

# **MSP Gang Programmer (MSP-GANG)**

## **User's Guide**



Literature Number: SLAU358P  
September 2011–Revised February 2019

<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 MSP Flash Devices Using the MSP Gang Programmer .....	13
2.1.1 Programming Using Interactive Mode .....	14
2.1.2 Programming From Image .....	20
2.1.3 Programming From Script.....	24
2.1.4 Programming in Standalone Mode.....	30
2.1.5 Memory Setup for GO, Erase, Program, Verify, and Read .....	33
2.1.6 Secure Device Setup and Memory Protection .....	35
2.1.7 Programming MCU With IP Encapsulated Segment .....	37
2.1.8 Serialization .....	38
2.1.9 Creating and Using Images .....	40
2.1.10 Programming From Image File .....	44
2.1.11 Programming From SD Card .....	45
2.1.12 File Extensions .....	45
2.1.13 Checksum Calculation.....	46
2.1.14 Commands Combined With the Executable File.....	46
2.2 Data Viewers.....	47
2.3 Status Messages .....	49
2.4 Self Test .....	53
2.5 Label .....	58
2.6 Preferences .....	59
2.6.1 USB ID Number .....	59
2.6.2 COM Port.....	59
2.6.3 LCD Contrast .....	59
2.6.4 Checksum – Gang430 Standard .....	59
2.7 Benchmarks.....	60
2.7.1 Benchmarks for MSP430F5xx.....	60
2.7.2 Benchmarks for MSP430FR5xx .....	61
2.7.3 Benchmarks for MSP430F2xx.....	61
2.7.4 Benchmarks for MSP432P401R .....	62
<b>3 Firmware .....</b>	<b>63</b>
3.1 Commands .....	63
3.2 Firmware Interface Protocol.....	64
3.3 Synchronization Sequence.....	64
3.4 Command Messages .....	64
3.4.1 Frame Structure .....	64
3.4.2 Checksum .....	66
3.5 Detailed Description of Commands .....	66
3.5.1 General .....	66
3.5.2 Commands Supported by the BOOT Loader.....	66

3.5.3	Commands Supported by Application Firmware .....	70
3.5.4	API Firmware Commands That Should Not be Used .....	73
<b>4</b>	<b>Dynamic Link Library for MSP-GANG Programmer .....</b>	<b>77</b>
4.1	Gang430.dll Wrapper Description .....	77
4.2	MSP-GANG.dll Description .....	77
4.2.1	MSPGANG_GetDataBuffers_ptr .....	78
4.2.2	MSPGANG_SetGangBuffer, MSPGANG_GetGangBuffer .....	79
4.2.3	MSPGANG_GetDevice .....	81
4.2.4	MSPGANG_LoadFirmware .....	83
4.2.5	MSPGANG_InitCom .....	83
4.2.6	MSPGANG_ReleaseCom .....	83
4.2.7	MSPGANG_GetErrorString .....	84
4.2.8	MSPGANG_SelectBaudrate .....	84
4.2.9	MSPGANG_GetDiagnostic .....	84
4.2.10	MSPGANG_MainProcess .....	85
4.2.11	MSPGANG_InteractiveProcess .....	85
4.2.12	MSPGANG_Interactive_Open_Target_Device .....	85
4.2.13	MSPGANG_Interactive_Close_Target_Device .....	86
4.2.14	MSPGANG_Interactive_DefReadTargets .....	86
4.2.15	MSPGANG_Interactive_ReadTargets .....	87
4.2.16	MSPGANG_Interactive_ReadBytes .....	88
4.2.17	MSPGANG_Interactive_WriteWord_to_RAM .....	88
4.2.18	MSPGANG_Interactive_WriteByte_to_RAM .....	89
4.2.19	MSPGANG_Interactive_WriteBytes_to_RAM .....	89
4.2.20	MSPGANG_Interactive_WriteBytes_to_FLASH .....	90
4.2.21	MSPGANG_Interactive_Copy_Gang_Buffer_to_RAM .....	90
4.2.22	MSPGANG_Interactive_Copy_Gang_Buffer_to_FLASH .....	91
4.2.23	MSPGANG_Interactive_EraseSectors .....	91
4.2.24	MSPGANG_Interactive_BlankCheck .....	92
4.2.25	MSPGANG_Interactive_DCO_Test .....	92
4.2.26	MSPGANG_SelectImage .....	93
4.2.27	MSPGANG_EraseImage .....	94
4.2.28	MSPGANG_CreateGangImage .....	94
4.2.29	MSPGANG_LoadImageBlock .....	95
4.2.30	MSPGANG_VerifyPSAImageBlock .....	96
4.2.31	MSPGANG_ReadImageBlock .....	96
4.2.32	MSPGANG_Read_Code_File .....	100
4.2.33	MSPGANG_Save_Config, MSPGANG_Load_Config, MSPGANG_Default_Config .....	100
4.2.34	MSPGANG_SetConfig, MSPGANG_GetConfig .....	101
4.2.35	MSPGANG_GetNameConfig, MSPGANG_SetNameConfig .....	110
4.2.36	MSPGANG_SetTmpGANG_Config .....	112
4.2.37	MSPGANG_GetLabel .....	113
4.2.38	MSPGANG_GetInfoMemory, MSPGANG_SetInfoMemory .....	114
4.2.39	MSPGANG_Get_qty_MCU_Type, MSPGANG_Set_MCU_Type, MSPGANG_Get_MCU_TypeName, MSPGANG_Get_qty_MCU_Family, MSPGANG_Get_MCU_FamilyName, MSPGANG_Get_MCU_Name .....	114
4.2.40	MSPGANG_Set_MCU_Name .....	115
4.2.41	MSPGANG_HW_devices .....	116
4.2.42	MSPGANG_GetProgressStatus .....	117
4.2.43	MSPGANG_GetAPIStatus .....	119
4.2.44	MSPGANG_Set_IO_State .....	120
4.2.45	MSPGANG_Convert_Address .....	122
4.2.46	MSPGANG_Memory_Header_text .....	122

4.2.47	MSPGANG_Interactive_ClrLockedDevice.....	122
4.2.48	MSPGANG_Get_Code_Info.....	123
4.2.49	MSPGANG_MakeSound.....	123
4.2.50	MSPGANG_Callback_ProgressBar.....	124
4.2.51	MSPGANG_GetPCHardwareFingerprint .....	126
4.2.52	MSPGANG_Flash_valid_addr .....	126
<b>5</b>	<b>Schematics .....</b>	<b>127</b>
5.1	Schematics .....	127
<b>6</b>	<b>Frequently Asked Questions .....</b>	<b>134</b>
6.1	Question: Why does device init, connect, or programming fail?.....	134
6.2	Question: Can I use single wires for connection between MSP-GANG and target device? .....	135
6.3	Question: How to serialize parts? .....	135
6.4	Question: How to have parts run after programming? .....	135
6.5	Question: What are possible reasons for the part to fail Verify step? .....	135
	<b>Revision History .....</b>	<b>136</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 .....	16
2-3.	Reset Options.....	17
2-4.	Verification Error.....	19
2-5.	Flash Memory Data .....	20
2-6.	Main MSP Gang Programmer Dialog GUI, From Image Mode .....	21
2-7.	Main MSP Gang Programmer Dialog GUI, From Image Mode and Custom Configuration Enabled .....	23
2-8.	Main MSP Gang Programmer Dialog GUI, From Script .....	25
2-9.	Main MSP Gang Programmer Dialog GUI, Standalone Mode .....	30
2-10.	Image Option.....	31
2-11.	Target Enable or Disable Option .....	32
2-12.	Memory Options, BSL Sectors Selected .....	34
2-13.	MSP430 Secure Device Options .....	35
2-14.	MSP432 Secure Device Options .....	36
2-15.	MSP432 Secure Device Options Details .....	37
2-16.	Memory Options Window.....	38
2-17.	Serialization .....	39
2-18.	Image Name Configuration Screen .....	42
2-19.	Image File Security Options .....	43
2-20.	Hardware Fingerprint of Computer in Use .....	43
2-21.	Programming From Image File .....	44
2-22.	Password for Image File.....	45
2-23.	Code File Data.....	47
2-24.	Comparison of Code and Flash Memory Data of the Target Microcontroller .....	48
2-25.	Self Test .....	54
2-26.	Information About the MSP Gang Programmer .....	58
2-27.	Preferences Selection Window .....	59
5-1.	MSP-GANG Simplified Schematic (1 of 4) .....	127
5-2.	MSP-GANG Simplified Schematic (2 of 4) .....	128
5-3.	MSP-GANG Simplified Schematic (3 of 4) .....	129
5-4.	MSP-GANG Simplified Schematic (4 of 4) .....	130
5-5.	Gang Splitter Schematic .....	131
5-6.	BSL Connection Schematic .....	132
5-7.	Schematic of MSP-GANG 14-20 Adapter .....	133
5-8.	Top View of MSP-GANG 14-20 Adapter (Order Separately From TI) .....	133

## List of Tables

2-1.	Benchmark Results – MSP430F5438A, 256kB Code .....	60
2-2.	Benchmark Results – MSP430F5438A, 250kB Code, Mode: From Image .....	60
2-3.	Benchmark Results – MSP430F5438A, 250kB Code, Mode: Interactive, Communication by USB .....	60
2-4.	Benchmark Results – MSP430FR5994, 256kB Code, Mode: From Image .....	61
2-5.	Benchmark Results – MSP430FR5994, 256kB Code, Mode: Interactive, Communication by USB .....	61
2-6.	Benchmark Results – MSP430F2619, 120kB Code, Mode: From Image .....	61
2-7.	Benchmark Results – MSP430F2619, 120kB Code, Mode: Interactive, Communication by USB .....	61
2-8.	Benchmark Results – MSP432P401R, 256kB Code, Mode: From Image.....	62
2-9.	Benchmark Results – MSP432P401R, 256kB Code, Mode: Interactive, Communication by USB.....	62
3-1.	Data Frame for Firmware Commands .....	65
5-1.	Gang Splitter Bill of Materials (BOM) .....	132

## ***Read This First***

---

---

---

### **If You Need Assistance**

If you have any feedback or questions, the Texas Instruments Product Information Center (PIC) and the [TI E2E™ Forum](#) provide support for the MSP430™ and SimpleLink™ MSP432™ microcontrollers and for the MSP-GANG. See the [TI website](#) for contact information for the PIC. Device-specific information is on the [MSP website](#).

### **Trademarks**

E2E, MSP430, SimpleLink, MSP432 are trademarks of Texas Instruments.  
Windows is a registered trademark of Microsoft Corporation.  
All other trademarks are the property of their respective owners.

### **Related Documentation From Texas Instruments**

The primary sources of MSP information are the device-specific data sheets and user's guides. The most current information is on the [MSP website](#).

Information specific to the MSP-GANG is at <http://www.ti.com/tool/msp-gang>.



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 A digital device, 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 or more of the following measures:

- \* Reorient or relocate the receiving antenna
- \* Increase the separation between equipment and receiver
- \* Connect the equipment into an outlet on a circuit different from that to which 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.**



**NOTE:** This equipment has been tested and found to comply with:

CISPR 24:1997 +A1:2001 +A2:2002 / EN 55024:1998 +A1:2001 +A2:2003 EMC Requirements

CISPR 24: 2010 / EN 55024:2010 - Electromagnetic Compatibility Requirements

CISPR 22:2008-09 / EN 55022:2006 +A1:2007, Class A - Information Technology Equipment

CISPR 22:2008-09 / EN 55022:2010+AC: 2011, Class A - Information Technology Equipment

IEC 61000-4-4 / EN 61000-4-4 - Electromagnetic Compatibility Requirements, Part 4: Electrical Fast Transient Requirements

IEC 1000-4-2 / EN 61000-4-2 -Electromagnetic Compatibility Requirements, Part 2: Electrostatic Discharge Requirements



## Introduction

---



The MSP Gang Programmer for the MSP430 and MSP432 microcontrollers can program up to eight of the same MSP flash or FRAM devices at one 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. [Figure 1-1](#) shows a top-level view of the MSP Gang Programmer.

The MSP Gang Programmer is not a gang programmer in the traditional sense, in that there are not eight sockets to program target devices. Instead, the MSP Gang Programmer connects to target devices that are mounted in the final circuit or system. The MSP Gang Programmer accesses the target devices through connectors that use JTAG, Serial-Wire Debug (SWD), Spy-Bi-Wire (SBW), or bootloader (BSL) signals.

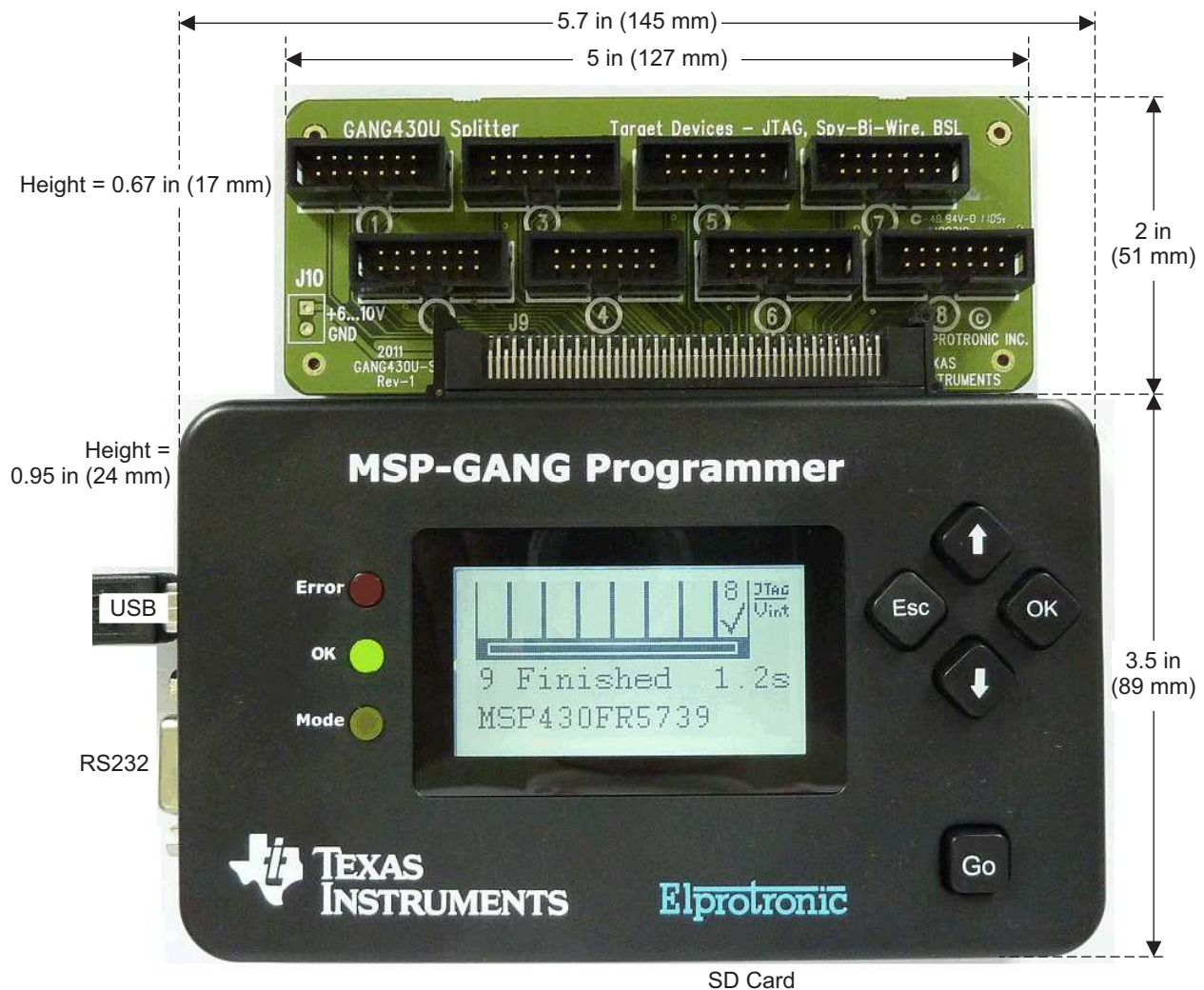
The MSP Gang Programmer includes an expansion board, called the Gang Splitter, that connects the MSP Gang Programmer to multiple target devices. Eight cables connect the Gang Splitter to eight target devices (through JTAG, SWD, SBW, or BSL connectors). For MSP432 MCUs, an adapter kit ([MSP-GANG-432ADPTR](#)) can convert from 14-pin JTAG connectors to 20-pin Arm connectors.

[Chapter 2](#) describes how to use the MSP Gang Programmer to program target devices. This chapter describes the modes of operation and how to choose the method of programming. This chapter also describes the user interface that defines how to program the target device.

[Chapter 3](#) describes firmware commands that give low-level control of the programming process. The commands correspond to specific actions that the programmer can perform. The MSP Gang Programmer connects to a host computer through a RS-232 or USB port to receive the commands. Often, you must use the commands in groups or in a specific order to ensure proper behavior.

[Chapter 4](#) describes `Gang430.dll`, `MSP-GANG.dll`, and the functions that are available through them.

[Chapter 5](#) contains an I/O schematic that shows how signals from the MSP Gang Programmer go to each target device through an MSP-standard JTAG, SWD, SBW, or BSL connector. To make a traditional gang programmer, you can change the circuit to connect the signals to the target device pins directly through a socket.



NOTE: Dimensions are approximate.

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

## 1.1 Software Installation

Use the latest software version, which can be downloaded from the [MSP-GANG Production Programmer tool folder](#). The MSP-GANG Programmer Software runs on Windows® 32 bit or 64 bit: Windows XP, Windows 7, Windows 8, and Windows 10.

To install MSP Gang Programmer software:

1. Unzip the installation package.
2. Run setup.exe in the root directory of the package.
3. Follow the instructions in the installation process.
4. When the setup program finishes, click the MSP Gang Programmer Read Me First icon to read important information about the MSP Gang Programmer.
5. The setup program also adds a program group and icons to the Windows desktop.
6. To start the MSP Gang Programmer software, click the icon.

## 1.2 Driver Installation

To install the required drivers:

1. Connect the MSP-GANG programmer to a PC USB port. When the Windows wizard starts, follow the instructions provided by wizard. When the wizard asks for the USB driver location, browse to the CD-ROM drive. Drivers are 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, the USB driver is not required. Run the Windows Device Manager to find for the COM port number to use with communication through 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 connects up to eight targets using the included 14-pin cables. The target MSP430 flash devices can be in stand-alone sockets or can be on an application PCB. The MSP Gang Programmer can connect to these devices through JTAG, SBW, or BSL signals.  
If the target device is an MSP432 MCU, use the adapter kit ([MSP-GANG-432ADPTR](#)) to convert from 14-pin JTAG connectors to 20-pin Arm connectors.
2. Connect the MSP Gang Programmer hardware to the computer USB port using a USB A-B cable.  
The USB port (5 V, 0.5 A) can supply the programmer.  
If the computer does not have a USB port, connect the programmer to a serial port (COM1 to COM255) using a 9-pin Sub-D connector.
3. If the MSP Gang Programmer is not connected through the USB port, or if the total current consumption of the programmed target devices exceeds 0.3 A, connect an external power supply to the programmer.

---

**NOTE: External Power Supply**

An external power supply must provide a voltage between 6 V and 10 V DC and must provide 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.

---

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

**CAUTION**

When an external power supply is used to power target devices, disconnect  $V_{CC}$  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, connect the  $V_{CC}$  from the target device to  $V_{extin}$  (pin 4) on the JTAG, SBW, SWD, or BSL connectors. The MSP Gang Programmer uses this voltage to detect the presence of an external power supply.

Set the desired  $V_{CC}$  level in the MSP Gang Programmer to the same voltage that powers the target device. This information is mandatory to provide correct I/O levels for the TMS, TCK, TDI, TDO, and RST 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 be unreliable.

5. The MSP Gang Programmer can be supplied from an external power supply connected to the DC connector or through 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 MSP Flash Devices Using the MSP Gang Programmer

The MSP Gang Programmer is capable of quickly and reliably programming MSP 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
- Stand Alone

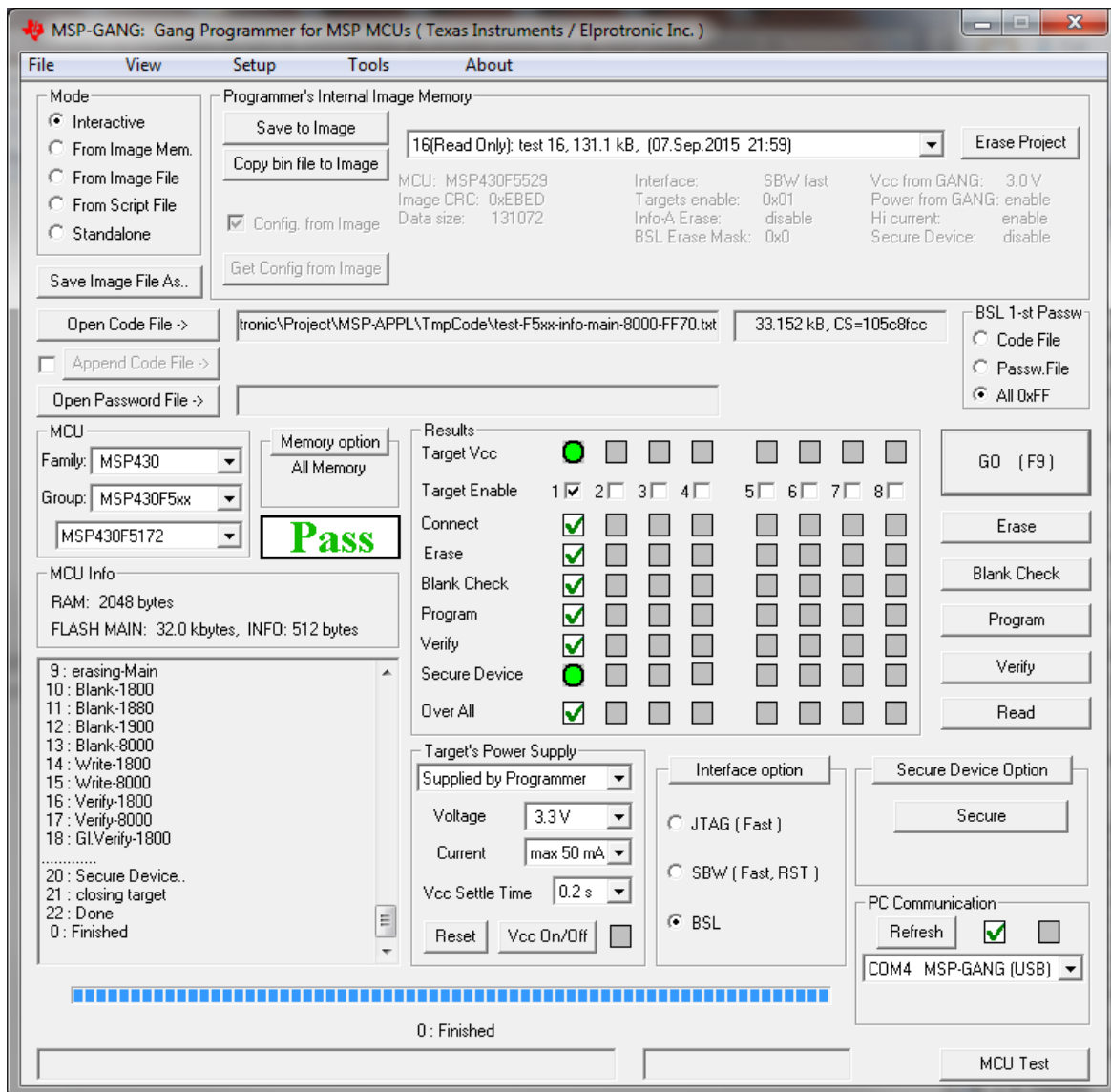
The Interactive mode is selected by default, and is the easiest to get started with, because it requires the least amount of preparation. After the user has mastered the Interactive mode it can be used to create images and script files, which can then be used with the From Image and From Script modes, 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 2.1.9](#).

The following sections 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 MSP Flash Devices using the Interactive Mode:

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



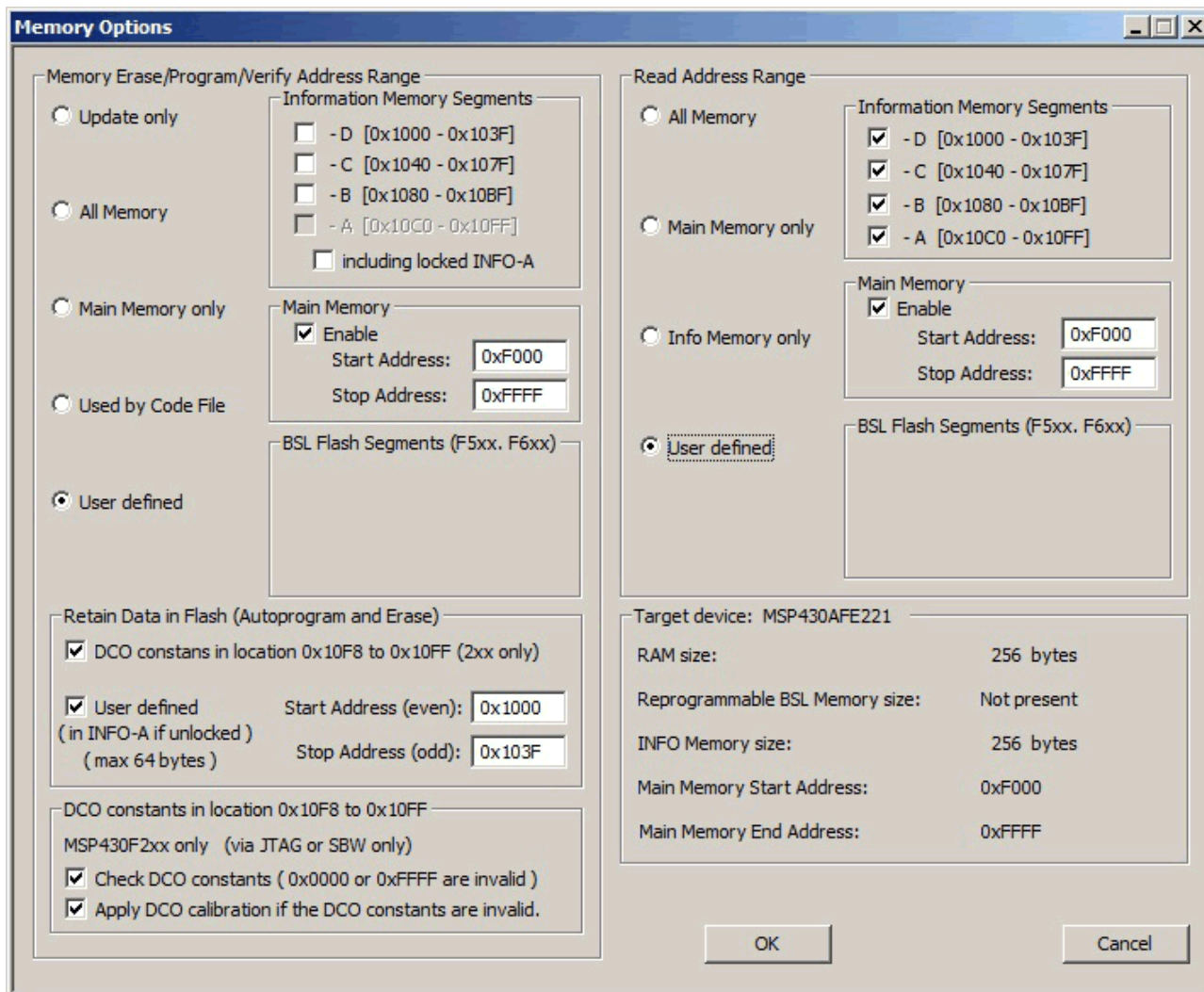
**Figure 2-1. Main MSP Gang Programmer Dialog GUI, Interactive Mode**

2. Select a target device using the MCU Family, then MCU Group, and then desired MCU Type.
3. Select the code file to be programmed into the devices using the Open Code File button or pulldown menu: File→Open Code File. The formats supported for the code file are TI (.txt) and Intel (.hex) and Motorola (.s19, .s28, .s37). Code size and checksum appear on the right side (for details on how the checksum is calculated, see [Section 2.1.13](#)).
4. Optionally add another code file to be programmed into the devices using the Append Code File button (check the box on the left to enable this option). This feature is useful for updating BSL firmware in 5xx or 6xx MCUs. The two code files are combined together to create one final code file. If a conflict is



detected, a warning appears; however, if programming proceeds without changes the second code file overwrites the conflict area. Code size and checksum appear on the right side.

5. Some MCUs (for example, the MSP430FR57xx) provide a method of disabling JTAG by programming a password to flash memory. The password should be specified as data to be programmed starting at 0xFF80 and up to 0xFFFF (where 0xFF80 must be 0xAAAA, 0xFF82 must be the size of the password in words, and 0xFF88-0xFFFF contains the password). The code file must contain password contents if you intend to lock JTAG using the password feature after programming. If the MCU is already locked using a previously programmed code file, then you must provide the password section (or entire old code file) using the Open Password File button if and only if the password section is different. Functionally, if the MCU is locked by password, the code file's password section is first used to attempt to unlock the MCU. If that fails, then the password file's contents are used to attempt to unlock the MCU. If both attempts fail, the MCU remains locked and JTAG access fails. Password file contents are not used to program the MCU.
6. 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.
7. 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.
8. In the Interface selector, choose the desired interface (JTAG, SBW, SWD, or BSL) and communication speed (fast, medium, or slow).
9. 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 how to use the memory configuration window).



**Memory Options**

**Memory Erase/Program/Verify Address Range**

☐ Update only

☐ All Memory

☐ Main Memory only

☐ Used by Code File

☒ User defined

**Information Memory Segments**

☐ - D [0x1000 - 0x103F]

☐ - C [0x1040 - 0x107F]

☐ - B [0x1080 - 0x10BF]

☐ - A [0x10C0 - 0x10FF]

☐ including locked INFO-A

**Main Memory**

☒ Enable

Start Address: 0xF000

Stop Address: 0xFFFF

**BSL Flash Segments (F5xx, F6xx)**

**Read Address Range**

☐ All Memory

☐ Main Memory only

☐ Info Memory only

☒ User defined

**Information Memory Segments**

☒ - D [0x1000 - 0x103F]

☒ - C [0x1040 - 0x107F]

☒ - B [0x1080 - 0x10BF]

☒ - A [0x10C0 - 0x10FF]

**Main Memory**

☒ Enable

Start Address: 0xF000

Stop Address: 0xFFFF

**BSL Flash Segments (F5xx, F6xx)**

**Retain Data in Flash (Autoprogram and Erase)**

☒ DCO constants in location 0x10F8 to 0x10FF (2xx only)

☒ User defined (in INFO-A if unlocked) (max 64 bytes)

Start Address (even): 0x1000

Stop Address (odd): 0x103F

**DCO constants in location 0x10F8 to 0x10FF**

MSP430F2xx only (via JTAG or SBW only)

☒ Check DCO constants ( 0x0000 or 0xFFFF are invalid )

☒ Apply DCO calibration if the DCO constants are invalid.

**Target device: MSP430AFE221**

RAM size: 256 bytes

Reprogrammable BSL Memory size: Not present

INFO Memory size: 256 bytes

Main Memory Start Address: 0xF000

Main Memory End Address: 0xFFFF

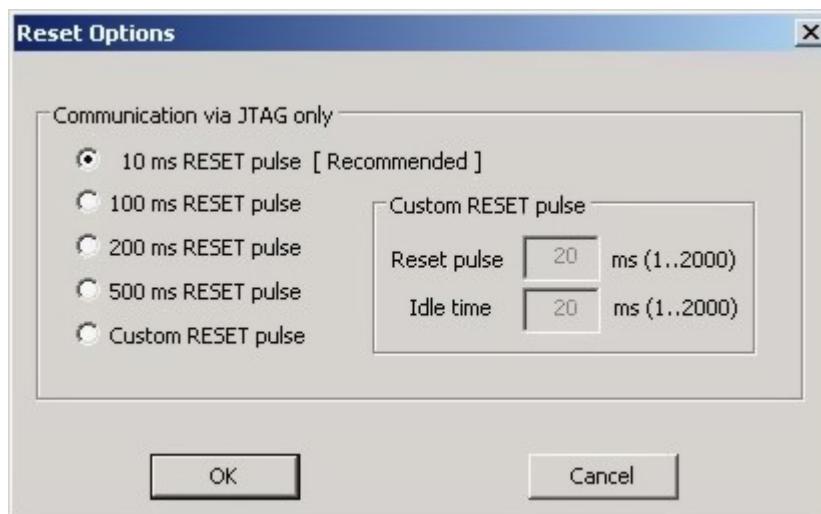
OK Cancel

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

**Figure 2-2. Memory Options**

10. 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. GO starts erase, blank check, program, verify, or blow fuse if selected. The progress and completion of the operation are displayed in the Results group. The result is shown as one of the following:

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

**NOTE:** When a FRAM MCU is selected, the blank check step is skipped. During global verification, main code contents and empty values are verified.

#### 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 to flash memory on the target device. Addresses specified in the code file 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

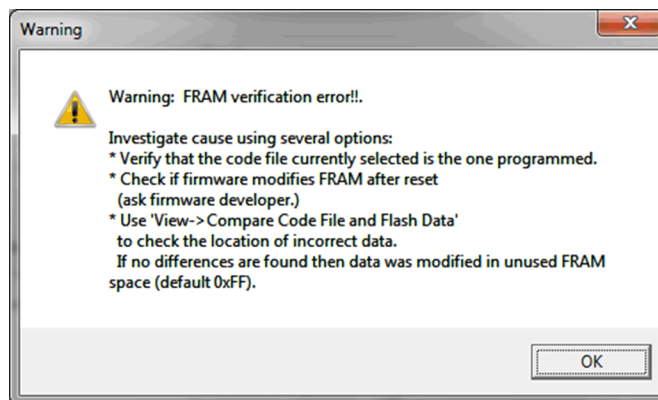
Click the Verify button in the Main Dialog GUI to verify that the contents of the target MCU's flash memory have been properly programmed. This function is best used after programming the same segment of memory, as performed using the button described above. Make sure that the same memory segments are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

Verification of selected flash memory is divided into two steps: (1) verify selected flash memory that only corresponds to the code file, and (2) verify selected flash memory that corresponds to the code file AND selected flash memory not included in the code file that should be empty (0xFF). Examples of selected flash memory include Main Memory, All Memory, or User defined, with the exception of Retain Data (if defined). Verified flash memory that only corresponds to the code file is displayed in the GUI using Verify-XXXX messages, where XXXX is the start address of a contiguous code segment. Verified flash memory that corresponds to the code file AND flash memory not included in the code file is displayed in the GUI using GI.Verify-XXXX messages; where XXXX is the start address of a contiguous code and empty data segment. Each contiguous segment is verified using a checksum (CS) and pseudo-signature analysis (PSA). Verification passes if the CS and PSA match between flash memory and the code file.

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 fails otherwise.

If the verification fails for any reason, TI recommends using an option from the pulldown menu View→Compare Code File and Flash Data. When this option is enabled, the contents of the Flash or FRAM memory is read and compared with used code file contents. Only bytes defined in the Code File contents are compared. All other byte contents taken from the Flash or FRAM are ignored, regardless of their content. If no errors are found by this verification, even the verification itself failed, then bytes outside of the code file (not programmed) have a value other than 0xFF. Check the firmware that was downloaded to Flash or FRAM to determine if the firmware is modifying the Flash or FRAM in unused memory space after MCU reset (for example, if the Flash or FRAM is used for additional memory space like EEPROM).

If the verification fails and the MCU has FRAM memory, then the following pop-up message is displayed.



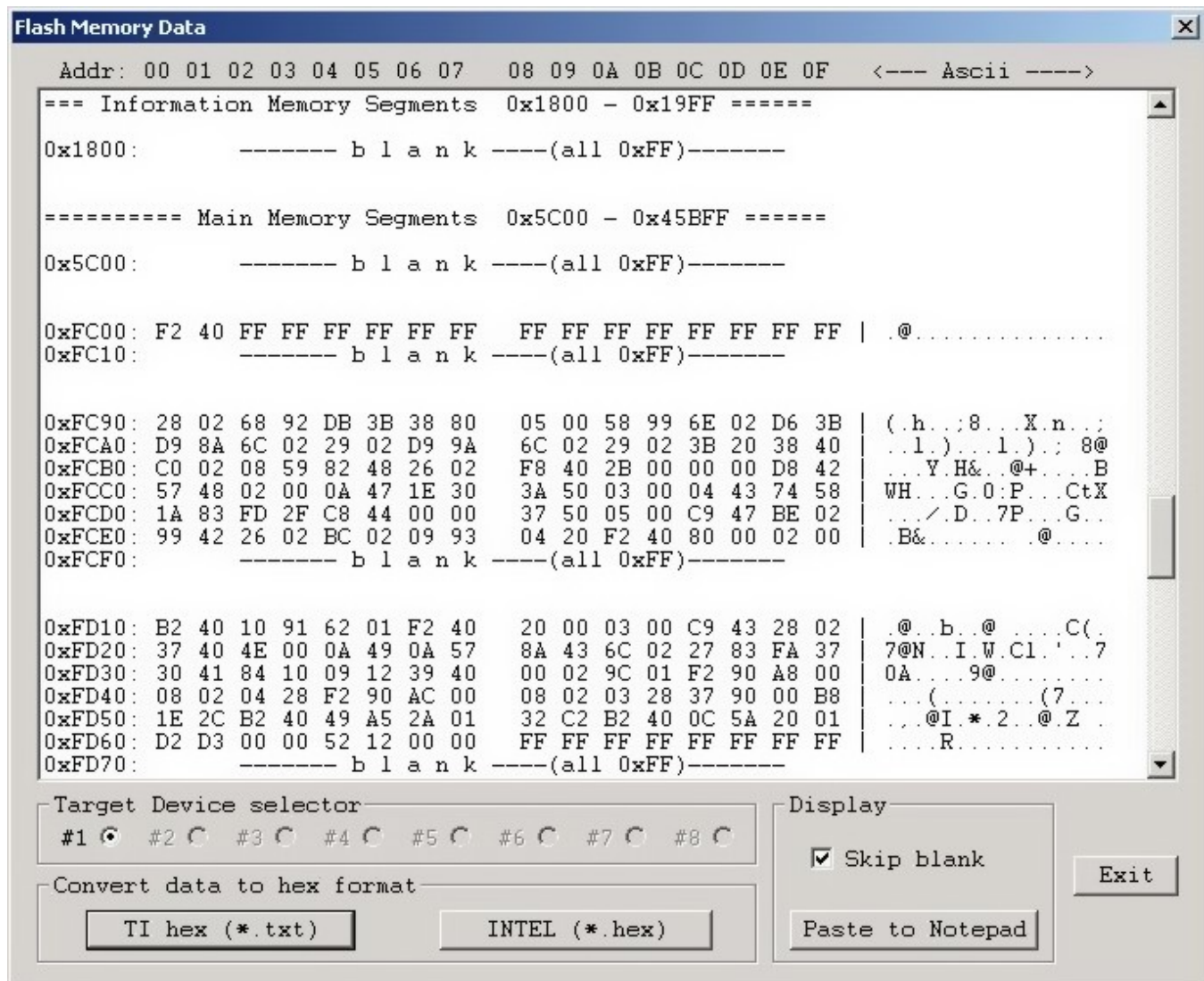
NOTE: Verification failed on MCU with FRAM type memory

**Figure 2-4. Verification Error**

#### 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 2-5](#). This window can be selected in the View→Flash Memory Data pulldown menu. The Flash Memory Data viewer, shown in [Figure 2-5](#), 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.



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 2-5. Flash Memory Data**

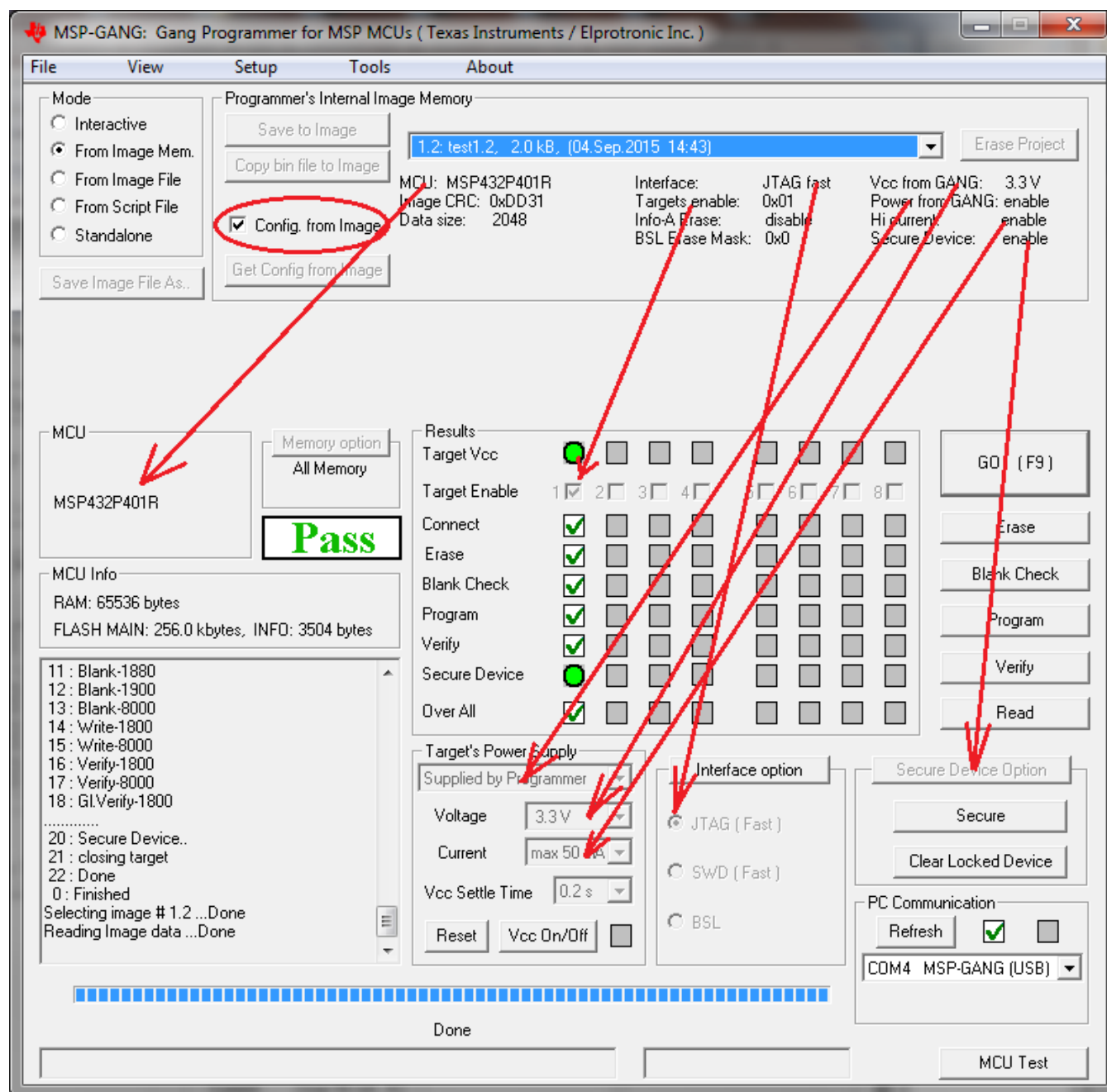
## 2.1.2 Programming From Image

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.

When an image has been created, it can be used to greatly simplify programming by using the procedure described in [Section 2.1.9](#). [Figure 2-6](#) 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 96 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.



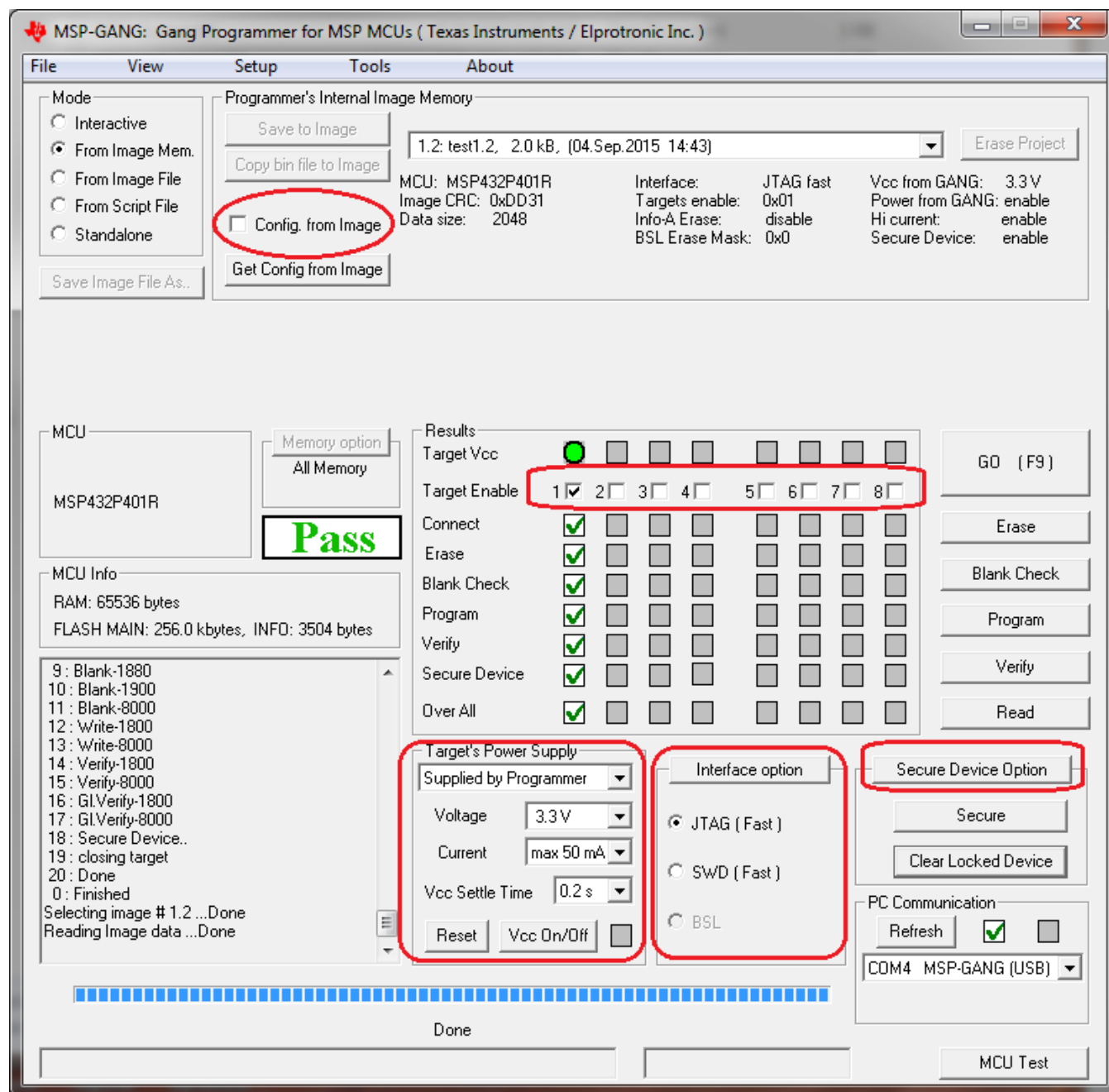
**NOTE:** This figure shows the From Image Mode (see the Mode section near the top left corner). The user can load an image from MSP Gang Programmer internal memory. Saved images contain all configuration necessary for programming and all code files. An image can be created using the Interactive Mode and saved to the programmer. One of 96 different images can be selected from internal memory, or one image from each external SD-Card can be used.

**Figure 2-6. Main MSP Gang Programmer Dialog GUI, From Image Mode**

[Figure 2-6](#) 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 96 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 2-7](#) 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 2.1.9](#).





NOTE: This figure shows 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.

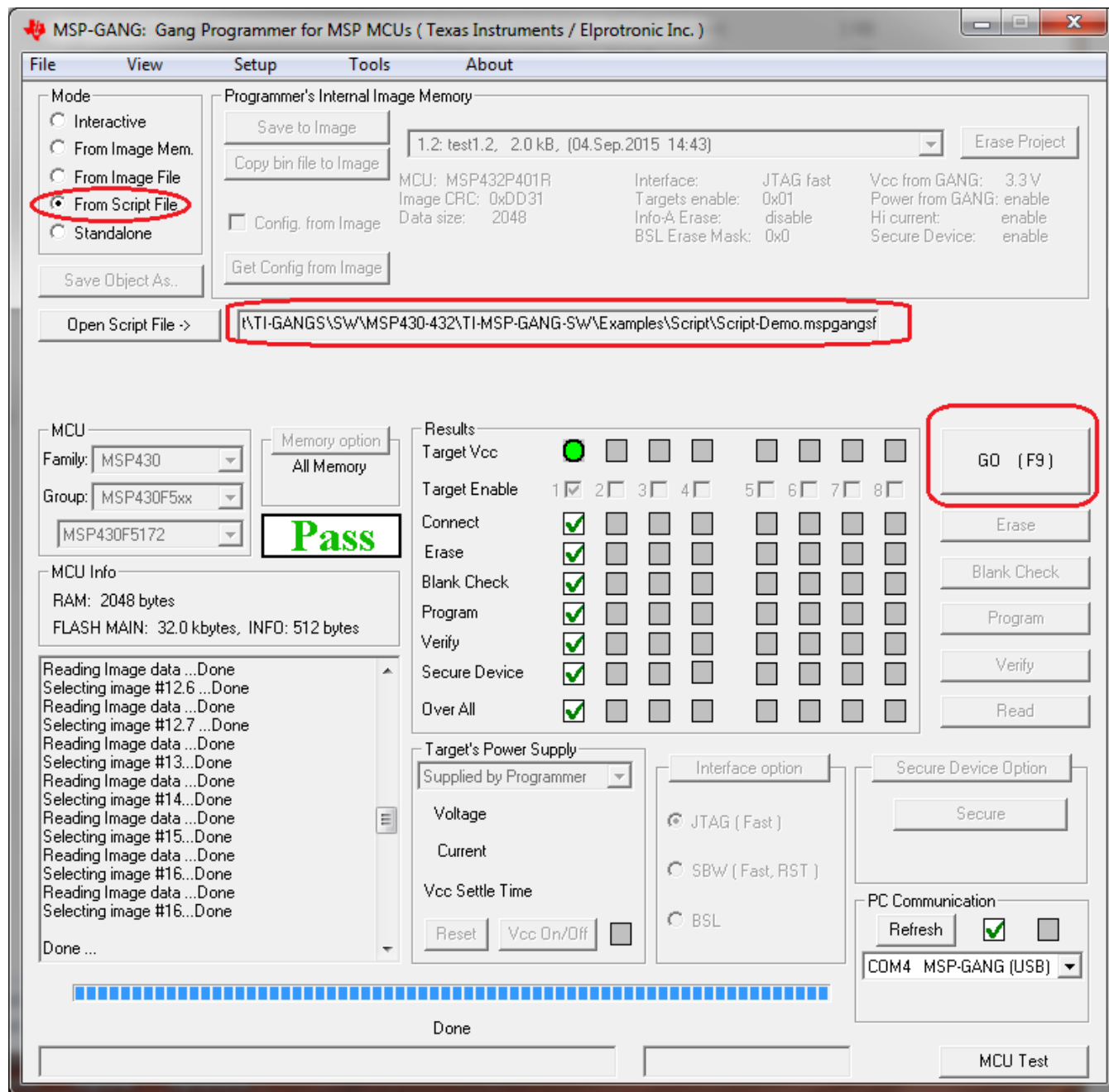
**Figure 2-7. Main MSP Gang Programmer Dialog GUI, From Image Mode and Custom Configuration Enabled**

### 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, and so on), which is sufficient for most nonrecursive programs.

Figure 2-8 shows the main dialog GUI where the From Script option is selected for programming (top left corner). A script file is selected using the Open Script File button and it specifies all configuration options, and the code files to be used for programming. A script can be created using any text editor and saved in a simple text file. Follow these guidelines to create a script.





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

**Figure 2-8. 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, and so on).

### 2.1.3.2 Command Syntax

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

---

**NOTE:** A comment cannot be placed after a filename.

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

---

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

---

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

For example, this would be illegal:

```
>START VCCOFF
```

---

### 2.1.3.3 Instructions

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

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

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

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

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

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

Example:

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

**GOTO label** – Jump to instruction immediately following the label.

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

**F\_LOADPASSWORDFILE filename** – Load JTAG password file. Provide a full path and filename.

**F\_FROMIMAGEMODE** – Switch to Image mode.

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

**RETURN** – Return from CALL.

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

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

The operation can be one of the following:

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

**F\_LOADCFGFILE filename** – Load configuration file. Provide a full path and filename.

**F\_LOADCODEFILE filename** – Load code file. Provide a full path and filename.

**F\_APPENDCODEFILE filename** – Append code file. Provide a full path and file name.

**F\_VCCOFF** – Turn  $V_{CC}$  OFF from programming adapter to target device.

**F\_VCCON** – Turn  $V_{CC}$  ON from programming adapter to target device.

---

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

---

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

**F\_RESET** – Perform RESET function from main dialog screen.

**F\_GO** – Perform GO function from main dialog screen.

**F\_ERASEFLASH** – Perform ERASE FLASH function from main dialog screen.

**F\_BLANKCHECK** – Perform BLANK CHECK function from main dialog screen.

**F\_WRITEFLASH** – Perform WRITE FLASH function from main dialog screen.

**F\_VERIFYFLASH** – Perform VERIFY FLASH function from main dialog screen.

**F\_BLOWFUSE** – Perform BLOW FUSE function from main dialog 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 96 from MSP Gang Programmer internal memory.

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

---

**NOTE:** The execution result can be saved in the result file. Contents of the file can be used by the application software if required. The result can be saved in the new file or append to the existing file. Following script line commands can be used for specifying the result file:

---

**F\_NEWRESULTFILENAME** – Provide a full path and name of the result file.

**F\_APPENDRESULTFILENAME** – Provide a full path and name of the file where the result should be appended.

**F\_COMMENTTOFILE** – Add a comment at the beginning of the result stream.

**F\_RESULTTOFILE** – Save result to the result file specified by F\_NEWRESULTFILENAME or F\_APPENDRESULTFILENAME. The following data is saved:

```

Finished task mask:      HHHH   (16 bits task mask)
Cumulative target mask: HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Requested target mask:  HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Connected target mask:  HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Erased target mask:     HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Blank Check target mask: HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Programmed target mask: HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Verified target mask:   HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Secured target mask:    HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
error_no:               error number
VTIO in mV:             VTio in mV
Vcc Error target mask:  HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Vcc Cumulative Err mask: HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
JTAG Init target mask:  HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Already Secured mask:   HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);
Wrong MCU ID mask:     HH   (8 bits target mask - 0x01-target-1,.. 0x80-target-8);

```

**TRACEOFF** – Disable tracing.

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

**END** – End of script.

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
F_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

; use file name and FULL PATH or relative path to MSP-Gang.dll file location
F_LOADCFGFILE Examples\Script\test.msppgangproj

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

; use file name and FULL PATH or relative path to MSP-Gang.dll file location
F_LOADCFGFILE Examples\Script\finalcode.msppgangproj

F_GO

; wait min 0.5 s before turning Vcc ON again
SLEEP 500

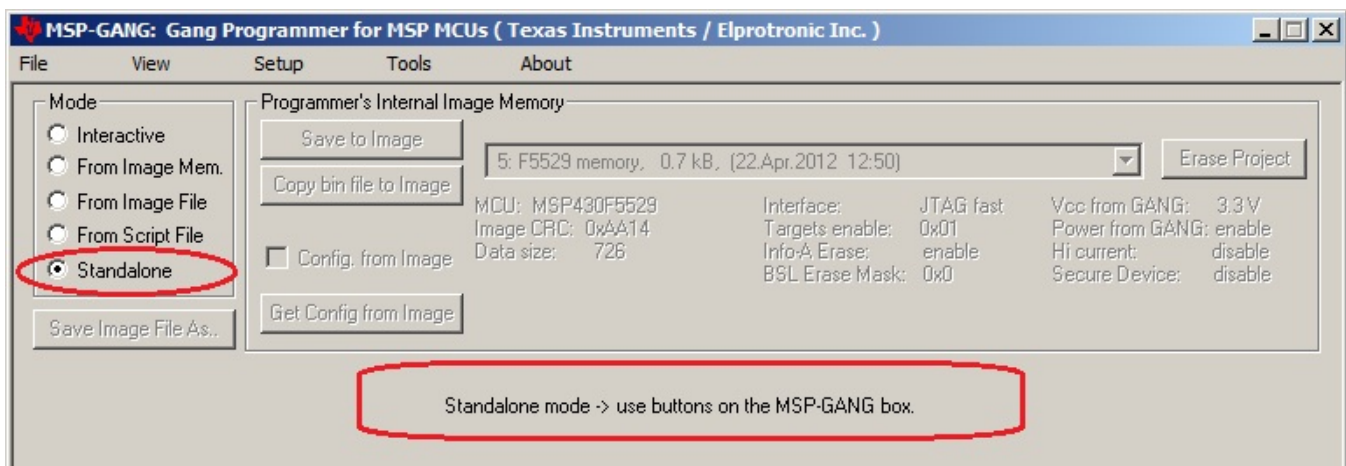
```

```
F_VCCON
SLEEP 10000
GOTO START

>finish
END
;=====
```

### 2.1.4 Programming in Standalone Mode

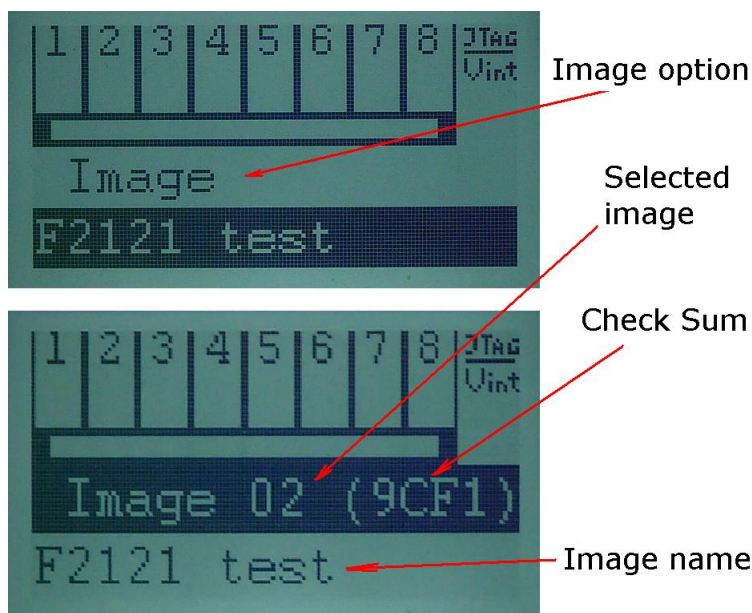
The MSP Gang Programmer supports the Standalone mode of programming target devices. In this mode, the MSP Gang Programmer can only use images for programming because they contain a complete configuration and code files necessary for the procedure. If the user has not already created an image then follow the procedure outlined in [Section 2.1.9](#). When viewed from the GUI, [Figure 2-9](#) 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 96 different images), or an external SD-Card, without the use of a desktop or laptop computer.

**Figure 2-9. 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 and down) and the enter button to select a desired image for programming. A description of the selected image is displayed on the bottom line, and it is the same description that was created in the GUI when the Save Image button was pressed (see [Figure 2-10](#)).

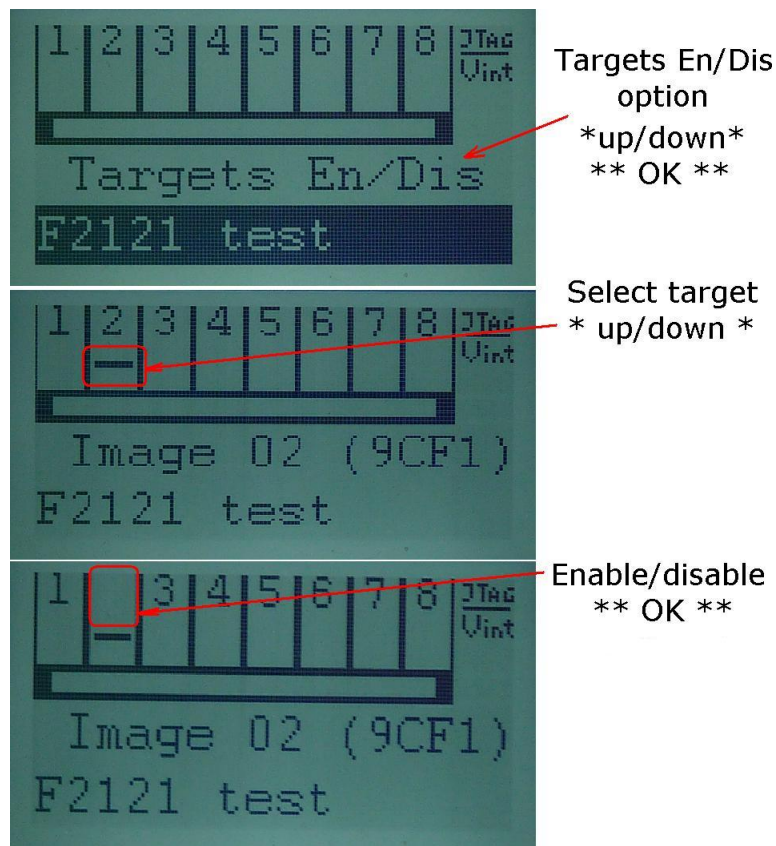


**Figure 2-10. Image Option**

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 2-11](#)), 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 2-11. Target Enable or 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 by 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**

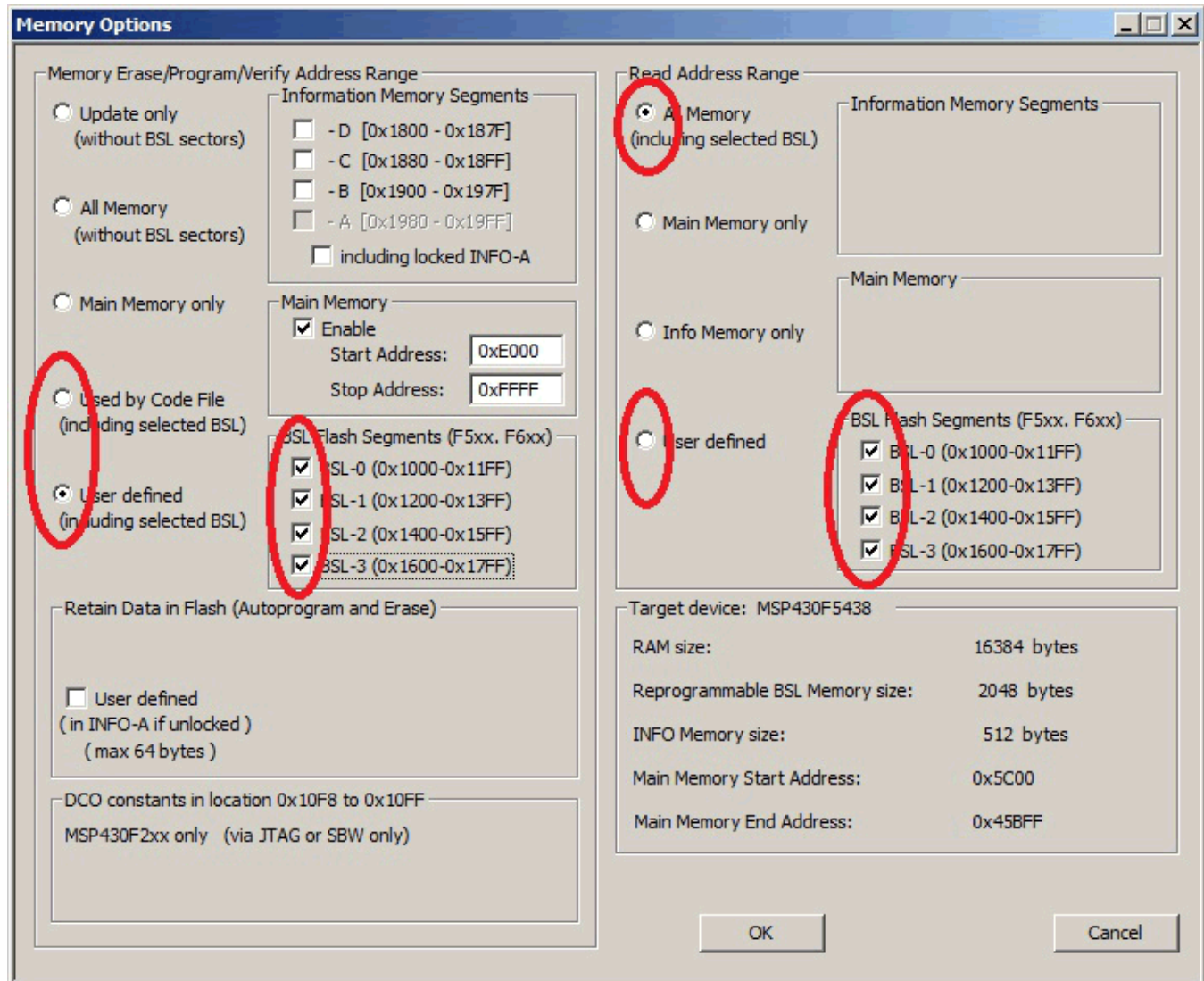
The number of bytes in all data blocks must be even. Words (two bytes) are used for writing and reading data. In case that the code file contains an odd number of bytes, the data segment is appended by a single byte containing a blank value of 0xFF. This value does not overwrite the current memory contents (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 MSP430F5xx and MSP430F6xx 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 2-12. Contents of BSL sectors can be read even when the All Memory option is selected.

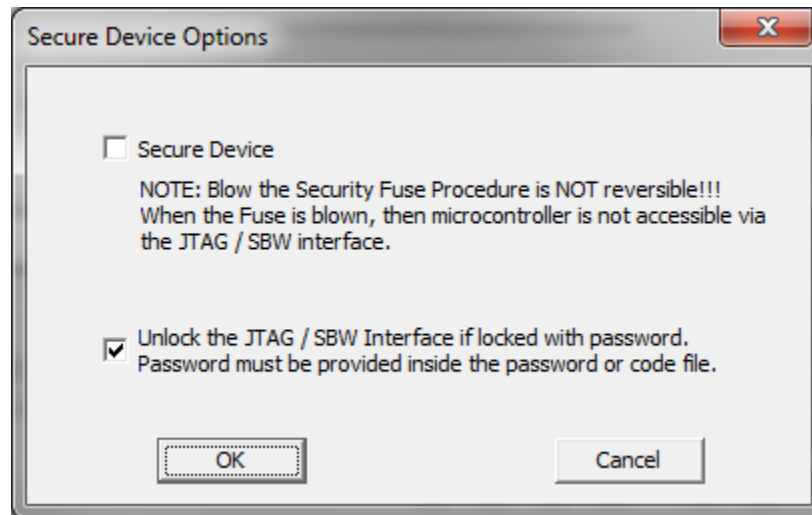


NOTE: The user can select which segments of memory are written to or read from. The selected configuration shows how the user can configure the programmer to overwrite segments of memory used by the Bootloader (BSL).

**Figure 2-12. Memory Options, BSL Sectors Selected**

### 2.1.6 Secure Device Setup and Memory Protection

The MSP430 family has an option to block access to the MCU through the JTAG and SBW interface. To select the Secure Device option, press the Secure Device Option button on the GUI or select the option from the pulldown menu under Setup→Secure Device. [Figure 2-13](#) shows the Secure Device Options window. When the Secure Device option is selected, the device is secured at the end of the GO programming procedure if all programming steps pass successfully. Otherwise, the device is not secured. For MSP430 devices, the Secure Device process is not reversible.



NOTE: Irreversible unless password option used. Can be done automatically after programming (at end of GO operation)

**Figure 2-13. MSP430 Secure Device Options**

In some MCUs, typically the FRAM family, a lower JTAG and SBW protection level is available. The JTAG and SBW can be protected by password that is saved in the MCU flash at the addresses 0xFF80 through 0xFFFF. If the password in the code file at this address is the same as the password saved inside the flash, then access to JTAG and SBW is unlocked, and flash can be reprogrammed. This is useful for updating firmware after initial programming.

However, if the device is secured using the Secure Device procedure, then unlocking by using this password is no longer possible. The Secure Device mechanism provides a higher level of protection.

The MSP432 family implements a different approach to memory protection. The MSP432 can provide protection for selected memory regions or to block communication. All protection options are described in the MSP432 technical reference manual and are implemented by programming the flash mailbox (see the *MSP432P4xx Family Technical Reference Manual* ([SLAU356](#)) for details). The MSP-GANG can program the flash mailbox according to user settings or directly from a code file. When the Secure Device Option button is selected for the MSP432 family, the Secure Device Options screen is displayed (see [Figure 2-14](#)).

Secure / Memory Protection Options for MSP432 MCU

☒ User-Defined - use values specified by the user for memory protection (shown on the left).
 ☐ Code File - read values from code file for memory protection (shown on the right).
 

Copy Values From File to User-Defined

Information Memory - Flash Boot-override Mailbox (commands executed after reset)

☒ Secure write enable
 

User-Defined

Code File

CMD (offset: 0x4) - enabled commands:

0x0008 0000

Details

0xFFFF FFFF

Device Security Mailbox Command Selection (see TRM chapter 4.7 for details)

User-Defined

Code File

JTAG\_SWD\_LOCK\_SECEEN

JTAG\_SWD\_LOCK\_SECEEN

SEC\_ZONE0\_EN

SEC\_ZONE1\_EN

SEC\_ZONE2\_EN

SEC\_ZONE3\_EN

BSL\_CONFIG

JTAG\_SWD\_LOCK\_ENC\_UPDATE

SEC\_ZONE0\_UPDATE

SEC\_ZONE1\_UPDATE

SEC\_ZONE2\_UPDATE

SEC\_ZONE3\_UPDATE

FACTORY\_RESET\_PARAMETERS

FACTORY\_RESET

(0x44-0x50) JTAG\_SWD\_LOCK\_UNENC\_PWD[0]

☒ Command Enable
 ☒ Command Enable

0xFFFF FFFF

0x0000 0000

0x0000 0000

0xFFFF FFFF

0xFFFF FFFF

0xFFFF FFFF

0x0000 0000

0x0000 0000

0xFFFF FFFF

0xFFFF FFFF

0xFFFF FFFF

0xFFFF FFFF

0xFFFF FFFF

0xFFFF FFFF

0x0000 0000

0x0000 0000

Note: Saved Protection bits can be erased if required, however the whole flash memory will be erased also (press "Clear Locked Device" button).

Clear Locked Device Options

Erase ALL Main Memory and

☒ Secure / Mail Box (0x200000-0x200FFF)
 ☐ BSL Memory (0x2002000 - 0x2003FFF)

Reload

OK

Cancel

NOTE: The flash mailbox can be programmed to provide memory protection of some memory, or communication can be blocked. Can be done automatically after programming (at end of GO operation)

**Figure 2-14. MSP432 Secure Device Options**

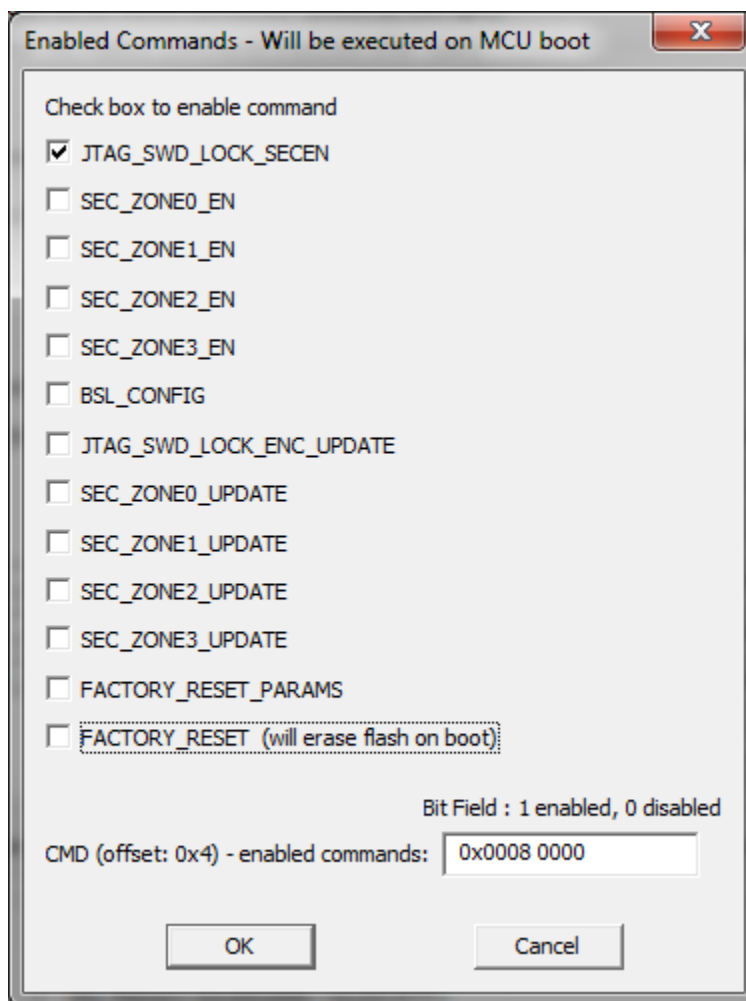
36 Operation

SLAU358P–September 2011–Revised February 2019

[Submit Documentation Feedback](#)

Copyright © 2011–2019, Texas Instruments Incorporated

Many types of protection options are available and can be set in the Enabled Commands screen (see [Figure 2-15](#)).



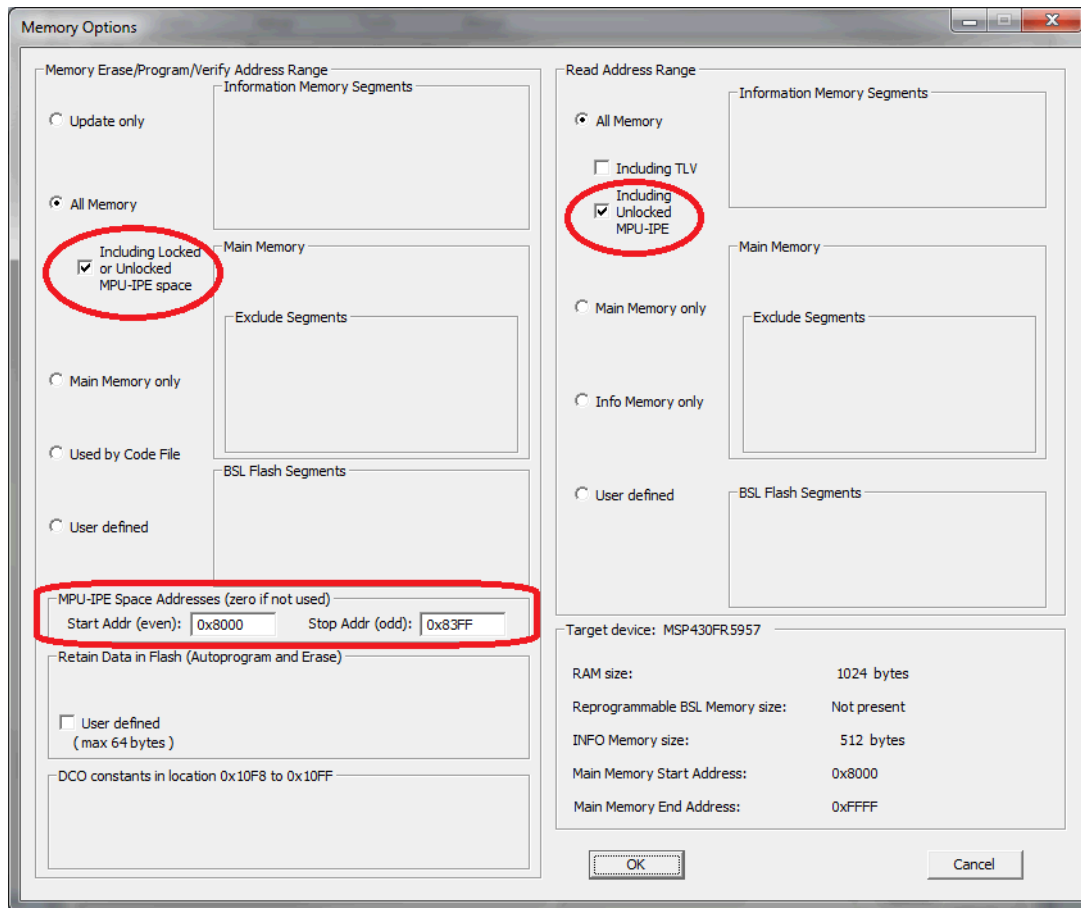
NOTE: The flash mailbox can be programmed with different instructions that provide memory protection, or block communication.

**Figure 2-15. MSP432 Secure Device Options Details**

### 2.1.7 Programming MCU With IP Encapsulated Segment

Some FRAM MCUs have the option to protect an address range in main memory. All data from protected memory space is read as 0x3FFF regardless of actual contents. When the protected memory is not locked, then the contents can be read "as is" if the option "Including Unlocked MPU-IPE" is selected. The programmer must have the address range of protected memory to be able to service the MCU correctly. The protected memory range must be specified in the MPU-IPE Space Addresses Group in the Memory Options window (see [Figure 2-16](#)). The protected memory space can be erased and reprogrammed when the option "Including Locked or Unlocked MPU-IPE space" is selected. That option can only be selected when the All Memory option is selected. When the memory region is protected and locked using the MPU-IPE features (see MCU family user guide or technical reference manual for details) then all memory is erased first before programming the MCU. The protected and locked MPU-IPE memory range can be erased and reprogrammed only when the JTAG or SBW communication is used. When BSL communication is used, the locked memory cannot be erased or reprogrammed. Through BSL, memory can only be erased and reprogrammed when the locking option is not used.

When a new code file is programmed with contents outside of the protected area, all memory (except protected memory) can be erased, blank checked, programmed, and verified. If the protected memory space is defined incorrectly, a blank check error will result, because 0x3FFF will be read instead of the expected 0xFFFF.



**Figure 2-16. Memory Options Window**

### 2.1.8 Serialization

Serialization implemented in the MSP-GANG creates a unique serial number (SN) or MAC address and saves it in the flash, FRAM, or dedicated MAC register in the target device. The SN or MAC address is new every time a new target device is programmed. The SN or MAC number can be generated automatically (incremented from the last number) or read from an external file every time before pressing the GO button.

To enable serialization, select *ENABLE Serialization* in the Serialization screen (see [Figure 2-17](#)). Specify the log file name where the all programmed SN and MAC numbers are saved. The SN or MAC number can be saved in any flash or FRAM location as specified in the *Start Address in Memory* field (see [Figure 2-17](#)). The address must be even, and the *Used size in bytes* (the size of the SN or MAC number) must also be an even number of bytes. The *In Memory Format* section specifies if the SN or MAC number is written LSB first or MSB first.



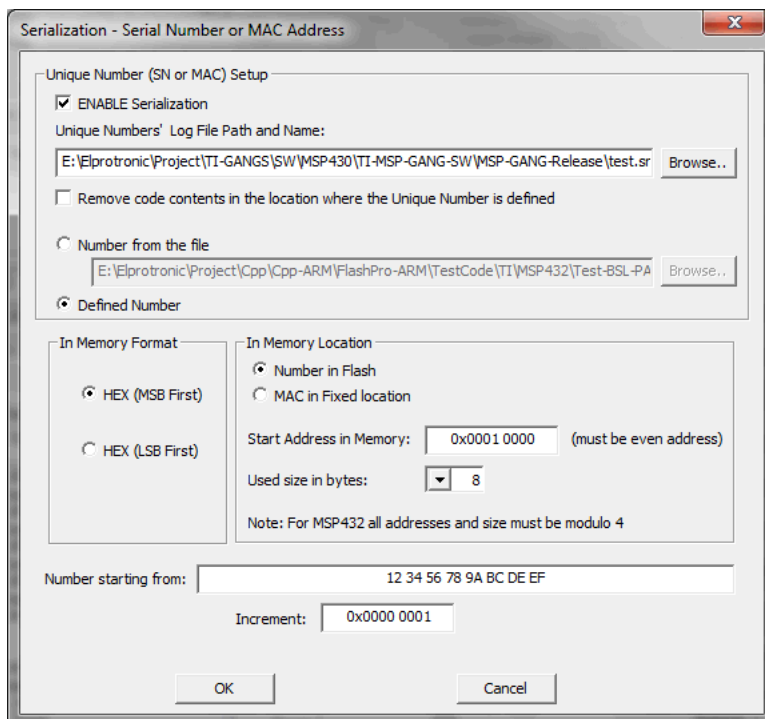


Figure 2-17. Serialization

If the *In Memory Format* option is *HEX (MSB First)*, the SN is saved to flash memory starting from the specified address (0x10000) as follows:

12 34 56 78 9A BC DE EF

If the *In Memory Format* option is *HEX (LSB First)*, the SN is saved to flash memory as follows:

EF DE BC 9A 78 56 34 12

In the report window and log file, the SN is always displayed in the same order as it is saved in memory starting from the lowest address to the highest. In this case, if the SN is saved in memory as MSB first, then the displayed SN in the report window, log file, and *Serialization* screen (see [Figure 2-17](#)) are the same.

If the SN or MAC number is generated automatically (the *Defined Number* option is selected), the number is generated starting with the value in the *Number starting from* field and incremented as specified in the *Increment* field. All numbers must be specified in hex format. When the target are programmed with the new numbers, the value in *Number starting from* is automatically updated and saved in the configuration for use in the next session. The user is responsible for tracking whether or not a particular SN or MAC number has been used. The programmer only applies the values set by the user.

When the *Number from the file* option is selected, up to 8 numbers (SN or MAC) must be in the user-specified file, which must have an extension of *.txt*. The file can contain up to 8 numbers that will be applied in the next programming session. The file must be saved and valid before the GO button is pressed. If additional targets are to be programmed, the file must be updated with the new number list. The following list is an example of the contents of the SN or MAC number file:

```
01 0A A3 B4 32 35 65 23
01 0A A3 B4 32 35 65 24
01 0A A3 B4 32 35 65 25
01 0A A3 B4 32 35 65 26
01 0A A3 B4 32 35 65 27
01 0A A3 B4 32 35 65 28
01 0A A3 B4 32 35 65 29
01 0A A3 B4 32 35 65 2A
```

The preceding example lists numbers that would be programmed to 8 target devices using a size of 8 bytes each. These hex numbers can represent integer values or ASCII text, depending on the application. In the case of ASCII, the text must be converted to hex in the file.

After a set of 8 targets is programmed, the user must update the file (the same file name) with new values. The following example lists a new set of 8 values:

```
01 0A A3 B4 32 35 65 2B
01 0A A3 B4 32 35 65 2C
01 0A A3 B4 32 35 65 2D
01 0A A3 B4 32 35 65 2E
01 0A A3 B4 32 35 65 2F
01 0A A3 B4 32 35 65 30
01 0A A3 B4 32 35 65 31
01 0A A3 B4 32 35 65 32
```

The programmer writes the numbers as provided to the specified flash or FRAM location. The provided numbers must be the same size (in bytes) as specified in the *Used size in bytes* option (see [Figure 2-17](#)).

The SN or MAC number can be also saved to a dedicated register, if available on the target MCU; for example, in the MSP432E4xx MCUs. In this case, select the *MAC in Fixed location* option. The address for the MAC number is hardcoded and displayed (read only and grayed out) for user review in the *Start Address in Memory* field (see [Figure 2-17](#)).

If the SN or MAC number is saved to flash or FRAM, the same address cannot be used by the program code (specified in the code file). The programmer will display a warning if it detects a conflict between address of the SN or MAC number and code. If the linker requires that the code file fill the SN or MAC number location with a dummy value, select the *Remove code contents in the location where the Unique Number is defined* option to overwrite this location with the correct SN or MAC number.

If location specified for the SN or MAC number is not empty (all 0xFF), the SN or MAC number is not written to the target. If the programmer detects any value other than 0xFF, the existing value is restored and the new SN or MAC number is ignored. This process keeps an existing SN or MAC number in the target if the number was already programmed. To overwrite an existing value, erase the device memory before programming.

---

**NOTE:** The MSP-GANG does not erase the existing SN or MAC number if the erase all memory option is used. The old SN or MAC number is restored after the erase, the same way that the defined retained bytes are restored.

---

The SN or MAC number can be erased; for example, if serialization is disabled. After erasing, the SN or MAC number location can be used as regular memory.

## 2.1.9 Creating and Using Images

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

Creating an image is done in Interactive Mode by following the same steps described in [Section 2.1.4](#), followed by pressing the "Save Image File As..." or "Save to Image" buttons. The first button saves the code files and configuration options as a binary file and image file locally on disk, and the second button saves this information directly to the MSP Gang Programmer internal memory. Note that to use the MSP Gang Programmer in Standalone mode, you need to program at least one image to internal memory or read a binary file from an SD card (using the SD card connector on the MSP Gang Programmer). If you intend to modify the contents of an image at a later date, it is advisable to save the configuration options as a project. Because an image is read-only, reading a project file is the only way to recreate images easily without reentering the configuration options from scratch. After the project is loaded, a change can be made and a new image with the same name can be created to overwrite the previous one.



**NOTE: Do not overwrite images unnecessarily during production**

The image flash memory has a specified 10000 endurance cycles. Therefore, over the lifetime of the product, each image can be reliably reprogrammed 10000 times. Reprogramming images should be done once per production setup, rather than per programming run. Reprogramming the image per programming run will quickly exhaust flash endurance cycles and result in errant behavior.

---

In total, 96 different images can be saved internally in the MSP Gang Programmer or one image can be saved on an SD card. Each image can be selected at any time to program the target devices. The 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. In order for the image file to be usable from the SD card, copy only the binary file (".spgangbin") to the SD card and preserve the proper extension (Note that binary files are not encrypted). For redistribution to a customer, the image file can be sent and encrypted with additional security features.

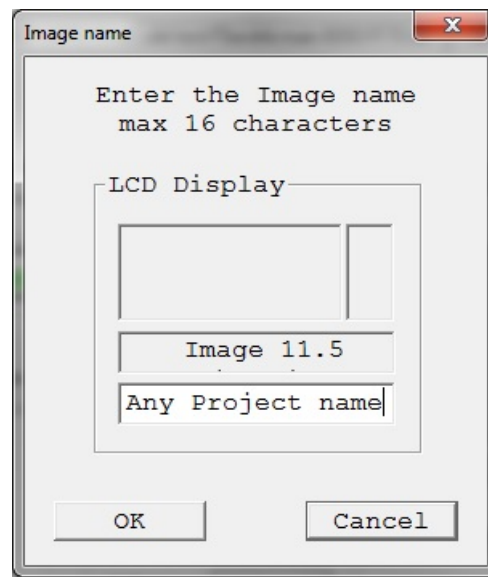
When a new image is saved to a file or to a MSP Gang Programmer internal memory, an image configuration screen appears (see [Figure 2-18](#)). Enter any name up to 16 characters. This name is displayed in the GUI image selector (see [Figure 2-1](#)) on the bottom line of the MSP Gang Programmer LCD screen when the corresponding image is selected. Press OK when the name is entered.

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 96 different images can be saved internally, or one image can be saved on an SD-Card. Each image can be selected at any time to program the target devices. The 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 2-18](#)). 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.

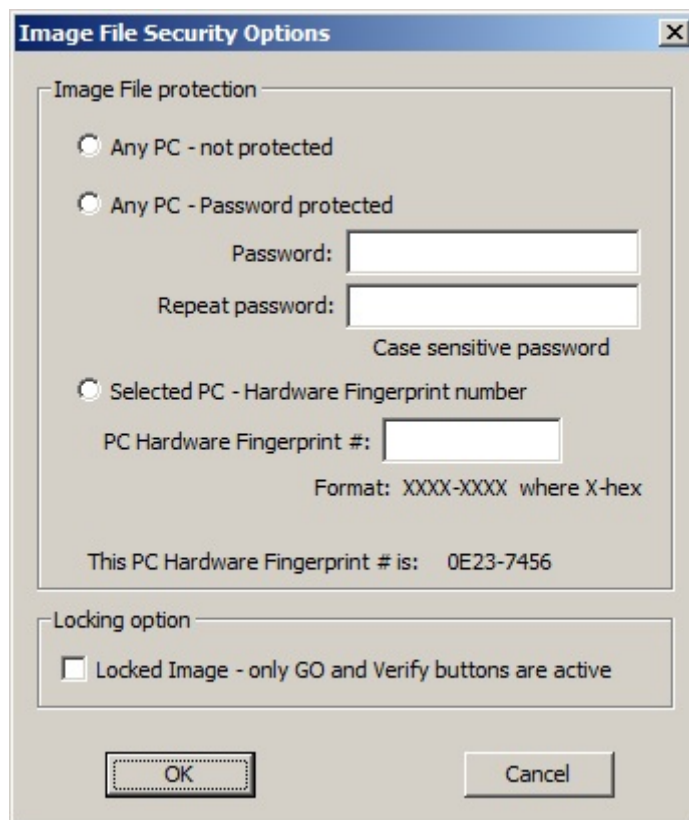


NOTE: The image name is limited to 16 characters. This name is shown on the LCD display of the MSP Gang Programmer, and Image pulldown menu in the GUI.

**Figure 2-18. Image Name Configuration Screen**

**NOTE:** Since version 1.2.1.0, the number of images has increased from 16 (512KB each) to 96 (64KB each). Total image memory has decreased from 8MB to 6MB. For compatibility purposes with older images, the numbering scheme for the new images uses an index and subindex format (for example, 1.0, 1.1, 1.2... 1.7, 2.0, 2.1, 2.2... 12.7). The first index selects the 512KB image memory block, and the subindex selects which 64KB portion of the 512KB block is used. Old images that occupy 512KB always have the subindex as 0 (for example, old Image 1 is now Image 1.0). Newly created images can occupy one 64KB block or more (for example, a 128KB image stored in image number 2.1, will be saved to occupy blocks 2.1 and 2.2). Images 13 to 16 will be removed in future versions; however, during the transition period, they can only be read or erased (that is, they are marked as read-only in the GUI).

The screen shown in [Figure 2-19](#) 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 2-19. 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 2-20](#) 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 2-20. 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.

### 2.1.10 Programming From Image File

An image file can be used to program target devices from a self-contained read-only file that has all the necessary configuration options and code files already included. By selecting the "From Image File" Mode you can use an image file created using the steps described in Section 2.1.9. If the image is password protected you are prompted to enter the password before you can use the image. Alternatively, if the image is restricted to be used on a specific PC you are unable to use the image unless your PC matches the hardware fingerprint (for instructions on how to use images from MSP Gang Programmer internal memory see Section 2.1.2).

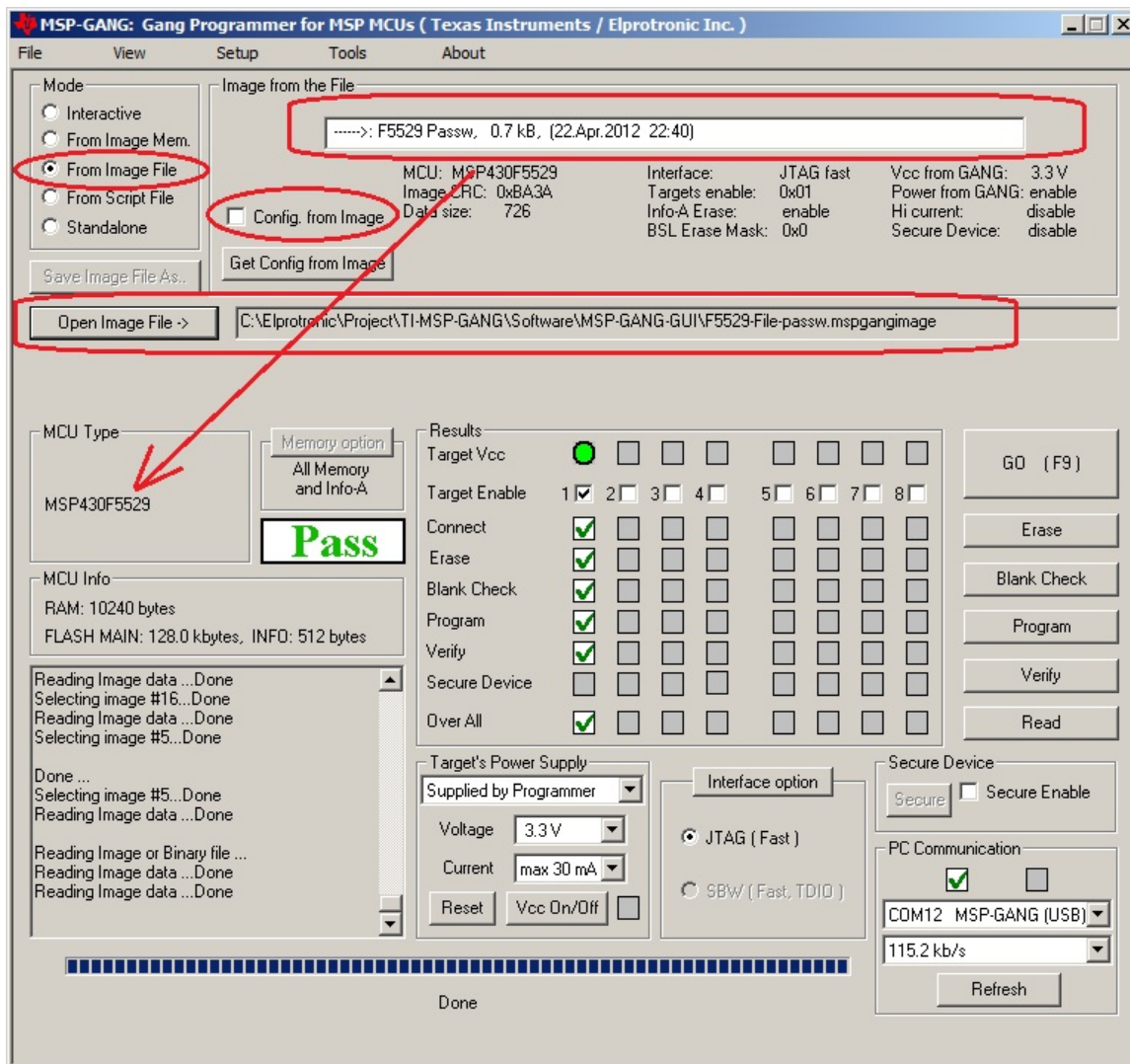
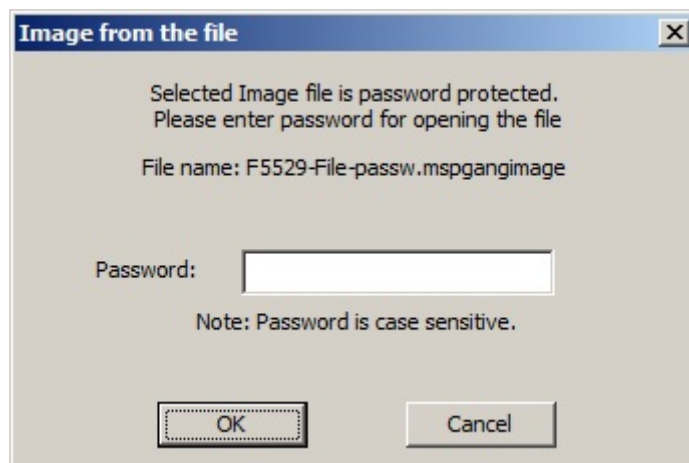


Figure 2-21. Programming From Image File



**Figure 2-22. Password for Image File**

### 2.1.11 Programming From SD Card

The MSP Gang Programmer can program target devices with an image loaded from an external SD card. To program from an external SD card, copy a binary file (".mspgangbin") created using steps described in [Section 2.1.9](#) to the root directory of the SD card (preserve the original extension of ".mspgangbin"). If multiple binary files are present in the root directory of the SD card, the first one found is used (the first one found is not necessarily the first one alphabetically). To ensure that the desired binary file is used, verify that only one binary file with the proper extension .mspgangbin is present in the root directory. The name of the selected file is displayed on the LCD screen of the MSP Gang Programmer.

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

### 2.1.12 File Extensions

MSP Gang Programmer software accepts the following file extensions:

Code hex files

*.txt	Texas Instruments
*.s19,*.s28,*.s37	Motorola
*.hex	Intel
*.a43	Intel hex format with extensions specified by IAR

Image files

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

Script files

*.mspgangsf	script file
-------------	-------------

Project configuration files

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

### 2.1.13 Checksum Calculation

The checksum (CS) that is displayed on the side of the code file name is used for internal verification. The CS is calculated as the 32-bit arithmetic sum of the 16-bit unsigned words in the code file, without considering the flash memory size or location. If any portion of the code file specifies only one byte instead of a 16-bit word, the missing byte is defined as 0xFF for the CS calculation.

The following formula is used.

```
DWORD CS;
DWORD XL, XH;

CS = 0;
for( addr = 0; addr < ADDR_MAX; addr = addr + 2 )
{
    if(( valid_code[ addr ] ) || ( valid_code[ addr+1 ]))
    {
        if( valid_code[ addr ] )
            XL = (DWORD) code[ addr ];
        else
            XL = 0xFF;

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

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

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

```
-----
@FC00
F2 40

@FC90
28 02 68 92 DB 3B 38 80 05 00 58

@FFFC
4E F9 B6 FA
q
-----
```

The CS is calculated as shown below:

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

### 2.1.14 Commands Combined With the Executable File

Programming executable file can be opened with the following commands:

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

For example:

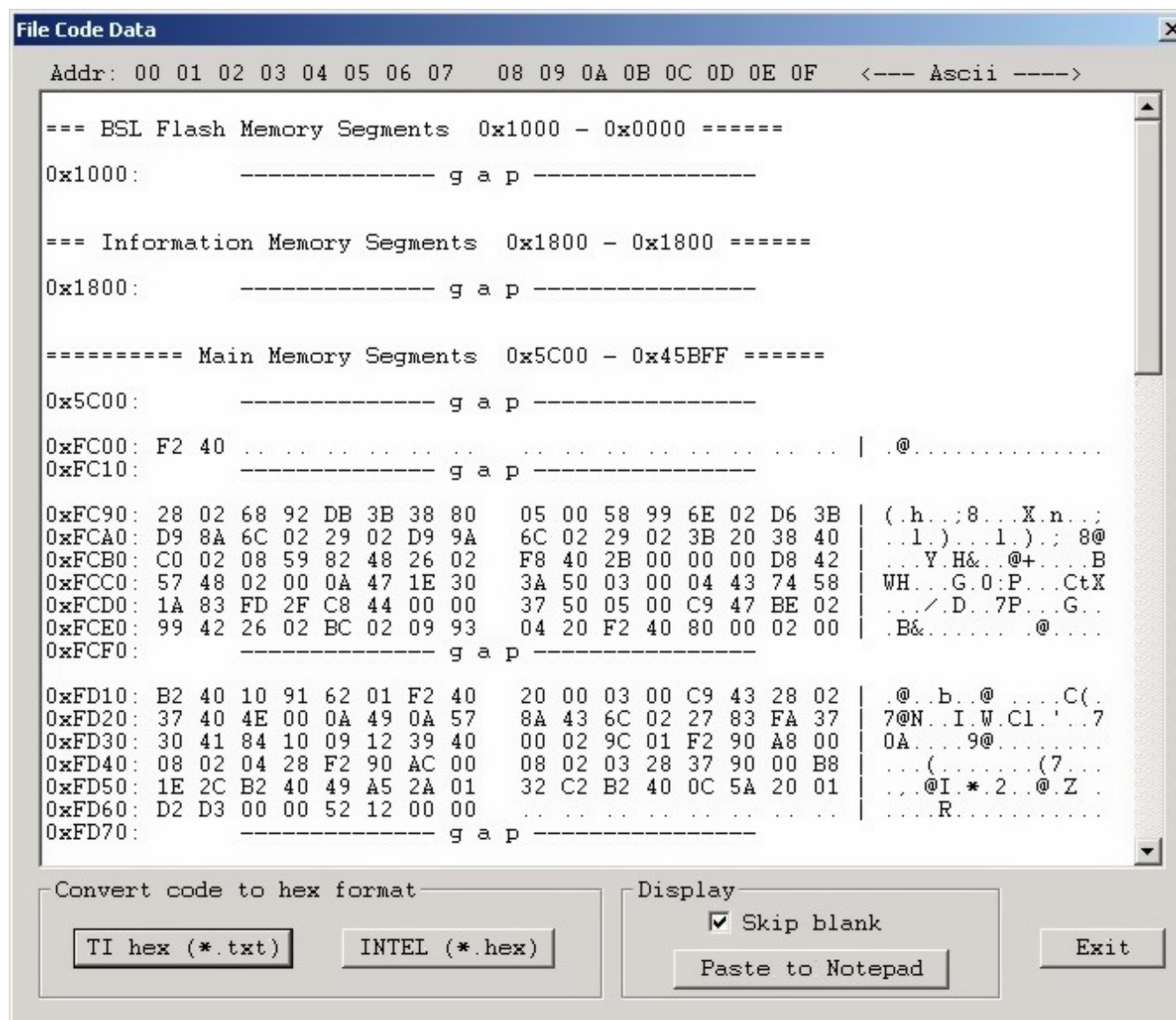
```
MSP-GANG.exe -sf test.mspgangsf
or
MSP-GANG.exe -prj test1.mspgangproj
or
MSP-GANG.exe -prj test1.mspgangproj -sf test.mspgangsf
```



## 2.2 Data Viewers

Data from code files and from flash memory can be viewed and compared in data viewers. Contents of the selected file can be viewed by selecting the View→Code File Data option from the drop-down menu. The Code data viewer, shown in Figure 2-23, displays the code address on the left side, data in hex format in the central column, the same data in ASCII format in the right column. Data in hex format is displayed from 0x00 to 0xFF for addresses corresponding to the code file. Data from other addresses is displayed as double dots (.). If code size exceeds flash memory size in the selected microcontroller, this warning message is displayed first.

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



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

Figure 2-23. Code File Data

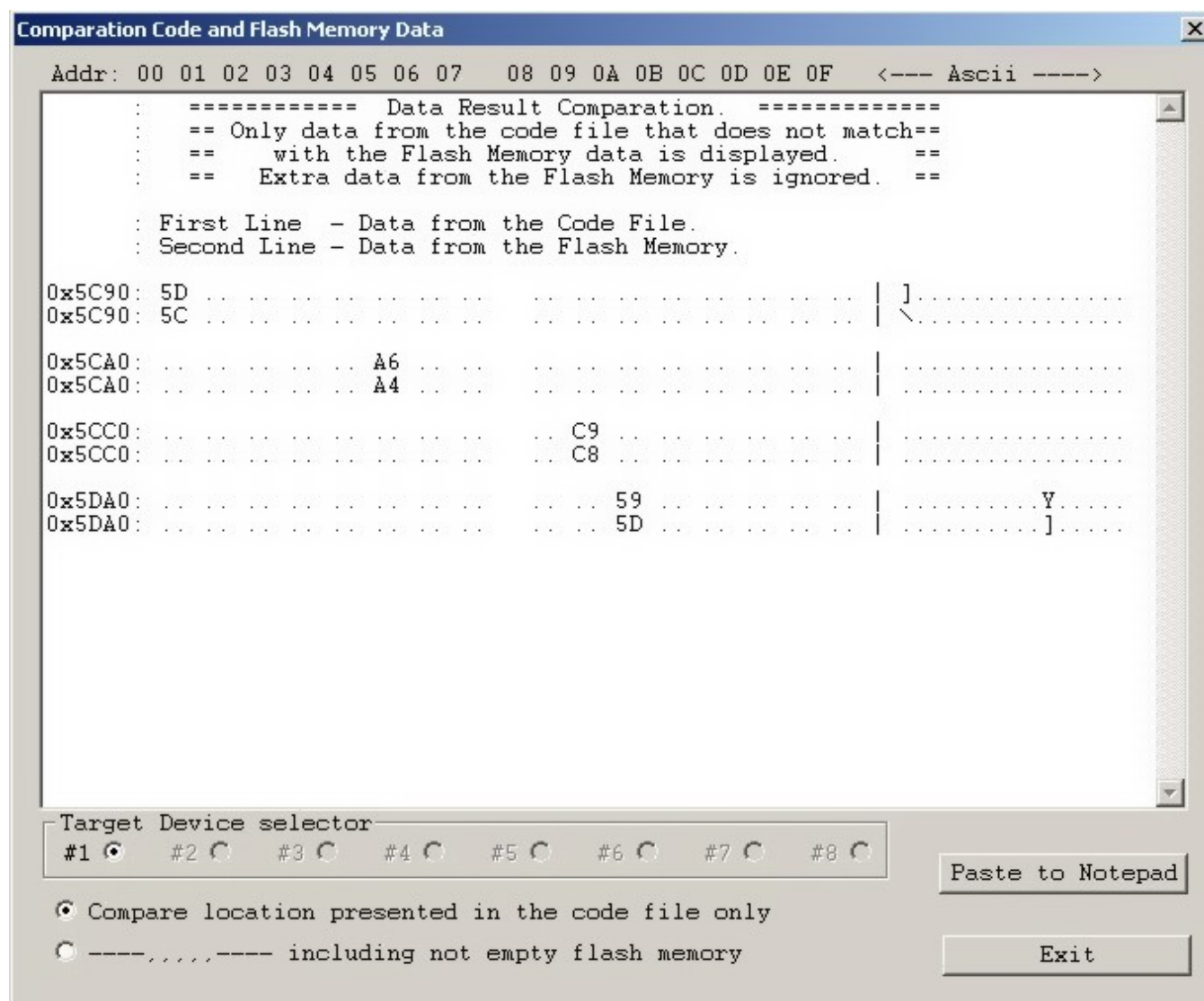
The contents of the code viewer can be converted to TI (\*.txt) or Intel (\*.hex) file format by clicking on the TI hex or INTEL button.



Contents of flash memory data can be viewed by selecting the View→Flash Memory Data option from the 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 2-23](#).

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 2-24](#).

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



NOTE: Only 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 2-24. Comparison of Code and Flash Memory Data of the Target Microcontroller**

## 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, or erase one block of the main memory. For that reason, many more tasks are displayed in the report window than are described above. For example, when programming the MSP430F5438 the following information would be displayed in the report window:

```
Executing Main Process...
```

```
.....
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 : Blank-1800
16 : Blank-1880
17 : Blank-1900
18 : Blank-5C00
19 : Blank-10000
20 : Blank-20000
21 : Blank-30000
22 : Blank-40000
23 : Write-FC00
24 : Write-FC90
25 : Write-FD10
26 : Write-FD80
27 : Write-FFE2
28 : Verify-FC00
29 : Verify-FC90
30 : Verify-FD10
31 : Verify-FD80
32 : Verify-FFE2
33 : Gl.Verify-1800
34 : Gl.Verify-5C00
35 : Gl.Verify-10000
36 : Gl.Verify-20000
37 : Gl.Verify-30000
38 : Gl.Verify-40000
39 : closing target
40 : Done
0 : Finished
```

This report indicates that sectors INFO-B, INFO-C, INFO-D, and the main memory block have been erased (tasks 2 to 14) blank checked (tasks 15 to 22), programmed (tasks 23 to 27) and verified (tasks 28 to 38). 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 512KB 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 512KB 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.

==== Errors from the BOOT loader. ====

ERROR # 001 - BOOT Firmware only is in the MSP-GANG! The API Firmware should be downloaded.  
 ERROR # 002 - API Firmware CRC is not present! The API Firmware should be reloaded.  
 ERROR # 003 - API Firmware CRC error! The API Firmware should be reloaded.  
 ERROR # 004 - BOOT CRC error in the MSP-GANG!

==== Errors from the MSP-GANG Firmware. ====

ERROR # 010 - CRC Access key. Key corrupted. Access to programmer is blocked.  
 ERROR # 011 - Invalid programmer's access key. Access to programmer is blocked.  
 ERROR # 012 - Unknown interface.  
 ERROR # 013 - Vcc is too low.  
 ERROR # 014 - Vcc is too high.  
 ERROR # 015 - VtIO is too low.  
 ERROR # 016 - VtIO is too high.  
 ERROR # 017 - Header CRC. Image header is corrupted.  
 ERROR # 018 - Script CRC. Image script is corrupted.  
 ERROR # 019 - Exceed script number.  
 ERROR # 020 - Script command unknown.  
 ERROR # 021 - MCU Fetch synch failed.  
 ERROR # 022 - CPU JTAG synch failed.  
 ERROR # 023 - MCU device initialization failed.  
 ERROR # 024 - RAM firmware download failed.  
 ERROR # 025 - Flash blank check failed.  
 ERROR # 026 - Flash read verify failed.  
 ERROR # 027 - Flash write failed.  
 ERROR # 028 - Image flash write initialization failed.  
 ERROR # 029 - Image flash lock failed.  
 ERROR # 030 - Invalid script type/name.  
 ERROR # 031 - Script size too big.  
 ERROR # 032 - Used wrong MCU.  
 ERROR # 033 - IR communication failed.  
 ERROR # 034 - Page info number out of range.  
 ERROR # 035 - Address too high.  
 ERROR # 036 - Target number out of range.  
 ERROR # 037 - Address not even.  
 ERROR # 038 - Size not even.  
 ERROR # 039 - DCO calibration frequency out of range.  
 ERROR # 040 - DCO calibration failed.  
 ERROR # 041 - Gang flash write failed.  
 ERROR # 042 - SD Card - Read Response Error  
 ERROR # 043 - SD Card - Boundary Address Error  
 ERROR # 044 - SD Card - Initialization timeout  
 ERROR # 045 - SD Card - Read timeout  
 ERROR # 046 - SD Card - Initialization Error  
 ERROR # 047 - SD Card - CRC7 Error  
 ERROR # 048 - SD Card - CRC16 Error  
 ERROR # 049 - SD Card - Write CRC Error  
 ERROR # 050 - SD Card - Data Write Error  
 ERROR # 051 - SD Card - Write Timeout  
 ERROR # 052 - SD Card - MBR Sector Error  
 ERROR # 053 - SD Card - Volume Error  
 ERROR # 054 - SD Card - Address in File Error  
 ERROR # 055 - SD Card - Read File Error  
 ERROR # 056 - SD Card - File not found  
 ERROR # 057 - Hardware Rev-0. option not supported.  
 ERROR # 058 - Flash PSA verification failed.  
 ERROR # 059 - Flash read (Rx) data error.  
 ERROR # 060 - Vpp too low to secure fuse.  
 ERROR # 061 - Secure key incorrect.  
 ERROR # 062 - Secure device failed.  
 ERROR # 063 - Target not open, no access.  
 ERROR # 064 - BSL password wrong index.  
 ERROR # 065 - BSL read (Rx) timeout.  
 ERROR # 066 - BSL firmware download failed.

```

ERROR # 067 - Unknown command.
ERROR # 068 - Fast BSL initialization failed.
ERROR # 069 - MSP-
GANG current overload. Reduce amount of target devices to 5 or select 30 mA output current.
ERROR # 070 - Wrong BSL passwords. Target cannot be unlocked.

==== Errors from the MSP-GANG DLL. ====

ERROR # 301 - Communication - Frame has errors !
ERROR # 302 - Unable to open COM port - already in use?
ERROR # 303 - Unable to close COM port !
ERROR # 304 - Unable to modify COM port state !
ERROR # 305 - Synchronization failed. Programmer connected?
ERROR # 306 - Timeout during operation - Correct COM port selected?
ERROR # 307 - Wrong baud rate specified !
ERROR # 308 - Communication Port baud rate change
ERROR # 309 - Communication port - diagnostic response error
ERROR # 310 - Open Comm port - invalid handle value
ERROR # 311 - Invalid Comm Port Setup
ERROR # 312 - Open Comm Port timeout
ERROR # 313 - Get Comm Port state error
ERROR # 321 - Command did not complete correctly !
ERROR # 322 - Command failed or not defined or Target not accessible !
ERROR # 323 - Could not read 'default.mspsgangcfg'!
ERROR # 324 - File contains invalid record !
ERROR # 325 - Unexpected end of file !
ERROR # 326 - Error during file I/O !
ERROR # 327 - Selected file is of unrecognizable format !
ERROR # 328 - Unable to open file !
ERROR # 329 - Function argument(s) out of range !
ERROR # 330 - Note: Boot downloaded
ERROR # 331 - WARNING: Temporary function blocked, due to used main polling
ERROR # 332 - Image Memory corrupted or erased ! Load Image.
ERROR # 333 - Target not accessible !
ERROR # 334 - Verification failed !
ERROR # 335 - Main Process Parameters not yet set ! Load Image.
ERROR # 336 - Could not erase Image Buffer !
ERROR # 337 - Could not load Image Buffer !
ERROR # 338 - Could not load Main Process Parameters !
ERROR # 339 - Could not select Baud Rate !
ERROR # 340 - WARNING: Could not set target voltage -
Short circuitry or settling time too small?
ERROR # 341 - Invalid firmware command !
ERROR # 342 - Power supply voltage too low !
ERROR # 343 - WARNING: Sense voltage out of range - Check pin MSP_VCC_IN of target connector !
ERROR # 344 - Wrong target device connected !
ERROR # 345 - No target device connected
ERROR # 346 - File(s) contains already specified data (code overwritten)
ERROR # 347 - Selected Image number out of range
ERROR # 348 - Could not open the configuration file.
ERROR # 349 - Script Header size error
ERROR # 350 - Image ID error. Image ignored, program terminated.
ERROR # 351 - Image contents (size, no of tasks) error. Program terminated.
ERROR # 352 - Image CRC error. Program terminated.
ERROR # 353 - WARNING: Code overwritten. Code from the file written to already used location.
ERROR # 354 - Code in the file contains invalid data.
ERROR # 355 - Open File error
ERROR # 356 - Extension or file name error
ERROR # 357 - Wrong password for opening the image file
ERROR # 358 - Wrong PC hardware fingerprint # for opening the image file
ERROR # 359 - Image file ID error or file corrupted
ERROR # 360 - Check Sum of the Image file error or file corrupted
ERROR # 361 - Wrong header in the image file or file corrupted
ERROR # 362 - Image file is not for the MSP-GANG programmer.
ERROR # 363 - Image file contents error or file corrupted.
ERROR # 364 - Unknown protection mode of the image file or file corrupted.

```

```

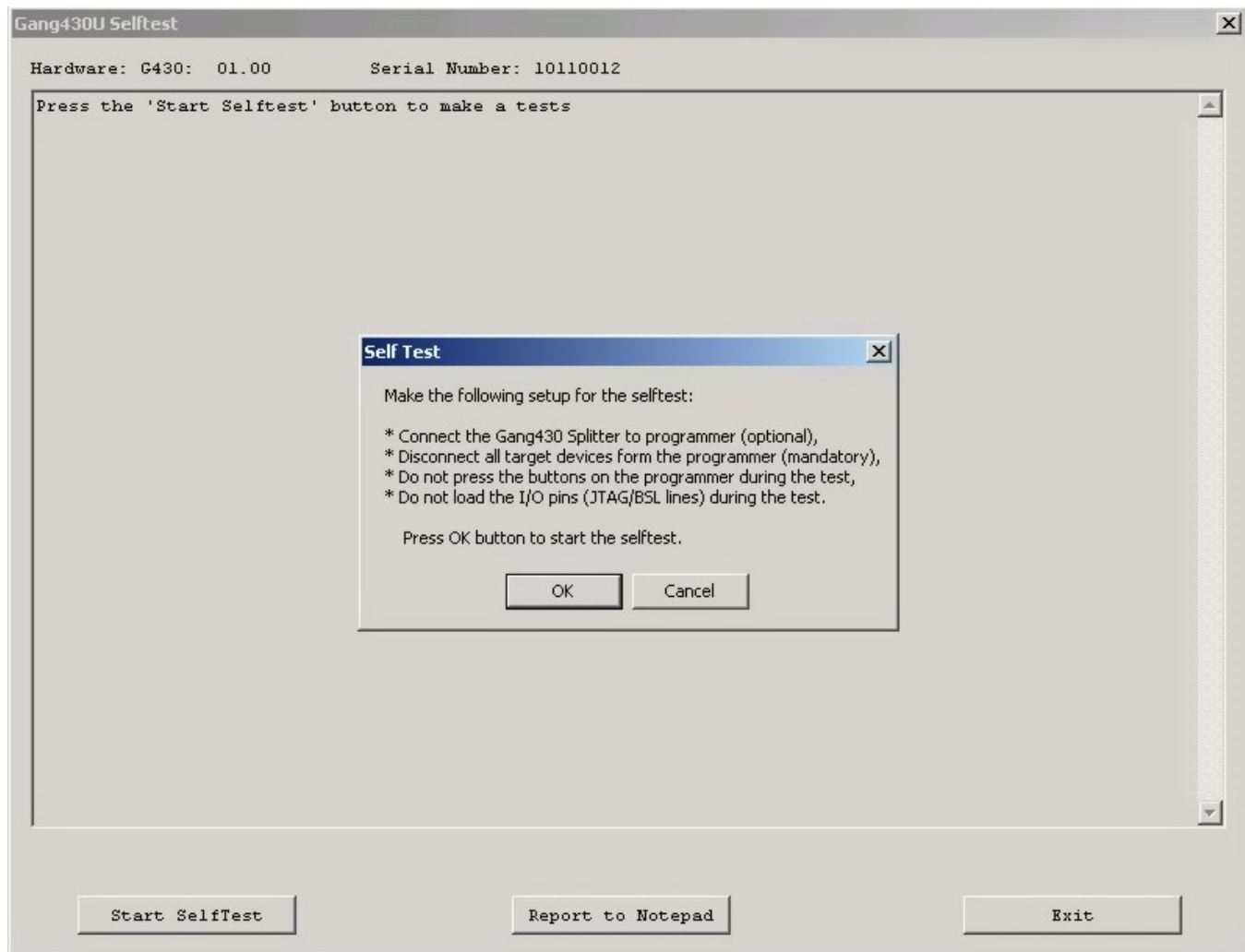
ERROR # 365 - Data offset in the image file error or file corrupted.
ERROR # 366 - Hex data conversion in the image file error or file corrupted.
ERROR # 367 - Image file corrupted.
ERROR # 368 - Image file cannot be unlocked
ERROR # 369 - Customized MCUs license file open error
ERROR # 370 -
    WARNING: Code specified for the BSL space location, but access to the BSL is locked.
ERROR # 371 - Info memory page number is out of range.
ERROR # 372 - COM ports scan number is too low.
ERROR # 373 - Selftest data size too high.
ERROR # 374 - Data size too high.
ERROR # 377 - Gang mask ZERO. No target devices enabled. Nothing to do.
ERROR # 378 - Address definition.
ERROR # 379 - Data size is below 2.
ERROR # 380 - Invalid DCO number.
ERROR # 381 - Command not implemented.
ERROR # 382 - Wrong Target number
ERROR # 383 - Code File error.
ERROR # 384 - Password File error.
ERROR # 385 - Nothing to program/verify - empty code in selected memory space.
ERROR # 386 - Code out of range of the selected MCU.
ERROR # 387 - Invalid name index.
ERROR # 388 - Warning: Part of the code ignored. Check the memory setup.
ERROR # 391 - Image data size too long
ERROR # 392 - Terminated by user
ERROR # 393 - Code specified in the Retain Data space
ERROR # 394 - Number of tasks out of range
ERROR # 395 - Data blocks in all tasks out of range
ERROR # 396 - WARNING: Code from file is out of range in selected MCU.
ERROR # 397 - Option not supported for selected MCU.
ERROR # 398 - Wrong destination.
ERROR # 399 - The Address and Size of the Locking Options out of range.
ERROR # 400 - Warning - empty data or string.
ERROR # 401 - Invalid input parameter(s).
ERROR # 402 - Selected Image is READ ONLY and cannot be reprogrammed.
ERROR # 403 - Serial number location and code overlap.
ERROR # 404 - Serial number size error.
ERROR # 405 - No serial number found in serial number file.
ERROR # 406 - Serial number address or size error.
ERROR # 407 - Code located in the MPU-IPE space without selected access to MPU-
    IPE in Memory Options.
ERROR # 408 - External Power Supply is required for selected MCU.
ERROR # 999 - Invalid error number !

```

## 2.4 Self Test

The MSP Gang Programmer Self Test program can test most of the hardware for correctness. Connect the programmer to a computer running MSP Gang Programmer software. If using a Gang Splitter, connect it to the 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 2-25](#), 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 self-test 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 2-25. Self Test**

The following is a typical self test report:

```
=== MSP-GANG Self test results ( Saturday, November 27, 2011, 19:08:43 ) ===
Adapter SN -----: 10110012
Hardware -----: G430: 01.01
Access key -----: MSP430 - Gang Programmer
Silicon Number --: 24D4 CC47 0400 1B00
API Firmware ----: MSP-Gang A430: 01.00.08.00
BOOT Firmware ---: G430BOOT B430: 01.00.01.00
GUI Software ----: MSP-Gang-GUI G430: 01.00.08.00
DLL Software ----: MSP-Gang-DLL D430: 01.00.08.00
===== 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: TDOI Tx-bus (\#1 HIGH) 0x01 ( 0x01 to 0x01) Result: 0x01 ... >> OK <<
137: TDOI Tx-bus (\#2 HIGH) 0x02 ( 0x02 to 0x02) Result: 0x02 ... >> OK <<
138: TDOI Tx-bus (\#3 HIGH) 0x04 ( 0x04 to 0x04) Result: 0x04 ... >> OK <<
139: TDOI Tx-bus (\#4 HIGH) 0x08 ( 0x08 to 0x08) Result: 0x08 ... >> OK <<
140: TDOI Tx-bus (\#5 HIGH) 0x10 ( 0x10 to 0x10) Result: 0x10 ... >> OK <<
141: TDOI Tx-bus (\#6 HIGH) 0x20 ( 0x20 to 0x20) Result: 0x20 ... >> OK <<
142: TDOI Tx-bus (\#7 HIGH) 0x40 ( 0x40 to 0x40) Result: 0x40 ... >> OK <<
143: TDOI Tx-bus (\#8 HIGH) 0x80 ( 0x80 to 0x80) Result: 0x80 ... >> OK <<
144: TDOI Tx-Rx (\#1 HIGH) 0x01 ( 0x01 to 0x01) Result: 0x01 ... >> OK <<
145: TDOI Tx-Rx (\#2 HIGH) 0x02 ( 0x02 to 0x02) Result: 0x02 ... >> OK <<
146: TDOI Tx-Rx (\#3 HIGH) 0x04 ( 0x04 to 0x04) Result: 0x04 ... >> OK <<
147: TDOI Tx-Rx (\#4 HIGH) 0x08 ( 0x08 to 0x08) Result: 0x08 ... >> OK <<
148: TDOI Tx-Rx (\#5 HIGH) 0x10 ( 0x10 to 0x10) Result: 0x10 ... >> OK <<
149: TDOI Tx-Rx (\#6 HIGH) 0x20 ( 0x20 to 0x20) Result: 0x20 ... >> OK <<
150: TDOI Tx-Rx (\#7 HIGH) 0x40 ( 0x40 to 0x40) Result: 0x40 ... >> OK <<
151: TDOI Tx-Rx (\#8 HIGH) 0x80 ( 0x80 to 0x80) Result: 0x80 ... >> OK <<
152: TDOI Rx-bus (\#1 HIGH) 0x01 ( 0x01 to 0x01) Result: 0x01 ... >> OK <<
153: TDOI Rx-bus (\#2 HIGH) 0x02 ( 0x02 to 0x02) Result: 0x02 ... >> OK <<
154: TDOI Rx-bus (\#3 HIGH) 0x04 ( 0x04 to 0x04) Result: 0x04 ... >> OK <<
155: TDOI Rx-bus (\#4 HIGH) 0x08 ( 0x08 to 0x08) Result: 0x08 ... >> OK <<
156: TDOI Rx-bus (\#5 HIGH) 0x10 ( 0x10 to 0x10) Result: 0x10 ... >> OK <<
157: TDOI Rx-bus (\#6 HIGH) 0x20 ( 0x20 to 0x20) Result: 0x20 ... >> OK <<
158: TDOI Rx-bus (\#7 HIGH) 0x40 ( 0x40 to 0x40) Result: 0x40 ... >> OK <<
159: TDOI Rx-bus (\#84 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 <<

```

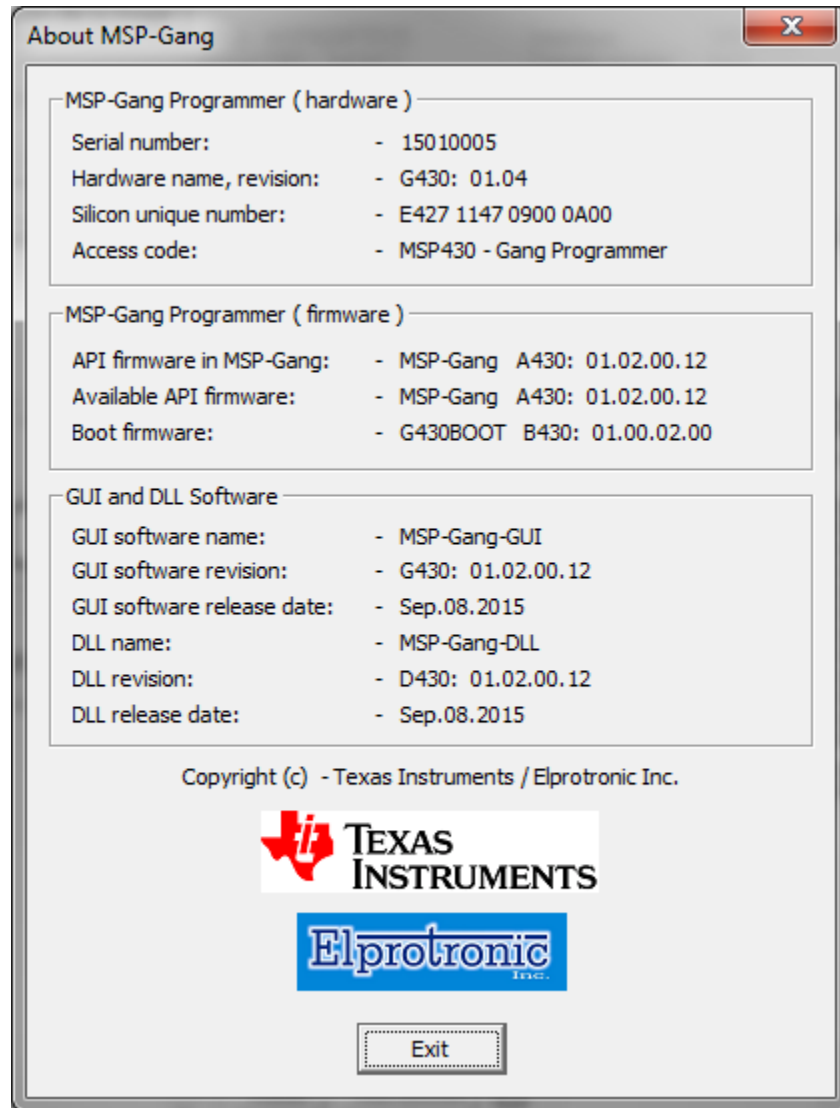
```

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.

```

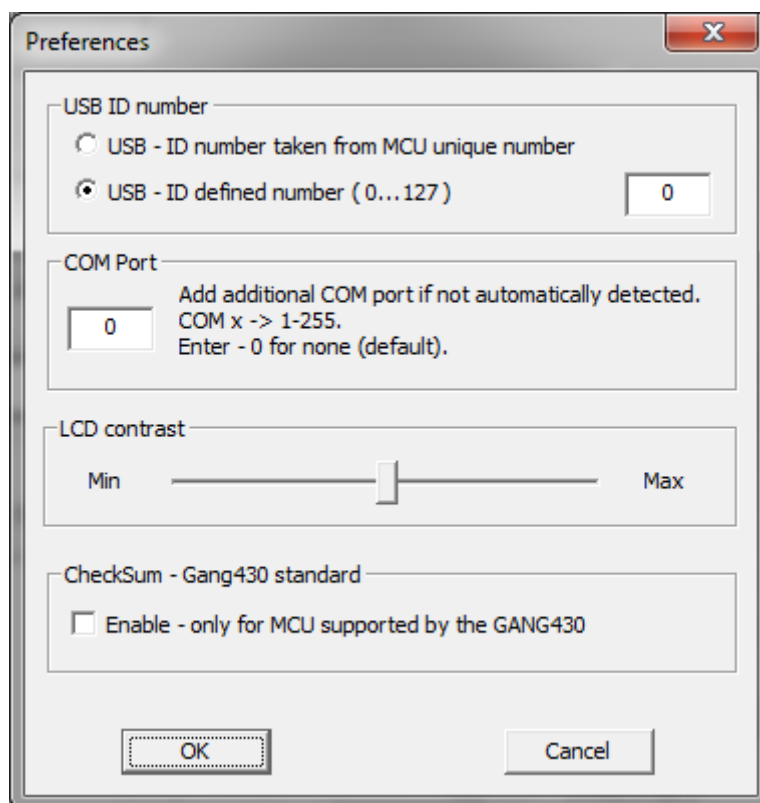
## 2.5 Label

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 2-26](#).



**Figure 2-26. Information About the MSP Gang Programmer**

## 2.6 Preferences



**Figure 2-27. Preferences Selection Window**

### 2.6.1 USB ID Number

This parameter specifies the ID number of the USB driver. A different number causes the MSP-GANG programmer to use a different COM port.

- Driver ID is taken from MSP-GANG serial number.
- Driver ID is chosen by the user. Valid values are 0 to 127, inclusive (default is 0). Set different values when connecting multiple programmers to the same PC.

### 2.6.2 COM Port

Manually add the specified COM port number to the list of selectable COM ports. Useful when a COM port is not detected by software and is not added to the available COM port list, but the user knows that the specific COM port is available. When set to 0, no extra COM port is added.

### 2.6.3 LCD Contrast

Contrast for the LCD display on the actual MSP-GANG Programmer. The saved value is stored internally inside the MSP-GANG Programmer.

### 2.6.4 Checksum – Gang430 Standard

Old versions of the MSP-GANG Programmer (named MSP-GANG430) used a different algorithm to calculate checksums. Enabling this option shows two checksums on the new MSP-GANG Programmer: both the old MSP-GANG430 checksum and the new MSP-GANG checksum are shown. The old checksum is shown only for MCUs supported by the old MSP-GANG430 Programmer.

## 2.7 Benchmarks

This section shows the results of timing benchmarks used on the MSP Gang Programmer to measure the programming speed. Each table shows the result of the benchmark when programming with the JTAG and SBW interfaces. Identical programming speed is seen whether programming one device or eight devices simultaneously, because programming each MCU is done in parallel.

### 2.7.1 Benchmarks for MSP430F5xx

**Table 2-1. Benchmark Results – MSP430F5438A, 256kB Code<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> Programming speed and verify speed without startup procedures (access to target device).

**Table 2-2. Benchmark Results – MSP430F5438A, 250kB Code, Mode: From Image<sup>(1)</sup>**

	Erase, Blank Check, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)
JTAG Fast	9.6	0.4	31
JTAG Med	16.6	0.5	17
JTAG Slow	32	0.6	8
SBW Fast	38	0.6	7
SBW Med	58	0.7	4.6
SBW Slow	110	1.1	2.5

<sup>(1)</sup> Programming speed and verify speed without startup procedures (access to target device).

**Table 2-3. Benchmark Results – MSP430F5438A, 250kB Code, Mode: Interactive, Communication by USB<sup>(1)</sup>**

	Erase, Blank Check, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)
JTAG Fast	13.5	0.5	21
JTAG Med	19.7	0.6	14
JTAG Slow	36	0.7	7.5
SBW Fast	42	0.7	6.3
SBW Med	61	0.8	4.3
SBW Slow	114	1.2	2.3

<sup>(1)</sup> Programming speed and verify speed without startup procedures (access to target device).



## 2.7.2 Benchmarks for MSP430FR5xx

**Table 2-4. Benchmark Results – MSP430FR5994, 256kB Code, Mode: From Image<sup>(1)</sup>**

	Erase, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)
JTAG Fast	5.3	0.9	67
JTAG Med	11.4	1.1	27
JTAG Slow	24.2	1.5	11.8
SBW Fast	29.6	1.6	9.5
SBW Med	44.5	2.1	6.2
SBW Slow	86.4	3.3	3.2

<sup>(1)</sup> Programming speed without startup procedures (access to target device).

**Table 2-5. Benchmark Results – MSP430FR5994, 256kB Code, Mode: Interactive, Communication by USB<sup>(1)</sup>**

	Erase, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)
JTAG Fast	8.0	1.0	40
JTAG Med	14.1	1.2	21
JTAG Slow	26.8	1.5	10.6
SBW Fast	32.3	1.7	8.7
SBW Med	47.1	2.1	5.8
SBW Slow	89	3.3	3.1

<sup>(1)</sup> Programming speed without startup procedures (access to target device).

## 2.7.3 Benchmarks for MSP430F2xx

**Table 2-6. Benchmark Results – MSP430F2619, 120kB Code, Mode: From Image<sup>(1)</sup>**

	Erase, Blank Check, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)
JTAG Fast	8.3	0.4	16.2
JTAG Med	15	0.5	8.6
JTAG Slow	25	0.6	5.1

<sup>(1)</sup> Programming speed without startup procedures (access to target device).

**Table 2-7. Benchmark Results – MSP430F2619, 120kB Code, Mode: Interactive, Communication by USB<sup>(1)</sup>**

	Erase, Blank Check, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)
JTAG Fast	10.4	0.5	12.7
JTAG Med	17	0.5	7.5
JTAG Slow	27	0.6	4.7

<sup>(1)</sup> Programming speed without startup procedures (access to target device).



## 2.7.4 Benchmarks for MSP432P401R

**Table 2-8. Benchmark Results – MSP432P401R, 256kB Code, Mode: From Image<sup>(1)</sup>**

	Erase, Blank Check, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)
JTAG Fast	5.3	0.3	55.6
JTAG Med	13.0	0.8	21.5
JTAG Slow	27.0	1.2	10.5
SWD Fast	6.6	0.7	45.0
SWD Med	15.0	0.9	19.0
SWD Slow	32.0	1.5	9.0

<sup>(1)</sup> Programming speed without startup procedures (access to target device).

**Table 2-9. Benchmark Results – MSP432P401R, 256kB Code, Mode: Interactive, Communication by USB<sup>(1)</sup>**

	Erase, Blank Check, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)
JTAG Fast	8.0	0.3	36.0
JTAG Med	15.5	0.5	18.0
JTAG Slow	29.0	0.8	9.5
SWD Fast	8.9	0.3	31.0
SWD Med	17.7	0.5	16.0
SWD Slow	34.0	0.9	8.0

<sup>(1)</sup> Programming speed without startup procedures (access to target device).

## ***Firmware***

---

### **3.1 Commands**

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

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

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

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

- Main process
- Interactive process
- Erase Image
- Read Info memory from 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
- Self-test

## 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 USB is several times faster than RS-232. Communication requires one start bit, eight data bits, even parity bit, and one stop bit. A software handshake is performed by a (not) acknowledge character.

## 3.3 Synchronization Sequence

To synchronize with the 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 -> 0xA5

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

### 3.4.1 Frame Structure

The data frame format follows the TI MSP 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.

The following abbreviations are used in [Table 3-1](#).

CMD	Command identification
R	Do not use this command. Used for internal communication.
T	Target number (1 to 8)
L1, L2	Number of bytes in AL through Dn. The valid values of these bytes are restricted as follows: L1 = L2, L1 < 255, L1 even.
A1, A2, A3	Block start address or erase (check) address or jump address LO or HI byte. The bytes are combined to generate a 24-bit word as follows: Address = A3 × 0x1000 + A2 × 0x100 + A1
LL, LH	Number of pure data bytes (maximum 250) or erase information LO or HI byte or block length of erase check (max is 0xFFFF).
D1...Dn	Data bytes
CKL, CKH	16-bit checksum LO or HI byte
xx	Can be any data
–	No character (data byte) received or transmitted
ACK	The acknowledge character returned by the MSP-GANG can be either DATA_ACK = 0x90 (frame was received correctly, command was executed successfully) or DATA_NAK = 0xA0 (frame not valid (for example, wrong checksum, L, L2), command is not defined, is not allowed).
PRS	DATA_IN_PROGRESS = 0xB0 – Tasks in progress. Use Get Progress Status (0xA5) command to get the status and check when task is finished.

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

MSP-GANG Firmware Command	PROMPT	CMD	L1	L2	A1	A2	A3	A4	LL	LH	D1	D2...Dn	CLK	CLH	ACK
"Hello"	0D												–	–	ACK
Boot Commands Disable	3E	2A	R	R	R	R	R	R	R	R	R	R	CKL	CKH	ACK
Boot Commands Enable	3E	2B	R	R	R	R	R	R	R	R	R	R	CKL	CKH	ACK
Diagnostic	3E	32	04	04	00	00	–	–	00	00	–	–	CKL	CKH	–
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 Firmware Update	3E	3B	R	R	R	R	R	R	R	R	R	R	CKL	CKH	ACK
Get Label	3E	40	04	04	00	00	–	–	00	00	–	–	CKL	CKH	–
Response-Get Label	80	00	8C	8C	D1	D2	D3	D4	D5	D6	D7	D8...D140	CKL	CKH	–
Get Progress Status	A5												–	–	
Response----,---,---	80	A5	D1	D2	D3	D4	D5	D6	D7	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	–6	00	D1	D2...Dn-6	CKL	CKH	ACK
Verify Image CRC	3E	45	08	08	A1	A2	A3	A4	LL	LH	D1	D2	CKL	CKH	ACK
Get Image Header	3E	47	06	06	A1	A2	00	00	n	00	–	–	CKL	CKH	–
Response---,---,---	80	0	n	n	D1	D2	–	–	D3	D4	D5	D6...Dn	CKL	CKH	
Read Gang Buffer	3E	49	4	4	T	0	–	–	n	0	–	–	CKL	CKH	–
Response---,---,---	80	0	n	n	D1	D2	D3	D4	D5	D6	D7	D8...Dn	CKL	CKH	
Write Gang Buffer	3E	4A	n+4	n+4	T	0	–	–	n	0	D1	D2...Dn	CKL	CKH	ACK
Disable API Interrupts	3E	4C	4	4	R	R	–	–	R	R	–	–	CKL	CKH	ACK
Select Image	3E	50	4	4	A1	0	–	–	0	0			CKL	CKH	ACK

<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.

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

Display Message	3E	54	n+4	n+4	A1	A2	–	–	n	00	D1	D2...Dn	CKL	CKH	ACK
Set IO State	3E	4E	0C	0C	VL	VH	–	–	08	00	D1	D2...D8	CKL	CKH	ACK
Set Temporary Configuration	3E	56	06	06	A1	A2	–	–	2	0	D1	D2	CKL	CKH	ACK
Get Gang Status	3E	58	04	04	A1	0	–	–	0	0	–	–	CKL	CKH	–
Response —,,—	80	0	n	n	D1	D2	D3	D4	D5	D6	D7	D8...Dn	CKL	CKH	
Remote Selftest	3E	71	n+6	n+6	A1	A2	A3	A4	n	0	D1	D2...Dn	CKL	CKH	
Response—,—,,—	80	0	n	n	D1	D2	D3	D4	D5	D6	D7	D8...Dn	CKL	CKH	

### 3.4.2 Checksum

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

$$\text{CHECKSUM} = \text{INV} [ (B1 + 256 \times B2) \text{ XOR } (B3 + 256 \times B4) \text{ XOR} \dots \text{XOR } ((Bn - 1) + 256 \times Bn) ]$$

or

$$\text{CKL} = \text{INV} [ B1 \text{ XOR } B3 \text{ XOR} \dots \text{XOR } Bn-1 ]$$

$$\text{CKH} = \text{INV} [ B2 \text{ XOR } B4 \text{ XOR} \dots \text{XOR } Bn ]$$

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

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

## 3.5 Detailed Description of Commands

### 3.5.1 General

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

### 3.5.2 Commands Supported by the BOOT Loader

#### 3.5.2.1 "Hello" 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 RS-232, the "Hello" command must be sent a minimum of three times through RS-232. After this, an ACK (0x90) is transmitted through RS-232. This sequence also works in reverse, to reestablish communication between RS-232 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 information memory update. Use the MSP-GANG executable GUI software for updating firmware or information memory update if required.

### 3.5.2.3 Boot Commands Enable

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

Do not use this command. This command is used during firmware or information memory update. Use the MSP-GANG executable GUI software for updating firmware or information 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 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 information memory update. Use the MSP-GANG executable GUI software for updating firmware or information 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 information memory update. Use the MSP-GANG executable GUI software for updating firmware or information 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 information memory update. Use the MSP-GANG executable GUI software for updating firmware or information 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 replies 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 an internal 8-level FIFO buffer for progress status (8 internal buffers of 50 bytes each). This allows messages to be retrieved even if status has been changed a few times in the interval of 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 or nack
bytes	8-9	(WORD)	Finished tasks mask
byte	10		Cumulative gang mask
byte	11		Request gang mask
byte	12		Connected gang mask
byte	13		Erased gang mask
byte	14		Blank check gang mask
byte	15		Programmed gang mask
byte	16		Verified gang mask
byte	17		Secured gang mask
bytes	18-23		Spare
byte	24		Error number
byte	25		Internal VTIO (VTIO = data × 32 mV)
byte	26		VCC gang status mask – A
byte	27		VCC gang status mask – B
byte	28		VCC error mask
byte	29		VCC cumulative error mask
byte	30		JTAG init err mask
byte	31		JTAG Fuse already blown mask
byte	32		Wrong MCU ID mask
byte	33		Progress bar (0 – 100%)
bytes	34-50		Comment text (comment currently displayed on the LCD display)

Where, bytes 8-9 are task mask bits:

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

### 3.5.3 Commands Supported by Application Firmware

Commands supported by the application firmware give access to 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 through the 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 using the DLL rather than directly through the 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 50 4 4 A1 0 0 0 CKL CKH
Rx -> 90 (ACK)
```

The Select Image command sets a number for the current image. After this command, all operations that the MSP-GANG performs use this image. The MSP-GANG supports 96 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 used with 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; for example, what targets are connected or erased. In the comment bytes (34-50) is the current process, and the same message as is displayed on the LCD display.

### 3.5.3.3 Set Temporary Configuration 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, high or low current, external VCC enable or disable, VCC settle time, communication interface (JTAG or SBW), enabled target devices and enable process mask (for example, erase or program verify). The Set Temporary Configuration in MSP-GANG command allows modification of these parameters.

When the Main Process command is finished, then the temporary setups are erased and the configuration from the image memory is restored. When the modified configuration should be used in the next run, then the temporary configuration should be transferred to 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 of off

Data	0x0001	ON
Data	0x0000	OFF

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

High (50 mA) current from programmer enable or disable

Data	0x0001	Enable
Data	0x0000	Disable

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

Set interface configuration

Data	0x0000	SBW through TDOI line
Data	0x0001	SBW through 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
------	------------------	-------------------------------

## CFG\_TMP\_BSL\_1ST\_PASSW (36)

Data	0x00	BSL_ANY_PASSW
Data	0x01	BSL_PASSW_FROM_CODE_FILE
Data	0x02	BSL_PASSW_FROM_PASSWORD_FILE
Data	0x03	BSL_EMPTY_PASSW

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

GET_APP_FLAGS	10
GET_LAST_STATUS	12
GET_LAST_ERROR_NO	14

### 3.5.3.5 Read 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 -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 4C 4 4 R R R R CKL CKH  
Rx -> ACK

---

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

---

### 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 Set IO State Command

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

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

V<sub>CC</sub> – V<sub>CC</sub> level in mV ( V<sub>CC</sub> = VH × 256 + VL)

D1 – Open destination buffer for output and transferred data for each target

0 = none

1 = TDI (target1 to target8)

2 = TDOI (target1 to target8)

3 = TMS (target1 to target8)

4 = RST (target1 to target8)

5 = BSL-RX (target1 to target8)

D2 – data transferred to the buffer above

b0 to b7 – target1 to target8

D3 – output enable bits: 0 = high impedance, 1 = output

b2 (0x04) – common RST – the same state for all eight targets (Note: if the RST buffer above is selected, then this state is ignored)

b3 (0x08) – common TEST – the same state for all eight targets

b4 (0x10) – common TCK – the same state for all eight targets

b5 (0x20) – common TMS – the same state for all eight targets (Note: if the TMS buffer above is selected, then this state is ignored)

D4 – output level on all targets: 0 = LOW, 1 = HIGH

b2 (0x04) – common RST – the same level for all eight targets (Note: if the RST buffer above is selected, then this state is ignored)

b3 (0x08) – common TEST – the same level for all eight targets

b4 (0x10) – common TCK – the same level for all eight targets

b5 (0x20) – common TMS – the same level for all eight targets (Note: if the TMS buffer above is selected, then this state is ignored)

D5 – V<sub>CC</sub> enable bits to each targets

b0 to b7 – target1 to target8

D6 – I<sub>CC</sub> HI enable: 0 = disable, 1 = enable

D7 – spare

D8 – spare

#### Example 1

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

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

then

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



## Example 2

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

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

then

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

## 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 through the MSP-GANG.dll. Because the newer 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 TI recommends using 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 is 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

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 through the 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 through the USB or RS-232 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 MSG-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.

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

#define DBUFFER_SIZE          0x210000
#define JTAG_PASSW_LEN        0x80
#define BSL_PASSW_LEN         0x20    //MSP430
#define ARM_BSL_PASSW_LEN     0x100    //MSP432
#define MAX_BSL_PASSW_LEN     0x100    //max from (ARM_BSL_PASSW_LEN,
BSL_PASSW_LEN)
#define MAX_PASSW_LEN         0x100    //max from (ARM_BSL_PASSW_LEN,
BSL_PASSW_LEN, JTAG_PASSW_LEN )
#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    GangRx[DBUFFER_SIZE][GANG_SIZE];    //data read from all targets
    BYTE    Tmp[DBUFFER_SIZE];                //used for second file cmp
    BYTE    Flag_ScrCode[DBUFFER_SIZE];        //0 - empty    1-Code1, 2-Code2, 4-
Appended Code in SourceCode[x];
        #define CODE1_FLAG          1
        #define CODE2_FLAG          2
        #define APPEND_CODE_FLAG    4
    BYTE    Flag_UsedCode[DBUFFER_SIZE];        //0 - empty    1-
valid data in UsedCode[x];
    BYTE    Flag_WrEn[DBUFFER_SIZE];            //0 - none    1-
write/verify enable in FlashMem[x]
    BYTE    Flag_EraseEn[DBUFFER_SIZE];        //0 - none    1-
erase enable in FlashMem[x]
    BYTE    Flag_RdEn[DBUFFER_SIZE];            //0 - none    1-
read enable in FlashMem[x]
    BYTE    Flag_Sp3[DBUFFER_SIZE];            //used internally
    BYTE    JTAG_Passsword[2][JTAG_PASSW_LEN];
    BYTE    BSL_Passsword[2][MAX_BSL_PASSW_LEN];
    BYTE    Flag_JTAG_Passw[2][JTAG_PASSW_LEN]; // [0][..]-
password from code file; [1][..]-password from password file
    BYTE    Flag_BSL_Passw[2][MAX_BSL_PASSW_LEN]; // [0][..]-
password from code file; [1][..]-password from password file
} DATA_BUFFERS;
```

```
extern    DATA_BUFFERS    dat;
```

### Syntax

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

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

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

### Example

Check if the code contents is specified at the MCU location and get the code contents at that location.

```
int    get_code_content( long MCU_addr, BYTE *data )
{
    long baddr, MCU_addr;
    BYTE data, used_code;

    baddr = MSPGANG_Convert_Address( MCU_TO_DATABUF, MCU_addr );
    if( baddr >= 0 )
    {
        if( DBuf->Flag_ScrCode[ baddr ] )
        {
            *data = DBuf->SourceCode[ baddr ];
            return(SUCCESS);
        }
        else
            return(EMPTY_DATA);
    }
    return(WRONG_MCV_ADDR);
}
```

## 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 when used it for data and 16 bytes when used for serialization.

```
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

```
#define    GANG_DATA_BUF_SIZE        128
#define    SN_GANG_BUF_SIZE          SN_DATA_MAX_SIZE
#define    TARGET_MASK                0x1F
#define    SN_DATA_FLAG              0x40
```

If used for data:

BYTE target	Target number (1 to 8)
BYTE size	Size of data (1 to 128)

If used for serialization:

BYTE target	Target number (1 to 8) ORed with SN_DATA_FLAG (SN_DATA_FLAG   Target_Number)
BYTE size	Size of data (1 to 16)

BYTE *data	Pointer to data buffer from where data is taken or to where the data should be saved
------------	--

## Result

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

## Example

Write unique 16 bytes of data to RAM or Flash

```

BYTE data[16];
MSPGANG_Interactive_Open_Target_Device( "test" );
for(target=1; target<=8; target++)
{
    for(k=0; k<16; k++)
        data[k] = enter_desired_data_per_target...
    MSPGANG_SetGangBuffer( target, 16, data );
}
MSPGANG_Interactive_Copy_GANG_Buffer_to_RAM( MC_addr, 16 ); //copy unique data to
RAM
//or
MSPGANG_Interactive_Copy_GANG_Buffer_to_FLASH( MC_addr, 16 ); //copy unique data to
FLASH

```

### 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

#### Data Format

```
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_prg_min;
    long    Vcc_run_min;
    long    BSL_passw_size;
    long    family_index;
    long    has_JTAG_password;
    long    has_unlockable_JTAG;
    long    has_SBW;
    long    has_4wire_JTAG;
    long    has_BSL;
    long    no_of_DCO_constants;
    long    RAM_start_addr;
    long    MCU_ID;
    long    Clk_Type;
    long    uses_SUC;
    long    MCU_Type;
    long    MCU_Type_index;
    long    ARM_FW_Type;
    long    MAC_RegAddr;
    long    MPU_Type;
    long    spare[11];
} 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 baud rate 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 per second
	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 or 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

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

#### Example

Get data from the info memory for the F2xx from each 8 target devices

```
DATA_BUFFERS *DBuf;
void *temp;
MSPGANG_GetDataBuffers_ptr((&temp));
DBuf = (DATA_BUFFERS *)temp;
long baddr, MCU_addr;
.....
//read data from all targets and save it in the internal DATA_BUFFERS
MSPGANG_Interactive_DefReadTargets( 0xFF, 0, 100, 0x10C0, 0x10FF);
//get the base address of data in the DATA_BUFFERS
baddr = MSPGANG_Convert_Address( MCU_TO_DATABUF, 0x10C0 );
if( baddr >= 0 )
.....
//data at the MCU_addr 0x10C0 to 0x10FF, target -> 1..8
data = DBuf->GangRx[ baddr + (MCU_addr-0x10C0)][target-1];
//get data for each target from internal buffer
.....
```

### 4.2.15 MSPGANG\_Interactive\_ReadTargets

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
------	------------

#### Example

Get data from each of the 8 target devices

```
DATA_BUFFERS *DBuf;
void *temp;
MSPGANG_GetDataBuffers_ptr((&temp));
DBuf = (DATA_BUFFERS *)temp;
long baddr, MCU_addr;
.....
//read data from all targets and save it in the internal DATA_BUFFERS
MSPGANG_Interactive_DefReadTargets( 0xFF, 0, 100, 0x10C0, 0x10FF);
//get the base address of data in the DATA_BUFFERS
baddr = MSPGANG_Convert_Address( MCU_TO_DATABUF, 0x10C0 );
if( baddr >= 0 )
    .....
    //data at the MCU_addr 0x10C0 to 0x10FF), target -> 1..8
    data = DBuf->GangRx[ baddr + (MCU_addr-0x10C0)][target-1];
    //get data for each target from internal buffer
    .....
```

#### 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
------	------------

##### Example

See [Section 4.2.2](#).

#### 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
------	------------

##### Example

See [Section 4.2.2](#).

#### 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
LONG EndAddr	address must be odd.

##### 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 the active image to work with. MSP-GANG supports up to 96 images. Image numbers (image size 64 kB each) are compatible with the old image numbering (0-15) that are 512 kB each. New image numbering style:

b7=1 - new numbering: force the new standard; for example, set b7=1 when subimage number = 0

b6-b4 - subimage number

b3-b0 - image number (0-15 = image 1-16)

Examples:

b7	b6-b4	b3-b0	
1	2	4	(image 5.2) (new numbering - when erased: one sector, 64kB)
0	0	4	(image 5) (old numbering - when erased: eight sectors, 512kB)
0	2	4	(image 5.2) (even without b7=1, new numbering; when erased, one sector, 64kB)

Old image numbering	New image numbering								In GUI			
0	0x80 or 0,	0x90,	0xA0,	0xB0,	0xC0,	0xD0,	0xE0,	0xF0	1.0,	1.1,	1.2,	1.3,...
1	0x81 or 1,	0x91,	0xA1,	0xB1,	0xC1,	0xD1,	0xE1,	0xF1	2.0,	2.1,	2.2,	2.3,...
2	0x82 or 2,	0x92,	0xA2,	0xB2,	0xC2,	0xD2,	0xE2,	0xF2	3.0,	3.1,	3.2,	3.3,...
3	0x83 or 3,	0x93,	0xA3,	0xB3,	0xC3,	0xD3,	0xE3,	0xF3	4.0,	4.1,	4.2,	4.3,...
4	0x84 or 4,	0x94,	0xA4,	0xB4,	0xC4,	0xD4,	0xE4,	0xF4	5.0,	5.1,	5.2,	5.3,...
5	0x85 or 5,	0x95,	0xA5,	0xB5,	0xC5,	0xD5,	0xE5,	0xF5	6.0,	6.1,	6.2,	6.3,...
6	0x86 or 6,	0x96,	0xA6,	0xB6,	0xC6,	0xD6,	0xE6,	0xF6	7.0,	7.1,	7.2,	7.3,...
7	0x87 or 7,	0x97,	0xA7,	0xB7,	0xC7,	0xD7,	0xE7,	0xF7	8.0,	8.1,	8.2,	8.3,...
8	0x88 or 8,	0x98,	0xA8,	0xB8,	0xC8,	0xD8,	0xE8,	0xF8	9.0,	9.1,	9.2,	9.3,...
9	0x89 or 9,	0x99,	0xA9,	0xB9,	0xC9,	0xD9,	0xE9,	0xF9	10.0,	10.1,	10.2,	10.3,...
10	0x8A or 10,	0x9A,	0xAA,	0xBA,	0xCA,	0xDA,	0xEA,	0xFA	11.0,	11.1,	11.2,	11.3,...
11	0x8B or 11,	0x9B,	0xAB,	0xBB,	0xCB,	0xDB,	0xEB,	0xFB	12.0,	12.1,	12.2,	12.3,...
12	- used for read only if the flash image is not used for internal firmware											
13	(transition time)											
14	--,--,---											
15	--,--,---											

### Syntax

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

### Arguments

LONG lImage                      Image number (0 to 0xFB)

### 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 is automatically erased first (MSPGANG\_EraseImage is called automatically, your application code does NOT need to call it explicitly). Use the following sequence for preparing and saving an image into image memory:

```
MSPGANG_CreateGangImage(name);
MSPGANG_SelectImage(1Image);
MSPGANG_EraseImage();
MSPGANG_LoadImageBlock();
MSPGANG_VerifyPSAImageBlock();
```

### Syntax

```
LONG MSPGANG_LoadImageBlock(void)
```

### Arguments

None

### Result

LONG      Error code

---

#### NOTE: Do not overwrite images unnecessarily during production

The image flash memory has a specified 10000 endurance cycles. Therefore, over the lifetime of the product, each image can be reliably reprogrammed 10000 times. Reprogramming images should be done once per production setup, rather than per programming run. Reprogramming the image per programming run will quickly exhaust flash endurance cycles and result in errant behavior.

---

```
//Ideally, load image once per setup. Reduce programming time and save flash endurance cycles.
//Loading an image usually takes longer than full target device programming.
MSPGANG_CreateGangImage(...);
MSPGANG_LoadImageBlock();
...
do
{
    ...
    MSPGANG_MainProcess(...);
    ...
} while(...);

//Avoid loading image inside loop if possible.
//Loading image per programming cycle wastes time and quickly uses up flash endurance cycles.
do
{
    MSPGANG_CreateGangImage(...);
    MSPGANG_LoadImageBlock();
    MSPGANG_MainProcess(...);
} while(...);
```

### 4.2.30 MSPGANG\_VerifyPSAImageBlock

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_VerifyPSAImageBlock(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 512KB) 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

#### Data Format

```
union _IMAGE_HEADER
{
    BYTE    bytes[IMAGE_HEADER_SIZE];
    WORD    words[IMAGE_HEADER_SIZE/2];

    struct
    {
        WORD    own_PSA;
        WORD    global_PSA;
        BYTE    year;
        BYTE    month;
        BYTE    day;
        BYTE    hour;
        BYTE    min;
        BYTE    sec;

#define GLOBAL_PSA_START_OFFSET    10
        // down - covered by global_PSA ----
#define SHORT_ID_2BYTE_OFFSET    10
        WORD    shortID;
#define CHUNKS_NO_2BYTE_OFFSET    12
        WORD    chunks;
#define IMAGE_DATA_2BYTE_OFFSET    14
        WORD    image_data_offset;
#define GLOBAL_SIZE_4BYTE_OFFSET    16
        DWORD    size;
        //global_size;
```



```

        WORD        ID_rev;                //20
        BYTE        ID_name[HEADER_ID_SIZE];                //22
        DWORD       DLL_ver;                //32
#define HEADER_COMMENT_ADDR        36
        char        comment[SCRIPT_TEXT_SIZE];
        WORD        used_tasks_mask;                //52
        BYTE        Interface;                //54    type (JTAG, SBW, BSL), speed(Fast,
Med, Slow)
        BYTE        GangMask;                //55
        BYTE        Vcc_PowerEn;                //56
        BYTE        Icc_HiEn;                //57
        WORD        Vcc_mV;                //58
        WORD        min_Vcc_mV;                //60
        WORD        max_Vcc_mV;                //62
        WORD        RST_time_ms;                //64
        WORD        RST_release_ms;                //66
        BYTE        InfoA_Erase_En;                //68
        BYTE        BSL_Erase_En_mask;                //69
        BYTE        SecureDev_En;                //70
        BYTE        DCO_Flags;                //71
#define DCO_RETAIN_EN        0x01
#define DCO_VALIDATION_EN        0x02
#define DCO_RECAL_EN        0x04
#define DCO_ONE_CONSTANTS        0x08
        BYTE        IO_cfg;                //72
#define SBW_VIA_RST_BIT        0x01
        BYTE        MemoryOption;                //73    for GUI only -
        for displaying used memory option. No impact in firmware
        BYTE        InterfaceSpeed;                //74    for GUI only -
        for displaying used speeds (JTAG/SBW/CJTAG/BSL). No impact in firmware
        BYTE        VccSettleTime;                //75    settle time *20ms

        BYTE        JTAG_unlockEn                : 1;    //76
        BYTE        HasLockedInfoA                : 1;
        BYTE        HasAutoEraseInBSL                : 1;
        BYTE        BSL_X_type                : 1;
        BYTE        spare_flag4                : 1;
        BYTE        spare_flag5                : 1;
        BYTE        spare_flag6                : 1;
        BYTE        spare_flag7                : 1;

        BYTE        ClrSegments;                //77    MSP432
        // #define    MSP432_CLR_LOCKING_INFOA_BIT        0x01
        // #define    MSP432_CLR_LOCKING_BSL_BIT        0x02

        BYTE        BSL_1st_Passw;                //78

        //BYTE        free[112-78];
    }prg;

    struct
    {
        BYTE        offset[IMAGE_HEADER_CTRL_OFFSET];                //offset 112
        BYTE        flags;                //0x70
#define IMAGE_LOCK        0x10                //must be the same bit as in LOCK_LD_PRJ
        BYTE        sp1;                //0x71
        BYTE        sp2;                //0x72
        BYTE        sp3;                //0x73
        BYTE        sp4;                //0x74

```

```

        BYTE      sp5;                      //0x75
        BYTE      sp6;                      //0x76
        BYTE      sp7;                      //0x77
        BYTE      sp8;                      //0x78
        BYTE      sp9;                      //0x79
        BYTE      sp10;                     //0x7A
        BYTE      sp11;                     //0x7B
        BYTE      sp12;                     //0x7C
        BYTE      sp13;                     //0x7D
        BYTE      sp14;                     //0x7E
        BYTE      sp15;                     //0x7F
    }ctrl;

    struct
    {
        BYTE      offset[IMAGE_HEADER_SIZE/2];

        char      MCU_name[SCRIPT_MCU_NAME_SIZE];    //0
        WORD      Id[2];                          //16
        WORD      SubId[2];                        //20
        WORD      MainEraseMode;                   //24
        WORD      minPVcc;                          //26
        WORD      RAM_size;                         //28
        WORD      SubIDAddr;                       //30
        BYTE      FRAM;                            //32
//#define      FRAM_NONE                        0
//#define      FRAM_ASIC                        1
//#define      FRAM_MSPXV2_57                    2
//#define      FRAM_APOLLO                      3
//#define      FRAM_MSPXV2_59                    4

// --- one byte
        BYTE      DefaultDCO                      : 1;    //33
        BYTE      ASIC                            : 1;
        BYTE      MPU                            : 1;
        BYTE      JTAG_Passw                      : 1;
        BYTE      BSLprogrammable                 : 1;
        BYTE      JTAG_Unlockable                 : 1;
        BYTE      BSL_16B_passw                  : 1;
        BYTE      F1_80                          : 1;    //spare
// --- one byte
        BYTE      TestPin;                        //34
        BYTE      CpuX;                          //35

        BYTE      Quick_W                        :1;    //36
        BYTE      Quick_R                        :1;
        BYTE      Quick_W_bug                   :1;
        BYTE      Quick_0x08                    :1;
        BYTE      Quick_0x10                    :1;
        BYTE      Quick_0x20                    :1;
        BYTE      Quick_0x40                    :1;
        BYTE      Quick_0x80                    :1;

        BYTE      FastFlash;                     //37

        BYTE      EnhVerify;                     //38
        BYTE      JTAG;                          //39

        BYTE      SpyBiWire;                     //40
        BYTE      Marginal;                      //41
    }

```

```

        BYTE      F5xx;                //42
        BYTE      MCU_Group;           //43

        WORD      RAM_addr;            //44
        BYTE      SYS_CLK;              //46 used for F5xx and up
//#define STANDARD 0 - for compatibility
#define Xv2_PLL 1 //as standard before
#define HF_8MHz 2 //FRAM FR57xx
#define HF_1MHz 3 //Apollo
#define HF2_8MHz 4 //FRAM FR58xx, FR59xx
#define Xv2_PLL_G60XX 5
#define DCO_16384HZ 6 //i2xxx

        BYTE      InfoA_type;          //47
//#define STANDARD 0 - for compatibility
#define I2XX_1K 1 //i2xxx 1K - 0x1000-0x13FF

        BYTE      FLASH_Type;          //48
//#define STANDARD 0 - for compatibility
#define SEGMENT_1K 1 //i2xxx 1K flash segment size
#define FLASH_SEGM_2K 2
#define FLASH_SEGM_4K 4
#define FLASH_SEGM_8K 8
#define FLASH_SEGM_16K 16
#define FLASH_SEGM_32K 32

        BYTE      Secure_Type;          //49
//#define STANDARD 0 - for compatibility
#define SUC 1 //i2xxx

        BYTE      Map;                  //50
        BYTE      SysClkDiv2;           //51
        BYTE      NMI_to_addr;          //52
#define NMI_TO_ADDR_NOT_SUPPORTED 0x80
        BYTE      FW_type;              //53

        #define FAMILY_XMS432P401 21 //XSP432P401 Rev-B obsolete -
        not supported
        #define FAMILY_MSP432P401 22
        #define FAMILY_MSP432P4111 23
#define FAMILY_MSP432 21
        BYTE      MCU_Type;              //54
#define MSP430F 0x01 //or can be 0
#define CC_RF_BIT_ID 0x10
#define MSP_FR_BIT_ID 0x20
#define MSP432_BIT_ID 0x80
        BYTE      free_1;                //55
        BYTE      free_2;                //56
        BYTE      free_3;                //57
        WORD      Id2[2];                //58
        BYTE      free_4;                //62

//    BYTE      free[128-48];
}device;
};

```

### 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\_SetConfig 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
#define INTERFACE_NONE 0
#define INTERFACE_JTAG 4
#define INTERFACE_SBW 8
#define INTERFACE_BSL 0xC
#define INTERFACE_TYPE_MAX_INDEX INTERFACE_BSL

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

#define CFG_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
#define SBW_VIA_TDOI_BIT 0x00
#define SBW_VIA_RST_BIT 0x01
// 0 - SBW_VIA_TDOI (pin 1) and TCK/TEST (pin-7/8)
// 1 - SBW_VIA_RST (pin 11) and TCK/TEST (pin-7/8)

#define CFG_POWERTARGETEN 6
#define EXTERNAL_POWER_WHOLE_RANGE 0 // external power supply -
```

```

whole range from Vccmin to Vccmax
#define    POWER_SUPPLIED_BY_MSPGANG        1    // targets supplied by
MSP-GANG
#define    EXTERNAL_POWER_IN_RANGE          2    // external power supply -
verified range - selected Vcc +/- 0.3V

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

#define    CFG_ICC_HI_EN                     8
//    disable      0    (up to 30mA from MSP-GANG to each targets)
//    enable       1    (up to 50mA from MSP-GANG to each targets)

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

#define    CFG_TARGET_EN_INDEX              10
//    Targets GANG enable mask - 0x00 ...0xFF. Enable all targets -
> 0xFF
#define    TARGET_1_MASK                    0x01
#define    TARGET_2_MASK                    0x02
#define    TARGET_3_MASK                    0x04
#define    TARGET_4_MASK                    0x08
#define    TARGET_5_MASK                    0x10
#define    TARGET_6_MASK                    0x20
#define    TARGET_7_MASK                    0x40
#define    TARGET_8_MASK                    0x80

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

#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/FRAM start erase address

```

```

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

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

#define          CFG_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
                #define          APPLICATION_NO_RESET          0
                #define          APPLICATION_TOGGLE_RESET      1
                #define          APPLICATION_TOGGLE_VCC        2
                #define          APPLICATION_JTAG_RESET        3
                #define          APPLICATION_RESET_MAX_INDEX    APPLICATION_JTAG_RESET

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

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

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

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

#define          CFG_BSL_ENH_ENABLE          29
                //      disable          0
                //      enable          1

#define          CFG_BSL_ENH_INDEX           30
                //for future usage
                #define          BSL_ENH_DISABLE            0
                #define          BSL_ENH_NONE                1
                #define          BSL_ENH_ERASE                2
                #define          BSL_ENH_MAX_INDEX            2

```



```

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

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

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

#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
                //      mask for 4 BSL segments - disable->0, enable->1
                //      bit 0 -> 0x01      BSL segment 1
                //      bit 1 -> 0x02      BSL segment 2
                //      bit 2 -> 0x04      BSL segment 3
                //      bit 3 -> 0x08      BSL segment 4

#define          CFG_BSL_FLASH_RD_EN                38
                //      mask for 4 BSL segments - disable->0, enable->1
                //      bit 0 -> 0x01      BSL segment 1
                //      bit 1 -> 0x02      BSL segment 2
                //      bit 2 -> 0x04      BSL segment 3
                //      bit 3 -> 0x08      BSL segment 4

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

#define          CFG_READDEFSTARTADDR               40
                // Memory READ start address

#define          CFG_READDEFSTOPADDR                41
                // Memory READ end address

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

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

#define          CFG_OPEN_FILE_TYPE                 44
                #define          CODE_FILE_INDEX          0
                #define          APPEND_FILE_INDEX        1
                #define          PASSW_FILE_INDEX         2

```

```

#define          SECONDCODE_FILE_INDEX      3
#define          CODE2_FILE_INDEX          4

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

#define          CFG_IMAGE_NO                46
//image number - 0...9

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

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

#define          CFG_COPY_CFG_FROM_MEMORY_EN 49
//      Direct (eg. Interactive)            0
//      From Image memory                   1

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

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

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

#define          CFG_CODE2_FILE_EN           53
//      disable          0
//      enable          1

#define          CFG_BSL_FIRST_PASSWORD      54
#define          BSL_ANY_PASSW              0
#define          BSL_PASSW_FROM_CODE_FILE    1
#define          BSL_PASSW_FROM_PASSWORD_FILE 2
#define          BSL_EMPTY_PASSW            3

#define          CFG_DEFINED_RETAIN_DATA_EN  55
//      disable          0
//      enable          1

#define          CFG_DEFINED_RETAIN_START_ADDR 56
//address must be even
#define          CFG_DEFINED_RETAIN_END_ADDR  57
//address must be odd
#define          DEFINED_RETAIN_DATA_MAX_SIZE 0x40
// END_ADDR - START_ADDR + 1 <= DEFINED_RETAIN_DATA_MAX_SIZE

```

```

#define          CFG_SECURE_CODE_SOURCE          58
enum {USER_SOURCE=0, FILE_SOURCE=1};

#define          CFG_MSP432_CLR_LOCKING_OPTIONS  59
#define          MSP432_CLR_LOCKING_INFOA_BIT    0x01
#define          MSP432_CLR_LOCKING_BSL_BIT      0x02
#define          MSP432_CLR_LOCKING_MASK        ( MSP432_CLR_LOCKING_INFOA_BIT |
MSP432_CLR_LOCKING_BSL_BIT )

#define          CFG_WRDEF_EXCLUDE_SECTIONS      60
//      mask for 4 Read_Defined excluded sections disable->0, enable->1
//      bit 0 -> 0x01      section 1
//      bit 1 -> 0x02      section 2
//      bit 2 -> 0x04      section 3
//      bit 3 -> 0x08      section 4

#define          CFG_RDDEF_EXCLUDE_SECTIONS      61
//      mask for 4 Read_Defined excluded sections disable->0, enable->1
//      bit 0 -> 0x01      section 1
//      bit 1 -> 0x02      section 2
//      bit 2 -> 0x04      section 3
//      bit 3 -> 0x08      section 4

#define          CFG_WRDEF_EXCLUDE_S1_START_ADDR 62
#define          CFG_WRDEF_EXCLUDE_S1_END_ADDR   63
#define          CFG_WRDEF_EXCLUDE_S2_START_ADDR 64
#define          CFG_WRDEF_EXCLUDE_S2_END_ADDR   65
#define          CFG_WRDEF_EXCLUDE_S3_START_ADDR 66
#define          CFG_WRDEF_EXCLUDE_S3_END_ADDR   67
#define          CFG_WRDEF_EXCLUDE_S4_START_ADDR 68
#define          CFG_WRDEF_EXCLUDE_S4_END_ADDR   69

#define          CFG_RDDEF_EXCLUDE_S1_START_ADDR 70
#define          CFG_RDDEF_EXCLUDE_S1_END_ADDR   71
#define          CFG_RDDEF_EXCLUDE_S2_START_ADDR 72
#define          CFG_RDDEF_EXCLUDE_S2_END_ADDR   73
#define          CFG_RDDEF_EXCLUDE_S3_START_ADDR 74
#define          CFG_RDDEF_EXCLUDE_S3_END_ADDR   75
#define          CFG_RDDEF_EXCLUDE_S4_START_ADDR 76
#define          CFG_RDDEF_EXCLUDE_S4_END_ADDR   77

#define          CFG_RD_TLV_EN                   78

#define          CFG_SERIALIZATION_EN            80
//      disable      0
//      enable       1

#define          CFG_SN_ADDRESS_IN_MEMORY        81
//address must be even

#define          CFG_SN_DATA_SIZE_IN_BYTES       82
//Size must be even 2...16
#define          SN_DATA_MAX_SIZE                16

#define          CFG_SN_REMOVE_CODE_FROM_SN_LOCATION 83
//      disable      0
//      enable       1

#define          CFG_SN_SOURCE                   84
#define          SN_SOURCE_DEFINED               0
#define          SN_SOURCE_FROM_FILE             1
// 0 - defined
// 1 - from file

```

```

#define          CFG_SN_FORMAT_IN_MEMORY          85
                #define          SN_FORMAT_LSB_FIRST          0
                #define          SN_FORMAT_MSB_FIRST          1

#define          CFG_SN_DATA_INCREMENT            86
#define          CFG_INIT_SN_DATA_0              87
                //Bits 0-31 of the SN init data
#define          CFG_INIT_SN_DATA_1              (CFG_INIT_SN_DATA_0+1)
                //Bits 32-63 of the SN init data
#define          CFG_INIT_SN_DATA_2              (CFG_INIT_SN_DATA_0+2)
                //Bits 64-91 of the SN init data
#define          CFG_INIT_SN_DATA_3              (CFG_INIT_SN_DATA_0+3)
                //Bits 92-127 of the SN init data
#define          CFG_SN_DESTINATION              91
                #define          NUMBER_TO_FLASH              0
                #define          NUMBER_TO_MAC_REG            1

#define          CFG_MPU_IPE_WR_LOCKED            92
                //    disable          0
                //    enable          1
#define          CFG_MPU_IPE_WR_UNLOCKED          93
                //    disable          0
                //    enable          1
#define          CFG_MPU_IPE_RD_UNLOCKED          94
                //    disable          0
                //    enable          1
#define          CFG_MPU_IPE_START_ADDR           95
#define          CFG_MPU_IPE_END_ADDR            96

#define          CFG_ADDITIONAL_COMPORT_NO        200
                // 0 - disable (default)
                // 1-255 added COM port number 1-255

// MSP432P protection configuration
#define          CFG_MSP432_MB_CMD                300
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_SECEN          301
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_INIT_VECT0      302
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_INIT_VECT1      303
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_INIT_VECT2      304
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_INIT_VECT3      305
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_SECKEYS0        306
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_SECKEYS1        307
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_SECKEYS2        308
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_SECKEYS3        309
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_SECKEYS4        310
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_SECKEYS5        311
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_SECKEYS6        312
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_AES_SECKEYS7        313
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_UNENC_PWD0          314
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_UNENC_PWD1          315
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_UNENC_PWD2          316
#define          CFG_MSP432_MB_JTAG_SWD_LOCK_UNENC_PWD3          317
#define          CFG_MSP432_MB_SEC_ZONE0_SECEN          318
#define          CFG_MSP432_MB_SEC_ZONE0_START_ADDR          319
#define          CFG_MSP432_MB_SEC_ZONE0_LENGTH          320
#define          CFG_MSP432_MB_SEC_ZONE0_AESINIT_VECT0          321
#define          CFG_MSP432_MB_SEC_ZONE0_AESINIT_VECT1          322
#define          CFG_MSP432_MB_SEC_ZONE0_AESINIT_VECT2          323

```

#define	CFG_MSP432_MB_SEC_ZONE0_AESINIT_VECT3	324
#define	CFG_MSP432_MB_SEC_ZONE0_SECKEYS0	325
#define	CFG_MSP432_MB_SEC_ZONE0_SECKEYS1	326
#define	CFG_MSP432_MB_SEC_ZONE0_SECKEYS2	327
#define	CFG_MSP432_MB_SEC_ZONE0_SECKEYS3	328
#define	CFG_MSP432_MB_SEC_ZONE0_SECKEYS4	329
#define	CFG_MSP432_MB_SEC_ZONE0_SECKEYS5	330
#define	CFG_MSP432_MB_SEC_ZONE0_SECKEYS6	331
#define	CFG_MSP432_MB_SEC_ZONE0_SECKEYS7	332
#define	CFG_MSP432_MB_SEC_ZONE0_UNENC_PWD0	333
#define	CFG_MSP432_MB_SEC_ZONE0_UNENC_PWD1	334
#define	CFG_MSP432_MB_SEC_ZONE0_UNENC_PWD2	335
#define	CFG_MSP432_MB_SEC_ZONE0_UNENC_PWD3	336
#define	CFG_MSP432_MB_SEC_ZONE0_ENCUPDATE_EN	337
#define	CFG_MSP432_MB_SEC_ZONE0_DATA_EN	338
#define	CFG_MSP432_MB_SEC_ZONE1_SECEN	339
#define	CFG_MSP432_MB_SEC_ZONE1_START_ADDR	340
#define	CFG_MSP432_MB_SEC_ZONE1_LENGTH	341
#define	CFG_MSP432_MB_SEC_ZONE1_AESINIT_VECT0	342
#define	CFG_MSP432_MB_SEC_ZONE1_AESINIT_VECT1	343
#define	CFG_MSP432_MB_SEC_ZONE1_AESINIT_VECT2	344
#define	CFG_MSP432_MB_SEC_ZONE1_AESINIT_VECT3	345
#define	CFG_MSP432_MB_SEC_ZONE1_SECKEYS0	346
#define	CFG_MSP432_MB_SEC_ZONE1_SECKEYS1	347
#define	CFG_MSP432_MB_SEC_ZONE1_SECKEYS2	348
#define	CFG_MSP432_MB_SEC_ZONE1_SECKEYS3	349
#define	CFG_MSP432_MB_SEC_ZONE1_SECKEYS4	350
#define	CFG_MSP432_MB_SEC_ZONE1_SECKEYS5	351
#define	CFG_MSP432_MB_SEC_ZONE1_SECKEYS6	352
#define	CFG_MSP432_MB_SEC_ZONE1_SECKEYS7	353
#define	CFG_MSP432_MB_SEC_ZONE1_UNENC_PWD0	354
#define	CFG_MSP432_MB_SEC_ZONE1_UNENC_PWD1	355
#define	CFG_MSP432_MB_SEC_ZONE1_UNENC_PWD2	356
#define	CFG_MSP432_MB_SEC_ZONE1_UNENC_PWD3	357
#define	CFG_MSP432_MB_SEC_ZONE1_ENCUPDATE_EN	358
#define	CFG_MSP432_MB_SEC_ZONE1_DATA_EN	359
#define	CFG_MSP432_MB_SEC_ZONE2_SECEN	360
#define	CFG_MSP432_MB_SEC_ZONE2_START_ADDR	361
#define	CFG_MSP432_MB_SEC_ZONE2_LENGTH	362
#define	CFG_MSP432_MB_SEC_ZONE2_AESINIT_VECT0	363
#define	CFG_MSP432_MB_SEC_ZONE2_AESINIT_VECT1	364
#define	CFG_MSP432_MB_SEC_ZONE2_AESINIT_VECT2	365
#define	CFG_MSP432_MB_SEC_ZONE2_AESINIT_VECT3	366
#define	CFG_MSP432_MB_SEC_ZONE2_SECKEYS0	367
#define	CFG_MSP432_MB_SEC_ZONE2_SECKEYS1	368
#define	CFG_MSP432_MB_SEC_ZONE2_SECKEYS2	369
#define	CFG_MSP432_MB_SEC_ZONE2_SECKEYS3	370
#define	CFG_MSP432_MB_SEC_ZONE2_SECKEYS4	371
#define	CFG_MSP432_MB_SEC_ZONE2_SECKEYS5	372
#define	CFG_MSP432_MB_SEC_ZONE2_SECKEYS6	373
#define	CFG_MSP432_MB_SEC_ZONE2_SECKEYS7	374
#define	CFG_MSP432_MB_SEC_ZONE2_UNENC_PWD0	375
#define	CFG_MSP432_MB_SEC_ZONE2_UNENC_PWD1	376
#define	CFG_MSP432_MB_SEC_ZONE2_UNENC_PWD2	377
#define	CFG_MSP432_MB_SEC_ZONE2_UNENC_PWD3	378
#define	CFG_MSP432_MB_SEC_ZONE2_ENCUPDATE_EN	379
#define	CFG_MSP432_MB_SEC_ZONE2_DATA_EN	380
#define	CFG_MSP432_MB_SEC_ZONE3_SECEN	381
#define	CFG_MSP432_MB_SEC_ZONE3_START_ADDR	382
#define	CFG_MSP432_MB_SEC_ZONE3_LENGTH	383

```
#define CFG_MSP432_MB_SEC_ZONE3_AESINIT_VECT0 384
#define CFG_MSP432_MB_SEC_ZONE3_AESINIT_VECT1 385
#define CFG_MSP432_MB_SEC_ZONE3_AESINIT_VECT2 386
#define CFG_MSP432_MB_SEC_ZONE3_AESINIT_VECT3 387
#define CFG_MSP432_MB_SEC_ZONE3_SECKEYS0 388
#define CFG_MSP432_MB_SEC_ZONE3_SECKEYS1 389
#define CFG_MSP432_MB_SEC_ZONE3_SECKEYS2 390
#define CFG_MSP432_MB_SEC_ZONE3_SECKEYS3 391
#define CFG_MSP432_MB_SEC_ZONE3_SECKEYS4 392
#define CFG_MSP432_MB_SEC_ZONE3_SECKEYS5 393
#define CFG_MSP432_MB_SEC_ZONE3_SECKEYS6 394
#define CFG_MSP432_MB_SEC_ZONE3_SECKEYS7 395
#define CFG_MSP432_MB_SEC_ZONE3_UNENC_PWD0 396
#define CFG_MSP432_MB_SEC_ZONE3_UNENC_PWD1 397
#define CFG_MSP432_MB_SEC_ZONE3_UNENC_PWD2 398
#define CFG_MSP432_MB_SEC_ZONE3_UNENC_PWD3 399
#define CFG_MSP432_MB_SEC_ZONE3_ENCUPDATE_EN 400
#define CFG_MSP432_MB_SEC_ZONE3_DATA_EN 401
#define CFG_MSP432_MB_BSL_EABLE 402
#define CFG_MSP432_MB_BSL_START_ADDR 403
#define CFG_MSP432_MB_BSL_HARD_INV_PARAMS 404
#define CFG_MSP432_MB_JTAG_SWD_LOCK_ENCPAYLOADADDR 405
#define CFG_MSP432_MB_JTAG_SWD_LOCK_ENCPAYLOADLEN 406
#define CFG_MSP432_MB_JTAG_SWD_LOCK_DST_ADDR 407
#define CFG_MSP432_MB_SEC_ZONE0_PAYLOADADDR 408
#define CFG_MSP432_MB_SEC_ZONE0_PAYLOADLEN 409
#define CFG_MSP432_MB_SEC_ZONE1_PAYLOADADDR 410
#define CFG_MSP432_MB_SEC_ZONE1_PAYLOADLEN 411
#define CFG_MSP432_MB_SEC_ZONE2_PAYLOADADDR 412
#define CFG_MSP432_MB_SEC_ZONE2_PAYLOADLEN 413
#define CFG_MSP432_MB_SEC_ZONE3_PAYLOADADDR 414
#define CFG_MSP432_MB_SEC_ZONE3_PAYLOADLEN 415
#define CFG_MSP432_MB_FACTORY_RESET_ENABLE 416
#define CFG_MSP432_MB_FACTORY_RESET_PWDEN 417
#define CFG_MSP432_MB_FACTORY_RESET_PWD0 418
#define CFG_MSP432_MB_FACTORY_RESET_PWD1 419
#define CFG_MSP432_MB_FACTORY_RESET_PWD2 420
#define CFG_MSP432_MB_FACTORY_RESET_PWD3 421
#define CFG_MSP432_MB_FACTORY_RESET_PASSWORD0 422
#define CFG_MSP432_MB_FACTORY_RESET_PASSWORD1 423
#define CFG_MSP432_MB_FACTORY_RESET_PASSWORD2 424
#define CFG_MSP432_MB_FACTORY_RESET_PASSWORD3 425
```

```
// MSP432E protection configuration
```

```
#define CFG_MSP432E_FMPREADEN0_DATA 430
#define CFG_MSP432E_FMPREADEN1_DATA 431
#define CFG_MSP432E_FMPREADEN2_DATA 432
#define CFG_MSP432E_FMPREADEN3_DATA 433
#define CFG_MSP432E_FMPREADEN4_DATA 434
#define CFG_MSP432E_FMPREADEN5_DATA 435
#define CFG_MSP432E_FMPREADEN6_DATA 436
#define CFG_MSP432E_FMPREADEN7_DATA 437
#define CFG_MSP432E_FMPREADEN8_DATA 438
#define CFG_MSP432E_FMPREADEN9_DATA 439
#define CFG_MSP432E_FMPREADEN10_DATA 440
#define CFG_MSP432E_FMPREADEN11_DATA 441
#define CFG_MSP432E_FMPREADEN12_DATA 442
#define CFG_MSP432E_FMPREADEN13_DATA 443
#define CFG_MSP432E_FMPREADEN14_DATA 444
```

#define	CFG_MSP432E_FMPREADEN15_DATA	445
#define	CFG_MSP432E_FMPPRGEN0_DATA	446
#define	CFG_MSP432E_FMPPRGEN1_DATA	447
#define	CFG_MSP432E_FMPPRGEN2_DATA	448
#define	CFG_MSP432E_FMPPRGEN3_DATA	449
#define	CFG_MSP432E_FMPPRGEN4_DATA	450
#define	CFG_MSP432E_FMPPRGEN5_DATA	451
#define	CFG_MSP432E_FMPPRGEN6_DATA	452
#define	CFG_MSP432E_FMPPRGEN7_DATA	453
#define	CFG_MSP432E_FMPPRGEN8_DATA	454
#define	CFG_MSP432E_FMPPRGEN9_DATA	455
#define	CFG_MSP432E_FMPPRGEN10_DATA	456
#define	CFG_MSP432E_FMPPRGEN11_DATA	457
#define	CFG_MSP432E_FMPPRGEN12_DATA	458
#define	CFG_MSP432E_FMPPRGEN13_DATA	459
#define	CFG_MSP432E_FMPPRGEN14_DATA	460
#define	CFG_MSP432E_FMPPRGEN15_DATA	461
#define	CFG_MSP432E_USER_REG0	462
#define	CFG_MSP432E_USER_REG1	463
#define	CFG_MSP432E_USER_REG2	464
#define	CFG_MSP432E_USER_REG3	465
#define	CFG_MSP432E_USERDEBUGDATA	466
#define	CFG_MSP432E_FP_REG_WREN	467
#define	CFG_MSP432E_USER_REG_WREN	468
#define	CFG_MSP432E_DBG_REG_WREN	469
#define	CFG_MSP432E_PROTECTION_SOURCE	470

#### 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
#define CODE_2_FILE_INDEX 6
#define IMAGE_FILE_INDEX 7
```

### 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 and data -----
#define   CFG_TMP_CLEAR           2
        //data - none

#define   CFG_TMP_TASK_MASK       4
        //--- task mask bits ---- - for all tasks - set 0xFFFF
        #define   CONNECT_TASK_BIT      0x0001
        #define   ERASE_TASK_BIT        0x0002
        #define   BLANKCHECK_TASK_BIT   0x0004
        #define   PROGRAM_TASK_BIT      0x0008
        #define   VERIFY_TASK_BIT       0x0010
        #define   SECURE_TASK_BIT       0x0020
        #define   DCO_CAL_TASK_BIT      0x0040
        //spare                      0x0080 to 0x4000
        #define   RST_AND_START_FW_BIT  0x8000

#define   CFG_TMP_VCC_VALUE       6
        // Vcc in mV - 1800 ...3600

#define   CFG_TMP_POWER_VCC_EN    8
        //   disable           0
        //   enable            1

#define   CFG_TMP_INTERFACE       10
        // (INTERFACE_JTAG | INTERFACE_FAST)
        // (INTERFACE_JTAG | INTERFACE_MED)
        // (INTERFACE_JTAG | INTERFACE_SLOW)
        // (INTERFACE_SBW | INTERFACE_FAST)
        // (INTERFACE_SBW | INTERFACE_MED)
        // (INTERFACE_SBW | INTERFACE_SLOW)

#define   CFG_TMP_GANG_MASK       12
        //   Targets GANG enable mask - 0x00 ...0xFF. Enable all targets -> 0xFF
        //   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

#define   CFG_TMP_VCC_ONOFF       14
        //   disable           0
```

```

//      enable      1

#define   CFG_LCD_CONTRAST      16
//      0x00 --0x3F

#define   CFG_TMP_ICC_HI_EN     18
//      disable      0
//      enable       1

#define   CFG_TMP_IO_INTERFACE  20
//      0 - SBW_VIA_TDOI (pin 1) and TCK/TEST (pin-7/8)
//      1 - SBW_VIA_RST (pin 11) and TCK/TEST (pin-7/8)

#define   CFG_TMP_RESET         22
//      disable      0
//      enable       1

#define   CFG_TMP_KEYBOARD_EN   24
//      disable      0
//      enable       1

#define   CFG_TMP_VCC_SETTLE_TIME 26
//      0...200 Vcc settle time in 20 ms increment (t = 0...4000 ms)

#define   CFG_TMP_CUMULATIVE_ST_EN 28
//      0 - disable
//      1 - enable ( default )

#define   CFG_WRAPPER_MASK      30
//      used in the DLL wrapper of the MSP-GANG430 only. Do not use it.

#define   CFG_WRAPPER_EN_KEY    32
//      used in the DLL wrapper of the MSP-GANG430 only. Do not use it.

#define   CFG_WRAPPER_GANG_MASK 34
//      used in the DLL wrapper of the MSP-GANG430 only. Do not use it.

#define   CFG_TMP_BSL_1ST_PASSW 36
//      #define BSL_ANY_PASSW 0
//      #define BSL_PASSW_FROM_CODE_FILE 1
//      #define BSL_PASSW_FROM_PASSWORD_FILE 2
//      #define BSL_EMPTY_PASSW 3

#define   CFG_DISABLE_TASK_MASK 38

```

#### 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 or from data buffer

##### Result

LONG            Error code

#### 4.2.39 MSPGANG\_Get\_qty\_MCU\_Type, MSPGANG\_Set\_MCU\_Type, MSPGANG\_Get\_MCU\_TypeName, MSPGANG\_Get\_qty\_MCU\_Family, MSPGANG\_Get\_MCU\_FamilyName, MSPGANG\_Get\_MCU\_Name

Set of functions that return the names of all supported MCUs, including the family, group, and MCU name.

##### Syntax

```
LONG WINAPI MSPGANG_Get_qty_MCU_Type( void );
LONG WINAPI MSPGANG_Set_MCU_Type( int type );
LONG WINAPI MSPGANG_Get_MCU_TypeName( LONG index, LPTSTR name );
LONG WINAPI MSPGANG_Get_qty_MCU_Family( void );
LONG WINAPI MSPGANG_Get_MCU_FamilyName( LONG index, LPTSTR name );
LONG WINAPI 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;
}
```

The following names and numbers are received using the previous functions:

```
MCU_Family_list
{ 1, " MSP430" },
```

```
{ 0x20, " MSP430FR" },
{ 0x10, " CC430" },
{ 0x80, " MSP432" },

MCU_Group_list
{ 1, 1, " MSP430F1xx" },
{ 20, 1, " MSP430F2xx" },
{ 22, 1, " MSP430AFE2xx" },
{ 21, 1, " MSP430G2xx" },
{ 23, 1, " MSP430i2xx" },
{ 25, 1, " MSP430Txxx" },
{ 40, 1, " MSP430F4xx" },
{ 41, 1, " MSP430FE4xx" },
{ 42, 1, " MSP430FG4xx" },
{ 43, 1, " MSP430FW4xx" },
{ 50, 1, " MSP430F5xx" },
{ 60, 1, " MSP430F6xx" },
{ 62, 1, " MSP430FG6xx" },
{ 55, 0x20, " MSP430FR2xx" },
{ 56, 0x20, " MSP430FR4xx" },
{ 57, 0x20, " MSP430FR5xx" },
{ 67, 0x20, " MSP430FR6xx" },
{ 51, 0x10, " CC-430F5xx" },
{ 61, 0x10, " CC-430F6xx" },
{ 0x84,0x80, " MSP432P4xx" },
{ 0x85,0x80, " MSP432E4xx" },
```

List of the MCU names in a selected group can be found as follows (in this example, a 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 1 → Target 2, ... bit 7 → Target 8

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

### Syntax

```
LONG 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

#### 4.2.44 MSPGANG\_Set\_IO\_State

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

##### Syntax

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

##### Arguments

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

##### Example 1

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

```
BYTE data[8] = { 04 60 00 00 7F 00 00 00 };  
MSPGANG_Set_IO_State( 3300, data );
```

then

```
data[1] = 0x61;  
MSPGANG_Set_IO_State( 3300, data );
```

**Example 2**

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

```
BYTE data[8] = { 00 00 04 00 7F 00 00 00 };  
MSPGANG_Set_IO_State( 3300, data );
```

then

```
data[4] = 0x04;  
MSPGANG_Set_IO_State( 3300, data );
```

#### 4.2.45 MSPGANG\_Convert\_Address

The MSPGANG\_Convert\_Address function converts the MCU data address to the data buffer address in the DLL (see DATA\_BUFFERS structure), where the data (flash, FRAM, RAM) are stored. Function is used in the MSP432 MCU where the MCU address is 32 bit. Currently the function is not used if the MSP430 MCU is used, but it can be used in the future if the memory space increases in the MCU.

##### Syntax

```
LONG WINAPI MSPGANG_Convert_Address( BYTE type, LONG Addr );
```

##### Arguments

BYTE type	MCU_TO_DATABUF 1 DATABUF_TO_MCU 2
LONG Addr	MCU address for type MCU_TO_DATABUF, or data buffer address for type DATABUF_TO_MCU.

##### Result

LONG	Data Buffer or MCU address. Result is positive if conversion of the address succeeds, and minus 1 (0xFFFFFFFF) if failed.
------	---

##### Example

See [Section 4.2.1](#), [Section 4.2.2](#), [Section 4.2.14](#), and [Section 4.2.15](#).

#### 4.2.46 MSPGANG\_Memory\_Header\_text

The MSPGANG\_Memory\_Header\_text displays the name of the data block specified at the MCU address. Function is used by GUI in the data viewer.

##### Syntax

```
LPTSTR WINAPI MSPGANG_Memory_Header_text( LONG Addr );
```

##### Arguments

LONG Addr	Beginning address in MCU of the selected data block
-----------	---

##### Result

LPTSTR	Text of data block name
--------	-------------------------

##### Example

```
char *header_txt;
// Selected MCU - MSP432P401R
header_txt = (char*)MSPGANG_Memory_Header_text( 0x8000 );
//Returned contents of the header_txt string ->
//      "Flash - Sectors      8 - 15  (0x08000 to 0x0FFFF)"
```

#### 4.2.47 MSPGANG\_Interactive\_ClrLockedDevice

Unlocks the MSP432 MCU, if it is locked. The whole main memory flash is erased. The information memory or BSL sectors can be erased if selected in configuration.

##### Syntax

```
LONG WINAPI MSPGANG_Interactive_ClrLockedDevice( void );
```

#### 4.2.48 MSPGANG\_Get\_Code\_Info

Gets the checksum or code size of the selected code.

##### Syntax

```
LONG WINAPI MSPGANG_Get_Code_Info( LONG type );
```

##### Arguments

LONG type	CODE_SIZE_INFO 1
	CODE_CHECK_SUM 2
	CODE2_SIZE_INFO 3
	CODE2_CHECK_SUM 4
	APPEND_CODE_SIZE_INFO 5
	APPEND_CODE_CHECK_SUM 6
	WHOLE_CODE_SIZE_INFO 7
	WHOLE_CODE_CHECK_SUM 8
	CS_PER_GANG430STD 9

##### Result

LONG	Checksum or code size of the selected code.
------	---

#### 4.2.49 MSPGANG\_MakeSound

The MSPGANG\_MakeSound make beep or sounds.

##### Syntax

```
void WINAPI MSPGANG_MakeSound( LONG type );
```

##### Arguments

LONG type	BEEP_OK 1
	BEEP_ERR 2
	BEEP_PRG_ERR 3
	BEEP_WARNING 4

### 4.2.50 MSPGANG\_CallBack\_ProgressBar

The MSPGANG\_CallBack\_ProgressBar function returns the current status during process execution. The function should be called from an interrupt or separate thread if the main function is executed.

#### Syntax

```
LONG WINAPI MSPGANG_CallBack_ProgressBar( void ** text, void ** history, BYTE
*G_status, BYTE *DLL_status );
```

#### Result

LONG If the result is negative, then the contents of the MSPGANG\_CallBack\_ProgressBar have not been updated.  
If the result is positive, then data has been updated.

#### Example

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

        BYTE cumulative; //byte offset - 8
        //target masks
        // TARGET_1_MASK 0x01
        // TARGET_2_MASK 0x02
        // TARGET_3_MASK 0x04
        // TARGET_4_MASK 0x08
        // TARGET_5_MASK 0x10
        // TARGET_6_MASK 0x20
        // TARGET_7_MASK 0x40
        // TARGET_8_MASK 0x80
        BYTE Rq_gang_mask; //byte offset - 9
        BYTE Connected_gang_mask; //byte offset - 10
        BYTE Erased_gang_mask; //byte offset - 11
        BYTE BlankCheck_gang_mask; //byte offset - 12
        BYTE Programmed_gang_mask; //byte offset - 13
        BYTE Verified_gang_mask; //byte offset - 14
        BYTE Secured_gang_mask; //byte offset - 15
        BYTE spare[6]; //byte offset - 16..21
        BYTE error_no; //byte offset - 22
        BYTE VTIO_32mV; //byte offset - 23
        BYTE VccSt_LOW; //byte offset - 24
        BYTE VccSt_HI; //byte offset - 25
        // VccSt_LOW, VccSt_HI provide 2 bits to each target.
        // Bit A for each target and bit B for each target.
        // Bits B A
```

```

//          0   0      Vcc below 0.7V
//          0   1      Vcc below Vcc min   ( 0.7 V < Vcc < Vcc min)
//          1   0      Vcc over Vcc min (OK status)
//          1   1      Vcc over 3.8V
BYTE        VccErr;                                //byte offset - 26
//current Vcc below min
BYTE        VccErr_Cumulative;                      //byte offset - 27
//Cumulative (during programminf) Vcc below min
BYTE        JTAG_init_err_mask;                     //byte offset - 28
BYTE        JTAG_Fuse_already_blowed_mask;          //byte offset - 29
BYTE        Wrong_MCU_ID_mask;                      //byte offset - 30
BYTE        Progress_bar;                           //byte offset - 31
// 0...100%
char        comment[SCRIPT_TEXT_SIZE];              //byte offset - 32..47
}st;
};

union DLL_STATUS
{
    BYTE bytes[DLL_STATUS_SIZE+4];
    struct
    {
        BYTE new_data;
        BYTE COM_status;
        //reserved for future
    }st;
};

void *text, *history;
GANG_PROGRESS_STATUS Gang_Status;
DLL_STATUS DLL_Status;
int pos;

pos = MSPGANG_Callback_ProgressBar( &text, &history, Gang_Status.bytes+2, DLL_Status.bytes );
if( pos >= 0 )
{
    ProgressBar->SetPos( pos );
    ...
}

```

#### 4.2.51 MSPGANG\_GetPCHardwareFingerprint

Reads the hardware fingerprint from the current PC. The function is used for projects that are protected with a password or a hardware fingerprint number.

##### Syntax

```
DWORD WINAPI MSPGANG_GetPCHardwareFingerprint( void );
```

##### Result

DWORD      Eight digit hardware fingerprint number taken from current PC

#### 4.2.52 MSPGANG\_Flash\_valid\_addr

Determines if the selected address space can be used for flash, FRAM, or OTP programming.

##### Syntax

```
LONG WINAPI MSPGANG_Flash_valid_addr(LONG dest, LONG start_addr, LONG size );
```

##### Arguments

LONG dest	Spare – not used
LONG start_addr	Memory region start address
LONG size	Number of bytes of memory region to be validated

##### Result

LONG      ERR\_NONE (0) if address is programmable  
Error code otherwise



## Schematics

### 5.1 Schematics

MSP-GANG-simplified.sch-1 - Sun Jan 15 14:25:06 2012

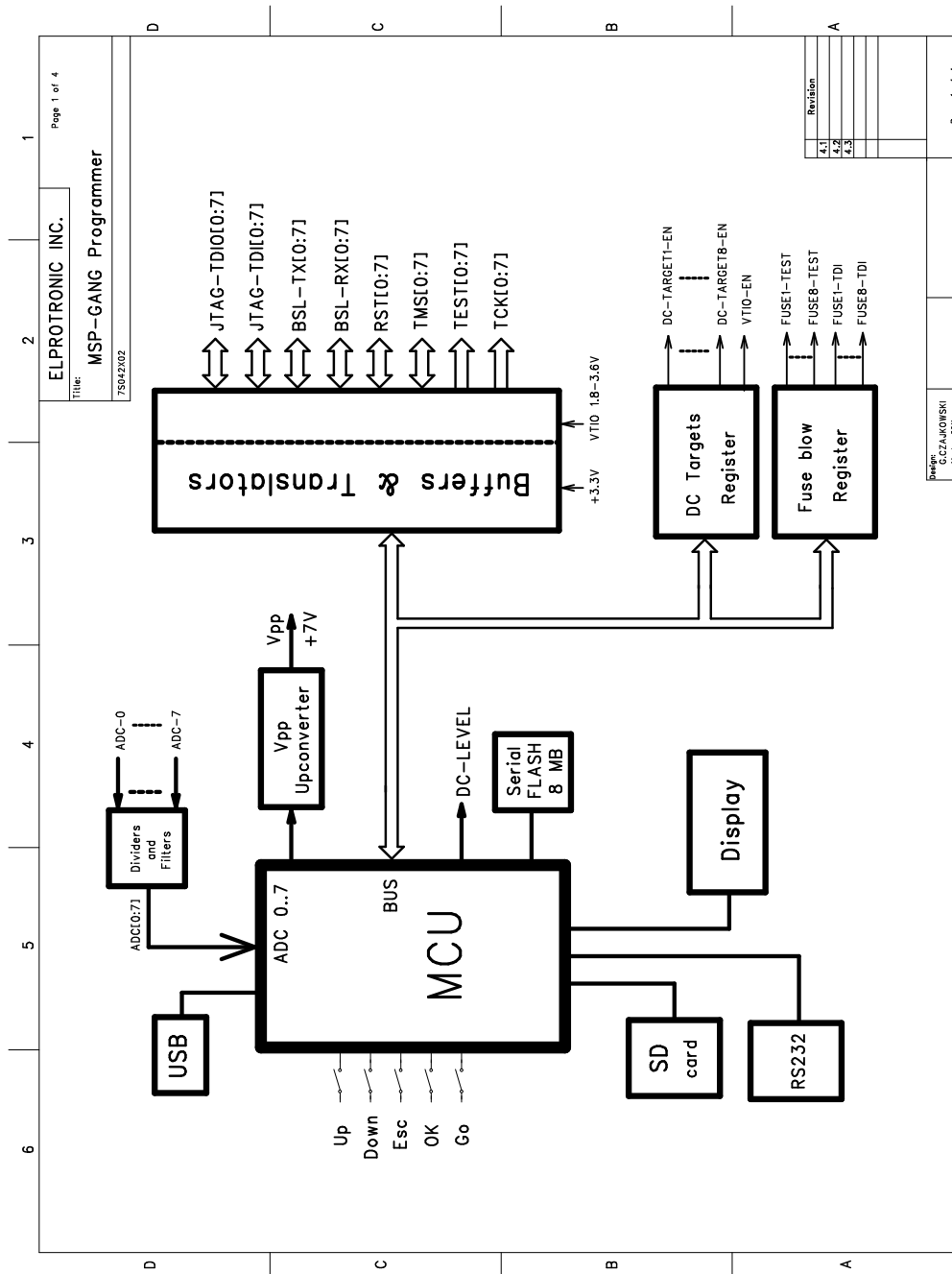


Figure 5-1. MSP-GANG Simplified Schematic (1 of 4)

MSP-GANG-simplified.sch-2 - Sun Jan 15 14:25:06 2012

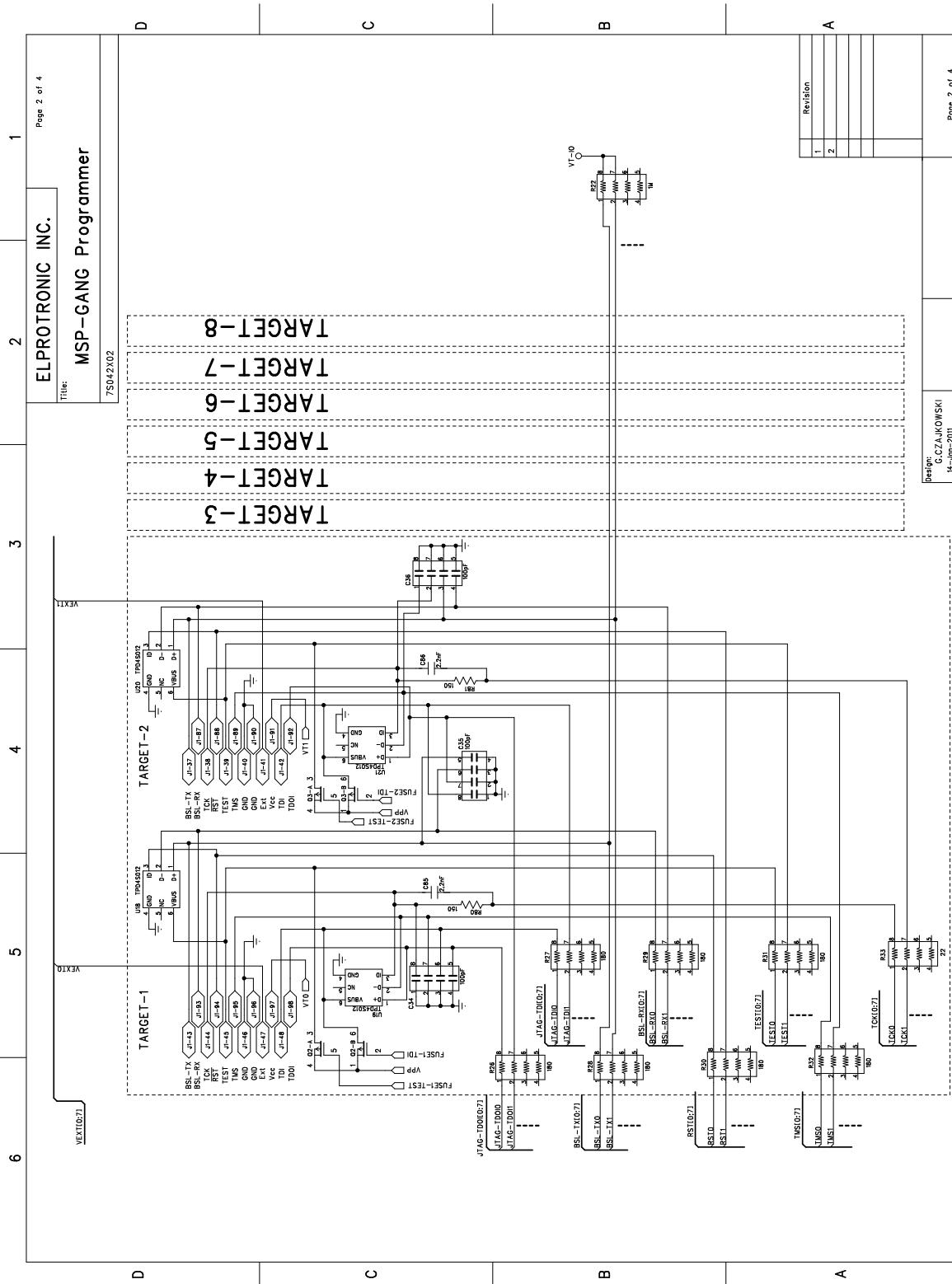


Figure 5-2. MSP-GANG Simplified Schematic (2 of 4)

MSP-GANG-simplified.sch-3 - Sun Jan 15 14:25:06 2012

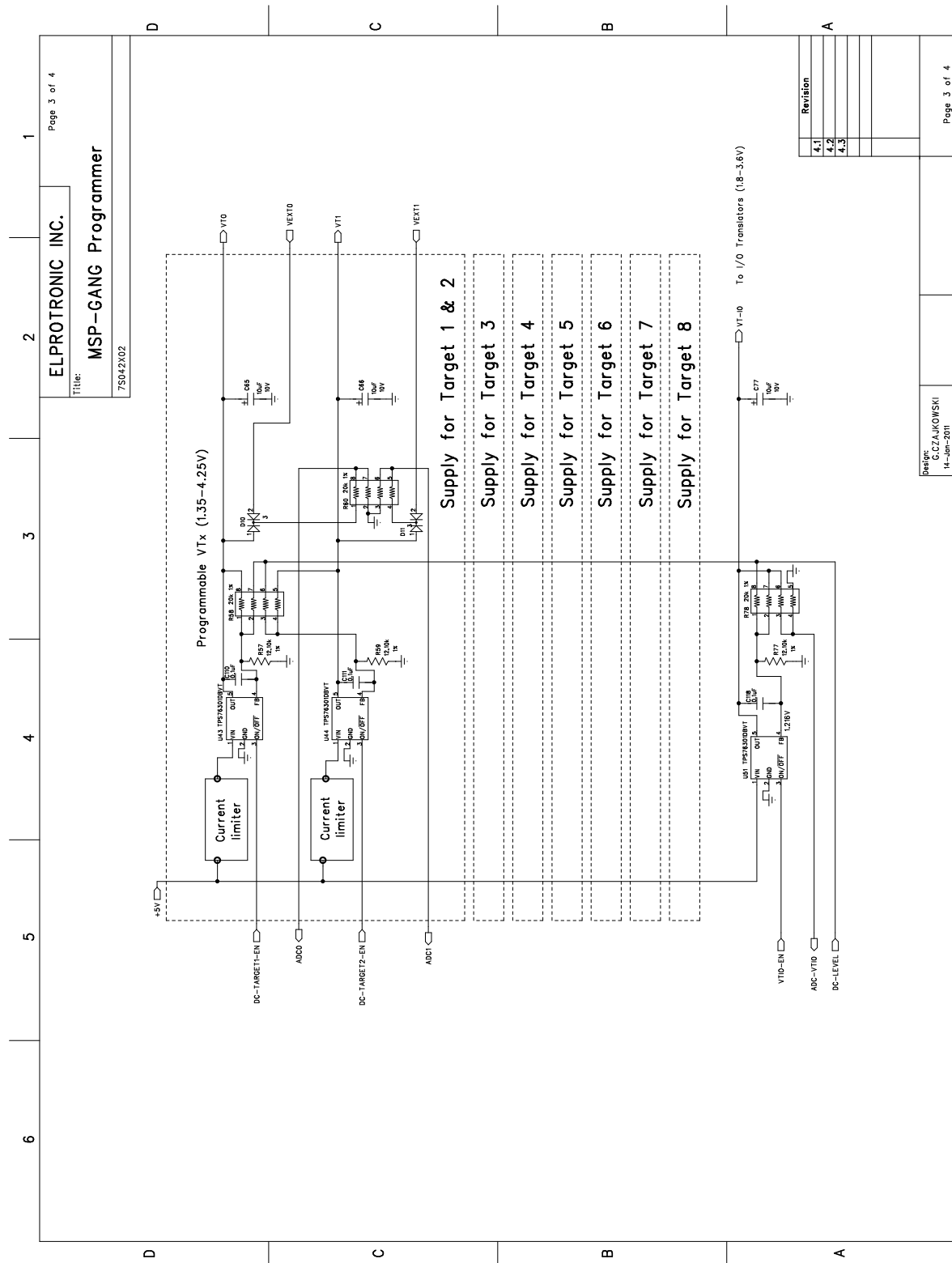


Figure 5-3. MSP-GANG Simplified Schematic (3 of 4)

MSP-GANG-simplified.sch-4 - Sun Jan 15 14:25:07 2012

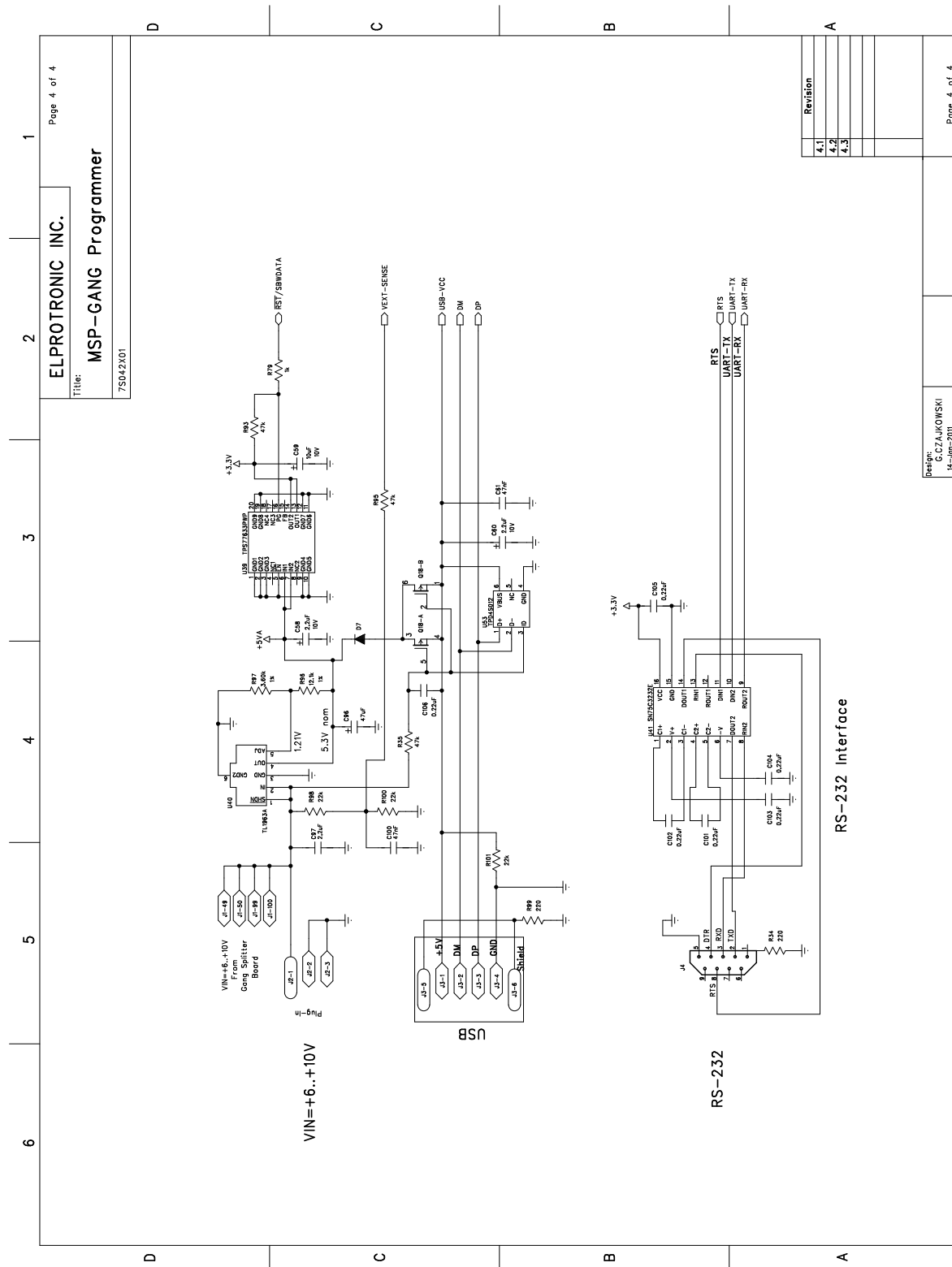


Figure 5-4. MSP-GANG Simplified Schematic (4 of 4)

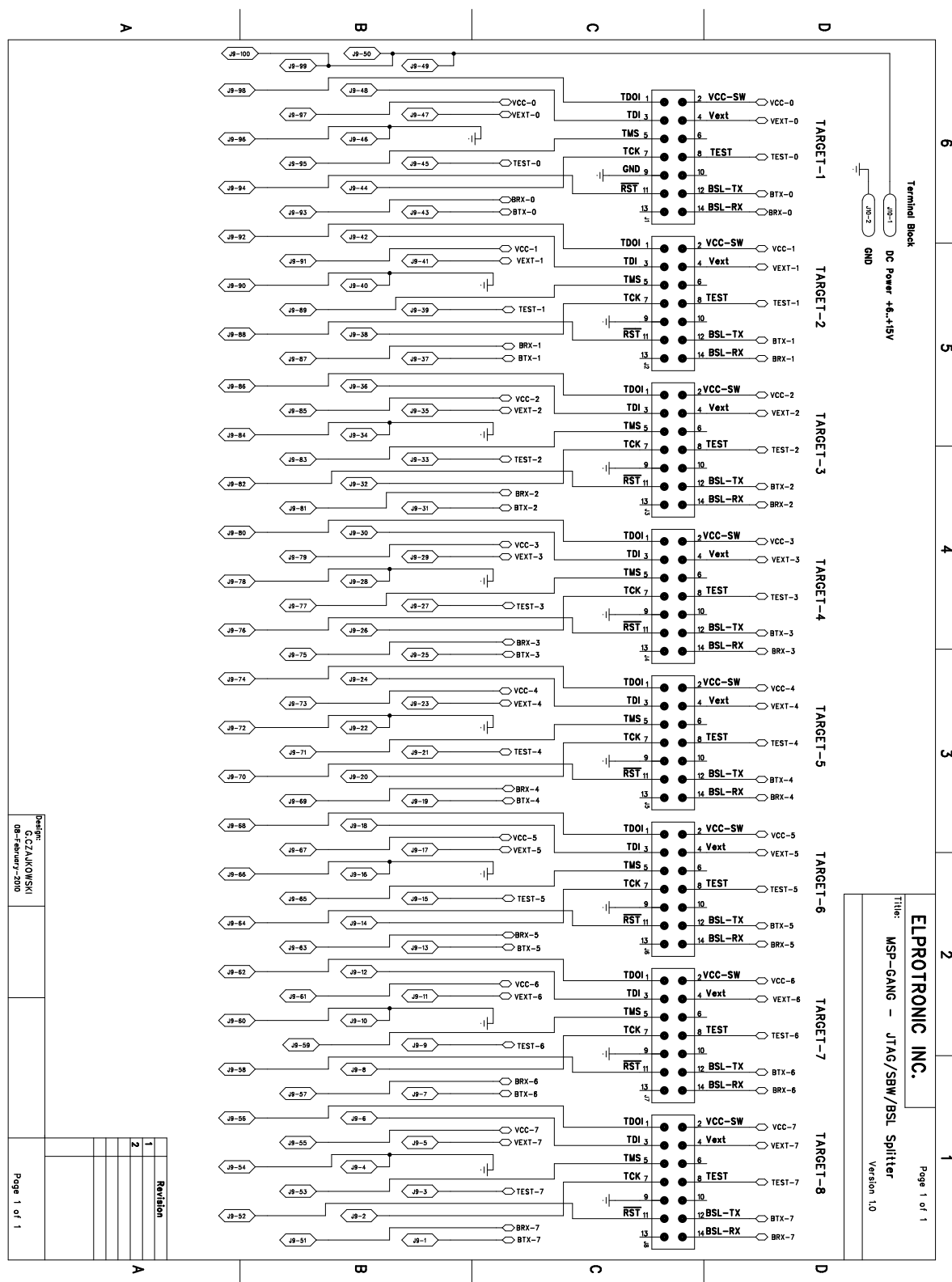
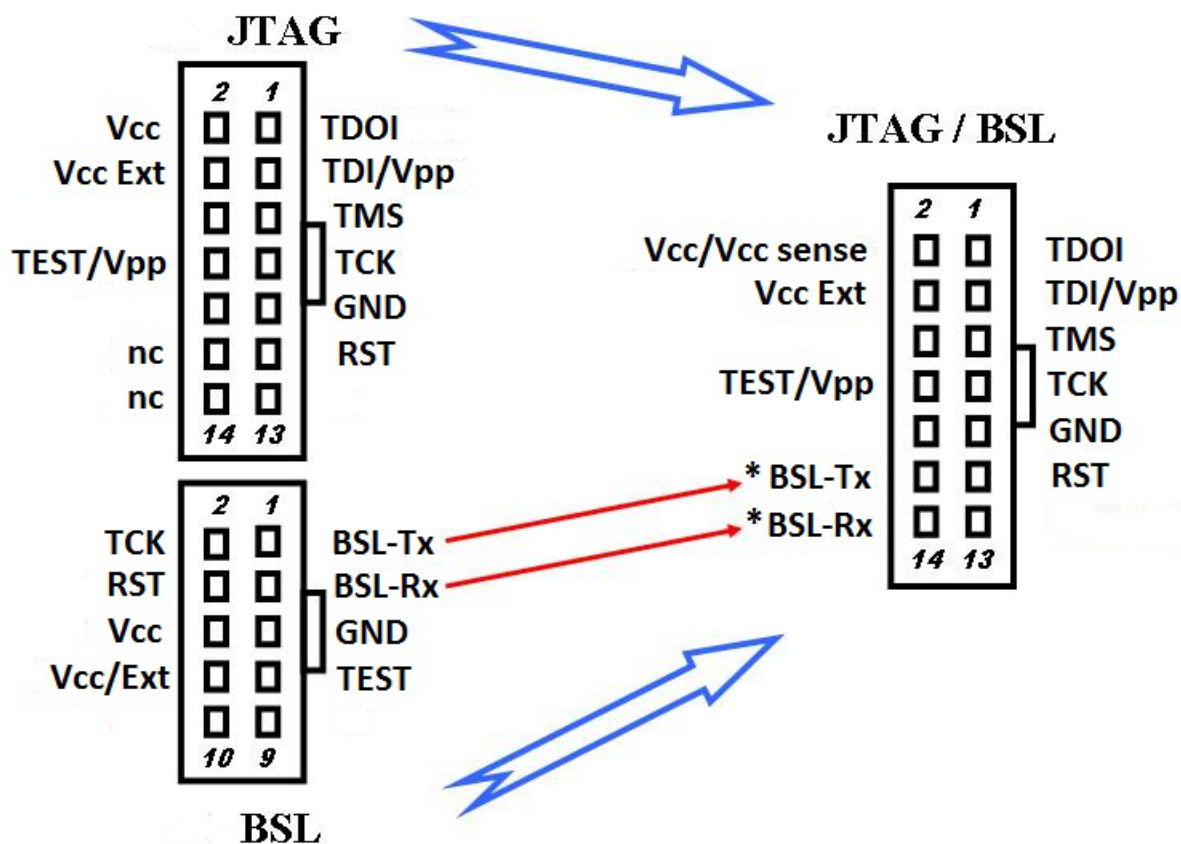


Figure 5-5. Gang Splitter Schematic

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

Item	Name	Drawing and Part Number	Quantity	Description
1	BLANK PC BOARD	MSP-GANG-SP rev-2	1	Blank PC Board
THROUGH HOLE COMPONENTS				
1	Connector	SBH11-PBPC-D07-ST-BK	1	14-pins Header Connector (Sullins)
2	Connector	SBH11-PBPC-D07-ST-BK	1	14-pins Header Connector (Sullins)
3	Connector	SBH11-PBPC-D07-ST-BK	1	14-pins Header Connector (Sullins)
4	Connector	SBH11-PBPC-D07-ST-BK	1	14-pins Header Connector (Sullins)
5	Connector	SBH11-PBPC-D07-ST-BK	1	14-pins Header Connector (Sullins)
6	Connector	SBH11-PBPC-D07-ST-BK	1	14-pins Header Connector (Sullins)
7	Connector	SBH11-PBPC-D07-ST-BK	1	14-pins Header Connector (Sullins)
8	Connector	SBH11-PBPC-D07-ST-BK	1	14-pins Header Connector (Sullins)
J9	Connector	TX24-100R-LT-H1E	1	100p-Receptacle Right Angle Connector (JAE Electronics)
J10	Connector	do not populate		2-pins terminal block
	Bumpers	SJ61A6	3	Bumpon, cylindrical 0.312 x 0.215, black


**Figure 5-6. BSL Connection Schematic**

Detailed description of the BSL connection can be found in *MSP430 Programming With the Bootloader (BSL)* (SLAU319). It is important to note that the MSP-GANG Programmer's Fast-BSL has much higher communication speed than standard BSL, 200 kbps compared to 9.6 kbps. Consequently, ensure that the hardware does not have additional delay on the BSL RX and TX lines beyond 0.5  $\mu$ s (a clock pulse duration of 2  $\mu$ s must be transmitted by the BSL RX and TX lines without degradation). Any additional filters or suppressors on the BSL RX and TX lines can degrade communication.

# MSP-GANG-432 ADPTR Rev.1.1

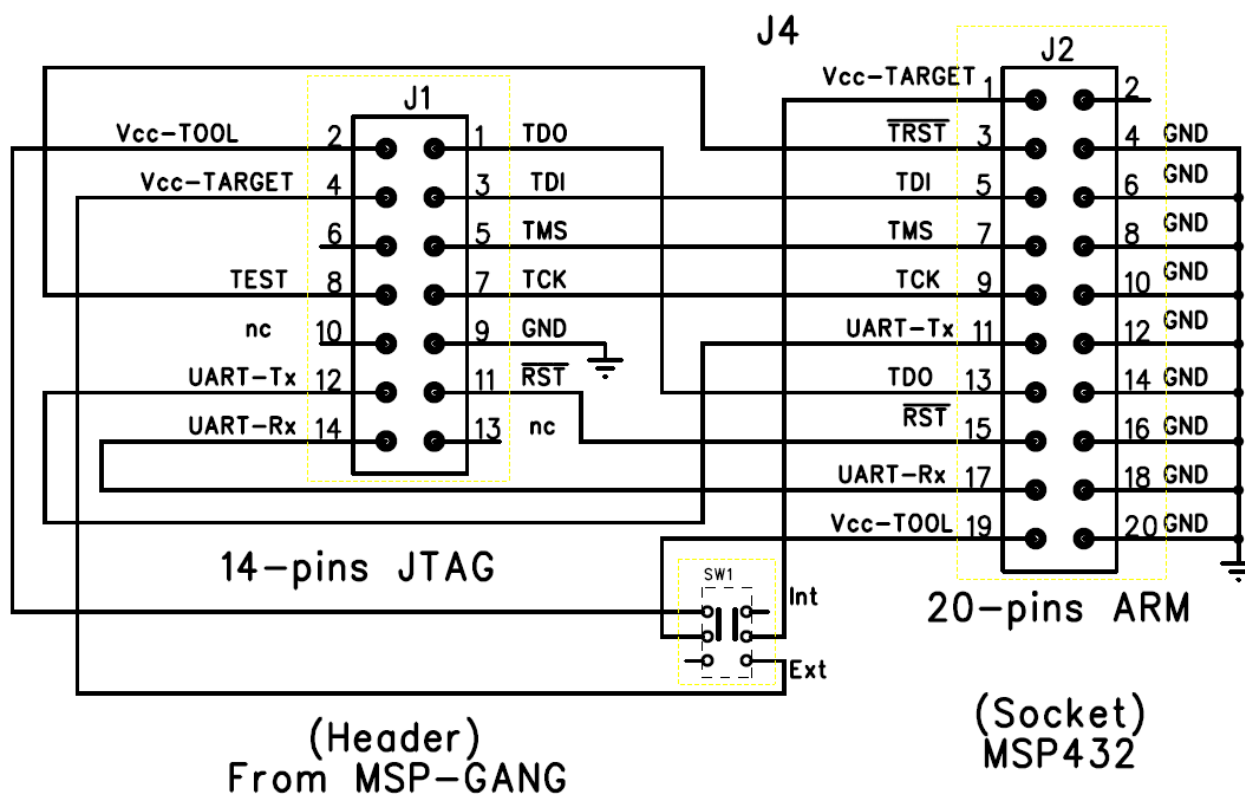
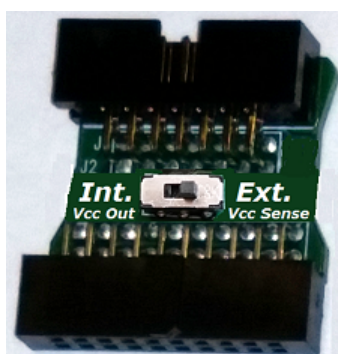


Figure 5-7. Schematic of MSP-GANG 14-20 Adapter



NOTE: Adapter should be plugged in on the 14-pin JTAG cable. The 20-pin end is connected to the MSP432 JTAG connector on the target board.

Figure 5-8. Top View of MSP-GANG 14-20 Adapter (Order Separately From TI)

## Frequently Asked Questions

---

---

### 6.1 Question: Why does device init, connect, or programming fail?

Answer: Frequently the cause is a bad connection between the MSP-GANG Programmer and the target device. A 14-wire ribbon cable is provided for JTAG/SBW or BSL connection between MSP-GANG and target device. The ribbon cable has an impedance of approximately 100  $\Omega$  and has DC lines in the ribbon cable to provide good isolation between signal wires (TDI, TCK, TMS, TDO). Each signal wire is separated by a DC wire to minimize crosstalk between the signal wires. The following pinout used in the MSP-GANG on each JTAG connector:

1	- TDO	(Signal wire)
2	- Vcc / Vcc sense	(DC wire)
3	- TDI	(Signal wire)
4	- Vcc sense	(DC wire)
5	- TMS	(Signal wire)
6	- n/c	(DC wire)
7	- TCK	(Signal wire)
8	- TEST	(DC wire in JTAG)
9	- GND	(DC wire)
10	- n/c	(DC wire)
11	- RESET	(DC wire in JTAG)
12	- BSL-Tx	(Signal wire)
13	- n/c	(DC wire)
14	- BSL-Rx	(Signal wire)

The provided 14-pin connector might not be ideal for some customers who want to minimize the number of wires and pinout order. When using a custom cable, make sure to address the issue of crosstalk between signal cables. Unfortunately, in many cases the custom cable does not provide good isolation between signal wires. As an example of a bad connection, the following uses an 8-wire ribbon cable for JTAG communication:

1	- Vcc / Vcc sense	(DC wire)
2	- TDO	(Signal wire)
3	- TDI	(Signal wire)
4	- TMS	(Signal wire)
5	- TCK	(Signal wire)
6	- TEST	(DC wire in JTAG)
7	- GND	(DC wire)
8	- RESET	(DC wire in JTAG)

On this connection, the TMS signal is coupled with the TCK and TDI lines and can generate additional TCK pulses on the TCK wire (rise time on TMS line can be seen on the TCK line also and can be detected by the MSP MCU as an additional unexpected TCK pulse).



## 6.2 Question: Can I use single wires for connection between MSP-GANG and target device?

Answer: Single wires are a poor type of connection and provide very bad quality. Single wires work like inductors connected between the MSP-GANG and target device, generating ripples on the target device side. If a ribbon cable cannot be used, then a twisted-pairs connection should be used instead. One wire on each twisted pair should be connected to a signal connection (for example, TDI or TCK), and the second wire connected to DC (GND or  $V_{CC}$ ) from both sides of the connection (one on the MSP-GANG side and the other on the target device side). For example, the following arrangement is acceptable:

```
1 - TDO      (Signal wire)      - first twisted pair
1 - Vcc / Vcc sense (DC wire)
2 - TDI      (Signal wire)      - second twisted pair
2 - Vcc sense (DC wire)
3 - TMS      (Signal wire)      - third twisted pair
3 - RESET    (DC wire)
4 - TCK      (Signal wire)      - fourth twisted pair
4 - GND      (DC wire)
```

If additional protecting components (such as a capacitor or suppressors) are used on the target device PCB, check the JTAG signal shape on the MSP MCU. The JTAG communication speed should be decreased (set to medium or slow) if required. Make sure that any ripple on the JTAG lines is smaller than 20% of the peak-to-peak signal level.

If connection cables are longer than 40 cm, then the ripple can be reduced by inserting 33- $\Omega$  resistors in series with TCK, TMS, and TDO in the middle of the connection wires. Do not provide series resistors in TEST and TDI lines if the device will be secured (blown the security fuse) for MSP families 1xx, 2xx, and 4xx. For blowing the security fuse in these devices, the programmer provides Vpp 6.5 V at 100 mA on the TEST or TDI lines to MSP MCU. An additional resistor inserted in these lines can reduce the maximum current provided to the MCU and the security fuse will not be blown.

## 6.3 Question: How to serialize parts?

Answer: The MSP-GANG GUI does not provide serialization; however, the provided MSP-GANG.dll allows to program unique data (for example, calibration or serialization) to each target device. An example to implement serialization using MSP-GANG.dll is available in this directory:

```
C:\Program Files (x86)\Texas Instruments\MSP-GANG\Examples\CPP_Applications_MSP_DLL
```

## 6.4 Question: How to have parts run after programming?

Answer: By default in the MSP-GANG Programmer, the RESET line is forced to low level, which prevents the target device running after being programmed. But that option can be modified using the pulldown menu:

- Setup > Finish Action

and selecting one of the options:

- Hardware reset (RST line) and start the application program
- OFF/ON the  $V_{CC}$  and start the application program

Application Program Run time is programmable from 1 to 120 seconds or infinite time.

## 6.5 Question: What are possible reasons for the part to fail Verify step?

Answer: If the part was programmed and verified in the GO step, and after the second time the Verify step failed, then in most cases the firmware is modifying flash contents when running for the first time. Usually the Info memory is modified in that case. Ask your software team if the firmware downloaded to the MCU is modifying the flash after the first run. If that is the case, then the firmware should be modified to contain only unmodified contents in the code file. To compare the code file contents and flash data (if modified), use this option from the pulldown menu:

- View > Compare Code File and Flash Data

## Revision History

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

Changes from July 12, 2018 to February 22, 2019	Page
• Updated FCC and CE compliance (changed 3 instances of "Class B" to "Class A") .....	8
• Added dimensions in <a href="#">Figure 1-1</a> , <i>Top View of the MSP Gang Programmer</i> .....	10

## IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale ([www.ti.com/legal/termsofsale.html](http://www.ti.com/legal/termsofsale.html)) or other applicable terms available either on [ti.com](http://ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2019, Texas Instruments Incorporated