fail !
Info: Task in progress
FATAL ERROR: Image Memory corrupted or erased ! Load Image.
ERROR: Target not accessible !
ERROR: Verification failed !
ERROR: Main Process Parameters not yet set ! Load Image.
ERROR: Could not erase Image Buffer !
ERROR: Could not load Image Buffer !
ERROR: Could not load Main Process Parameters !
ERROR: Could not select Baud Rate !
WARNING: Could not set target voltage - Short circuitry or settling time too small?
ERROR: Invalid firmware command !
ERROR: Power supply voltage too low !
WARNING: Sense voltage out of range - Check pin MSP\_VCC\_IN of target connector !
ERROR: Wrong target device connected !
ERROR: No target device connected
ERROR: File(s) contains already specified data (code overwritten)
ERROR: Selected image number out of range
ERROR: Could not open the configuration file.
ERROR: Script Header size error
ERROR: Image ID error. Image ignored, program terminated.
ERROR: Image contents (size, no of tasks) error. Program terminated.
ERROR: Image CRC error. Program terminated.
WARNING: Code overwritten. Code from the file written to already used location.
ERROR: Code in the file contains invalid data.
ERROR: Open File error
ERROR: Extension or file name error
ERROR: Wrong password for opening the image file
ERROR: Wrong PC hardware fingerprint \# for opening the image file
```

```
ERROR: Image file ID error or file corrupted
ERROR: Check Sum of the Image file error or file corrupted
ERROR: Wrong header in the image file or file corrupted
ERROR: Image file is not for the \productName programmer.
ERROR: Image file contents error or file corrupted.
ERROR: Unknown protection mode of the image file or file corrupted.
ERROR: Data offset in the image file error or file corrupted.
ERROR: Hex data conversion in the image file error or file corrupted.
ERROR: Image file corrupted.
ERROR: Image file cannot be unlocked
ERROR: Customized MCUs license file open error
WARNING: Code specified for the BSL space location, but access to the BSL is locked
ERROR: Info memory page number is out of range
ERROR: COM ports scan number is too low
ERROR: Self test data size too high.
ERROR: data size too high.
ERROR: Gang mask ZERO. Nothing to do.
ERROR: Address definition.
ERROR: data size is below 2.
```

## 2.4 Self Test

The MSP Gang Programmer Self Test program allows to test most of the hardware for correctness. Connect the programmer to a computer running MSP Gang Programmer software. If the user is utilizing a Gang Splitter, then connect it to MSP Gang Programmer hardware (this allows the Self Test to find short circuits in the Gang Splitter). 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 drop-down menu. Press the Start Self Test button, as shown in [Figure 2-17](#), to begin. If the Self Test reports any problems then it is advisable to send the test report to TI technical support for assistance.



NOTE: Use the MSP 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 MSP Gang Programmer.

**Figure 2-17. Self Test**

The following is a typical self test report:

```

=== GANG430U Self test results ( Saturday, November 27, 2010, 19:08:43 ) ===
Adapter SN -----: 10110012
Hardware -----: G430: 01.00
Access key -----: MSP430 - Gang Programmer
Silicon Number --: 24D4 CC47 0400 1B00
API Firmware ----: G430Beta A430: 01.00.00.07
BOOT Firmware ---: G430BOOT B430: 01.00.00.00
GUI Software ----: Gang430U-GUI G430: 01.00.00.00
DLL Software ----: Gang430U-DLL D430: 01.00.00.00
===== Test results =====
No. name parameter limits result status
1: Vcc Target-1 (ALL OFF) 0.00 V ( 0.00 to 0.30) Result: 0.02 V ... >> OK <<
2: Vcc Target-2 (ALL OFF) 0.00 V ( 0.00 to 0.30) Result: 0.01 V ... >> OK <<
3: Vcc Target-3 (ALL OFF) 0.00 V ( 0.00 to 0.30) Result: 0.01 V ... >> OK <<
4: Vcc Target-4 (ALL OFF) 0.00 V ( 0.00 to 0.30) Result: 0.01 V ... >> OK <<
5: Vcc Target-5 (ALL OFF) 0.00 V ( 0.00 to 0.30) Result: 0.01 V ... >> OK <<
6: Vcc Target-6 (ALL OFF) 0.00 V ( 0.00 to 0.30) Result: 0.01 V ... >> OK <<
7: Vcc Target-7 (ALL OFF) 0.00 V ( 0.00 to 0.30) Result: 0.01 V ... >> OK <<
8: Vcc Target-8 (ALL OFF) 0.00 V ( 0.00 to 0.30) Result: 0.01 V ... >> OK <<
9: Translators VT (OFF) 0.00 V ( 0.00 to 0.50) Result: 0.01 V ... >> OK <<

```

```

10: Translators VT (ON 1.8V) 1.80 V ( 1.60 to 2.00) Result: 1.80 V ... >> OK <<
11: Translators VT (ON 2.7V) 2.70 V ( 2.50 to 2.90) Result: 2.68 V ... >> OK <<
12: Translators VT (ON 3.6V) 3.60 V ( 3.40 to 3.80) Result: 3.58 V ... >> OK <<
13: Vpp Voltage-in 10.00 V ( 8.00 to 12.00) Result: 9.96 V ... >> OK <<
14: Vpp Voltage 7.00 V ( 6.50 to 7.30) Result: 6.90 V ... >> OK <<
15: Internal Vcc-3.3V 3.30 V ( 3.20 to 3.40) Result: 3.30 V ... >> OK <<
16: Vcc Target-1 (ALL ON 1.8V) 1.80 V ( 1.60 to 2.00) Result: 1.78 V ... >> OK <<
17: Vcc Target-2 (ALL ON 1.8V) 1.80 V ( 1.60 to 2.00) Result: 1.76 V ... >> OK <<
18: Vcc Target-3 (ALL ON 1.8V) 1.80 V ( 1.60 to 2.00) Result: 1.79 V ... >> OK <<
19: Vcc Target-4 (ALL ON 1.8V) 1.80 V ( 1.60 to 2.00) Result: 1.79 V ... >> OK <<
20: Vcc Target-5 (ALL ON 1.8V) 1.80 V ( 1.60 to 2.00) Result: 1.78 V ... >> OK <<
21: Vcc Target-6 (ALL ON 1.8V) 1.80 V ( 1.60 to 2.00) Result: 1.80 V ... >> OK <<
22: Vcc Target-7 (ALL ON 1.8V) 1.80 V ( 1.60 to 2.00) Result: 1.76 V ... >> OK <<
23: Vcc Target-8 (ALL ON 1.8V) 1.80 V ( 1.60 to 2.00) Result: 1.75 V ... >> OK <<
24: Vcc Target-1 (ALL ON 2.7V) 2.70 V ( 2.50 to 2.90) Result: 2.67 V ... >> OK <<
25: Vcc Target-2 (ALL ON 2.7V) 2.70 V ( 2.50 to 2.90) Result: 2.66 V ... >> OK <<
26: Vcc Target-3 (ALL ON 2.7V) 2.70 V ( 2.50 to 2.90) Result: 2.68 V ... >> OK <<
27: Vcc Target-4 (ALL ON 2.7V) 2.70 V ( 2.50 to 2.90) Result: 2.69 V ... >> OK <<
28: Vcc Target-5 (ALL ON 2.7V) 2.70 V ( 2.50 to 2.90) Result: 2.67 V ... >> OK <<
29: Vcc Target-6 (ALL ON 2.7V) 2.70 V ( 2.50 to 2.90) Result: 2.70 V ... >> OK <<
30: Vcc Target-7 (ALL ON 2.7V) 2.70 V ( 2.50 to 2.90) Result: 2.65 V ... >> OK <<
31: Vcc Target-8 (ALL ON 2.7V) 2.70 V ( 2.50 to 2.90) Result: 2.64 V ... >> OK <<
32: Vcc Target-1 (ALL ON 3.6V) 3.60 V ( 3.40 to 3.80) Result: 3.59 V ... >> OK <<
33: Vcc Target-2 (ALL ON 3.6V) 3.60 V ( 3.40 to 3.80) Result: 3.56 V ... >> OK <<
34: Vcc Target-3 (ALL ON 3.6V) 3.60 V ( 3.40 to 3.80) Result: 3.59 V ... >> OK <<
35: Vcc Target-4 (ALL ON 3.6V) 3.60 V ( 3.40 to 3.80) Result: 3.60 V ... >> OK <<
36: Vcc Target-5 (ALL ON 3.6V) 3.60 V ( 3.40 to 3.80) Result: 3.59 V ... >> OK <<
37: Vcc Target-6 (ALL ON 3.6V) 3.60 V ( 3.40 to 3.80) Result: 3.61 V ... >> OK <<
38: Vcc Target-7 (ALL ON 3.6V) 3.60 V ( 3.40 to 3.80) Result: 3.58 V ... >> OK <<
39: Vcc Target-8 (ALL ON 3.6V) 3.60 V ( 3.40 to 3.80) Result: 3.54 V ... >> OK <<
40: Vcc discharge (100ms)Target-1 3.60 V ( 1.00 to 2.70) Result: 2.10 V ... >> OK <<
41: Vcc discharge (100ms)Target-2 3.60 V ( 1.00 to 2.70) Result: 2.00 V ... >> OK <<
42: Vcc discharge (100ms)Target-3 3.60 V ( 1.00 to 2.70) Result: 2.07 V ... >> OK <<
43: Vcc discharge (100ms)Target-4 3.60 V ( 1.00 to 2.70) Result: 2.04 V ... >> OK <<
44: Vcc discharge (100ms)Target-5 3.60 V ( 1.00 to 2.70) Result: 2.08 V ... >> OK <<
45: Vcc discharge (100ms)Target-6 3.60 V ( 1.00 to 2.70) Result: 2.13 V ... >> OK <<
46: Vcc discharge (100ms)Target-7 3.60 V ( 1.00 to 2.70) Result: 2.02 V ... >> OK <<
47: Vcc discharge (100ms)Target-8 3.60 V ( 1.00 to 2.70) Result: 2.01 V ... >> OK <<
48: Vcc Target-1 ( \#1 ON ) 0.00 V ( 3.10 to 3.50) Result: 3.27 V ... >> OK <<
49: Vcc Target-2 ( \#1 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
50: Vcc Target-3 ( \#1 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
51: Vcc Target-4 ( \#1 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
52: Vcc Target-5 ( \#1 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
53: Vcc Target-6 ( \#1 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
54: Vcc Target-7 ( \#1 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
55: Vcc Target-8 ( \#1 ON ) 3.30 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
56: Vcc Target-1 ( \#2 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.27 V ... >> OK <<
57: Vcc Target-2 ( \#2 ON ) 0.00 V ( 3.10 to 3.50) Result: 3.26 V ... >> OK <<
58: Vcc Target-3 ( \#2 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
59: Vcc Target-4 ( \#2 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
60: Vcc Target-5 ( \#2 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
61: Vcc Target-6 ( \#2 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
62: Vcc Target-7 ( \#2 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
63: Vcc Target-8 ( \#2 ON ) 3.30 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
64: Vcc Target-1 ( \#3 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
65: Vcc Target-2 ( \#3 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
66: Vcc Target-3 ( \#3 ON ) 0.00 V ( 3.10 to 3.50) Result: 3.28 V ... >> OK <<
67: Vcc Target-4 ( \#3 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
68: Vcc Target-5 ( \#3 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
69: Vcc Target-6 ( \#3 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
70: Vcc Target-7 ( \#3 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
71: Vcc Target-8 ( \#3 ON ) 3.30 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
72: Vcc Target-1 ( \#4 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
73: Vcc Target-2 ( \#4 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
74: Vcc Target-3 ( \#4 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.26 V ... >> OK <<
75: Vcc Target-4 ( \#4 ON ) 0.00 V ( 3.10 to 3.50) Result: 3.29 V ... >> OK <<

```



```

76: Vcc Target-5 ( \#4 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
77: Vcc Target-6 ( \#4 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
78: Vcc Target-7 ( \#4 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
79: Vcc Target-8 ( \#4 ON ) 3.30 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
80: Vcc Target-1 ( \#5 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
81: Vcc Target-2 ( \#5 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
82: Vcc Target-3 ( \#5 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
83: Vcc Target-4 ( \#5 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
84: Vcc Target-5 ( \#5 ON ) 0.00 V ( 3.10 to 3.50 ) Result: 3.27 V ... >> OK <<
85: Vcc Target-6 ( \#5 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
86: Vcc Target-7 ( \#5 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
87: Vcc Target-8 ( \#5 ON ) 3.30 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
88: Vcc Target-1 ( \#6 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
89: Vcc Target-2 ( \#6 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
90: Vcc Target-3 ( \#6 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
91: Vcc Target-4 ( \#6 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
92: Vcc Target-5 ( \#6 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
93: Vcc Target-6 ( \#6 ON ) 0.00 V ( 3.10 to 3.50 ) Result: 3.30 V ... >> OK <<
94: Vcc Target-7 ( \#6 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
95: Vcc Target-8 ( \#6 ON ) 3.30 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
96: Vcc Target-1 ( \#7 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
97: Vcc Target-2 ( \#7 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
98: Vcc Target-3 ( \#7 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
99: Vcc Target-4 ( \#7 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
100: Vcc Target-5 ( \#7 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
101: Vcc Target-6 ( \#7 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
102: Vcc Target-7 ( \#7 ON ) 0.00 V ( 3.10 to 3.50 ) Result: 3.26 V ... >> OK <<
103: Vcc Target-8 ( \#7 ON ) 3.30 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
104: Vcc Target-1 ( \#8 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
105: Vcc Target-2 ( \#8 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
106: Vcc Target-3 ( \#8 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
107: Vcc Target-4 ( \#8 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
108: Vcc Target-5 ( \#8 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
109: Vcc Target-6 ( \#8 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.25 V ... >> OK <<
110: Vcc Target-7 ( \#8 ON ) 0.00 V ( 0.00 to 0.50 ) Result: 0.26 V ... >> OK <<
111: Vcc Target-8 ( \#8 ON ) 3.30 V ( 3.10 to 3.50 ) Result: 3.23 V ... >> OK <<
112: BSL RX bus ( \#1 HIGH ) 0x01 ( 0x01 to 0x01 ) Result: 0x01 ... >> OK <<
113: BSL RX bus ( \#2 HIGH ) 0x02 ( 0x02 to 0x02 ) Result: 0x02 ... >> OK <<
114: BSL RX bus ( \#3 HIGH ) 0x04 ( 0x04 to 0x04 ) Result: 0x04 ... >> OK <<
115: BSL RX bus ( \#4 HIGH ) 0x08 ( 0x08 to 0x08 ) Result: 0x08 ... >> OK <<
116: BSL RX bus ( \#5 HIGH ) 0x10 ( 0x10 to 0x10 ) Result: 0x10 ... >> OK <<
117: BSL RX bus ( \#6 HIGH ) 0x20 ( 0x20 to 0x20 ) Result: 0x20 ... >> OK <<
118: BSL RX bus ( \#7 HIGH ) 0x40 ( 0x40 to 0x40 ) Result: 0x40 ... >> OK <<
119: BSL RX bus ( \#8 HIGH ) 0x80 ( 0x80 to 0x80 ) Result: 0x80 ... >> OK <<
120: BSL TX bus ( \#1 HIGH ) 0x01 ( 0x01 to 0x01 ) Result: 0x01 ... >> OK <<
121: BSL TX bus ( \#2 HIGH ) 0x02 ( 0x02 to 0x02 ) Result: 0x02 ... >> OK <<
122: BSL TX bus ( \#3 HIGH ) 0x04 ( 0x04 to 0x04 ) Result: 0x04 ... >> OK <<
123: BSL TX bus ( \#4 HIGH ) 0x08 ( 0x08 to 0x08 ) Result: 0x08 ... >> OK <<
124: BSL TX bus ( \#5 HIGH ) 0x10 ( 0x10 to 0x10 ) Result: 0x10 ... >> OK <<
125: BSL TX bus ( \#6 HIGH ) 0x20 ( 0x20 to 0x20 ) Result: 0x20 ... >> OK <<
126: BSL TX bus ( \#7 HIGH ) 0x40 ( 0x40 to 0x40 ) Result: 0x40 ... >> OK <<
127: BSL TX bus ( \#8 HIGH ) 0x80 ( 0x80 to 0x80 ) Result: 0x80 ... >> OK <<
128: TDI bus ( \#1 HIGH ) 0x01 ( 0x01 to 0x01 ) Result: 0x01 ... >> OK <<
129: TDI bus ( \#2 HIGH ) 0x02 ( 0x02 to 0x02 ) Result: 0x02 ... >> OK <<
130: TDI bus ( \#3 HIGH ) 0x04 ( 0x04 to 0x04 ) Result: 0x04 ... >> OK <<
131: TDI bus ( \#4 HIGH ) 0x08 ( 0x08 to 0x08 ) Result: 0x08 ... >> OK <<
132: TDI bus ( \#5 HIGH ) 0x10 ( 0x10 to 0x10 ) Result: 0x10 ... >> OK <<
133: TDI bus ( \#6 HIGH ) 0x20 ( 0x20 to 0x20 ) Result: 0x20 ... >> OK <<
134: TDI bus ( \#7 HIGH ) 0x40 ( 0x40 to 0x40 ) Result: 0x40 ... >> OK <<
135: TDI bus ( \#8 HIGH ) 0x80 ( 0x80 to 0x80 ) Result: 0x80 ... >> OK <<
136: TDIO Tx-bus ( \#1 HIGH ) 0x01 ( 0x01 to 0x01 ) Result: 0x01 ... >> OK <<
137: TDIO Tx-bus ( \#2 HIGH ) 0x02 ( 0x02 to 0x02 ) Result: 0x02 ... >> OK <<
138: TDIO Tx-bus ( \#3 HIGH ) 0x04 ( 0x04 to 0x04 ) Result: 0x04 ... >> OK <<
139: TDIO Tx-bus ( \#4 HIGH ) 0x08 ( 0x08 to 0x08 ) Result: 0x08 ... >> OK <<
140: TDIO Tx-bus ( \#5 HIGH ) 0x10 ( 0x10 to 0x10 ) Result: 0x10 ... >> OK <<
141: TDIO Tx-bus ( \#6 HIGH ) 0x20 ( 0x20 to 0x20 ) Result: 0x20 ... >> OK <<

```



```

142: TDIO Tx-bus (\#7 HIGH) 0x40 ( 0x40 to 0x40) Result: 0x40 ... >> OK <<
143: TDIO Tx-bus (\#8 HIGH) 0x80 ( 0x80 to 0x80) Result: 0x80 ... >> OK <<
144: TDIO Tx-Rx (\#1 HIGH) 0x01 ( 0x01 to 0x01) Result: 0x01 ... >> OK <<
145: TDIO Tx-Rx (\#2 HIGH) 0x02 ( 0x02 to 0x02) Result: 0x02 ... >> OK <<
146: TDIO Tx-Rx (\#3 HIGH) 0x04 ( 0x04 to 0x04) Result: 0x04 ... >> OK <<
147: TDIO Tx-Rx (\#4 HIGH) 0x08 ( 0x08 to 0x08) Result: 0x08 ... >> OK <<
148: TDIO Tx-Rx (\#5 HIGH) 0x10 ( 0x10 to 0x10) Result: 0x10 ... >> OK <<
149: TDIO Tx-Rx (\#6 HIGH) 0x20 ( 0x20 to 0x20) Result: 0x20 ... >> OK <<
150: TDIO Tx-Rx (\#7 HIGH) 0x40 ( 0x40 to 0x40) Result: 0x40 ... >> OK <<
151: TDIO Tx-Rx (\#8 HIGH) 0x80 ( 0x80 to 0x80) Result: 0x80 ... >> OK <<
152: TDIO Rx-bus (\#1 HIGH) 0x01 ( 0x01 to 0x01) Result: 0x01 ... >> OK <<
153: TDIO Rx-bus (\#2 HIGH) 0x02 ( 0x02 to 0x02) Result: 0x02 ... >> OK <<
154: TDIO Rx-bus (\#3 HIGH) 0x04 ( 0x04 to 0x04) Result: 0x04 ... >> OK <<
155: TDIO Rx-bus (\#4 HIGH) 0x08 ( 0x08 to 0x08) Result: 0x08 ... >> OK <<
156: TDIO Rx-bus (\#5 HIGH) 0x10 ( 0x10 to 0x10) Result: 0x10 ... >> OK <<
157: TDIO Rx-bus (\#6 HIGH) 0x20 ( 0x20 to 0x20) Result: 0x20 ... >> OK <<
158: TDIO Rx-bus (\#7 HIGH) 0x40 ( 0x40 to 0x40) Result: 0x40 ... >> OK <<
159: TDIO Rx-bus (\#8 HIGH) 0x80 ( 0x80 to 0x80) Result: 0x80 ... >> OK <<
160: VEXT bus (\#1 HIGH) 0x01 ( 0x01 to 0x01) Result: 0x01 ... >> OK <<
161: VEXT bus (\#2 HIGH) 0x02 ( 0x02 to 0x02) Result: 0x02 ... >> OK <<
162: VEXT bus (\#3 HIGH) 0x04 ( 0x04 to 0x04) Result: 0x04 ... >> OK <<
163: VEXT bus (\#4 HIGH) 0x08 ( 0x08 to 0x08) Result: 0x08 ... >> OK <<
164: VEXT bus (\#5 HIGH) 0x10 ( 0x10 to 0x10) Result: 0x10 ... >> OK <<
165: VEXT bus (\#6 HIGH) 0x20 ( 0x20 to 0x20) Result: 0x20 ... >> OK <<
166: VEXT bus (\#7 HIGH) 0x40 ( 0x40 to 0x40) Result: 0x40 ... >> OK <<
167: VEXT bus (\#8 HIGH) 0x80 ( 0x80 to 0x80) Result: 0x80 ... >> OK <<
168: VEXT bus (All-ON delay 10us) 0xFF ( 0xFF to 0xFF) Result: 0xFF ... >> OK <<
169: VEXT bus (All-ON delay 5 ms) 0xFF ( 0x00 to 0x00) Result: 0x00 ... >> OK <<
170: Keys buffer (All pull-up) 0x1F ( 0x1F to 0x1F) Result: 0x1F ... >> OK <<
171: Access to LCD RAM (0xAA) 0xAA ( 0xAA to 0xAA) Result: 0xAA ... >> OK <<
172: Access to LCD RAM (0x99) 0x99 ( 0x99 to 0x99) Result: 0x99 ... >> OK <<
173: Image Flash Access (get ID) 0x02 ( 0x01 to 0x02) Result: 0x01 ... >> OK <<
174: TDI Fuse keys (\#1 ON) 0x01 ( 0x01 to 0x01) Result: 0x01 ... >> OK <<
175: TDI Fuse keys (\#2 ON) 0x02 ( 0x02 to 0x02) Result: 0x02 ... >> OK <<
176: TDI Fuse keys (\#3 ON) 0x04 ( 0x04 to 0x04) Result: 0x04 ... >> OK <<
177: TDI Fuse keys (\#4 ON) 0x08 ( 0x08 to 0x08) Result: 0x08 ... >> OK <<
178: TDI Fuse keys (\#5 ON) 0x10 ( 0x10 to 0x10) Result: 0x10 ... >> OK <<
179: TDI Fuse keys (\#6 ON) 0x20 ( 0x20 to 0x20) Result: 0x20 ... >> OK <<
180: TDI Fuse keys (\#7 ON) 0x40 ( 0x40 to 0x40) Result: 0x40 ... >> OK <<
181: TDI Fuse keys (\#8 ON) 0x80 ( 0x80 to 0x80) Result: 0x80 ... >> OK <<
182: TEST Fuse keys (All OFF ) 0.00 ( 0.00 to 0.30) Result: 0.00 ... >> OK <<
183: TEST Fuse keys (\#1 ON) 1.00 ( 0.80 to 3.00) Result: 1.47 ... >> OK <<
184: TEST Fuse keys (\#2 ON) 2.00 ( 0.80 to 3.00) Result: 1.46 ... >> OK <<
185: TEST Fuse keys (\#3 ON) 3.00 ( 0.80 to 3.00) Result: 1.54 ... >> OK <<
186: TEST Fuse keys (\#4 ON) 4.00 ( 0.80 to 3.00) Result: 1.62 ... >> OK <<
187: TEST Fuse keys (\#5 ON) 5.00 ( 0.80 to 3.00) Result: 1.78 ... >> OK <<
188: TEST Fuse keys (\#6 ON) 6.00 ( 0.80 to 3.00) Result: 1.91 ... >> OK <<
189: TEST Fuse keys (\#7 ON) 7.00 ( 0.80 to 3.00) Result: 2.01 ... >> OK <<
190: TEST Fuse keys (\#8 ON) 8.00 ( 0.80 to 3.00) Result: 2.04 ... >> OK <<
===== Finished =====
* Test pass - no errors.

```

## 2.5 Label

Information and MSP Gang Programmer software and hardware can be displayed by accessing the About drop-down menu. Select the About→About option to display information similar to that shown in Figure 2-18.



**Figure 2-18. Information About the MSP Gang Programmer**

## 2.6 Benchmark

This section shows the result of a timing benchmark used on the MSP Gang Programmer to measure the programming speed for a MCU from the F5xx family. The specific target device is the:

MSP430F5438A, 256kB flash – code to be programmed is 256kB

[Table 2-1](#) shows the result of the benchmark when programming with the JTAG and SBW interfaces. Identical programming speed results when one or eight devices are programmed simultaneously. This is because programming each MCU is done in parallel.

**Table 2-1. Benchmark Results<sup>(1)</sup>**

Interface	Erase, Blank Check, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)	Verify Speed (kB/s)
JTAG - FAST	8.7	0.25	32	1000
JTAG - MED	15.8	0.28	17	900
JTAG - SLOW	31.3	0.36	8.5	700
SBW - FAST	27.4	0.34	9.7	750
SBW - MED	47.6	0.45	5.5	570
SBW - SLOW	99.0	0.72	2.6	350

<sup>(1)</sup> Results of performing programming tasks on the MSP430F5438 using the MSP Gang Programmer. Programming and verification speed of the older devices (F1xx, F2xx, or F4xx) are approximately two times slower.



### 3.1 Commands

The GANG430U can be controlled via 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 MSP-GANG
- Write Info memory to MSP-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

## 3.2 Firmware Interface Protocol

The MSP 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 MSP 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 MSP Gang Programmer, because it is several times faster than via 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 MSP-GANG Programmer the host serial handler transmits a SYNC (CR) character (0x0D) to the MSP-GANG Programmer. The MSP-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 send 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 MSP-GANG 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 MSP430 serial standard protocol (SSP) rules, extended with a preceding synchronization sequence (SS), as described in [Section 3.3](#). The MSP 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
- Acknowledge done by the MSP Gang Programmer is mandatory.



**Table 3-1. Data Frame for Firmware Commands<sup>(1)(2)</sup>**

GANG430 Firmware Command	PRO MPT	CMD	L1	L2	A1	A2	A3	A4	LL	LH	D1	D2...Dn	CLK	CLH	ACK
"Hello"	0D												-	-	ACK
Boot cmd disable	3E	2A	R	R	R	R	R	R	R	R	R	R	CKL	CKH	ACK
Boot cmd 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	-
Diagnostic response	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 FW 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	D7	D8	D9...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
Get Access Key St	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 Int.	3E	4A	4	4	R	R	-	-	R	R	-	-	CKL	CKH	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 Tmp Gang Cfg	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	

<sup>(1)</sup> All numbers are bytes in hexadecimal notation. ACK is sent by the MSP-GANG.

<sup>(2)</sup> PROMPT = 0x3E means data frame expected.



## 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" Command

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 MSP-GANG. When there is no response, then the baud rate should be changed. After power-up, the USB interface is used for communication with the MSP GANG; 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.

#### 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 info memory update. Use the MSP-GANG executable GUI software for updating firmware or Info 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 info memory update. Use the MSP-GANG executable GUI software for updating firmware or Info memory update if required.

#### 3.5.2.4 Get Diagnostic Command

The Get 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 "G430BOOT"

D21: Comma (,)

D22 to D30: Zero-terminated application firmware name "MSP-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 Select Baud Rate Command

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

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

### 3.5.2.6 Erase Firmware Command

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

Do not use this command. This command is used during firmware or info memory update. Use the MSP-GANG executable GUI software for updating firmware or Info memory update if required.

### 3.5.2.7 Load Firmware Command

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

Do not use this command. This command is used during firmware or info memory update. Use the MSP-GANG executable GUI software for updating firmware or Info memory update if required.

### 3.5.2.8 Exit from Firmware Update Command

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

Do not use this command. This command is used during firmware or info memory update. Use the MSP-GANG executable GUI software for updating firmware or Info memory update if required.

### 3.5.2.9 Get Label Command

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 "G430BOOT"
D27: Comma ','
D28-D36: Zero-terminated application firmware name "MSP-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 "G430"  
 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 MSP-GANG is busy with other tasks. It will reply to the command without interrupting the currently serviced process. Some commands that have the long execution time requires use the Get Progress Status command for monitoring the current state. For example, the Main Process command that can be executed a few seconds or more, responding with character "In Progress 0xB0" as fast as the command has been received and accepted. The communication link has been released and ready to use the Get Progress Status command. Now the current status and progress data can be monitored by polling the Get Progress Status command. Contents of the progress status contains current task number, chunk number, and information about what tasks have been already finished (erase, blank check, program, verify and more). Additionally, the comment 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 and for the status on the PC screen to appear the same as it is in the MSP-GANG LCD display. The internal firmware the progress status buffer is always updated when the 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 all 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 the 20-ms interval. The MSP-GANG has internal 8 level FIFO buffer for progress status (8 internal buffers - 50 bytes each). This allows messages to be retrieved even if status has been changed a few times in interval 20 ms, 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 get 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 MSP-GANG 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 MSP-GANG 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 / 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:

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:

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

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

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, 27 (Vcc status) provide 2 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 target device. All features provided by the MSP-GANG programmer and available in the MSP-GANG GUI and MSP-GANG DLL are accessible by these functions. Some of the commands that allows control of the MSP-GANG programmer are described in the following sections; however, commands that provide data transfer and script information between MSP-GANG and MSP-GANG DLL are not described here. Users should use the GUI software package (MSP-GANG executable and MSP-GANG 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 its is recommended to use the MSP-GANG DLL and control the MSP-GANG programmer via the DLL rather than directly via RS-232 or USB interface using the low-level communication protocol. The MSP-GANG DLL allows full control of the MSP-GANG programmer.



### 3.5.3.1 Select Image Command

```
Tx -> 3E 40 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 MSP-GANG performs use this image. The GANG430 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.

---

### 3.5.3.2 Main Process Command

```
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 via a polling technique for monitoring 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: what targets are connected, erased, etc. In the comment bytes (34-50) is the current process: the same message as is displayed on the LCD display.

### 3.5.3.3 Set Temporary Configuration in MSP-GANG Command

```
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, Hi/Low current, External Vcc enable/disable, Vcc settle time, communication interface (JTAG/SBW), enabled target devices and enable process mask (erase, program verify, etc.). The Set Temporary Configuration in MSP-GANG command allows to modify 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 MSP-GANG again before starting the Main Process command.

The Set Temporary Configuration in MSP-GANG 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

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 MSP-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/off

Data	0x0001	ON
Data	0x0000	OFF

### CFG\_TMP\_ICC\_HI\_EN (18)

High (50 mA) current from programmer enable/disable

Data	0x0001	Enable
Data	0x0000	Disable

### CFG\_TMP\_IO\_INTERFACE (20)

Set interface configuration

Data	0x0000	SBW via TDIO 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 Selected Status Command

```
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 Selected Status command gets the selected status or results from the MSP-Gang programmer. The following numbers (A1) are available:

GET_APP_FLAGS	10
GET_LAST_STATUS	12
GET_LAST_ERROR_NO	14

See the description of the MSPGANG\_GetAPIStatus function ([Section 4.2.43](#)) for details of the B0...Bn byte contents.

#### 3.5.3.5 Read from Gang Data Buffer Command

```
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 MSP-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 128 bytes for each target device - Buffer[8] [128];

T = Target device number, 1 to 8  
n = Number of bytes taken from the Buffer[T-1] [..]

#### 3.5.3.6 Write to Gang Data Buffer Command

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

Write bytes to selected target's Buffer -> Buffer[8] [128]

T = Target device number, 1 to 8  
n = Number of bytes written to Buffer[T-1] [..]

### 3.5.4 API Firmware Commands That Should Not be Used

#### 3.5.4.1 Interactive Process Command

```
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 Command

```
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 Read Info Memory from MSP-GANG Command

```
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 to MSP-GANG Info Memory Command

```
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 Command

```
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 Write to Image Block Command

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

The Write to Image Block command loads the data bytes into the image buffer of the MSP-GANG. Do not use this function in your application. Use MSP-GANG GUI and MSP-GANG DLL for writing data into the internal image buffer.

#### 3.5.4.7 Verify Image Check Sum Command

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

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

### 3.5.4.8 Read Image Header Command

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 Command

Tx -> 3E 4A 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.

---

### 3.5.4.10 Display Message on LCD Display Command

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 Self-Test Command

Tx -> 3E 71 ... .. 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.

---





## Dynamic Link Library for MSP-GANG Programmer

---

---

### 4.1 Gang430.dll Wrapper Description

The Gang430.dll wrapper allows application software prepared for the old MSP430 Gang programmer to control the new MSP-GANG programmer via the MSP-GANG.dll. Because the MSP-GANG programmer has different functionality and features than the old MSP430 Gang Programmer, not all features provided in the old programmer are supported in the same way by the MSP-GANG programmer. The Gang430.dll wrapper allows an easy transition to the new programmer when using an old application, but it is recommended to use MSP-GANG.dll for remote control of the MSP-GANG programmer to have access to all features provided by the programmer.

When Gang430.dll is used, the following files must be located in the same directory where the application software is located:

- Gang430.dll - DLL wrapper with the same name as the previous Gang430.dll
- Gang430.ini - Initialization file for compatibility with the old structure
- MSP-GANG.dll - New DLL that has access to MSP-GANG Programmer

Examples of using the GANG430.dll as a wrapper around the new MSP-Gang.dll are provided and can be found in these locations (if the default installation directory was used):

C:\Program Files\Texas Instruments\MSP-GANG\Examples\C\_Applications\_Wrapper  
and  
C:\Program Files\Texas Instruments\MSP-GANG\Examples\Cpp\_Applications\_Wrapper

To use these examples, also copy the MSG-Gang.dll into the working directory.

#### Limitation

The MSP-GANG works in interactive mode. The image is not saved in the memory; however, the save image option must be used as it is in the old Gang430.dll. An image is saved inside the DLL only (very fast) and used when the "Start" command is executed. If USB communication is used, then programming is fast. RS-232 communication is, of course, slower than USB, but it still faster than the previous MSP430 Gang Programmer.

See the *MSP430 Gang Programmer (MSP-GANG430) User's Guide* ([SLAU101](#)) for list of commands used in Gang430.dll.

### 4.2 MSP-GANG.dll Description

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

- MSP-GANG.h: This file is the header file for the MSP-GANG.dll, and provides the function prototypes, typedefs, #defines, and data structures for the functions of the MSP-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.
- MSP-GANG.lib: This file is the library file for the MSP-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 MSP-GANG DLL functions have the same "MSPGANG\_" prefix in the function name. It is easy in the application software determine what functions are used with the MSP-GANG.dll. The following sections describe each function.

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

```
C:\Program Files\Texas Instruments\MSP-GANG\Examples\C_Applications_MSP_DLL
and
C:\Program Files\Texas Instruments\MSP-GANG\Examples\Cpp_Applications_MSP_DLL
```

These examples show how to configure the MSP-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 MSP-Gang.dll into the working directory.

#### 4.2.1 MSPGANG\_GetDataBuffers\_ptr

MSPGANG\_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    0x50000
#define JTAG_PASSW_LEN  0x80
#define BSL_PASSW_LEN   0x20
#define FLASH_END_ADDR  (DBUFFER_SIZE-1)
#define FLASH_BUF_LEN   DBUFFER_SIZE
#define GANG_SIZE       8

typedef struct
{
    BYTE SourceCode[DBUFFER_SIZE];           // source code from the file
    BYTE UsedCode[DBUFFER_SIZE];             // combined data (source code, serialization etc)
    BYTE CommonTx[DBUFFER_SIZE];            // data used to writing - same data to all targets
    BYTE GangTx[DBUFFER_SIZE][GANG_SIZE];   // selective data used to writing
    BYTE GangRx[DBUFFER_SIZE][GANG_SIZE];   // data read from all targets
    BYTE Tmp[DBUFFER_SIZE];
    BYTE JTAG_Passsw[2][JTAG_PASSW_LEN];
    BYTE BSL_Passsw[2][BSL_PASSW_LEN];
    BYTE Flag_ScrCode[DBUFFER_SIZE];        // 0-empty 1-valid data in SourceCode[x]
    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_JTAG_Passw[2][JTAG_PASSW_LEN]; // spare
    BYTE Flag_BSL_Passw[2][BSL_PASSW_LEN];  // 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;
MSPGANG_GetDataBuffers_ptr((&temp));
DBuf = (DATA_BUFFERS *)temp;
```

#### Syntax

```
LONG MSPGANG_GetDataBuffers_ptr(void ** x)
```

### 4.2.2 MSPGANG\_SetGangBuffer, MSPGANG\_GetGangBuffer

The MSP-GANG Programmer contains a temporary data buffer that can be used for writing and reading data to each target device. Buffer size is 128 bytes for each target device.

```
Buffer[8] [128];
```

MSPGANG\_SetGangBuffer writes data to selected Buffer. MSPGANG\_GetGangBuffer reads contents from the selected buffer.

#### Syntax

```
LONG MSPGANG_SetGangBuffer(BYTE target, BYTE size, BYTE *data)  
LONG MSPGANG_GetGangBuffer(BYTE target, BYTE size, BYTE *data)
```

#### Arguments

BYTE target	Target number (1 to 8)
BYTE size	Size of data (1 to 128)
BYTE *data	Pointer to data buffer from where data is taken or to where the data should be saved

#### Result

LONG	Error code
------	------------

### 4.2.3 MSPGANG\_GetDevice

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

#### Syntax

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

#### Arguments

LPTSTR lpszDeviceName	MCU name. The device name; for example, 'MSP430F5438A' for desired MCU or (blank) for currently selected MCU
void *lpData	Pointer to internal structure

#### Result

LONG      Error code

```
typedef struct
{
    long Group;
    long IsFRAM;
    long RAM_size;
    long no_of_info_segm;
    long info_segm_size;
    long info_start_addr;
    long info_end_addr;
    long info_A_locked;
    long MainMem_start_addr;
    long MainMem_end_addr;
    long no_of_BSL_segm;
    long BSL_segm_size;
    long BSL_start_addr;
    long BSL_end_addr;
    long Vcc_prgr_min;
    long Vcc_run_min;
    long BSL_passw_size;
    long family_index;
    long has_JTAG_password;
    long spare[25];
} DEVICE_INFO;
```

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

```
DEVICE_INFO *Device;
void *temp;
    MSPGANG_GetDevice('', &temp);
    Device = (DEVICE_INFO *)temp;
```

#### 4.2.4 MSPGANG\_LoadFirmware

Load firmware from MSP-GANG.dll to MSP-GANG Programmer,

---

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

---

##### Syntax

LONG MSPGANG\_LoadFirmware(void)

#### 4.2.5 MSPGANG\_InitCom

MSPGANG\_InitCom opens a communications port, sets the baudrate and checks if the MSP-GANG Programmer is present.

##### Syntax

LONG MSPGANG\_InitCom(LPTSTR lpszPort, LONG lBaud)

##### Arguments

char * lpszComPort	Name of the port
LONG lBaudRate	Baud rate

##### Result

LONG      Error code

#### 4.2.6 MSPGANG\_ReleaseCom

Release communications port

##### Syntax

LONG MSPGANG\_ReleaseCom(void)

##### Arguments

None

##### Result

LONG      Error code

### 4.2.7 MSPGANG\_GetErrorString

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

#### Syntax

```
LPTSTR MSPGANG_GetErrorString(LONG lErrorNumber)
```

#### Arguments

LONG lErrorNumber    Error number

#### Result

LPTSTR    Error string

### 4.2.8 MSPGANG\_SelectBaudrate

MSPGANG\_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 MSPGANG_SelectBaudrate(LONG lBaud)
```

#### Arguments

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

#### Result

LONG            Error code

### 4.2.9 MSPGANG\_GetDiagnostic

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

#### Syntax

```
LONG MSPGANG_GetDiagnostic(void **lpData)
```

#### Arguments

void \*\* lpData        Pointer to data buffer

#### Result

LONG            Error code



#### 4.2.10 MSPGANG\_MainProcess

MSPGANG\_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 MSPGANG_MainProcess(LONG timeout)
```

##### Arguments

LONG timeout      In seconds

##### Result

LONG      Error code

#### 4.2.11 MSPGANG\_InteractiveProcess

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

##### Syntax

```
LONG MSPGANG_InteractiveProcess(LONG timeout)
```

##### Arguments

LONG timeout      In seconds

##### Result

LONG      Error code

#### 4.2.12 MSPGANG\_Interactive\_Open\_Target\_Device

MSPGANG\_Interactive\_Open\_Target\_Device is used in the interactive mode and in initializing access to target devices (setting Vcc, checking fuse, and initializing JTAG/SBW 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 MSPGANG_Interactive_Open_Target_Device(LPTSTR name)
```

##### Arguments

LPTSTR name

##### Result

LONG      Error code

#### 4.2.13 MSPGANG\_Interactive\_Close\_Target\_Device

MSPGANG\_Interactive\_Close\_Target\_Device is used in the interactive mode and in closing access to target devices.

##### Syntax

```
LONG MSPGANG_Interactive_Close_Target_Device(void)
```

##### Result

LONG      Error code

#### 4.2.14 MSPGANG\_Interactive\_DefReadTargets

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_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 MSPGANG_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

#### 4.2.15 MSPGANG\_Interactive\_ReadTargets

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_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 MSPGANG_Interactive_ReadTargets(BYTE mask)
```

##### Arguments

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

##### Result

LONG      Error code

#### 4.2.16 MSPGANG\_Interactive\_ReadBytes

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_Interactive\_ReadBytes reads contents from one selected target device and saves it in the desired data buffer.

##### Syntax

```
LONG MSPGANG_Interactive_ReadBytes(BYTE target_no, LONG addr, LONG size, BYTE *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 bytes
BYTE *data	Pointer to buffer where data would be saved

##### Result

LONG      Error code

#### 4.2.17 MSPGANG\_Interactive\_WriteWord\_to\_RAM

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_Interactive\_WriteWord\_to\_RAM writes one word (16 bits) to any RAM or I/O location. The address must be even.

##### Syntax

```
LONG MSPGANG_Interactive_WriteWord_to_RAM(LONG addr, LONG data)
```

##### Arguments

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

##### Result

LONG      Error code

#### 4.2.18 MSPGANG\_Interactive\_WriteByte\_to\_RAM

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_Interactive\_WriteByte\_to\_RAM writes one byte to any RAM or I/O location.

##### Syntax

```
LONG MSPGANG_Interactive_WriteByte_to_RAM(LONG addr, BYTE data)
```

##### Arguments

LONG addr	RAM address location
BYTE data	Data (8 bits)

##### Result

LONG	Error code
------	------------

#### 4.2.19 MSPGANG\_Interactive\_WriteBytes\_to\_RAM

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_Interactive\_WriteBytes\_to\_RAM writes 'size' number of bytes to any RAM or I/O location. The starting address must be even.

##### Syntax

```
LONG MSPGANG_Interactive_WriteBytes_to_RAM(LONG addr, LONG size, BYTE * data)
```

##### Arguments

LONG addr	RAM address location
LONG size	Number of bytes to be written
BYTE * data	Data block

##### Result

LONG	Error code
------	------------

#### **4.2.20 MSPGANG\_Interactive\_WriteBytes\_to\_FLASH**

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_Interactive\_WriteBytes\_to\_FLASH writes 'size' number of bytes to any flash location. The starting address must be even.

##### **Syntax**

```
LONG MSPGANG_Interactive_WriteBytes_to_FLASH(LONG addr, LONG size, BYTE * data)
```

##### **Arguments**

LONG addr	RAM address location
LONG size	Number of bytes to be written
BYTE * data	Data block

##### **Result**

LONG	Error code
------	------------

#### **4.2.21 MSPGANG\_Interactive\_Copy\_Gang\_Buffer\_to\_RAM**

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_Interactive\_Copy\_Gang\_Buffer\_to\_RAM writes 'size' number of bytes from the internal Gang\_Buffer[8][128] 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 send to MSP-GANG first. See MSPGANG\_GetGangBuffer and MSPGANG\_SetGangBuffer functions for details.

##### **Syntax**

```
LONG MSPGANG_Interactive_Copy_GANG_Buffer_to_RAM(LONG addr, LONG size)
```

##### **Arguments**

LONG addr	RAM address location
LONG size	Number of bytes to be written (up to 128)

##### **Result**

LONG	Error code
------	------------

#### 4.2.22 MSPGANG\_Interactive\_Copy\_Gang\_Buffer\_to\_FLASH

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_Interactive\_Copy\_Gang\_Buffer\_to\_FLASH writes 'size' number of bytes from the internal Gang\_Buffer[8][128] 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 send to MSP-GANG first. See MSPGANG\_GetGangBuffer and MSPGANG\_SetGangBuffer functions for details.

##### Syntax

```
LONG MSPGANG_Interactive_Copy_GANG_Buffer_to_FLASH(LONG addr, LONG size)
```

##### Arguments

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

##### Result

LONG	Error code
------	------------

#### 4.2.23 MSPGANG\_Interactive\_EraseSectors

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_Interactive\_EraseSectors erases flash sectors starting from the sector with address location StartAddr and ending with the sector with EndAddr location.

##### Syntax

```
LONG MSPGANG_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
------	------------

#### 4.2.24 MSPGANG\_Interactive\_BlankCheck

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_Interactive\_BlankCheck verifies all flash contents starting from StartAddr and ending with EndAddr are 0xFF.

##### Syntax

```
LONG MSPGANG_Interactive_BlankCheck(LONG StartAddr, LONG EndAddr)
```

##### Arguments

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

##### Result

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

#### 4.2.25 MSPGANG\_Interactive\_DCO\_Test

Note: The target device must be opened first if not open yet (see MSPGANG\_Interactive\_Open\_Target\_Device, [Section 4.2.12](#)).

MSPGANG\_Interactive\_DCO\_Test takes data from INFO memory location 0x10F8 to 0x10FF, writing one selected word to DCO registers and checking the DCO frequency in real time for up to eight targets simultaneously. Test results in kHz are saved in the \*result\_in\_kHz buffer.

##### Syntax

```
LONG MSPGANG_Interactive_DCO_Test(BYTE DCO_no, LONG *result_in_kHz);
```

##### Arguments

BYTE DCO no	DCO number data taken from the Info memory. 0 = data for DCO taken from 0x10FE 1 = data for DCO taken from 0x10FC 2 = data for DCO taken from 0x10FA 3 = data for DCO taken from 0x10F8
LONG * results	Pointer to long buffer size for 8 targets (LONG DCO[8])

##### Result

LONG	Error code
------	------------



#### 4.2.26 MSPGANG\_SelectImage

MSPGANG\_SelectImage sets an active image to work with. MSP-GANG supports 16 images.

##### Syntax

```
LONG MSPGANG_SelectImage(LONG lImage)
```

##### Arguments

LONG lImage      Image number (0 to 15)

##### Result

LONG      Error code

#### 4.2.27 MSPGANG\_EraseImage

MSPGANG\_EraseImage clears (presets with 0xFF) active image memory. Use the MSPGANG\_SelectImage function to select desired image memory.

##### Syntax

```
LONG MSPGANG_EraseImage(void)
```

##### Result

LONG      Error code

#### 4.2.28 MSPGANG\_CreateGangImage

MSPGANG\_CreateGangImage creates a command script and the data to be written to target devices according to current MSP-GANG configuration. After the image data is prepared, then it can be saved in the selected image memory by calling the MSPGANG\_LoadImageBlock function.

##### Syntax

```
LONG MSPGANG_CreateGangImage(LPTSTR name)
```

##### Arguments

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

##### Result

LONG      Error code

### 4.2.29 MSPGANG\_LoadImageBlock

MSPGANG\_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:

```
MSPGANG_CreateGangImage(name);  
MSPGANG_SelectImage(lImage);  
MSPGANG_EraseImage();  
MSPGANG_LoadImageBlock();  
MSPGANG_VerifyPSAImageBlock();
```

#### Syntax

```
LONG MSPGANG_LoadImageBlock(void)
```

#### Arguments

None

#### Result

LONG      Error code

### 4.2.30 MSPGANG\_VerifyPSImageBlock

MSPGANG\_VerifyPSAImageBlock verifies the checksum of all blocks used in the selected image memory. The image memory number should be selected first using MSPGANG\_SelectImage function.

#### Syntax

```
LONG MSPGANG_LoadImageBlock(void)
```

#### Arguments

None

#### Result

LONG      Error code

### 4.2.31 MSPGANG\_ReadImageBlock

MSPGANG\_ReadImageBlock reads the header from the selected image memory. A maximum of 254 bytes can be read. Access to the remaining image memory (up to 512 kBytes) is blocked.

#### Syntax

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

#### Arguments

LONG address

LONG size

void \*lpData      Pointer to byte buffer where the result is saved

#### Result

LONG      Error code

### 4.2.32 MSPGANG\_Read\_Code\_File

MSPGANG\_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 MSPGANG\_SetConfig function.

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

When the MSPGANG\_Read\_Code\_File is executed, the flag set by MSPGANG\_SetConfig(CFG\_OPEN\_FILE\_TYPE, CODE\_FILE\_INDEX) is set to the default value of Read Code File.

#### Syntax

```
LONG MSPGANG_Read_Code_File(LPTSTR FullPath)
```

#### Arguments

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

#### Result

LONG Error code

### 4.2.33 MSPGANG\_Save\_Config, MSPGANG\_Load\_Config, MSPGANG\_Default\_Config

The current configuration file can be saved using the MSPGANG\_Save\_Config function and recalled when required using the MSPGANG\_Load\_Config function. The current configuration can be erased and the default configuration loaded by calling the MSPGANG\_Default\_Config function. When the new configuration is loaded, some of the parameters can be modified item-by-item using MSPGANG\_Set\_Config and can be read from the configuration item-by-item using MSPGANG\_GetConfig. The MSP-GANG configuration can also be created using the MSP-GANG GUI software (MSP-GANG-exe) by setting desired programmer setup, verifying if all works, then saving the configuration using the Save Setup as... option. The setup used in the GUI can be restored in the DLL when the above mentioned configuration file is downloaded using MSPGANG\_Load\_Config function.

#### Syntax

```
LONG MSPGANG_Save_Config(LPTSTR filename)
LONG MSPGANG_Load_Config(LPTSTR filename)
LONG MSPGANG_Default_Config(void)
```

#### Arguments

LPTSTR filename Path to the configuration file

#### Result

LONG Error code

### 4.2.34 MSPGANG\_SetConfig, MSPGANG\_GetConfig

#### Syntax

```
LONG MSPGANG_SetConfig(LONG index, LONG data)
```

#### Arguments

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

#### Result

LONG	Error code
------	------------

#### Syntax

```
LONG MSPGANG_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
	//INTERFACE_NONE (0)	
	//INTERFACE_JTAG (4)	
	//INTERFACE_SBW (8)	
	//INTERFACE_BSL (0xC)	
#define	CFG_JTAG_SPEED	1
	//INTERFACE_FAST (0)	
	//INTERFACE_MED (1)	
	//INTERFACE_SLOW (2)	
#define	CFG_SBW_SPEED	2
	//INTERFACE_FAST (0)	
	//INTERFACE_MED (1)	
	//INTERFACE_SLOW (2)	
#define	CFG_BSL_SPEED	3
	//INTERFACE_FAST (0)	
	//INTERFACE_MED (1)	
	//INTERFACE_SLOW (2)	
#define	CFG_IO_INTERFACE	4
	//SBW_VIA_TDOI_BIT (0)	
	//SBW_VIA_RST_BIT (1)	
#define	CFG_POWERTARGETEN	6
	//disable (0)	
	//enable (1)	
#define	CFG_VCCINDEX	7
	//Vcc in mV 0...3600	
#define	CFG_ICC_HI_EN	8
	//disable (0)	
	//enable (1)	
#define	CFG_BLOWFUSE	9
	//disable (0)	
	//enable (1)	
#define	CFG_TARGET_EN_INDEX	10
	//gang mask - one bit for each target	
	// target # 1 0x01	

```

        // target # 2 0x02
        // target # 3 0x04
        // target # 8 0x80
        // eg: enable all targets -> 0xFF
#define CFG_IMG_INTERFACE (0 + 0x1000)
#define CFG_IMG_JTAG_SPEED (1 + 0x1000)
#define CFG_IMG_SBW_SPEED (2 + 0x1000)
#define CFG_IMG_BSL_SPEED (3 + 0x1000)
#define CFG_IMG_IO_INTERFACE (4 + 0x1000)
#define CFG_IMG_POWERTARGETEN (6 + 0x1000)
#define CFG_IMG_VCCINDEX (7 + 0x1000)
#define CFG_IMG_ICC_HI_EN (8 + 0x1000)
#define CFG_IMG_BLOWFUSE (9 + 0x1000)
#define CFG_IMG_TARGET_EN_INDEX (10 + 0x1000)
#define CFG_FLASHERASEMODE 11
        //ERASE_NONE_MEM_INDEX (0)
        //ERASE_ALL_MEM_INDEX (1)
        //ERASE_PRG_ONLY_MEM_INDEX (2)
        //ERASE_INFILE_MEM_INDEX (3)
        //ERASE_DEF_CM_INDEX (4)
#define CFG_ERASEINFOA 12
        //disable (0)
        //enable (1)
#define CFG_ERASEINFOB 13
        //disable (0)
        //enable (1)
#define CFG_ERASEINFOC 14
        //disable (0)
        //enable (1)
#define CFG_ERASEINFOD 15
        //disable (0)
        //enable (1)
#define CFG_MASSERASE_AND_INFOA_EN 16
        //disable (0)
        //enable (1)
#define CFG_ERASESTARTADDR 17
        //FLASH Start Address
#define CFG_ERASESTOPADDR 18
        //FLASH End Address
#define CFG_FLASHREADMODE 19
        //READ_ALL_MEM_INDEX (0)
        //READ_PRGMEM_ONLY_INDEX (1)
        //READ_INFOMEM_ONLY_INDEX (2)
        //READ_DEF_MEM_INDEX (3)
#define CFG_READINFOA 20
        //disable (0)
        //enable (1)
#define CFG_READINFOB 21
        //disable (0)
        //enable (1)
#define CFG_READINFOC 22
        //disable (0)
        //enable (1)
#define CFG_READINFOD 23
        //disable (0)
        //enable (1)
#define CFG_FINALACTION_MODE 24
        //APPLICATION_NO_RESET (0)
        //APPLICATION_TOGGLE_RESET (1)
        //APPLICATION_TOGGLE_VCC (2)
        //APPLICATION_JTAG_RESET (3)
#define CFG_BEEPMODE 25
        //BEEP_PCSPK_EN_BIT (1)
        //BEEP_OK_EN_BIT (2)
        //BEEP_SOUND_EN_BIT (4)
#define CFG_DEFERASEMAINEN 26

```

```

        //disable (0)
        //enable (1)
#define CFG_CUSTOMRESETPULSETIME 27
        //time in ms
#define CFG_CUSTOMRESETIDLETIME 28
        //time in ms
#define CFG_RETAIN_CAL_DATA_INDEX 31
        //disable (0)
        //enable (1)
#define CFG_FINALACTIONRUNTIME 32
        //time in seconds
#define CFG_FINALACTIONVCCOFFTIME 33
        //time in ms
#define CFG_DCO_CONST_2XX_VERIFY_EN 35
        //disable (0)
        //enable (1)
#define CFG_DCOCAL_2XX_EN 36
        //disable (0)
        //enable (1)
#define CFG_BSL_FLASH_WR_EN 37
        //disable (0)
        //enable (1)
#define CFG_BSL_FLASH_RD_EN 38
        //disable (0)
        //enable (1)
#define CFG_READMAINMEMEN 39
        //disable (0)
        //enable (1)
#define CFG_READDEFSTARTADDR 40
        //Read Start Address
#define CFG_READDEFSTOPADDR 41
        //Read Start Address
#define CFG_COMPORT_NO 42
        //COM port - number 0...255
        //COM1 -> has number 0
        //.....
        //COM256 -> has number 255
#define CFG_UART_SPEED 43
        //9600 baud (0) (default)
#define CFG_OPEN_FILE_TYPE 44
        //CODE_FILE_INDEX (0)
        //APPEND_FILE_INDEX (1)
        //PASSW_FILE_INDEX (2)
        //SECONDCODE_FILE_INDEX (3)
#define CFG_IMAGE_NO 46
        //image # 0...15
#define CFG_RESETTIME 47
        //RESET_10MS_INDEX (0)
        //RESET_100MS_INDEX (1)
        //RESET_200MS_INDEX (2)
        //RESET_500MS_INDEX (3)
        //RESET_CUSTOM_INDEX (4)
#define CFG_VCC_SETTLE_TIME 51
        //(settle time in ms)/20
        //settle time 0...200 (0...4000ms)

```

### 4.2.35 MSPGANG\_GetNameConfig, MSPGANG\_SetNameConfig

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

#### Syntax

```
LPTSTR MSPGANG_GetNameConfig(LONG index)
```

#### Arguments

LONG index      See list of indexes below

#### Result

LPTSTR          File name

#### Syntax

```
LONG MSPGANG_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
```



### 4.2.36 MSPGANG\_SetTmpGANG\_Config

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

#### Syntax

```
LONG MSPGANG_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
```

### 4.2.37 MSPGANG\_GetLabel

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

#### Syntax

```
LONG MSPGANG_GetLabel(BYTE *Data)
```

#### Arguments

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

#### Result

LONG            Error code

#### 4.2.38 MSPGANG\_GetInfoMemory, MSPGANG\_SetInfoMemory

Reads or writes 128 bytes 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 MSPGANG_GetInfoMemory(BYTE page, BYTE *data)  
LONG MSPGANG_SetInfoMemory(BYTE page, BYTE *data)
```

##### Arguments

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

##### Result

LONG	Error code
------	------------

#### 4.2.39 MSPGANG\_Get\_qty\_MCU\_Family, MSPGANG\_Get\_MCU\_FamilyName, MSPGANG\_Check\_MCU\_Name, MSPGANG\_Get\_MCU\_Name

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

##### Syntax

```
LONG MSPGANG_Get_qty_MCU_Family(void)
LONG MSPGANG_Get_MCU_FamilyName(LONG index, LPTSTR name)
LONG MSPGANG_Check_MCU_Name(LPTSTR name)
LONG MSPGANG_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 = MSPGANG_Get_qty_MCU_Family(); //get no of MCU groups
for(k=0; k<n; k++)
{
    P = MSPGANG_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:

```
{ 1, "MSP430F1xx" },
{ 20, "MSP430F2xx" },
{ 22, "MSP430AFE2xx" },
{ 21, "MSP430G2xx" },
{ 40, "MSP430F4xx" },
{ 41, "MSP430FE4xx" },
{ 42, "MSP430FG4xx" },
{ 43, "MSP430FW4xx" },
{ 50, "MSP430F5xx" },
{ 57, "MSP430FR5xx" },
{ 60, "MSP430F6xx" },
{ 51, "CC-430F5xx" },
{ 61, "CC-430F6xx" }
```

List of the MCU names in selected group can be taken as follows (as an example - list of the MCUs from the MSP430F5xx group (group number 50)):

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

#### 4.2.40 MSPGANG\_Set\_MCU\_Name

The MSPGANG\_Set\_MCU\_Name allows to select desired target MCU.

##### Syntax

```
LONG MSPGANG_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

#### 4.2.41 MSPGANG\_HW\_devices

The MSPGANG\_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;
MSPGANG_HW_devices(MAX_COM_SIZE, (void **) &AvailableComPorts);
```

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

##### Syntax

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

##### Arguments

LONG max  
void \*\*AvailableComPorts

##### Result

LONG            Error code

#### 4.2.42 MSPGANG\_GetProgressStatus

MSPGANG\_GetProgressStatus gets progress status from MSP-GANG. 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 7 → Target 8

For example, when the `connected_gang_mask == 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 MSPGANG_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;
    WORD          chunk_ctr;
    BYTE          run;
    BYTE          ack;
    WORD          Finished_tasks_mask;
                 //--- 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
                 // DCO_CAL_TASK_BIT       0x0040
                 // spare                   0x0080 to 0x4000
                 // RST_AND_START_FW_BIT   0x8000
    BYTE          cumulative;
                 //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          Connected_gang_mask;
    BYTE          Erased_gang_mask;
    BYTE          BlankCheck_gang_mask;
    BYTE          Programmed_gang_mask;
    BYTE          Verified_gang_mask;
    BYTE          Secured_gang_mask;
    BYTE          spare[6];
    BYTE          error_no;
    BYTE          VTIO_32mV;
  }
}
```

```
BYTE VccSt_LOW;
BYTE VccSt_HI;
    // 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 0.7 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
BYTE VccErr;
    // current Vcc below min
BYTE VccErr_Cumulative;
    // Cumulative (during programming) Vcc below min
BYTE JTAG_init_err_mask;
BYTE JTAG_Fuse_already_blowed_mask;
BYTE Wrong_MCU_ID_mask;
BYTE Progress_bar;
    // 0...100%
char comment[SCRIPT_TEXT_SIZE];
}st;
};
```

### 4.2.43 MSPGANG\_GetAPIStatus

MSPGANG\_GetAPIStatus gets the selected status or results from the MSP-Gang programmer. The following numbers (no) are available:

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

#### Syntax

```
LONG MSPGANG_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 MSP-GANG for any command
             error numbers 1...255 - see error list numbers
```



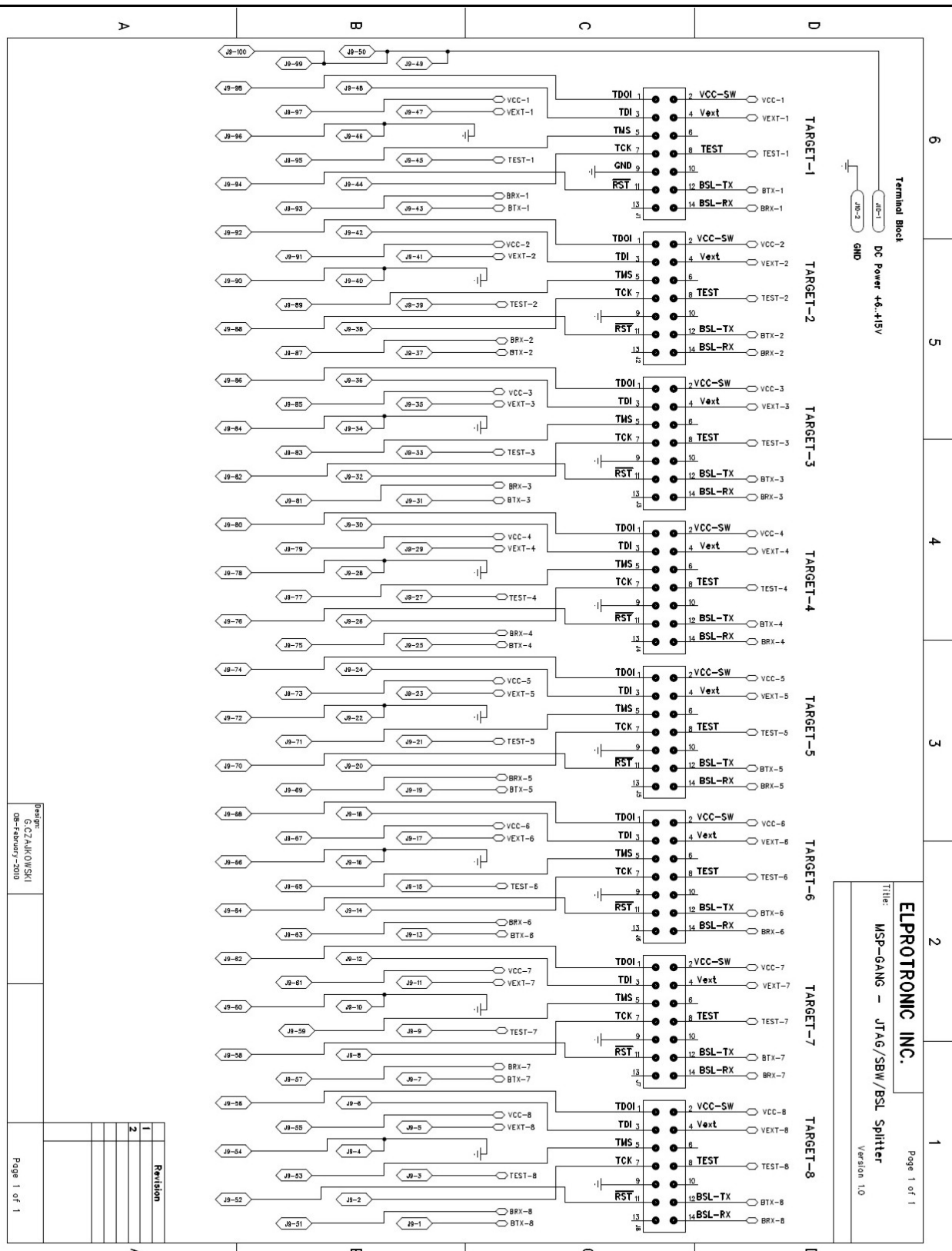


## **Schematics**

---

---

---



Gang430v2\_Splitter.sch-1 - Tue Jun 28 16:04:11 2011

Figure 5-1. Gang Splitter Schematic



**Revision History**

SLAU358	Initial release
---------	-----------------

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

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

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

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

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

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

### Products

Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>
DLP® Products	<a href="http://www.dlp.com">www.dlp.com</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>
Clocks and Timers	<a href="http://www.ti.com/clocks">www.ti.com/clocks</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>
RFID	<a href="http://www.ti-rfid.com">www.ti-rfid.com</a>
OMAP Mobile Processors	<a href="http://www.ti.com/omap">www.ti.com/omap</a>
Wireless Connectivity	<a href="http://www.ti.com/wirelessconnectivity">www.ti.com/wirelessconnectivity</a>

### Applications

Communications and Telecom	<a href="http://www.ti.com/communications">www.ti.com/communications</a>
Computers and Peripherals	<a href="http://www.ti.com/computers">www.ti.com/computers</a>
Consumer Electronics	<a href="http://www.ti.com/consumer-apps">www.ti.com/consumer-apps</a>
Energy and Lighting	<a href="http://www.ti.com/energy">www.ti.com/energy</a>
Industrial	<a href="http://www.ti.com/industrial">www.ti.com/industrial</a>
Medical	<a href="http://www.ti.com/medical">www.ti.com/medical</a>
Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Space, Avionics and Defense	<a href="http://www.ti.com/space-avionics-defense">www.ti.com/space-avionics-defense</a>
Transportation and Automotive	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
Video and Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>

TI E2E Community Home Page

[e2e.ti.com](http://e2e.ti.com)

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2011, Texas Instruments Incorporated