MSP Gang Programmer (MSP-GANG)

User's Guide

Literature Number: SLAU358H
September 2011–Revised February 2015

Contents

Preface **7**

1 Introduction **10**

1.1 Software Installation 11

1.2 Driver Installation 12

1.3 Hardware Installation 12

2 Operation **15**

2.1 Programming MSP430 Flash Devices Using the MSP Gang Programmer 15

2.1.1 Programming Using Interactive Mode 16

2.1.2 Programming From Image 21

2.1.3 Programming From Script 25

2.1.4 Programming in Standalone Mode 29

2.1.5 Memory Setup for GO, Erase, Program, Verify, and Read 32

2.1.6 Creating and Using Images 34

2.1.7 Programming From Image File 36

2.1.8 Programming From SD Card 38

2.1.9 File Extensions 38

2.1.10 Checksum Calculation 38

2.2 Data Viewers 39

2.3 Status Messages 42

2.4 Self Test 46

2.5 Label 51

2.6 Preferences 52

2.6.1 USB ID Number 52

2.6.2 COM Port 52

2.6.3 LCD Contrast 52

2.6.4 CheckSum - Gang430 Standard 52

2.7 Benchmarks 53

2.7.1 Benchmarks for MSP430F5xx 53

2.7.2 Benchmarks for MSP430FR57xx 54

2.7.3 Benchmarks for MSP430F2xx 54

3 Firmware **56**

3.1 Commands 56

3.2 Firmware Interface Protocol 57

3.3 Synchronization Sequence 57

3.4 Command Messages 57

3.4.1 Frame Structure 57

3.4.2 Checksum 59

3.5 Detailed Description of Commands 59

3.5.1 General 59

3.5.2 Commands Supported by the BOOT Loader 59

3.5.3 Commands Supported by Application Firmware 63

3.5.4 API Firmware Commands That Should Not be Used 66

4 Dynamic Link Library for MSP-GANG Programmer **71**

4.1 Gang430.dll Wrapper Description 71

4.2 MSP-GANG.dll Description 71
4.2.1 MSPGANG_GetDataBuffers_ptr 72
4.2.2 MSPGANG_SetGangBuffer, MSPGANG_GetGangBuffer 73
4.2.3 MSPGANG_GetDevice 74
4.2.4 MSPGANG_LoadFirmware 75
4.2.5 MSPGANG_InitCom 75
4.2.6 MSPGANG_ReleaseCom 75
4.2.7 MSPGANG_GetErrorString 76
4.2.8 MSPGANG_SelectBaudrate 76
4.2.9 MSPGANG_GetDiagnostic 76
4.2.10 MSPGANG_MainProcess 77
4.2.11 MSPGANG_InteractiveProcess 77
4.2.12 MSPGANG_Interactive_Open_Target_Device 77
4.2.13 MSPGANG_Interactive_Close_Target_Device 78
4.2.14 MSPGANG_Interactive_DefReadTargets 78
4.2.15 MSPGANG_Interactive_ReadTargets 78
4.2.16 MSPGANG_Interactive_ReadBytes 79
4.2.17 MSPGANG_Interactive_WriteWord_to_RAM 79
4.2.18 MSPGANG_Interactive_WriteByte_to_RAM 80
4.2.19 MSPGANG_Interactive_WriteBytes_to_RAM 80
4.2.20 MSPGANG_Interactive_WriteBytes_to_FLASH 81
4.2.21 MSPGANG_Interactive_Copy_Gang_Buffer_to_RAM 81
4.2.22 MSPGANG_Interactive_Copy_Gang_Buffer_to_FLASH 82
4.2.23 MSPGANG_Interactive_EraseSectors 82
4.2.24 MSPGANG_Interactive_BlankCheck 83
4.2.25 MSPGANG_Interactive_DCO_Test 83
4.2.26 MSPGANG_SelectImage 84
4.2.27 MSPGANG_EraseImage 84
4.2.28 MSPGANG_CreateGangImage 84
4.2.29 MSPGANG_LoadImageBlock 85
4.2.30 MSPGANG_VerifyPSAImageBlock 86
4.2.31 MSPGANG_ReadImageBlock 86
4.2.32 MSPGANG_Read_Code_File 90
4.2.33 MSPGANG_Save_Config, MSPGANG_Load_Config, MSPGANG_Default_Config 90
4.2.34 MSPGANG_SetConfig, MSPGANG_GetConfig 91
4.2.35 MSPGANG_GetNameConfig, MSPGANG_SetNameConfig 96
4.2.36 MSPGANG_SetTmpGANG_Config 97
4.2.37 MSPGANG_GetLabel 97
4.2.38 MSPGANG_GetInfoMemory, MSPGANG_SetInfoMemory 98
4.2.39 MSPGANG_Get_qty_MCU_Family, MSPGANG_Get_MCU_FamilyName, MSPGANG_Check_MCU_Name, MSPGANG_Get_MCU_Name 99
4.2.40 MSPGANG_Set_MCU_Name 100
4.2.41 MSPGANG_HW_devices 100
4.2.42 MSPGANG_GetProgressStatus 101
4.2.43 MSPGANG_GetAPIStatus 103
4.2.44 MSPGANG_Set_IO_State 104

5 Schematics 107

6 Frequently Asked Questions 115

6.1 Question: Why does device init, connect, or programming fail? 115
6.2 Question: Can I use single wires for connection between MSP-GANG and target device? 116
6.3 Question: How to serialize parts? 116
6.4 Question: How to have parts run after programming? 116
6.5 Question: What are possible reasons for the part to fail Verify step? 116

Revision History **117**

List of Figures

1-1. Top View of the MSP Gang Programmer 11
2-1. Main MSP Gang Programmer Dialog GUI, Interactive Mode 16
2-2. Memory Options 18
2-3. Reset Options 19
2-4. Flash Memory Data 21
2-5. Main MSP Gang Programmer Dialog GUI, From Image Mode 22
2-6. Main MSP Gang Programmer Dialog GUI, From Image Mode and Custom Configuration Enabled 24
2-7. Main MSP Gang Programmer Dialog GUI, From Script 25
2-8. Main MSP Gang Programmer Dialog GUI, Standalone Mode 29
2-9. Image Option 30
2-10. Target Enable or Disable Option 31
2-11. Memory Options, BSL Sectors Selected 33
2-12. Image Name Configuration Screen 35
2-13. Image File Security Options 35
2-14. Hardware Fingerprint of Computer in Use 36
2-15. Programming From Image File 37
2-16. Password for Image File 37
2-17. Code File Data 40
2-18. Comparison of Code and Flash Memory Data of the Target Microcontroller 41
2-19. Self Test 47
2-20. Information About the MSP Gang Programmer 51
2-21. Preferences Selection Window 52
5-1. MSP-GANG Simplified Schematic (1 of 4) 107
5-2. MSP-GANG Simplified Schematic (2 of 4) 108
5-3. MSP-GANG Simplified Schematic (3 of 4) 109
5-4. MSP-GANG Simplified Schematic (4 of 4) 110
5-5. Gang Splitter Schematic 111
5-6. BSL Connection Schematic 112

List of Tables

2-1. Benchmark Results – MSP430F5438A, 256kB Code 53
2-2. Benchmark Results – MSP430F5438A, 250kB Code, Mode: From Image 53
2-3. Benchmark Results – MSP430F5438A, 250kB Code, Mode: Interactive, Communication by USB 53
2-4. Benchmark Results – MSP430FR5738, 15kB Code, Mode: From Image 54
2-5. Benchmark Results – MSP430FR5738, 15kB Code, Mode: Interactive, Communication by USB 54
2-6. Benchmark Results – MSP430F2619, 120kB Code, Mode: From Image 54
2-7. Benchmark Results – MSP430F2619, 120kB Code, Mode: Interactive, Communication by USB 54
3-1. Data Frame for Firmware Commands 58
5-1. Gang Splitter Bill of Materials (BOM) 112

Read This First

If You Need Assistance

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

Related Documentation from Texas Instruments

The primary sources of MSP430 information are the device-specific data sheets and user's guides. The most current information is on the MSP430 web site at www.ti.com/msp430.

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

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

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

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

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

Introduction

The MSP Gang Programmer is an MSP430 device programmer that can program up to eight identical MSP430 flash or FRAM devices at the same time. The MSP Gang Programmer connects to a host PC using a standard RS-232 or USB connection and provides flexible programming options that allow the user to fully customize the process. A top-level view of the MSP Gang Programmer can be seen in Figure 1-1.

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

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

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

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

Chapter 4 describes Gang430.dll and MSP-GANG.dll, including the functions available through them.

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

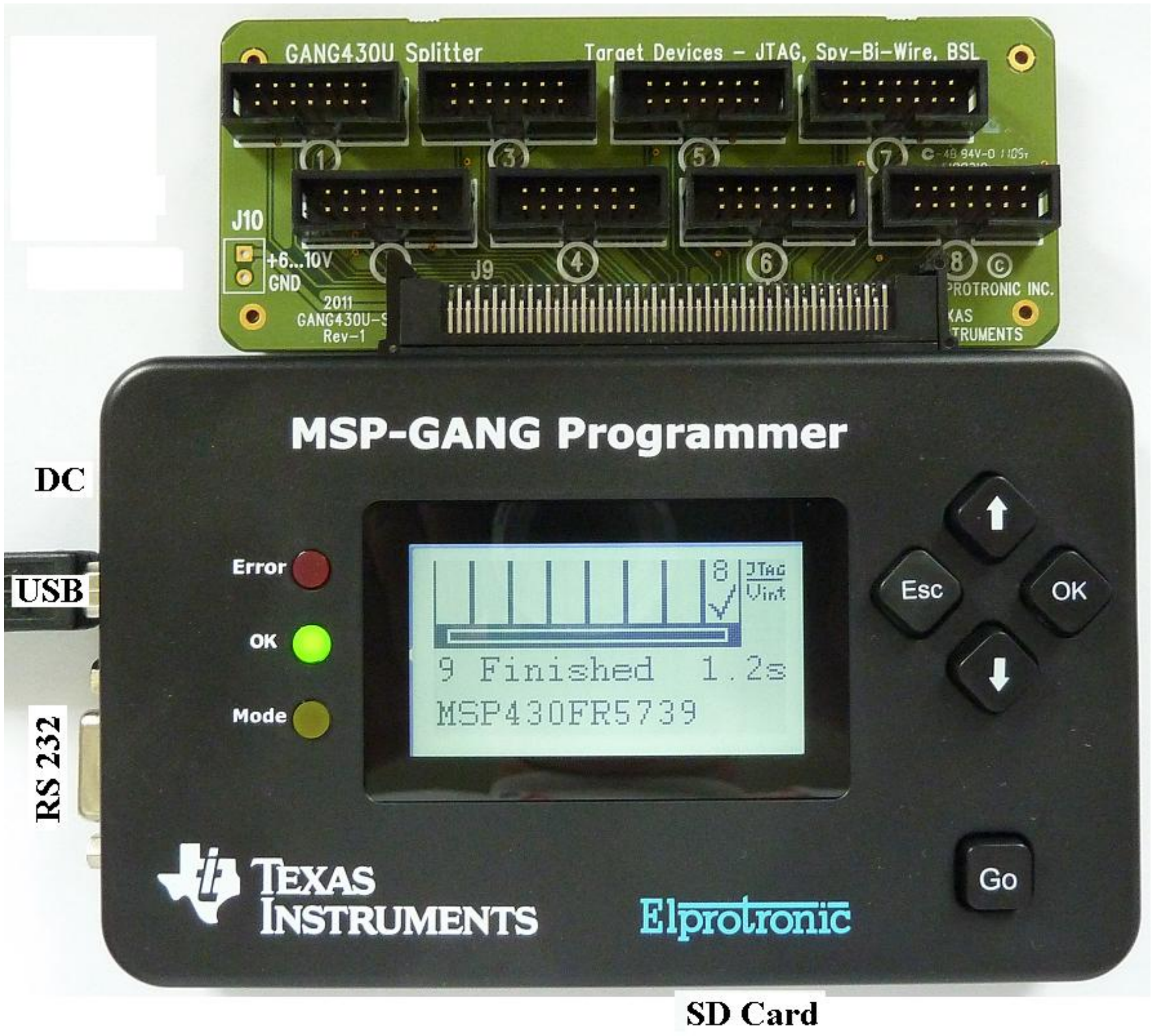


Figure 1-1. Top View of the MSP Gang Programmer

1.1 Software Installation

TI recommends that you use the latest software version, which can be downloaded from the MSP430 website at www.ti.com/msp-gang.

To install MSP Gang Programmer software:

1. Insert the MSP430 CD-ROM into the CD-ROM drive of the host computer. Setup automatically opens the default browser and displays the MSP430 start page.

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

1.2 Driver Installation

To install the required drivers:

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

   C:\Program Files\Texas Instruments\MSP-GANG\Driver
2. If the RS-232 interface is used for communication with MSP-GANG, then the additional driver is not required. Check the Device Manager 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 provides connectivity for up to eight targets using the included 14-pin cables. The target MSP430 flash devices can be in standalone sockets or can be on an application's PCB. These devices can be accessed through JTAG or Spy-Bi-Wire (SBW) signals.
2. Connect the MSP Gang Programmer hardware to a computer's USB port using a USB A-B cable.

   The programmer can be fully supplied from the computer's USB port (5 V, 0.5 A). The programmer can also be connected to a serial port (COM1 to COM255) using a 9-pin Sub-D connector, if the computer does not have a USB port.
3. An external power supply is required to power the MSP Gang Programmer if it is not connected through the USB port or if the total current consumption of the programmed target devices exceeds 0.3 A.

NOTE: External Power Supply

If an external power supply is used then it must provide a voltage between 6 V and 10 V dc and must be capable of providing a minimum current of 800 mA. The center post of the power supply connector on the MSP Gang Programmer is the positive-voltage terminal. The programmer indicates the status of the power supply connection by using system LEDs and the LCD back light.

NOTE: Maximum Signal Path Length: 50 cm

The maximum length of a signal path between the 14-pin JTAG or SBW connector on the Gang Splitter and the target device is 50 cm.

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

CAUTION

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

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

Also, the same voltage value that is used for powering the target device should be set in the MSP Gang Programmer as the desired V_{CC} level. This information is mandatory to provide correct I/O levels for the TMS, TCK, TDI, TDO, 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 become 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 MSP430 Flash Devices Using the MSP Gang Programmer

The MSP Gang Programmer is capable of quickly and reliably programming MSP430 flash devices using an RS-232 or USB interface. There are four ways to use the programmer to achieve this task and these include:

- Interactive
- From Image
- From Script
- Standalone

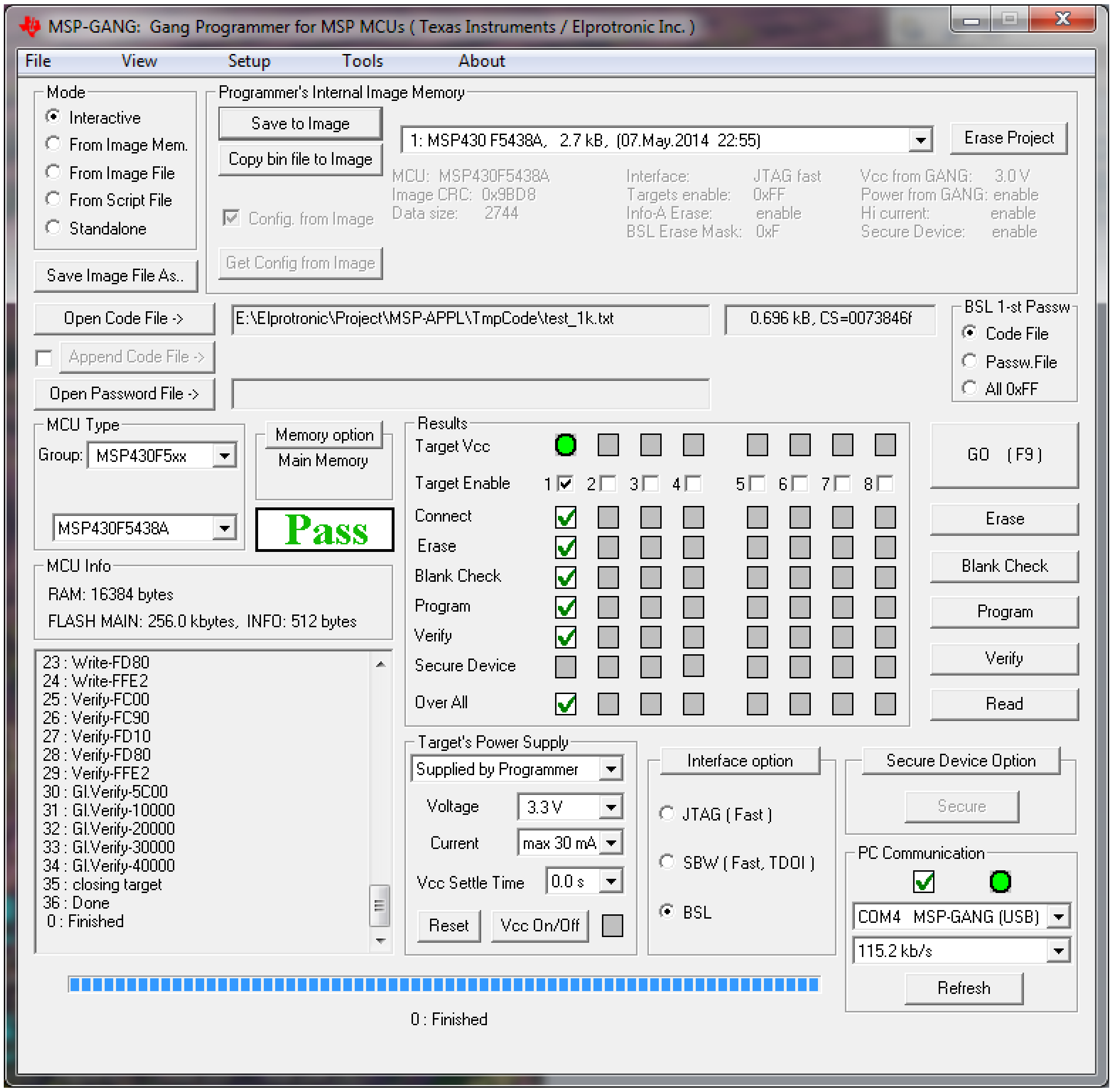
The Interactive mode is selected by default, and is the easiest to get started with, because it requires the least amount of preparation. After the user has mastered the Interactive mode it can be used to create images and script files, which can then be used with the From Image and From Script modes,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

The following 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 MSP430 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.



Mode: [✓ Interactive] From Image Mem. From Image File From Script File Standalone

Programmer's Internal Image Memory: 1: MSP430 F5438A, 2.7 kB, (07.May.2014 22:55)

[✓ Config. from Image]

```
MCU:  MSP430F5438A     Interface:        JTAG fast   Vcc from GANG:    3.0 V
Image CRC:  0x9BD8     Targets enable:   0xFF        Power from GANG:  enable
Data size:   2744      Info-A Erase:     enable      Hi current:       enable
                       BSL Erase Mask:   0xF         Secure Device:    enable
```

E:\Elprotronic\Project\MSP-APPL\TmpCode\test_1k.txt — 0.696 kB, CS=0073846f

BSL 1-st Passw: [✓ Code File] Passw.File All 0xFF

MCU Type — Group: MSP430F5xx; MSP430F5438A

Pass

MCU Info
RAM: 16384 bytes
FLASH MAIN: 256.0 kbytes, INFO: 512 bytes

```
23 : Write-FD80
24 : Write-FFE2
25 : Verify-FC00
26 : Verify-FC90
27 : Verify-FD10
28 : Verify-FD80
29 : Verify-FFE2
30 : Gl.Verify-5C00
31 : Gl.Verify-10000
32 : Gl.Verify-20000
33 : Gl.Verify-30000
34 : Gl.Verify-40000
35 : closing target
36 : Done
 0 : Finished
```

Results

	1	2	3	4	5	6	7	8
Target Vcc	●							
Target Enable	✓							
Connect	✓							
Erase	✓							
Blank Check	✓							
Program	✓							
Verify	✓							
Secure Device								
Over All	✓							

Target's Power Supply: Supplied by Programmer
Voltage: 3.3 V
Current: max 30 mA
Vcc Settle Time: 0.0 s

Interface: JTAG (Fast) SBW (Fast, TDOI) [✓ BSL]

PC Communication: ✓ ● COM4 MSP-GANG (USB) 115.2 kb/s

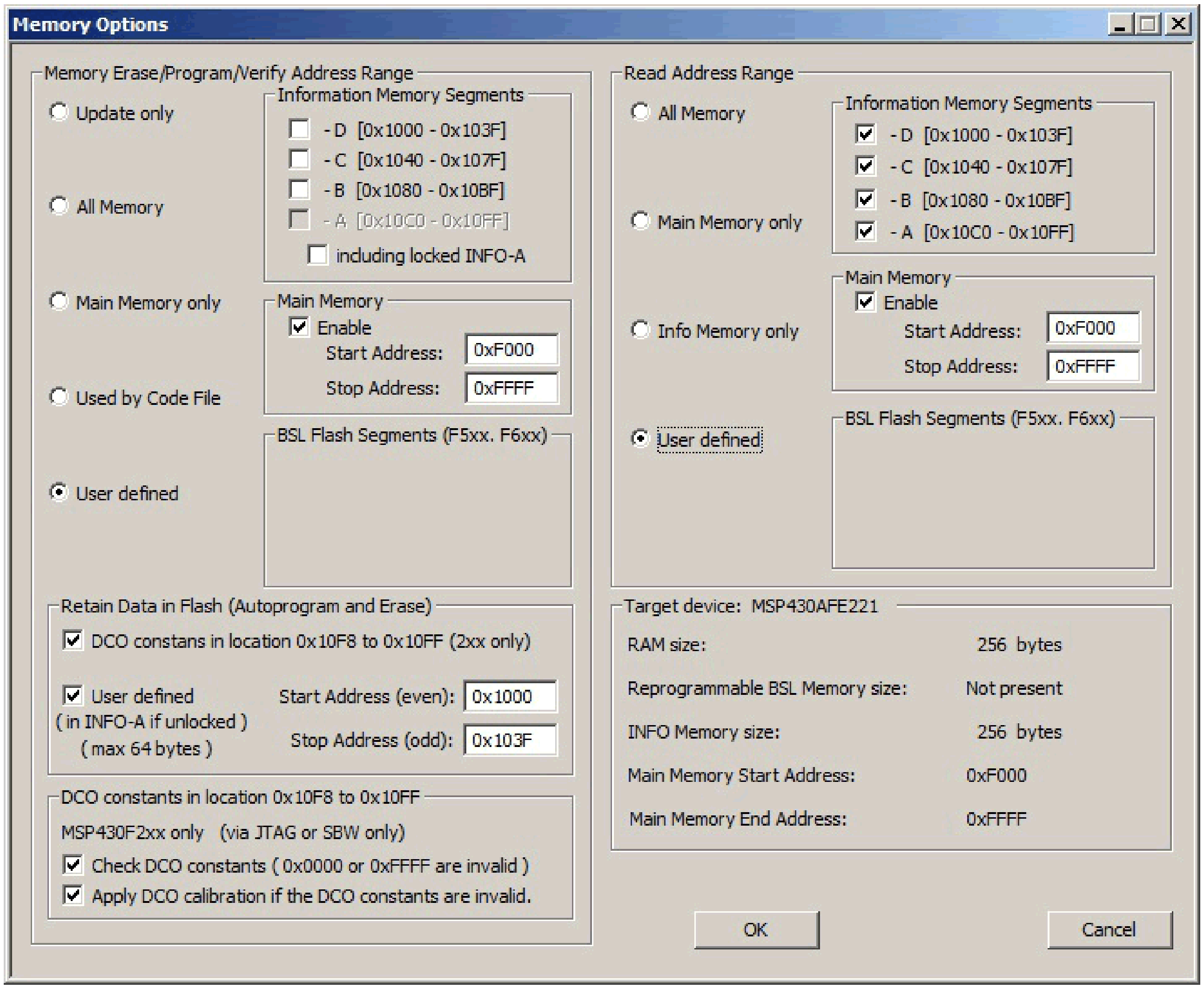
0 : Finished

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

2. Select a target device using the MCU Type menu (select MCU group and then desired MCU type).
3. Select the code file to be programmed into the 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 2.1.10).
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 or Spy-Bi-Wire) 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)

Retain Data in Flash (Autoprogram and Erase)

[✓ DCO constans 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.]

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)

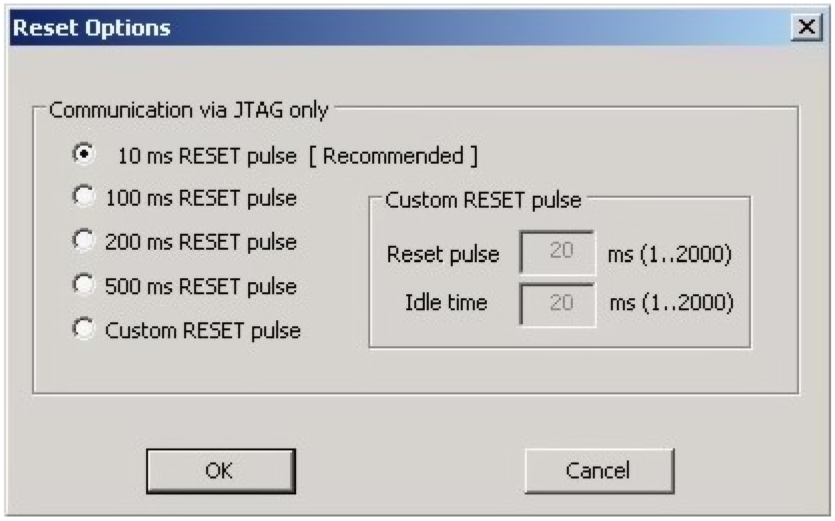
Target device: MSP430AFE221

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

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.



Communication via JTAG only

- [✓ 10 ms RESET pulse [Recommended]]
- 100 ms RESET pulse
- 200 ms RESET pulse
- 500 ms RESET pulse
- Custom RESET pulse

Custom RESET pulse

Reset pulse 20 ms (1..2000)

Idle time 20 ms (1..2000)

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

Figure 2-3. Reset Options

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

2.1.1.1 GO

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

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

2.1.1.2 Erase

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

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

2.1.1.5 Verify

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

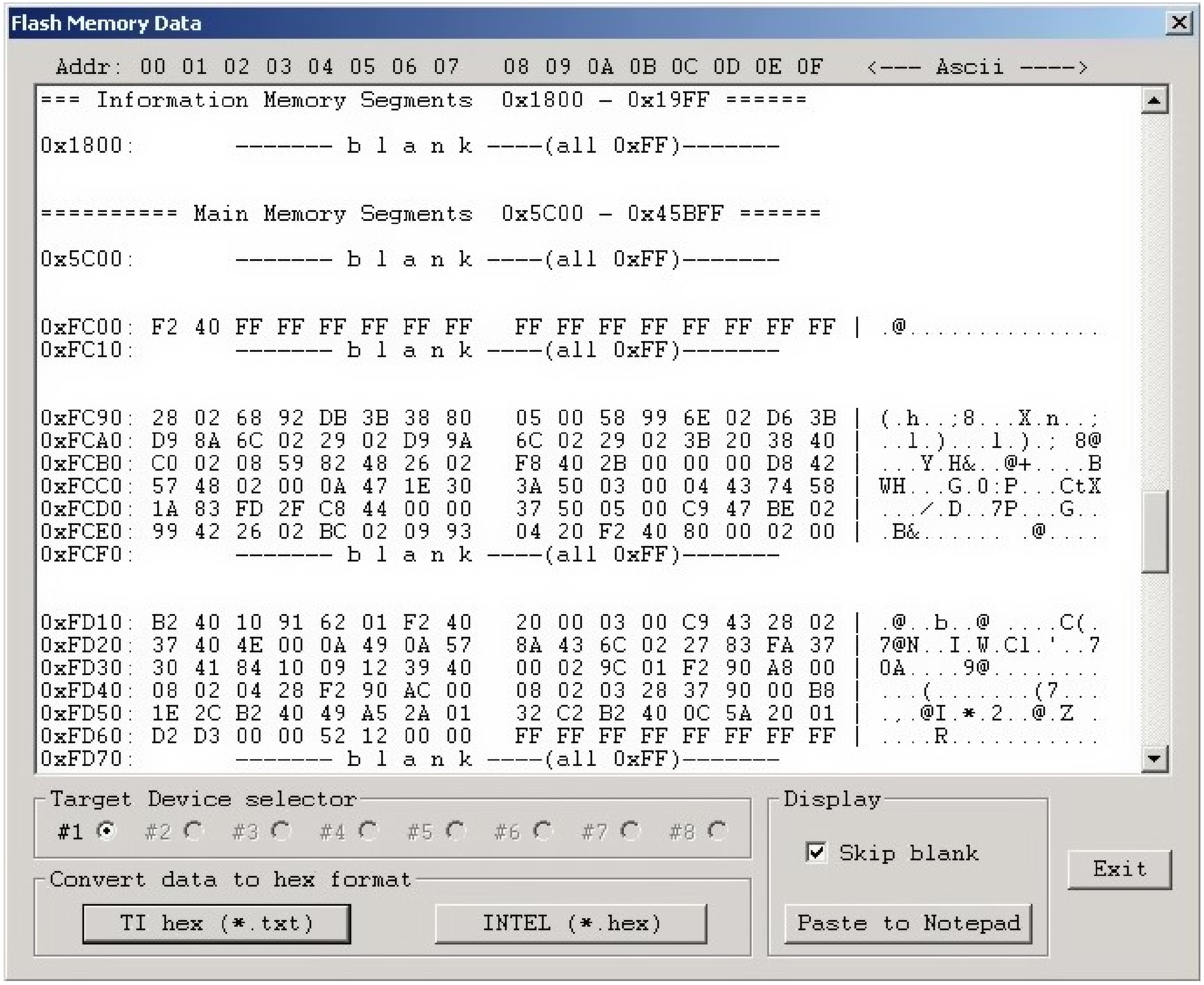
Verification of selected flash memory is divided into two steps: (1) verify selected flash memory that only corresponds to the code file, and (2) verify selected flash memory that corresponds to the code file AND selected flash memory not included in the code file that should be empty (0xFF). Examples 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 Gl.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.

2.1.1.6 Read

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

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



Flash Memory Data

```
 Addr: 00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F   <--- Ascii ---->
=== Information Memory Segments  0x1800 - 0x19FF ======

0x1800:       ------- b l a n k ----(all 0xFF)-------


========== Main Memory Segments  0x5C00 - 0x45BFF ======

0x5C00:       ------- b l a n k ----(all 0xFF)-------


0xFC00: F2 40 FF FF FF FF FF FF   FF FF FF FF FF FF FF FF | .@..............
0xFC10:       ------- b l a n k ----(all 0xFF)-------


0xFC90: 28 02 68 92 DB 3B 38 80   05 00 58 99 6E 02 D6 3B | (.h..;8...X.n..;
0xFCA0: D9 8A 6C 02 29 02 D9 9A   6C 02 29 02 3B 20 38 40 | ..l.)...l.).; 8@
0xFCB0: C0 02 08 59 82 48 26 02   F8 40 2B 00 00 00 D8 42 | ...Y.H&..@+....B
0xFCC0: 57 48 02 00 0A 47 1E 30   3A 50 03 00 04 43 74 58 | WH...G.0:P...CtX
0xFCD0: 1A 83 FD 2F C8 44 00 00   37 50 05 00 C9 47 BE 02 | .../.D..7P...G..
0xFCE0: 99 42 26 02 BC 02 09 93   04 20 F2 40 80 00 02 00 | .B&...... .@....
0xFCF0:       ------- b l a n k ----(all 0xFF)-------


0xFD10: B2 40 10 91 62 01 F2 40   20 00 03 00 C9 43 28 02 | .@..b..@ ....C(.
0xFD20: 37 40 4E 00 0A 49 0A 57   8A 43 6C 02 27 83 FA 37 | 7@N..I.W.Cl.'..7
0xFD30: 30 41 84 10 09 12 39 40   00 02 9C 01 F2 90 A8 00 | 0A....9@........
0xFD40: 08 02 04 28 F2 90 AC 00   08 02 03 28 37 90 00 B8 | ...(.......(7...
0xFD50: 1E 2C B2 40 49 A5 2A 01   32 C2 B2 40 0C 5A 20 01 | .,.@I.*.2..@.Z .
0xFD60: D2 D3 00 00 52 12 00 00   FF FF FF FF FF FF FF FF | ....R...........
0xFD70:       ------- b l a n k ----(all 0xFF)-------
```

Target Device selector: [✓ #1] #2 #3 #4 #5 #6 #7 #8

Display: [✓] Skip blank

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

Figure 2-4. Flash Memory Data

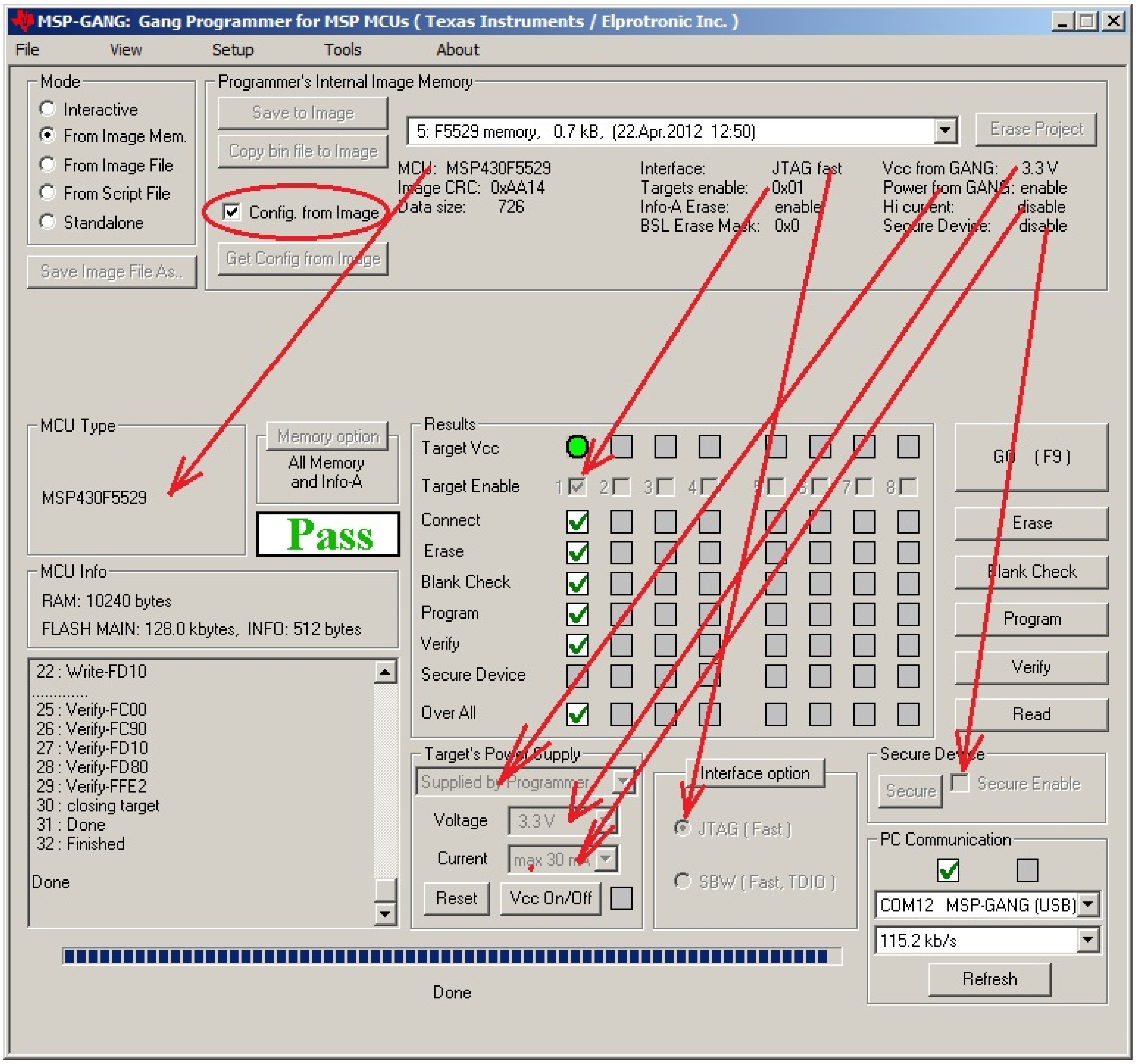
2.1.2 Programming From Image

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

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

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

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



Mode: Interactive, [✓ From Image Mem.], From Image File, From Script File, Standalone

Programmer's Internal Image Memory: 5: F5529 memory, 0.7 kB, (22.Apr.2012 12:50)

[✓ Config. from Image]

MCU:	MSP430F5529	Interface:	JTAG fast	Vcc from GANG:	3.3 V
Image CRC:	0xAA14	Targets enable:	0x01	Power from GANG:	enable
Data size:	726	Info-A Erase:	enable	Hi current:	disable
		BSL Erase Mask:	0x0	Secure Device:	disable

MCU Type: MSP430F5529

All Memory and Info-A

Pass

MCU Info
RAM: 10240 bytes
FLASH MAIN: 128.0 kbytes, INFO: 512 bytes

```
22 : Write-FD10
............
25 : Verify-FC00
26 : Verify-FC90
27 : Verify-FD10
28 : Verify-FD80
29 : Verify-FFE2
30 : closing target
31 : Done
32 : Finished

Done
```

Results

	1	2	3	4	5	6	7	8
Target Vcc	●							
Target Enable	✓							
Connect	✓							
Erase	✓							
Blank Check	✓							
Program	✓							
Verify	✓							
Secure Device								
Over All	✓							

Target's Power Supply: Supplied by Programmer; Voltage 3.3 V; Current max 30 mA

Interface option: [✓ JTAG (Fast)], SBW (Fast, TDIO)

Secure Device: Secure Enable

PC Communication: COM12 MSP-GANG (USB); 115.2 kb/s

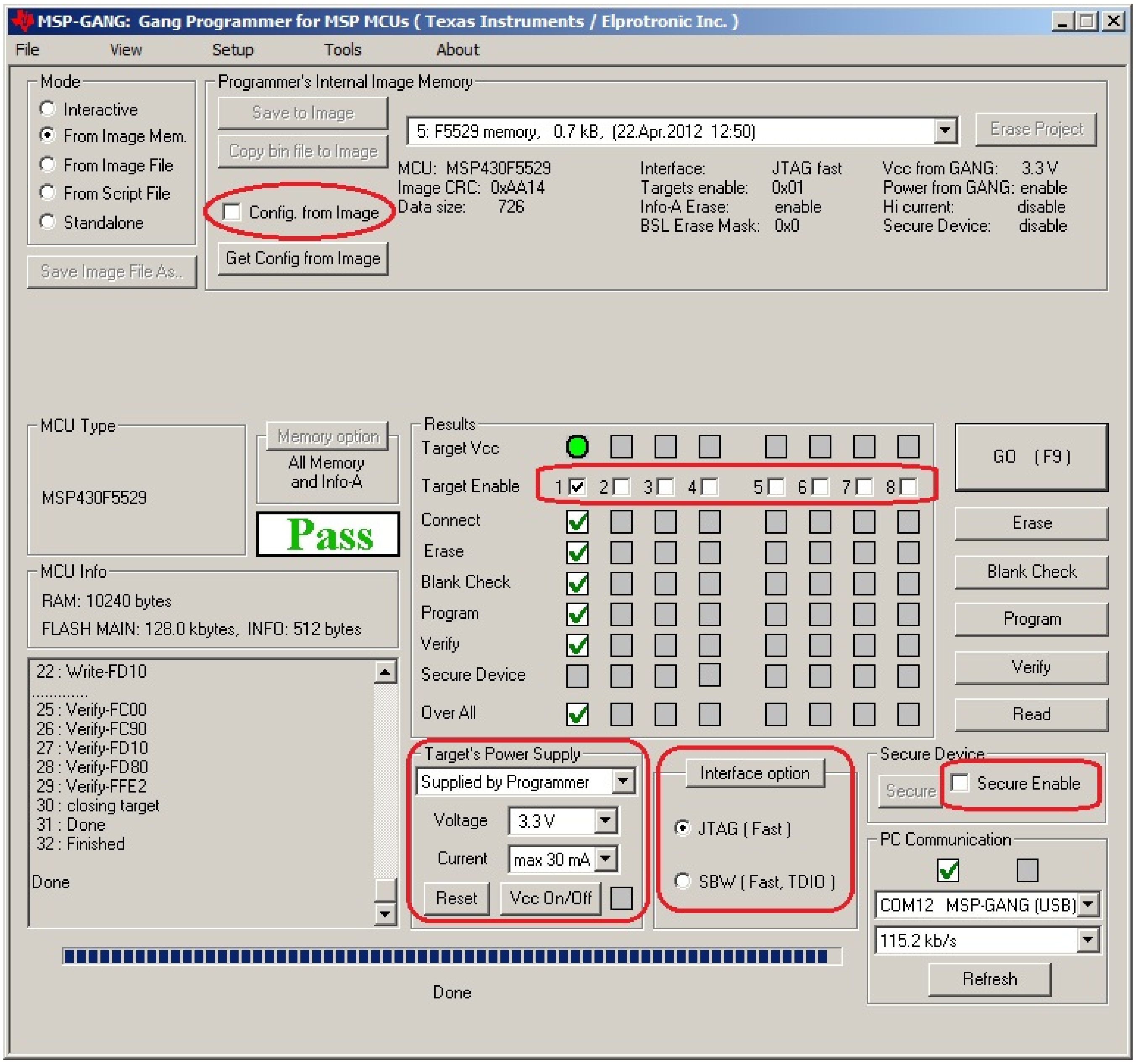
Done

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 16 different images can be selected from internal memory, or one image from each external SD-Card can be used.

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

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

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



Mode: Interactive; [✓ From Image Mem.]; From Image File; From Script File; Standalone

Programmer's Internal Image Memory: 5: F5529 memory, 0.7 kB, (22.Apr.2012 12:50)

Config. from Image: unchecked

```
MCU:  MSP430F5529      Interface:         JTAG fast   Vcc from GANG:    3.3 V
Image CRC:  0xAA14     Targets enable:    0x01        Power from GANG:  enable
Data size:     726     Info-A Erase:      enable      Hi current:       disable
                       BSL Erase Mask:    0x0         Secure Device:    disable
```

MCU Type: MSP430F5529

All Memory and Info-A

Pass

MCU Info: RAM: 10240 bytes; FLASH MAIN: 128.0 kbytes, INFO: 512 bytes

```
22 : Write-FD10
............
25 : Verify-FC00
26 : Verify-FC90
27 : Verify-FD10
28 : Verify-FD80
29 : Verify-FFE2
30 : closing target
31 : Done
32 : Finished

Done
```

Results:

	1	2	3	4	5	6	7	8
Target Vcc	●							
Target Enable	✓							
Connect	✓							
Erase	✓							
Blank Check	✓							
Program	✓							
Verify	✓							
Secure Device								
Over All	✓							

Target's Power Supply: Supplied by Programmer; Voltage 3.3 V; Current max 30 mA

Interface option: [✓ JTAG (Fast)]; SBW (Fast, TDIO)

Secure Device: Secure Enable (unchecked)

PC Communication: COM12 MSP-GANG (USB); 115.2 kb/s

Done

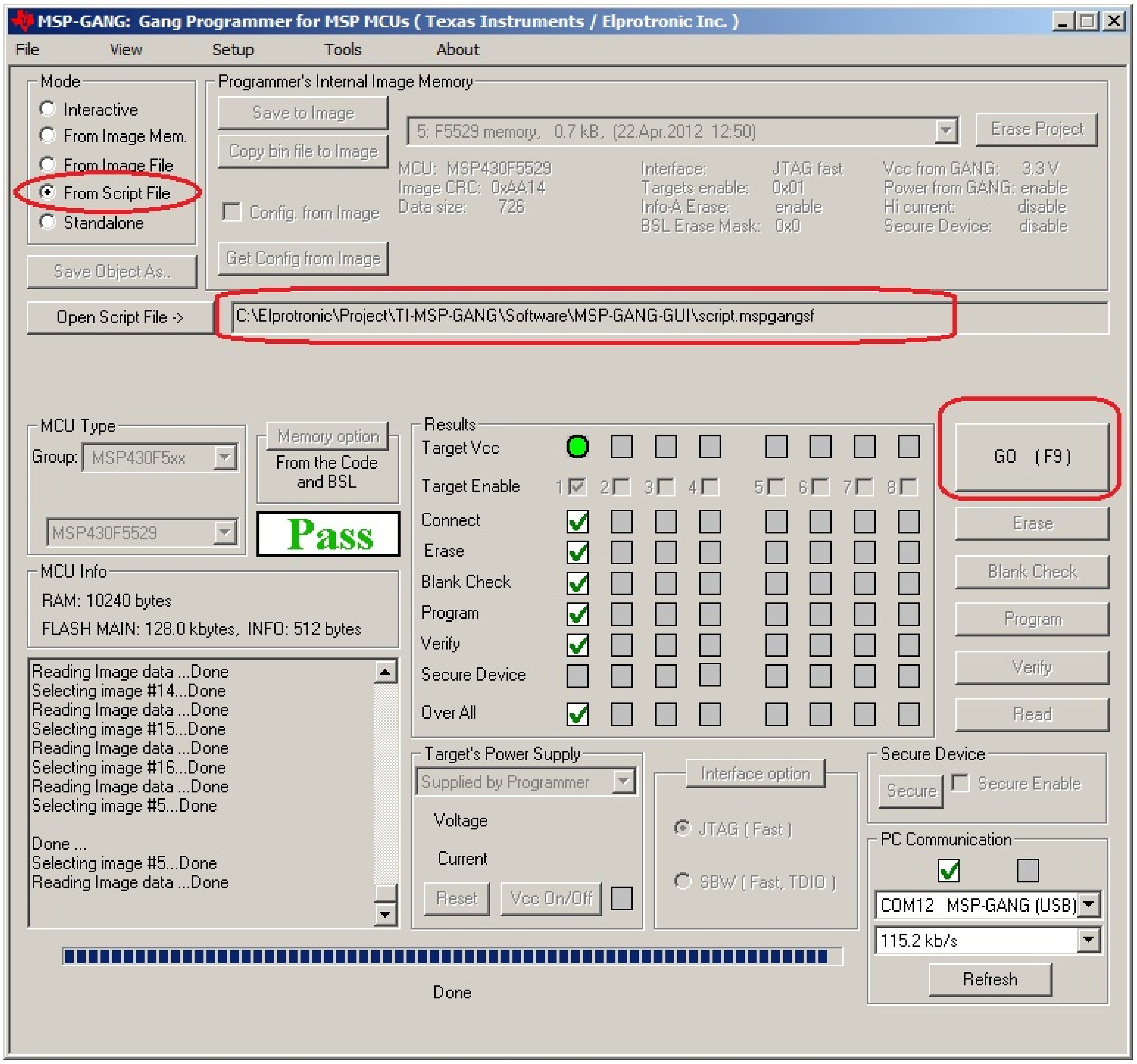
NOTE: This figure shows the From Image Mode (top left corner). The Config. from Image option is disabled in this example, allowing the user to change various but not all configuration settings from the image. The configuration options that can be changed are highlighted in red. One of the options that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

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

2.1.3 Programming From Script

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

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



Mode: [✓ From Script File]

5: F5529 memory, 0.7 kB, (22.Apr.2012 12:50)

MCU:	MSP430F5529	Interface:	JTAG fast	Vcc from GANG:	3.3 V
Image CRC:	0xAA14	Targets enable:	0x01	Power from GANG:	enable
Data size:	726	Info-A Erase:	enable	Hi current:	disable
		BSL Erase Mask:	0x0	Secure Device:	disable

C:\Elprotronic\Project\TI-MSP-GANG\Software\MSP-GANG-GUI\script.mspgangsf

Group: MSP430F5xx

MSP430F5529

Pass

RAM: 10240 bytes

FLASH MAIN: 128.0 kbytes, INFO: 512 bytes

Target Enable: 1 ✓ 2 3 4 5 6 7 8

```
Reading Image data ...Done
Selecting image #14...Done
Reading Image data ...Done
Selecting image #15...Done
Reading Image data ...Done
Selecting image #16...Done
Reading Image data ...Done
Selecting image #5...Done

Done ...
Selecting image #5...Done
Reading Image data ...Done
```

Supplied by Programmer

COM12 MSP-GANG (USB)

115.2 kb/s

Done

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

Figure 2-7. Main 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 dialogue screen.

F_GO – Perform GO function from main dialogue screen.

F_ERASEFLASH – Perform ERASE FLASH function from main dialogue screen.

F_BLANKCHECK – Perform BLANK CHECK function from main dialogue screen.

F_WRITEFLASH – Perform WRITE FLASH function from main dialogue screen.

F_VERIFYFLASH – Perform VERIFY FLASH function from main dialogue screen.

F_BLOWFUSE – Perform BLOW FUSE function from main dialogue screen.

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

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

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

F_STANDALONEMODE – Switch to Standalone mode.

F_INTERACTIVEMODE – Switch to Interactive mode.

TRACEOFF – Disable tracing.

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

END – End of script.

The following example script executes this sequence of commands:

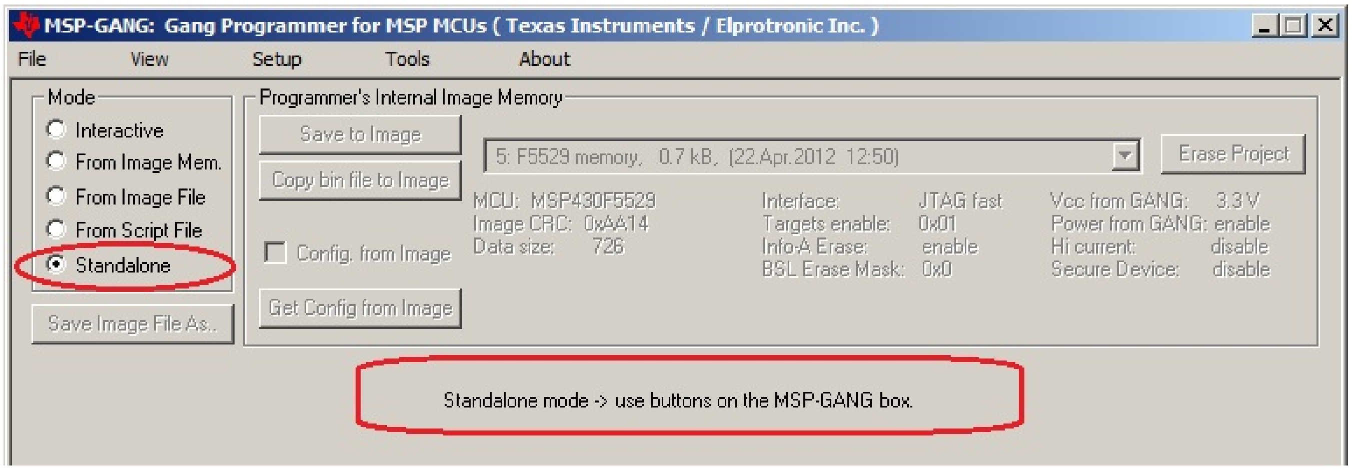
1. Label START is created.
2. V_{CC} from programmer to target device is turned OFF.
3. Message box notifies the user of V_{CC} setting and asks for permission to proceed with buttons OK and CANCEL. The program halts here until a button is pressed.
4. If CANCEL was pressed then GOTO finish label (ends the script).
5. If CANCEL was not pressed (in this case this implies that OK was pressed) then load configuration file test-A.g430cfg to the MSP Gang Programmer. Configuration file test-A.cfg should be prepared before running this script using Interactive mode.
6. Message box asks the user to proceed. The program halts until OK is pressed.
7. The MSP Gang Programmer programs the target device using the GO function.
8. Message box asks the user if the test succeeded giving a YES or NO choice.
9. If NO was pressed then GOTO START label (start of script).
10. If NO was not pressed (in this case this implies that YES was pressed) then load configuration file finalcode.g430cfg to the MSP Gang Programmer.
11. The MSP Gang Programmer programs the target device using the GO function. The new configuration changes the code file.
12. Script jumps to the beginning using GOTO START. This can be used to wait for the next target device to be connected.
13. Label finish is created.
14. Script ends.

```
;======================================================
; Script file - demo program
;------------------------------------------------------
>START
F_VCCOFF
MESSAGEBOX OKCANCEL
"VCC if OFF now. Connect the test board."
"When ready press the button:"
" "
"OK - to test the board"
"CANCEL - to exit from program"

IF BUTTONCANCEL GOTO finish

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

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.mspgangproj

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



Mode: ( ) Interactive ( ) From Image Mem. ( ) From Image File ( ) From Script File [✓ Standalone]

5: F5529 memory, 0.7 kB, (22.Apr.2012 12:50)

MCU: MSP430F5529
Image CRC: 0xAA14
Data size: 726

Interface: JTAG fast
Targets enable: 0x01
Info-A Erase: enable
BSL Erase Mask: 0x0

Vcc from GANG: 3.3 V
Power from GANG: enable
Hi current: disable
Secure Device: disable

Standalone mode -> use buttons on the MSP-GANG box.

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

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

After images have been download to the internal memory or after an SD card with a valid image is connected to the MSP Gang Programmer, proceed with programming in Standalone mode. Use the arrow buttons (up and down) and the enter button to select a desired image for programming. A description of the selected image is displayed on the bottom line, and it is the same description that was created in the GUI when the Save Image button was pressed (see Figure 2-9).

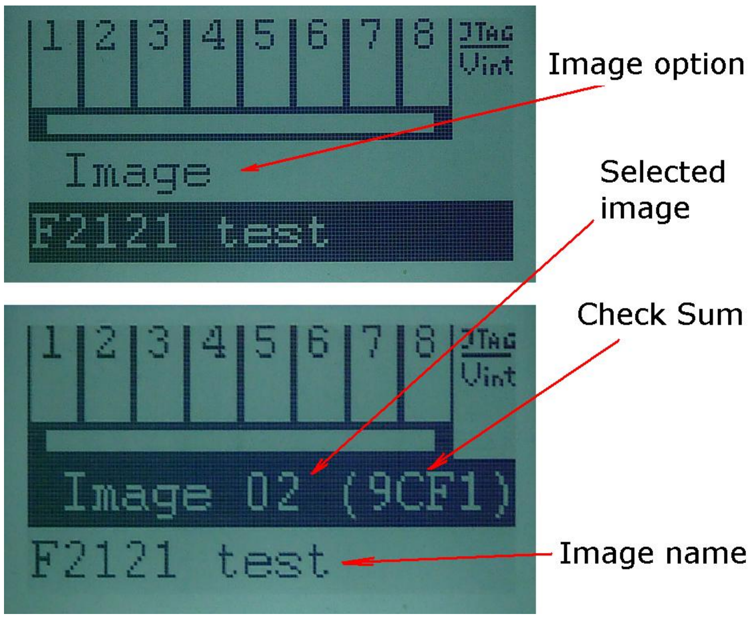


Figure 2-9. Image Option

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

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

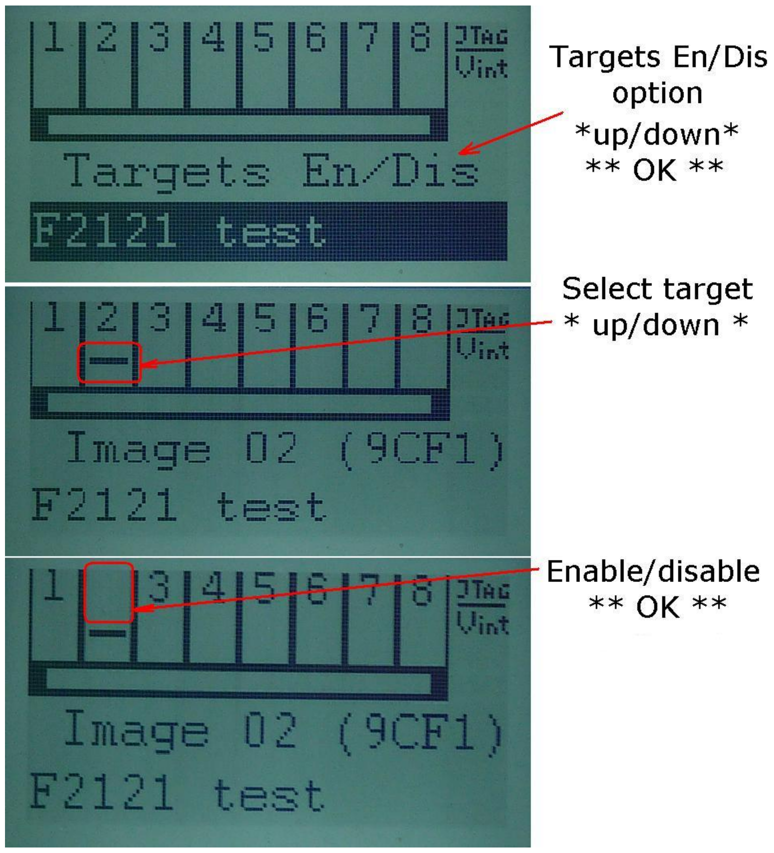


Figure 2-10. Target Enable or Disable Option

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

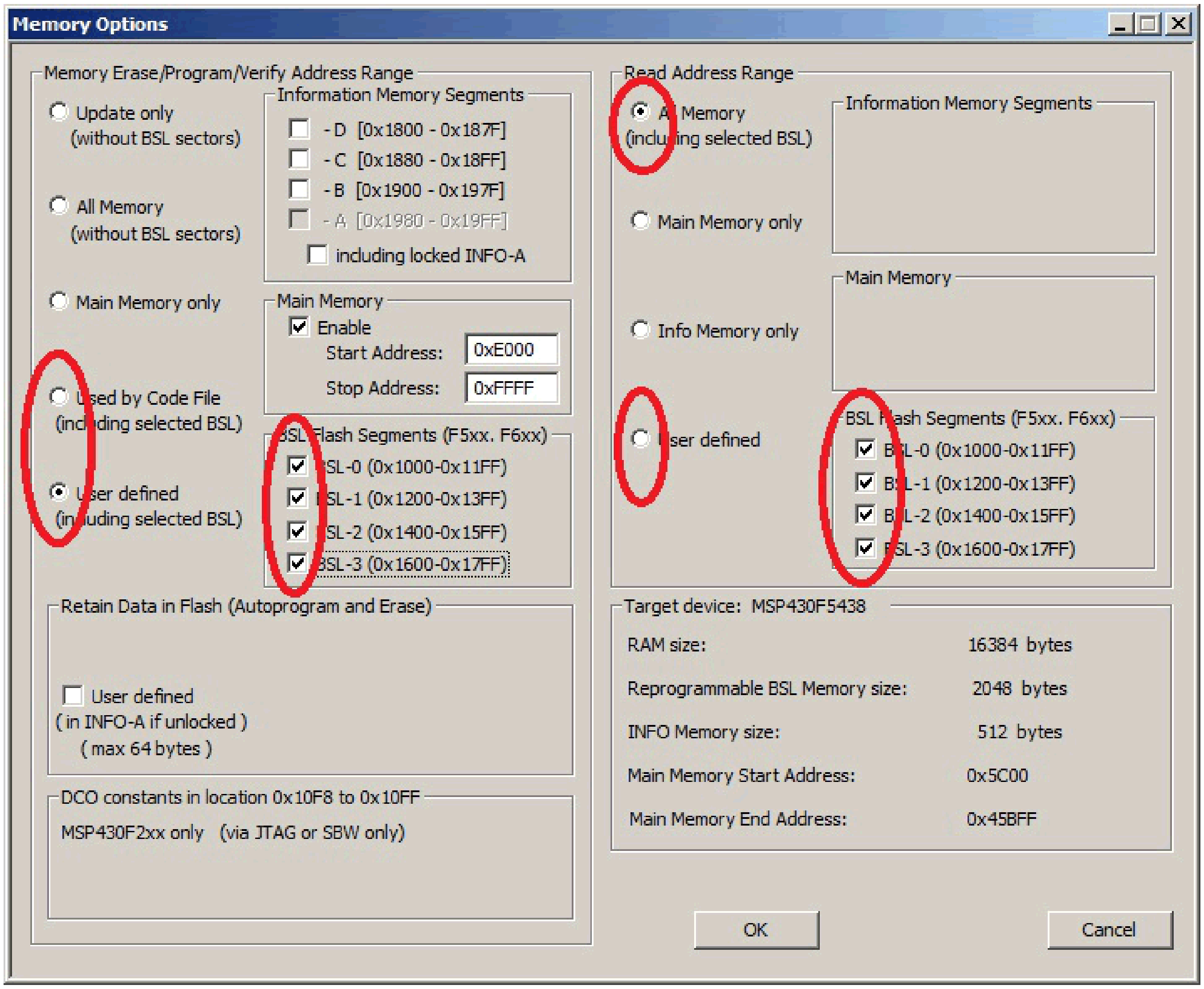
   NOTE: Even Number of Bytes

   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 2-11. Contents of BSL sectors can be read even when the All Memory option is selected.


Memory Options

Memory Erase/Program/Verify Address Range
- Update only (without BSL sectors)
- All Memory (without BSL sectors)
- Main Memory only
- Used by Code File (including selected BSL)
- [✓ User defined (including selected BSL)]

Information Memory Segments
- [ ] - D [0x1800 - 0x187F]
- [ ] - C [0x1880 - 0x18FF]
- [ ] - B [0x1900 - 0x197F]
- [ ] - A [0x1980 - 0x19FF]
- [ ] including locked INFO-A

Main Memory
- [✓] Enable
- Start Address: 0xE000
- Stop Address: 0xFFFF

BSL Flash Segments (F5xx. F6xx)
- [✓] BSL-0 (0x1000-0x11FF)
- [✓] BSL-1 (0x1200-0x13FF)
- [✓] BSL-2 (0x1400-0x15FF)
- [✓] BSL-3 (0x1600-0x17FF)

Retain Data in Flash (Autoprogram and Erase)
- [ ] User defined (in INFO-A if unlocked) (max 64 bytes)

DCO constants in location 0x10F8 to 0x10FF
MSP430F2xx only (via JTAG or SBW only)

Read Address Range
- [✓ All Memory (including selected BSL)]
- Main Memory only
- Info Memory only
- User defined

Information Memory Segments

Main Memory

BSL Flash Segments (F5xx. F6xx)
- [✓] BSL-0 (0x1000-0x11FF)
- [✓] BSL-1 (0x1200-0x13FF)
- [✓] BSL-2 (0x1400-0x15FF)
- [✓] BSL-3 (0x1600-0x17FF)

Target device: MSP430F5438

RAM size:	16384 bytes
Reprogrammable BSL Memory size:	2048 bytes
INFO Memory size:	512 bytes
Main Memory Start Address:	0x5C00
Main Memory End Address:	0x45BFF

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

Figure 2-11. Memory Options, BSL Sectors Selected

2.1.6 Creating and Using Images

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, 16 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 2-12). 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 16 different images can be saved internally, or one image can be saved on an SD-Card. Each image can be selected at any time to program the target devices. The MSP Gang Programmer also allows the image to be saved in a file, either to be saved on an SD-Card or to be sent to a customer. When the code file and configuration are ready to be saved, press the Save Image button to save to MSP Gang Programmer internal memory, or the Save Image to file button to save to a file.

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

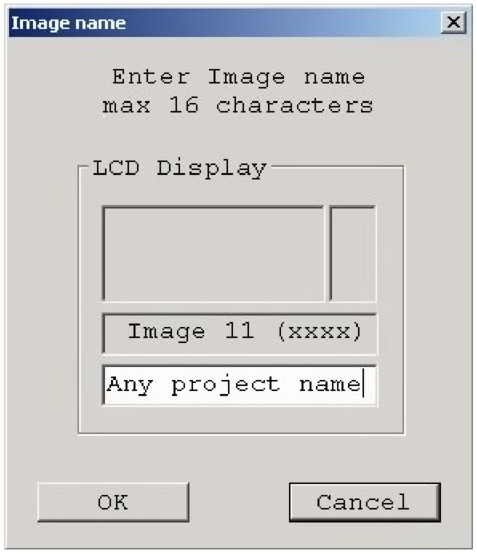


Image name

Enter Image name
max 16 characters

LCD Display

Image 11 (xxxx)

Any project name

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

Figure 2-12. Image Name Configuration Screen

The screen shown in Figure 2-13 allows the user configure what type of security is used to protect the image file. Three options are available; however, for all three options the contents of the code file are always encrypted and cannot be read.

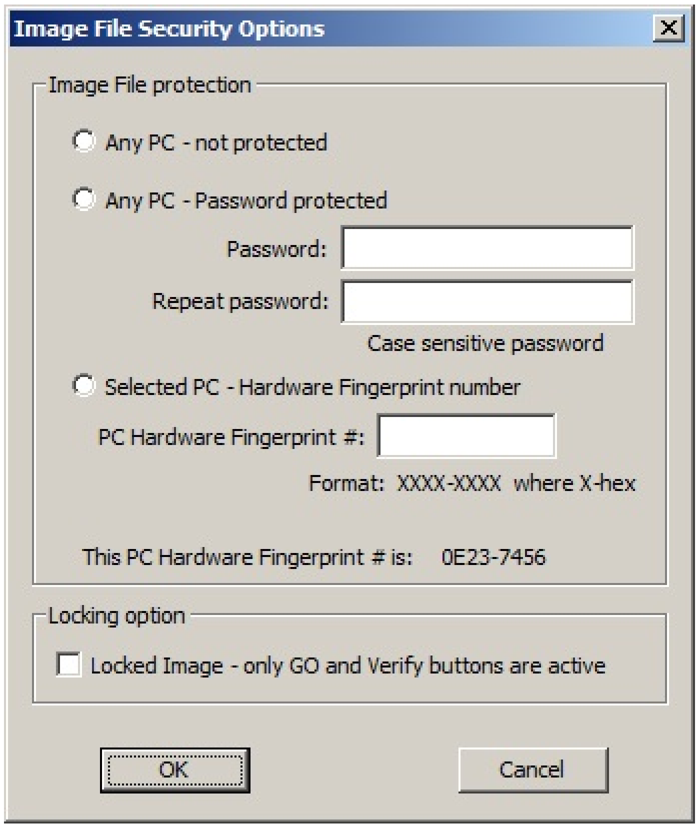


Image File Security Options

Image File protection

Any PC - not protected

Any PC - Password protected

Password:

Repeat password:

Case sensitive password

Selected PC - Hardware Fingerprint number

PC Hardware Fingerprint #:

Format: XXXX-XXXX where X-hex

This PC Hardware Fingerprint # is: 0E23-7456

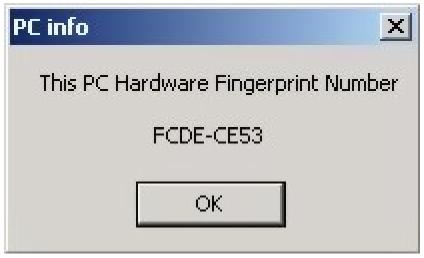
Locking option

Locked Image - only GO and Verify buttons are active

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

Figure 2-13. Image File Security Options

1. **Any PC**– Configuration can be opened on any computer using MSP Gang Programmer software. It can be used for programming only.
2. **Any PC - Password protected** – Configuration can be opened on any computer using the MSP Gang Programmer software, but only after the desired password has been entered.
3. **Selected PC - Hardware Fingerprint number** – Image can be opened only on the dedicated computer with the same hardware fingerprint number as the number entered in the edited line above. Figure 2-14 shows a window with the hardware fingerprint number. An example usage scenario would involve calling an intended user to provide the hardware fingerprint number of their computer and entering it within this configuration window. This restricts opening this image to only the dedicated computer running MSP Gang Programmer software.



This PC Hardware Fingerprint Number

FCDE-CE53

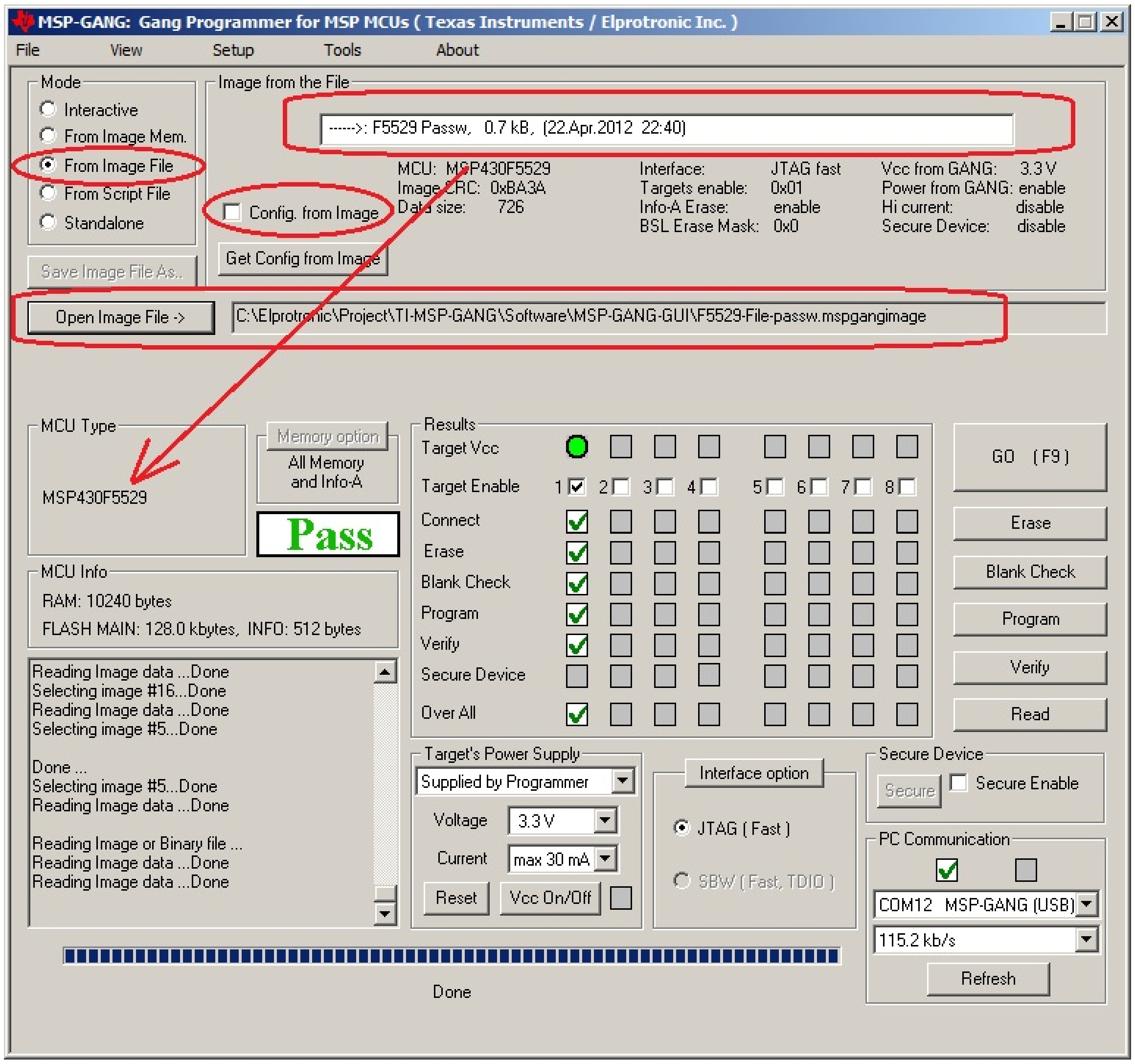
NOTE: The fingerprint can be used to secure the project where, for example, only a computer with a matching hardware fingerprint can be used to view and edit the project.

Figure 2-14. Hardware Fingerprint of Computer in Use

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

2.1.7 Programming From Image File

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



Mode: [✓ From Image File]

Image from the File: ------>: F5529 Passw, 0.7 kB, (22.Apr.2012 22:40)

[ ] Config. from Image

MCU: MSP430F5529 Image CRC: 0xBA3A Data size: 726

Interface: JTAG fast Targets enable: 0x01 Info-A Erase: enable BSL Erase Mask: 0x0

Vcc from GANG: 3.3 V Power from GANG: enable Hi current: disable Secure Device: disable

C:\Elprotronic\Project\TI-MSP-GANG\Software\MSP-GANG-GUI\F5529-File-passw.mspgangimage

MCU Type: MSP430F5529

Memory option: All Memory and Info-A

MCU Info: RAM: 10240 bytes FLASH MAIN: 128.0 kbytes, INFO: 512 bytes

```
Reading Image data ...Done
Selecting image #16...Done
Reading Image data ...Done
Selecting image #5...Done

Done ...
Selecting image #5...Done
Reading Image data ...Done

Reading Image or Binary file ...
Reading Image data ...Done
Reading Image data ...Done
```

Results	1	2	3	4	5	6	7	8
Target Vcc	●							
Target Enable	✓							
Connect	✓							
Erase	✓							
Blank Check	✓							
Program	✓							
Verify	✓							
Secure Device								
Over All	✓							

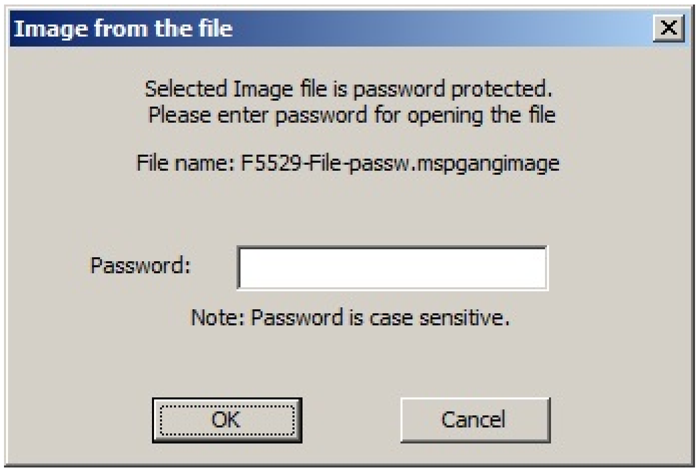
Target's Power Supply: Supplied by Programmer; Voltage: 3.3 V; Current: max 30 mA

Interface option: [✓ JTAG (Fast)] SBW (Fast, TDIO)

Secure Device: [ ] Secure Enable

PC Communication: COM12 MSP-GANG (USB); 115.2 kb/s

Figure 2-15. Programming From Image File



Selected Image file is password protected.
Please enter password for opening the file

File name: F5529-File-passw.mspgangimage

Password:

Note: Password is case sensitive.

Figure 2-16. Password for Image File

2.1.8 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 2.1.6 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.

2.1.9 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

2.1.10 Checksum Calculation

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

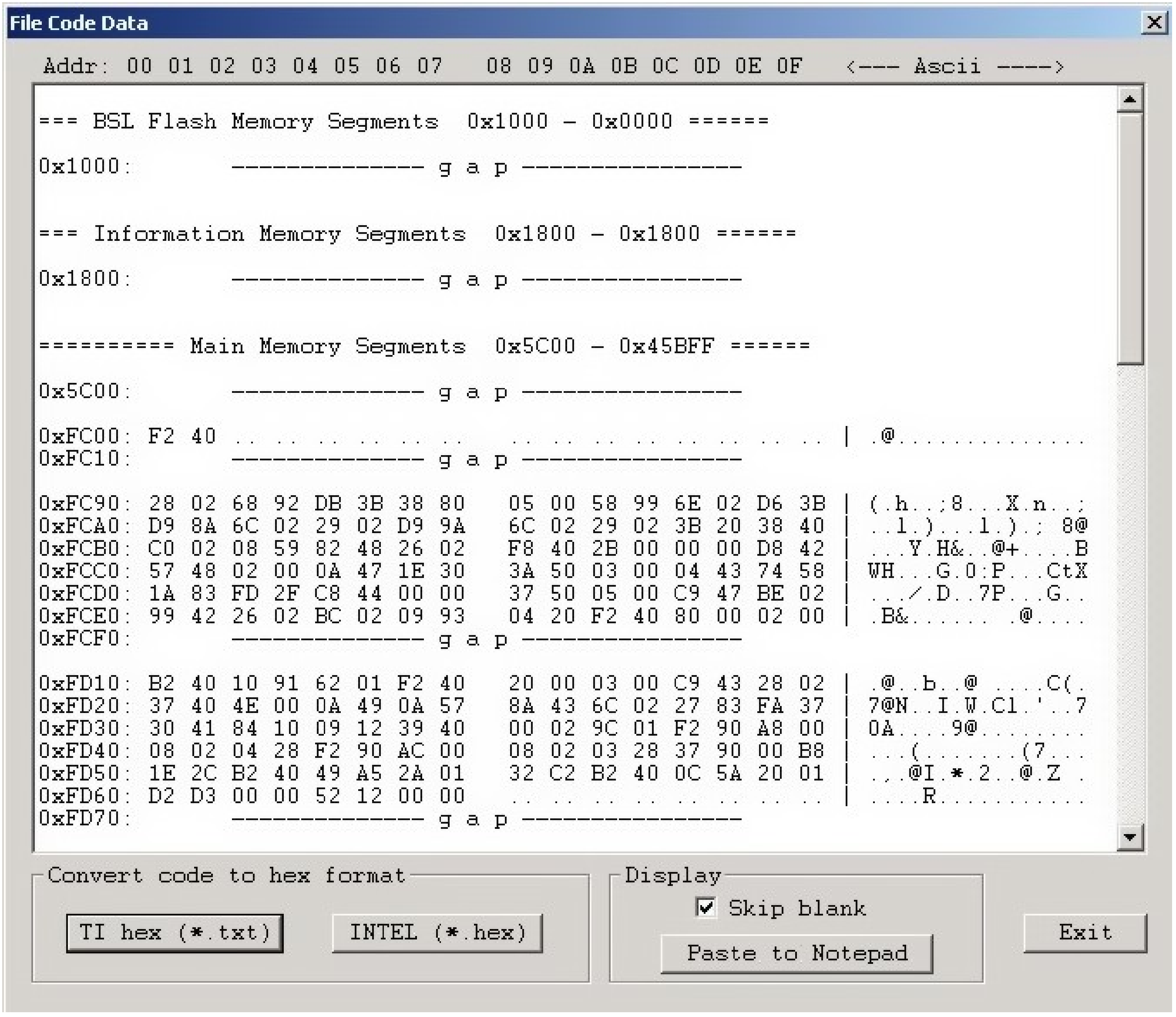
The CS is calculated as shown below:

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

2.2 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 2-17, 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 MSP430.



```
Addr: 00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F   <--- Ascii ---->

=== BSL Flash Memory Segments  0x1000 - 0x0000 ======

0x1000:         --------------- g a p -----------------


=== Information Memory Segments  0x1800 - 0x1800 ======

0x1800:         --------------- g a p -----------------


========== Main Memory Segments  0x5C00 - 0x45BFF ======

0x5C00:         --------------- g a p -----------------

0xFC00: F2 40 .. .. .. .. .. ..   .. .. .. .. .. .. .. .. | .@..............
0xFC10:         --------------- g a p -----------------

0xFC90: 28 02 68 92 DB 3B 38 80   05 00 58 99 6E 02 D6 3B | (.h..;8...X.n..;
0xFCA0: D9 8A 6C 02 29 02 D9 9A   6C 02 29 02 3B 20 38 40 | ..l.)...l.).; 8@
0xFCB0: C0 02 08 59 82 48 26 02   F8 40 2B 00 00 00 D8 42 | ...Y.H&..@+....B
0xFCC0: 57 48 02 00 0A 47 1E 30   3A 50 03 00 04 43 74 58 | WH...G.0:P...CtX
0xFCD0: 1A 83 FD 2F C8 44 00 00   37 50 05 00 C9 47 BE 02 | .../.D..7P...G..
0xFCE0: 99 42 26 02 BC 02 09 93   04 20 F2 40 80 00 02 00 | .B&...... .@....
0xFCF0:         --------------- g a p -----------------

0xFD10: B2 40 10 91 62 01 F2 40   20 00 03 00 C9 43 28 02 | .@..b..@ ....C(.
0xFD20: 37 40 4E 00 0A 49 0A 57   8A 43 6C 02 27 83 FA 37 | 7@N..I.W.Cl.'..7
0xFD30: 30 41 84 10 09 12 39 40   00 02 9C 01 F2 90 A8 00 | 0A....9@........
0xFD40: 08 02 04 28 F2 90 AC 00   08 02 03 28 37 90 00 B8 | ...(.......(7...
0xFD50: 1E 2C B2 40 49 A5 2A 01   32 C2 B2 40 0C 5A 20 01 | .,.@I.*.2..@.Z .
0xFD60: D2 D3 00 00 52 12 00 00   .. .. .. .. .. .. .. .. | ....R...........
0xFD70:         --------------- g a p -----------------
```

[✓ Skip blank]

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

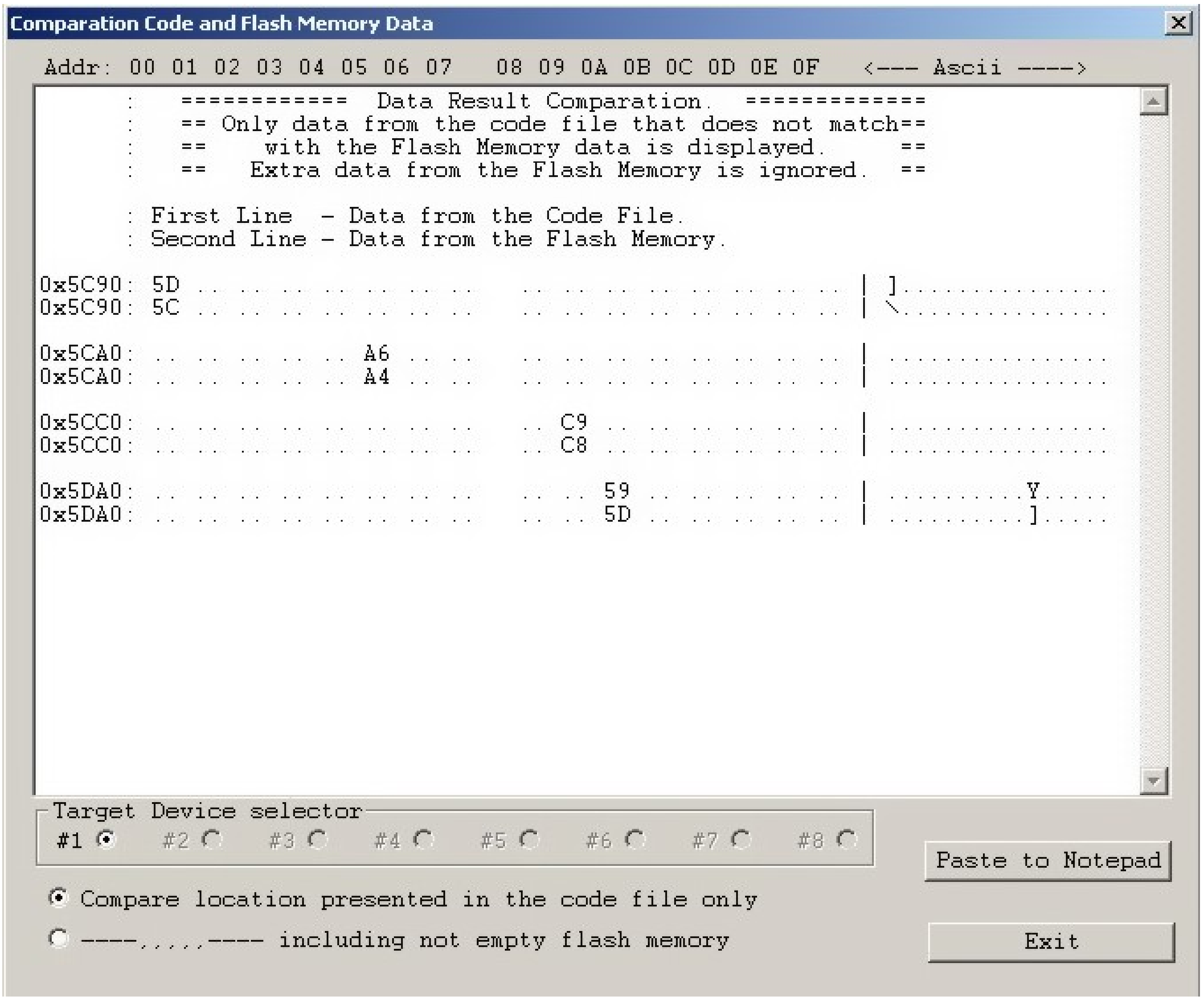
Figure 2-17. Code File Data

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 2-17.

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 2-18.

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.



Comparation Code and Flash Memory Data

```
Addr: 00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F   <--- Ascii ---->
       :   ============  Data Result Comparation.  =============
       :   == Only data from the code file that does not match==
       :   ==    with the Flash Memory data is displayed.     ==
       :   ==   Extra data from the Flash Memory is ignored.  ==

       : First Line  - Data from the Code File.
       : Second Line - Data from the Flash Memory.

0x5C90: 5D .. .. .. .. .. .. ..   .. .. .. .. .. .. .. .. | ]...............
0x5C90: 5C .. .. .. .. .. .. ..   .. .. .. .. .. .. .. .. | \...............

0x5CA0: .. .. .. .. .. A6 .. ..   .. .. .. .. .. .. .. .. | ................
0x5CA0: .. .. .. .. .. A4 .. ..   .. .. .. .. .. .. .. .. | ................

0x5CC0: .. .. .. .. .. .. .. ..   .. C9 .. .. .. .. .. .. | ................
0x5CC0: .. .. .. .. .. .. .. ..   .. C8 .. .. .. .. .. .. | ................

0x5DA0: .. .. .. .. .. .. .. ..   .. .. 59 .. .. .. .. .. | ..........Y.....
0x5DA0: .. .. .. .. .. .. .. ..   .. .. 5D .. .. .. .. .. | ..........].....
```

[✓ Compare location presented in the code file only]

----,.....---- including not empty flash memory

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 2-18. 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 is 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 512 kbytes in one image), the MSP Gang Programmer gives the user significant flexibility to perform custom programming procedures. If for any reason the code files and task scripts require more than 512 kbytes of memory, then the next image memory can be taken and combined with the first one for one larger image block (1Mbyte or more). The MSP Gang Programmer has internal flash memory of 8Mbyte that can, if desired, all be used to form one image with a memory footprint of 8Mbyte.

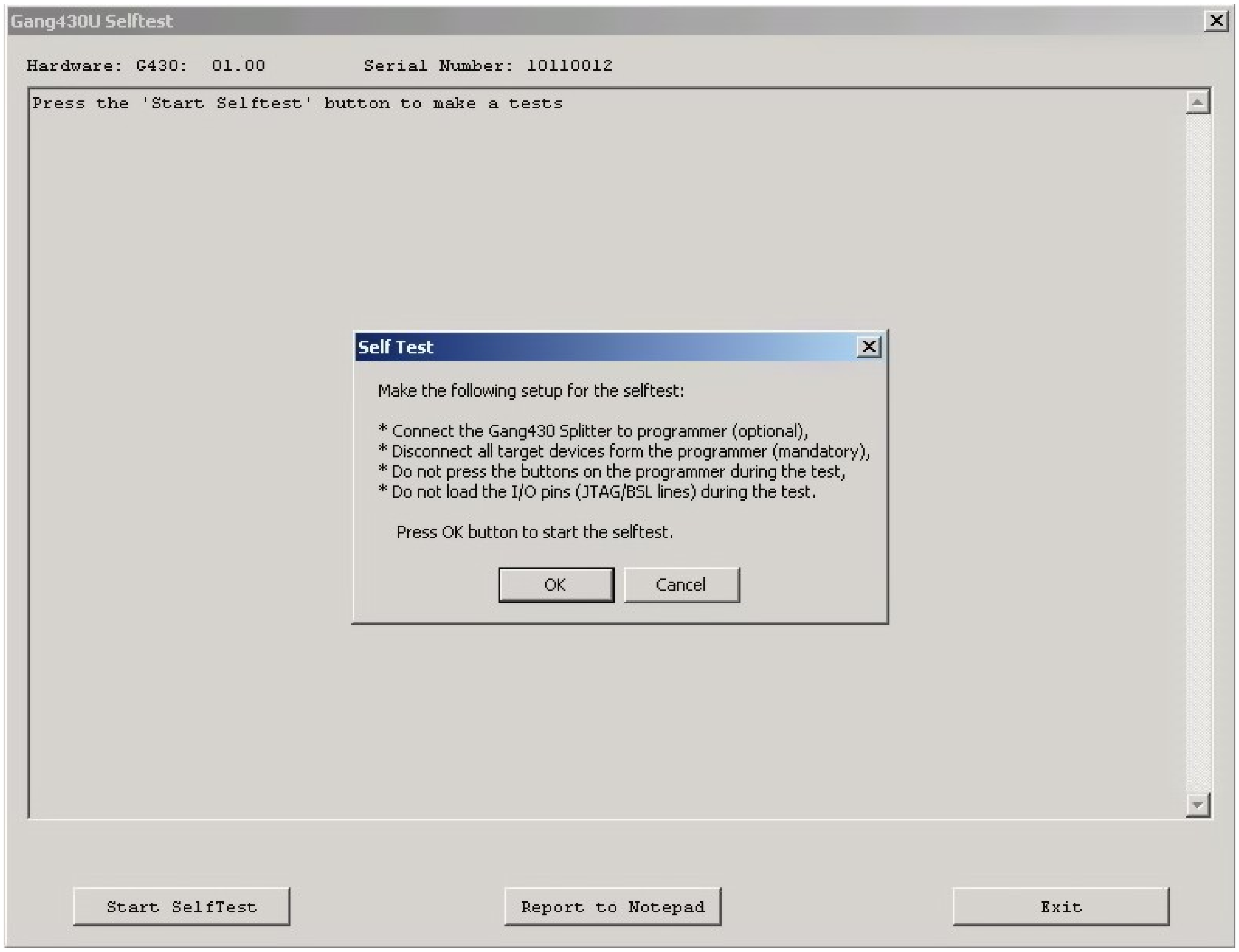
Error messages are displayed similarly to status messages, however, programming is terminated if the error is related to all target devices. Subsequently, if the problem is resolved or the faulty target device is disabled, then the programming procedure can be restarted to complete the programming process.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

```
==== 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
ERROR # 018   - Script CRC
ERROR # 019   - Exceed script no
ERROR # 020   - Script command ?
ERROR # 021   - MCU Fetch synch
ERROR # 022   - CPU JTAG synch
ERROR # 023   - MCU device init.
ERROR # 024   - RAM FW download
ERROR # 025   - Blank check err
ERROR # 026   - Read verify err
ERROR # 027   - Flash write err
ERROR # 028   - Image FL WR init
ERROR # 029   - Image Flash lock
ERROR # 030   - Invalid Script T
ERROR # 031   - Size too high
ERROR # 032   - Used wrong MCU
ERROR # 033   - IR Interrupted
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 Cal.Freq. out of range
ERROR # 040   - DCO Cal.Failed
ERROR # 041   - Gang Flash write error
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   - Verification Error
ERROR # 059   - Rx data error
ERROR # 060   - Secure Vpp low
ERROR # 061   - Secure Key Error
ERROR # 062   - Secure Device Er
ERROR # 063   - Target not open
ERROR # 064   - Wrong index
ERROR # 065   - BSL Rx Timeout
ERROR # 066   - BSL firmware download error
ERROR # 067   - Wrong command
ERROR # 068   - FBSL initialization error
ERROR # 069   - MSP-
GANG 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.mspgangcfg'!
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   - Extention 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. 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 # 999   - Invalid error number !
```

2.4 Self Test

The MSP Gang Programmer Self Test program allows to test most of the hardware for correctness. Connect the programmer to a computer running MSP Gang Programmer software. If the user is utilizing a Gang Splitter, then connect it to MSP Gang Programmer hardware (this allows the Self Test to find short circuits in the Gang Splitter). Disconnect all target devices, because any connected devices can modify the test results and make them invalid.

Activate the Self Test by choosing the Tools→Self Test option from the drop-down menu. Press the Start Self Test button, as shown in Figure 2-19, to begin. If the Self Test reports any problems then it is advisable to send the test report to TI technical support for assistance.



Hardware: G430: 01.00 Serial Number: 10110012

Press the 'Start Selftest' button to make a tests

Make the following setup for the selftest:

* Connect the Gang430 Splitter to programmer (optional),
* Disconnect all target devices form the programmer (mandatory),
* Do not press the buttons on the programmer during the test,
* Do not load the I/O pins (JTAG/BSL lines) during the test.

Press OK button to start the selftest.

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

Figure 2-19. 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 <<
```

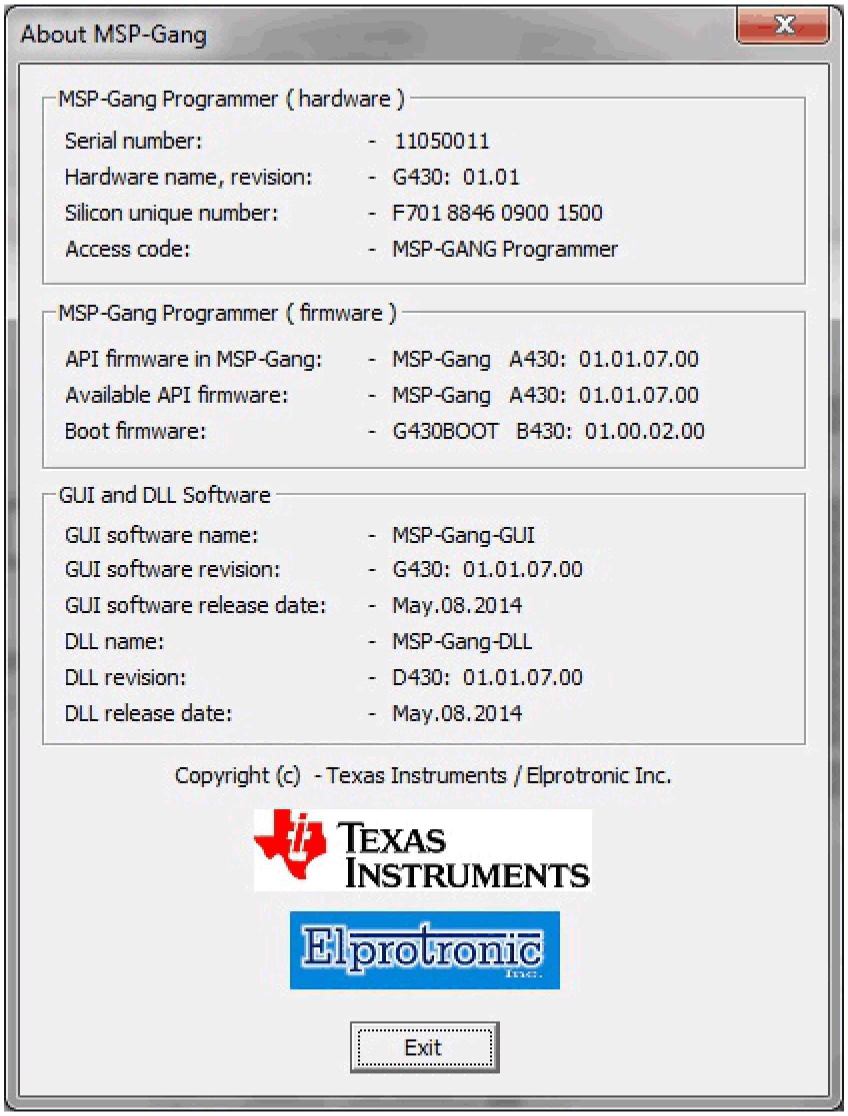
```
68: Vcc Target-5 ( \#3 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
69: Vcc Target-6 ( \#3 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
70: Vcc Target-7 ( \#3 ON ) 0.00 V ( 0.00 to 0.50) Result: 0.25 V ... >> OK <<
71: Vcc Target-8 ( \#3 ON ) 3.30 V ( 0.00 to 0.50) Result: 0.25 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 2-20.



MSP-Gang Programmer (hardware)

Serial number:	-	11050011
Hardware name, revision:	-	G430: 01.01
Silicon unique number:	-	F701 8846 0900 1500
Access code:	-	MSP-GANG Programmer

MSP-Gang Programmer (firmware)

API firmware in MSP-Gang:	-	MSP-Gang A430: 01.01.07.00
Available API firmware:	-	MSP-Gang A430: 01.01.07.00
Boot firmware:	-	G430BOOT B430: 01.00.02.00

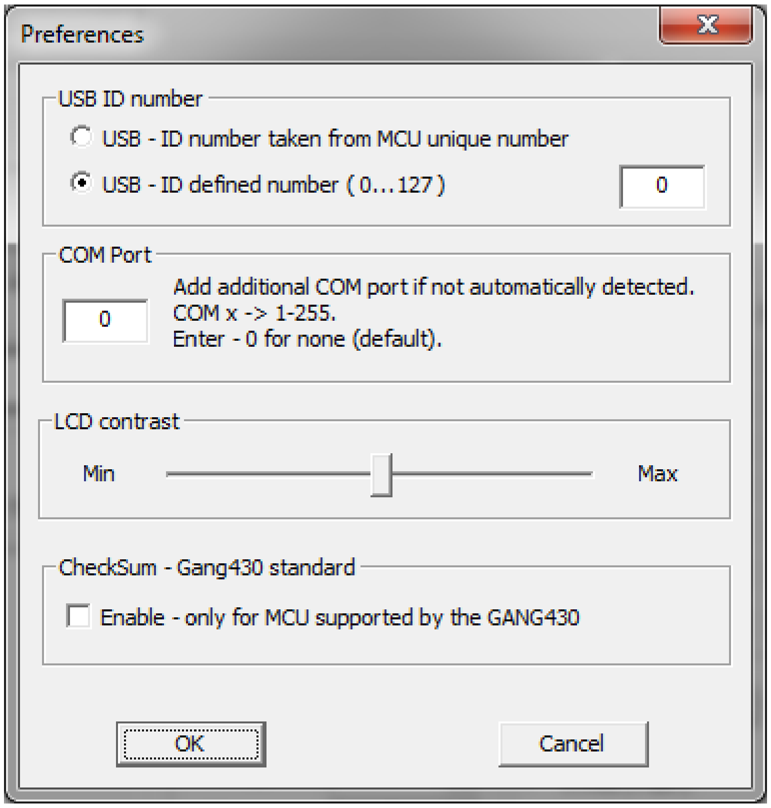
GUI and DLL Software

GUI software name:	-	MSP-Gang-GUI
GUI software revision:	-	G430: 01.01.07.00
GUI software release date:	-	May.08.2014
DLL name:	-	MSP-Gang-DLL
DLL revision:	-	D430: 01.01.07.00
DLL release date:	-	May.08.2014

Copyright (c) - Texas Instruments / Elprotronic Inc.

Figure 2-20. Information About the MSP Gang Programmer

2.6 Preferences


Preferences

USB ID number

USB - ID number taken from MCU unique number

[✓ USB - ID defined number (0...127)] 0

COM Port

0 Add additional COM port if not automatically detected.
COM x -> 1-255.
Enter - 0 for none (default).

LCD contrast

Min Max

CheckSum - Gang430 standard

Enable - only for MCU supported by the GANG430

Figure 2-21. 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[1]

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

[1] Programming speed and verify speed without startup procedures (access to target device).

Table 2-2. Benchmark Results – MSP430F5438A, 250kB Code, Mode: From Image[1]

	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

[1] 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[1]

	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

[1] Programming speed and verify speed without startup procedures (access to target device).

2.7.2 Benchmarks for MSP430FR57xx

Table 2-4. Benchmark Results – MSP430FR5738, 15kB Code, Mode: From Image[1]

	Erase, Blank Check, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)
JTAG Fast	0.9	0.3	35
JTAG Med	1.4	0.3	18
JTAG Slow	2.5	0.4	8.5
SBW Fast	2.9	0.4	7.5
SBW Med	4.2	0.5	4.8
SBW Slow	7.8	0.7	2.5

[1] Programming speed without startup procedures (access to target device).

Table 2-5. Benchmark Results – MSP430FR5738, 15kB Code, Mode: Interactive, Communication by USB[1]

	Erase, Blank Check, Program, and Verify (s)	Verify (s)	Programming Speed (kB/s)
JTAG Fast	1.2	0.3	22
JTAG Med	1.7	0.3	15
JTAG Slow	2.8	0.4	8
SBW Fast	3.3	0.4	6.5
SBW Med	4.7	0.5	4.5
SBW Slow	8.5	0.7	2.3

[1] 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[1]

	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

[1] Programming speed without startup procedures (access to target device).

Table 2-7. Benchmark Results – MSP430F2619, 120kB Code, Mode: Interactive, Communication by USB[1]

	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

[1] 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
- Selftest

3.2 Firmware Interface Protocol

The MSP Gang Programmer supports a UART communication protocol at baud rates from 9.6 to 115.2 kbaud in half duplex mode. The default baud rate at startup is 9.6 kbaud. This allows for communication between the MSP Gang Programmer and devices that have a lower communication speed than the maximum 115.2 kbaud. It is recommended that after startup the communication speed be increased to the common maximum for both devices to enable faster communication. If the control device has a USB interface with a virtual COM port, then it is recommended to use USB for communication between the control device and the MSP Gang Programmer, because 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 MSP430 serial standard protocol (SSP) rules, extended with a preceding synchronization sequence (SS), as described in Section 3.3. The MSP Gang Programmer is considered the receiver in Table 3-1, which details the data frame for firmware commands. The redundancy of some parameters results from the adaptation of the SSP or to save boot ROM space.

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

- The first eight bytes (HDR through LH) are mandatory (– represents dummy data).
- Data bytes D1 to Dn are optional.
- Two bytes (CKL and CKH) for checksum are mandatory
- Acknowledge done by the MSP Gang Programmer is mandatory.

The following abbreviations are used in Table 3-1.

CMD	Command identification
R	Do not use this command. Used for internal communication.
T	Target number (1 to 8)
L1, L2	Number of bytes in AL through Dn. The valid values of these bytes are restricted as follows: L1 = L2, L1 < 255, L1 even.
A1, A2, A3	Block start address or erase (check) address or jump address LO or HI byte. The bytes are combined to generate a 24-bit word as follows: Address = A3 × 0x10000 + A2 × 0x100 + A1
LL, LH	Number of pure data bytes (maximum 250) or erase information LO or HI byte or block length of erase check (max is 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[(1)] [(2)]

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	n-6	00	D1	D2...Dn-6	CKL	CKH	ACK
Verify Image CRC	3E	45	08	08	A1	A2	A3	A4	LL	LH	D1	D2	CKL	CKH	ACK
Get Image Header	3E	47	06	06	A1	A2	00	00	n	00	-	-	CKL	CKH	-
Response–,,,--	80	0	n	n	D1	D2	-	-	D3	D4	D5	D6...Dn	CKL	CKH	
Read Gang Buffer	3E	49	4	4	T	0	-	-	n	0	-	-	CKL	CKH	-
Response–,,,--	80	0	n	n	D1	D2	D3	D4	D5	D6	D7	D8...Dn	CKL	CKH	
Write Gang Buffer	3E	4A	n+4	n+4	T	0	-	-	n	0	D1	D2...Dn	CKL	CKH	ACK

(1) All numbers are bytes in hexadecimal notation. ACK is sent by the MSP-GANG.
(2) PROMPT = 0x3E means data frame expected.

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

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

3.4.2 Checksum

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

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

or

CKL = INV [B1 XOR B3 XOR...XOR Bn–1]

CKH = INV [B2 XOR B4 XOR...XOR Bn]

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

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

3.5 Detailed Description of Commands

3.5.1 General

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

3.5.2 Commands Supported by the BOOT Loader

3.5.2.1 "Hello" 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 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 16 images, 0 through 15. The default image after power on is 0.

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

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

3.5.3.2 Main Process Command

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

The Main Process command begins the gang programming cycle, using the operations defined in the SD or internal image memory. The result of the command execution can be determined using the Get Progress Status command described in Section 3.5.2. It should be noted that the Main Progress commands responds as soon as the command is accepted with byte In Progress (0xB0). When the byte In Progress is received, then the Get Progress Status command should be 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

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 n-6 0 D1 ... Dn-6 CKL CKH
Rx -> ACK or NACK
```

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

3.5.4.7 Verify Image Check Sum Command

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

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

3.5.4.8 Read Image Header Command

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

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

3.5.4.9 Disable API Interrupts Command

```
Tx -> 3E 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).

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

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

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

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

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

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

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

D7 - spare

D8 - spare

Example 1

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

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

then

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

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

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

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

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

C:\Program Files\Texas Instruments\MSP-GANG\Examples\C_Applications_Wrapper
and
C:\Program Files\Texas Instruments\MSP-GANG\Examples\Cpp_Applications_Wrapper

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

Limitation

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

The MSP-GANG.dll is a Dynamic Link Library (DLL) that provides functions for controlling the MSP-GANG Programmer. The MSP-GANG.dll controls the Gang Programmer 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.

```
#define DBUFFER_SIZE            0x90000
#define JTAG_PASSW_LEN          0x80
#define BSL_PASSW_LEN           0x20
#define FLASH_END_ADDR          (DBUFFER_SIZE-1)
#define FLASH_BUF_LEN           DBUFFER_SIZE
#define GANG_SIZE               8

typedef struct
{
    BYTE    SourceCode[DBUFFER_SIZE];        //source code from the file
    BYTE    UsedCode[DBUFFER_SIZE];          //combined data (source code,
serialization etc)
    BYTE    CommonTx[DBUFFER_SIZE];          //data used to writing -
 the same data to all targets
    BYTE    GangTx[DBUFFER_SIZE][GANG_SIZE]; //selective data used to writing
    BYTE    GangRx[DBUFFER_SIZE][GANG_SIZE]; //data read from all targets
    BYTE    Tmp[DBUFFER_SIZE];
    BYTE    JTAG_Passsword[2][JTAG_PASSW_LEN];
    BYTE    BSL_Passsword[2][BSL_PASSW_LEN];
    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_Sp1[DBUFFER_SIZE];         //spare
    BYTE    Flag_Sp2[DBUFFER_SIZE];         //spare
    BYTE    Flag_Sp3[DBUFFER_SIZE];         //
    BYTE    Flag_JTAG_Passw[2][JTAG_PASSW_LEN]; // [0][..]-password from code file;
                                                // [1][..]-password from password file
    BYTE    Flag_BSL_Passw[2][BSL_PASSW_LEN];   // [0][..]-password from code file;
                                                // [1][..]-password from password file
} DATA_BUFFERS;
extern  DATA_BUFFERS  dat;
```

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

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

Syntax

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

4.2.2 MSPGANG_SetGangBuffer, MSPGANG_GetGangBuffer

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

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

MSPGANG_SetGangBuffer writes data to selected Buffer. MSPGANG_GetGangBuffer reads contents from the selected buffer.

Syntax

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

Arguments

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

Result

LONG	Error code

4.2.3 MSPGANG_GetDevice

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

Syntax

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

Arguments

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

Result

LONG	Error code

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

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

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

4.2.4 MSPGANG_LoadFirmware

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

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

Syntax

```
LONG MSPGANG_LoadFirmware(void)
```

4.2.5 MSPGANG_InitCom

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

Syntax

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

Arguments

char * lpszComPort	Name of the port
LONG lBaudRate	Baud rate

Result

LONG	Error code

4.2.6 MSPGANG_ReleaseCom

Release communications port

Syntax

```
LONG MSPGANG_ReleaseCom(void)
```

Arguments

None

Result

LONG	Error code

4.2.7 MSPGANG_GetErrorString

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

Syntax

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

Arguments

LONG lErrorNumber	Error number

Result

LPTSTR	Error string

4.2.8 MSPGANG_SelectBaudrate

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

Syntax

```
LONG MSPGANG_SelectBaudrate(LONG lBaud)
```

Arguments

LONG lBaud	Baud rate in bytes 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

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

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

Syntax

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

Arguments

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

Result

LONG	Error code

4.2.15 MSPGANG_Interactive_ReadTargets

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

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

Syntax

```
LONG MSPGANG_Interactive_ReadTargets(BYTE mask)
```

Arguments

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

Result

LONG	Error code

4.2.16 MSPGANG_Interactive_ReadBytes

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

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

Syntax

```
LONG MSPGANG_Interactive_ReadBytes(BYTE target_no, LONG addr, LONG size, BYTE *data)
```

Arguments

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

Result

LONG	Error code

4.2.17 MSPGANG_Interactive_WriteWord_to_RAM

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

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

Syntax

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

Arguments

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

Result

LONG	Error code

4.2.18 MSPGANG_Interactive_WriteByte_to_RAM

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

MSPGANG_Interactive_WriteByte_to_RAM writes one byte to any RAM or I/O location.

Syntax

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

Arguments

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

Result

LONG	Error code

4.2.19 MSPGANG_Interactive_WriteBytes_to_RAM

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

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

Syntax

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

Arguments

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

Result

LONG	Error code

4.2.20 MSPGANG_Interactive_WriteBytes_to_FLASH

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

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

Syntax

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

Arguments

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

Result

LONG	Error code

4.2.21 MSPGANG_Interactive_Copy_Gang_Buffer_to_RAM

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

MSPGANG_Interactive_Copy_Gang_Buffer_to_RAM writes 'size' number of bytes from the internal Gang_Buffer[8][128] to RAM - simultaneously to all active target devices. Data for each target can be different. Contents from Gang_Buffer[0][n] are written to target 1, contents from Gang_Buffer[1][n] are written to target 2, and contents from Gang_Buffer[7][n] are written to target 8.

Data in the Gang_Buffer should be prepared and send to MSP-GANG first. See MSPGANG_GetGangBuffer and MSPGANG_SetGangBuffer functions for details.

Syntax

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

Arguments

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

Result

LONG	Error code

4.2.22 MSPGANG_Interactive_Copy_Gang_Buffer_to_FLASH

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

MSPGANG_Interactive_Copy_Gang_Buffer_to_FLASH writes 'size' number of bytes from the internal Gang_Buffer[8][128] to FLASH, simultaneously to all active target devices. Data for each target can be different (for example, calibration data or serial numbers). Contents from Gang_Buffer[0][n] are written to target 1, contents from Gang_Buffer[1][n] are written to target 2, and contents from Gang_Buffer[7][n] are written to target 8.

Data in the Gang_Buffer should be prepared and send to MSP-GANG first. See MSPGANG_GetGangBuffer and MSPGANG_SetGangBuffer functions for details.

Syntax

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

Arguments

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

Result

LONG	Error code

4.2.23 MSPGANG_Interactive_EraseSectors

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

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

Syntax

```
LONG MSPGANG_Interactive_EraseSectors(LONG StartAddr, LONG EndAddr)
```

Arguments

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

Result

LONG	Error code

4.2.24 MSPGANG_Interactive_BlankCheck

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

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

Syntax

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

Arguments

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

Result

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

4.2.25 MSPGANG_Interactive_DCO_Test

Note: The target device must be opened first if not open yet (see MSPGANG_Interactive_Open_Target_Device, Section 4.2.12).

MSPGANG_Interactive_DCO_Test takes data from INFO memory location 0x10F8 to 0x10FF, writing one selected word to DCO registers and checking the DCO frequency in real time for up to eight targets simultaneously. Test results in kHz are saved in the *result_in_kHz buffer.

Syntax

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

Arguments

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

Result

LONG	Error code

4.2.26 MSPGANG_SelectImage

MSPGANG_SelectImage sets an active image to work with. MSP-GANG supports 16 images.

Syntax

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

Arguments

LONG lImage	Image number (0 to 15)

Result

LONG	Error code

4.2.27 MSPGANG_EraseImage

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

Syntax

```
LONG MSPGANG_EraseImage(void)
```

Result

LONG	Error code

4.2.28 MSPGANG_CreateGangImage

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

Syntax

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

Arguments

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

Result

LONG	Error code

4.2.29 MSPGANG_LoadImageBlock

MSPGANG_LoadImageBlock saves the previously prepared image contents into the selected image memory. The selected image memory 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(lImage);
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 512 kbytes) is blocked.

Syntax

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

Arguments

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

Result

LONG	Error code

Data Format

```
#define        HEADER_ID_SIZE            10            //fixed size - do not modify
#define        SCRIPT_TEXT_SIZE        16            //fixed size - do not modify
#define        SCRIPT_MCU_NAME_SIZE        16            //fixed size - do not modify

#define        IMAGE_HEADER_CTRL_OFFSET    112

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        Unlock_Info;            //77
    #define        UNLOCK_INFOA            0x40
    #define        UNLOCK_INFO            0x80
        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;                //45  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;                //46
           //#define STANDARD            0 - for compatibility
           #define    I2XX_1K        1        //i2xxx 1K - 0x1000-0x13FF

        BYTE        FLASH_Type;                        //47
               //#define STANDARD            0 - for compatibility
           #define    SEGMENT_1K    1        //i2xxx 1K flash segment size

        BYTE        Secure_Type;                    //48
           //#define STANDARD            0 - for compatibility
           #define    SUC                1         //i2xxx



     //    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
          //INTERFACE_NONE      0
          //INTERFACE_JTAG      4
          //INTERFACE_SBW       8
          //INTERFACE_BSL       0xC
          //INTERFACE_TYPE_MAX_INDEX     INTERFACE_BSL

#define     CFG_JTAG_SPEED                    1
          //INTERFACE_FAST      0
          //INTERFACE_MED       1
          //INTERFACE_SLOW      2
          //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
          // 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
          //EXTERNAL_POWER_WHOLE_RANGE        0   // external power supply -
 whole range from Vccmin to Vccmax
          //POWER_SUPPLIED_BY_MSPGANG         1   // targets supplied by MSP-GANG
          //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
          //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_FLASHERASEMODE               11
          //ERASE_NONE_MEM_INDEX      0
          //ERASE_ALL_MEM_INDEX       1
          //ERASE_PRG_ONLY_MEM_INDEX  2
          //ERASE_INFILE_MEM_INDEX    3
          //ERASE_DEF_CM_INDEX        4
          //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
```

```
          //READ_ALL_MEM_INDEX          0
          //READ_PRGMEM_ONLY_INDEX      1
          //READ_INFOMEM_ONLY_INDEX     2
          //READ_DEF_MEM_INDEX          3
          //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
          //APPLICATION_NO_RESET        0
          //APPLICATION_TOGGLE_RESET    1
          //APPLICATION_TOGGLE_VCC      2
          //APPLICATION_JTAG_RESET      3
          //APPLICATION_RESET_MAX_INDEX   APPLICATION_JTAG_RESET

#define     CFG_BEEPMODE                    25
          //sum of following bits
          //BEEP_PCSPK_EN_BIT      1   //Beep via PC Speaker enable
          //BEEP_OK_EN_BIT         2   //Beep when OK enable
          //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
          //BSL_ENH_DISABLE     0
          //BSL_ENH_NONE        1
          //BSL_ENH_ERASE       2
          //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
          //UART_9600           0
          //UART_19200          1
          //UART_38400          2
          //UART_57600          3
          //UART_115200         4

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

#define     CFG_USE_SCRIPT_FILE             45
          //disable     0
```

```
                  //enable      1

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

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

     #define        CFG_PROJECT_SOURCE                 48
                  //INTERACTIVE_MODE        0
                  //FROM_IMAGE_MEMORY_MODE  1
                  //STANDALONE_MODE     2
                  //FROM_IMAGE_FILE_MODE    3
                  //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
                  //RUNNING_SCRIPT_NONE     0
                  //RUNNING_SCRIPT_ONLINE   1
                  //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
                  //BSL_ANY_PASSW           0
                  //BSL_PASSW_FROM_CODE_FILE        1
                  //BSL_PASSW_FROM_PASSWORD_FILE    2
                  //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
                  // END_ADDR - START_ADDR + 1  <= DEFINED_RETAIN_DATA_MAX_SIZE
                  //DEFINED_RETAIN_DATA_MAX_SIZE    0x40
```

4.2.35 MSPGANG_GetNameConfig, MSPGANG_SetNameConfig

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

Syntax

`LPTSTR MSPGANG_GetNameConfig(LONG index)`

Arguments

LONG index	See list of indexes below

Result

LPTSTR	File name

Syntax

`LONG MSPGANG_SetNameConfig(LONG index, LPTSTR name)`

Arguments

LONG index	See list of indexes below
LPTSTR file_name	

Result

LONG	Error code

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

4.2.36 MSPGANG_SetTmpGANG_Config

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

Syntax

```
LONG MSPGANG_SetTmpGANG_Config(LONG no, LONG data)
```

Arguments

LONG no	Index list of indexes below
LONG data	

Result

LONG	Error code

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

4.2.37 MSPGANG_GetLabel

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

Syntax

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

Arguments

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

Result

LONG	Error code

4.2.38 MSPGANG_GetInfoMemory, MSPGANG_SetInfoMemory

Reads or writes 128 bytes to the internal Information memory. Information memory contains configuration data such as LCD contrast and USB port configuration, and it is not intended to be modified by the user. Use the GUI software to set the Information memory.

Syntax

```
LONG MSPGANG_GetInfoMemory(BYTE page, BYTE *data)
LONG MSPGANG_SetInfoMemory(BYTE page, BYTE *data)
```

Arguments

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

Result

LONG	Error code

4.2.39 MSPGANG_Get_qty_MCU_Family, MSPGANG_Get_MCU_FamilyName, MSPGANG_Check_MCU_Name, MSPGANG_Get_MCU_Name

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

Syntax

```
LONG MSPGANG_Get_qty_MCU_Family(void)
LONG MSPGANG_Get_MCU_FamilyName(LONG index, LPTSTR name)
LONG MSPGANG_Check_MCU_Name(LPTSTR name)
LONG MSPGANG_Get_MCU_Name(LONG group_index, LONG index, LPTSTR name)
```

Use these functions in the following order:

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

MCU_FAMILY MCU_family_list[30];

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

MCU_NAME MCU_name_list[100];

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

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

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

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

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

4.2.40 MSPGANG_Set_MCU_Name

The MSPGANG_Set_MCU_Name allows to select desired target MCU.

Syntax

```
LONG MSPGANG_Set_MCU_Name(LPTSTR name);
```

Arguments

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

Result

LONG	Error code

4.2.41 MSPGANG_HW_devices

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

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

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

Syntax

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

Arguments

LONG max
void **AvailableComPorts

Result

LONG	Error code

4.2.42 MSPGANG_GetProgressStatus

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

bit 0 → Target 1, bit 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_blown_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

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

Example 1

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

```
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 );
```

Schematics

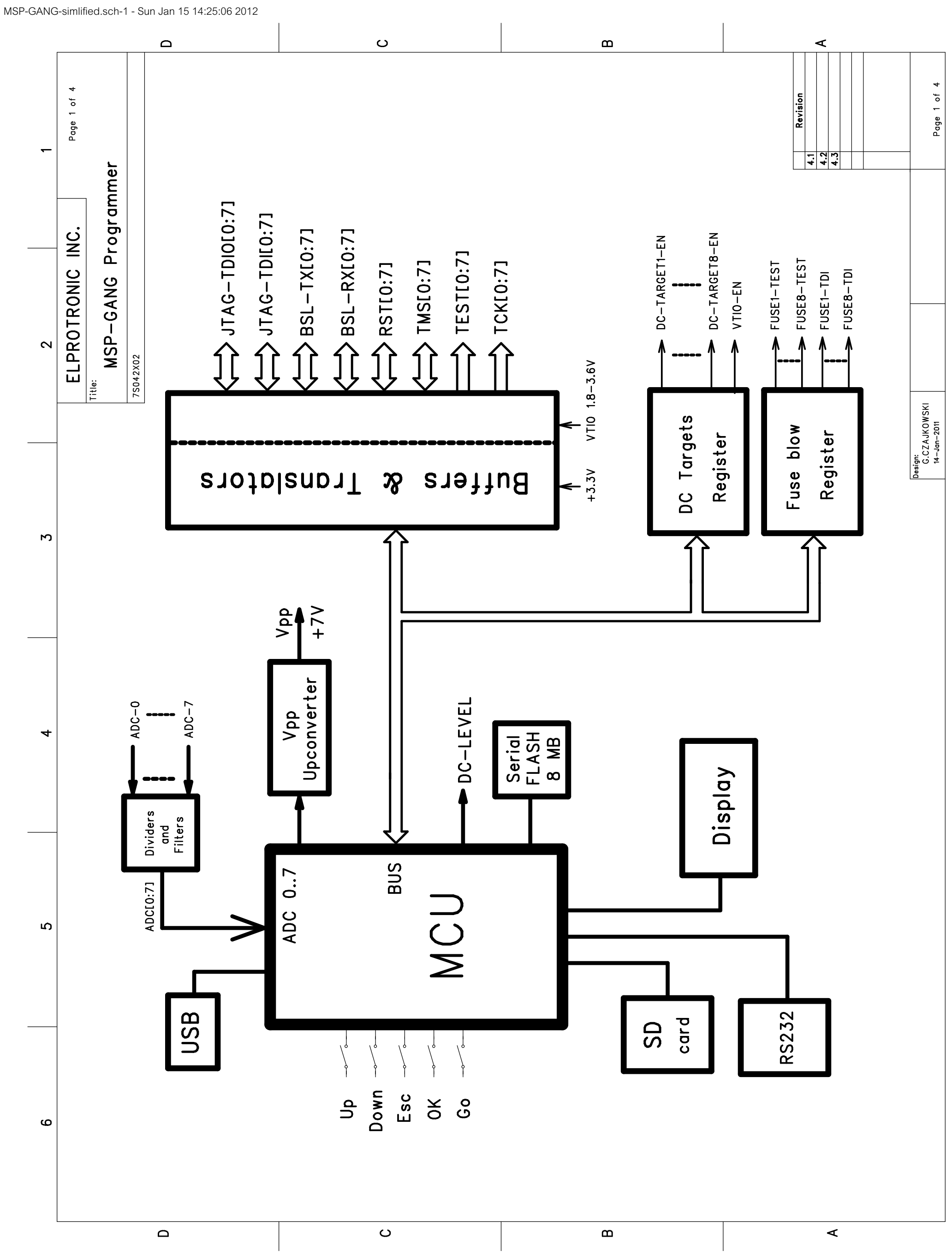


Figure 5-1. MSP-GANG Simplified Schematic (1 of 4)

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

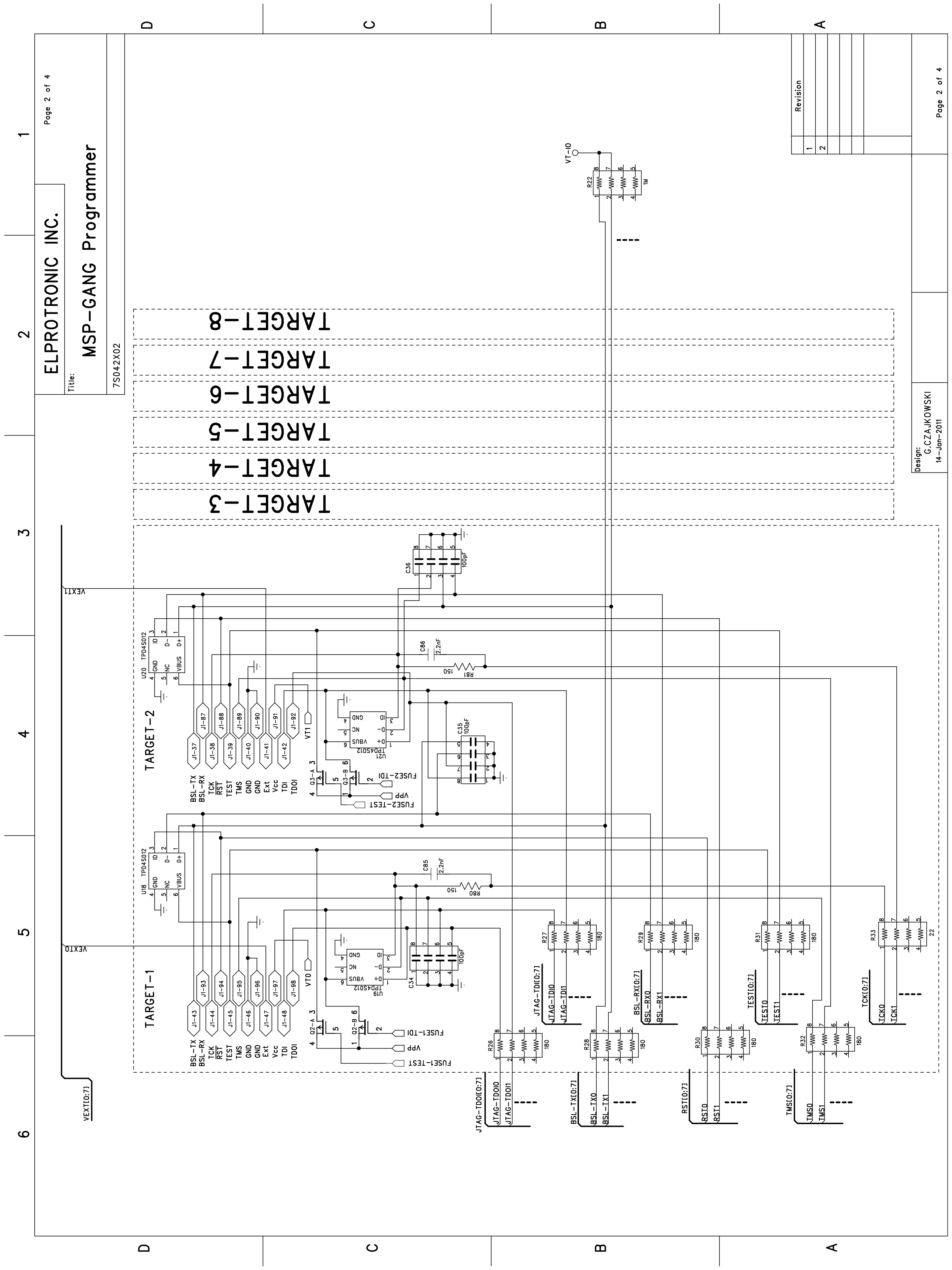


Figure 5-2. MSP-GANG Simplified Schematic (2 of 4)

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

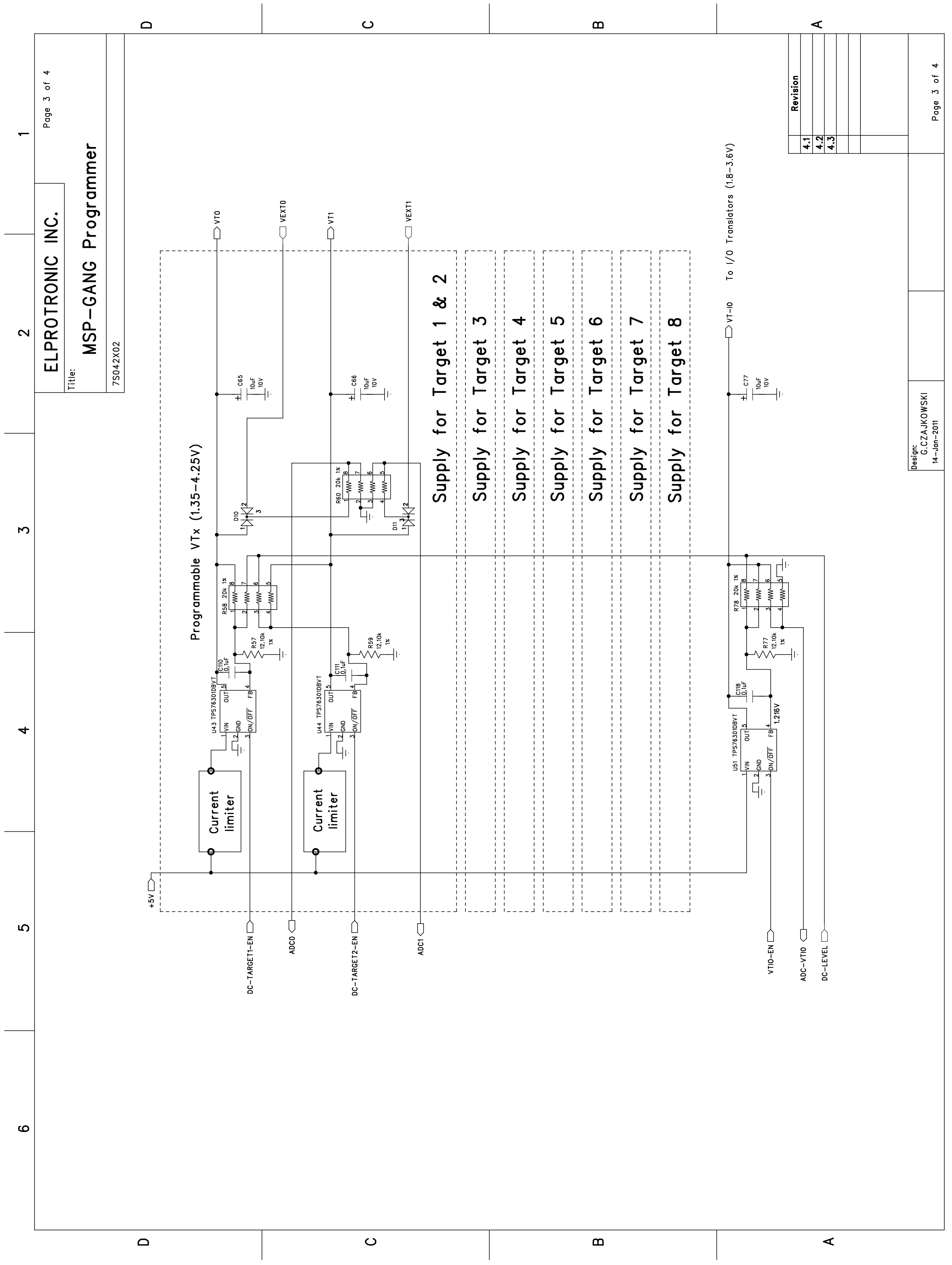


Figure 5-3. MSP-GANG Simplified Schematic (3 of 4)

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

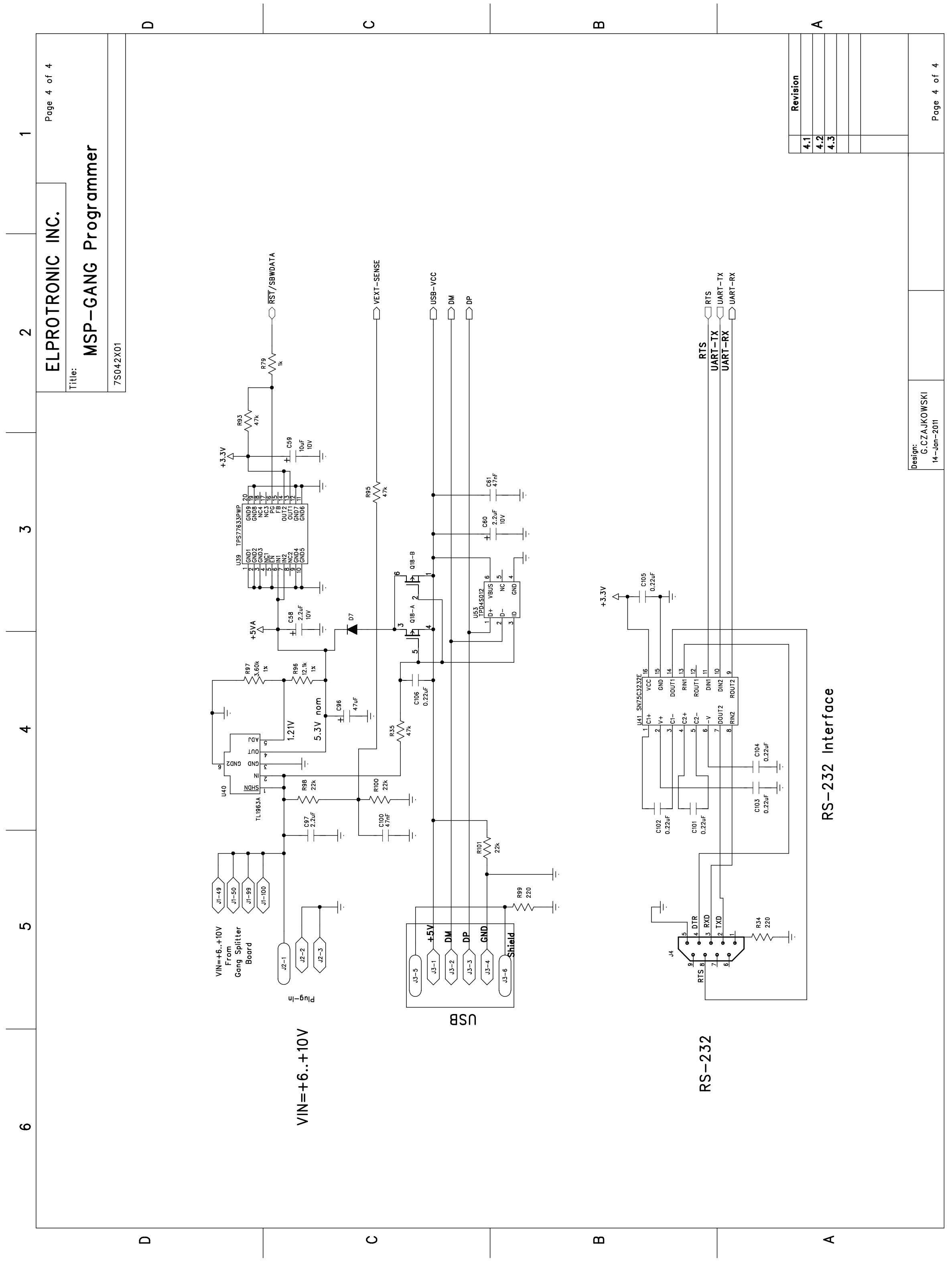


Figure 5-4. MSP-GANG Simplified Schematic (4 of 4)

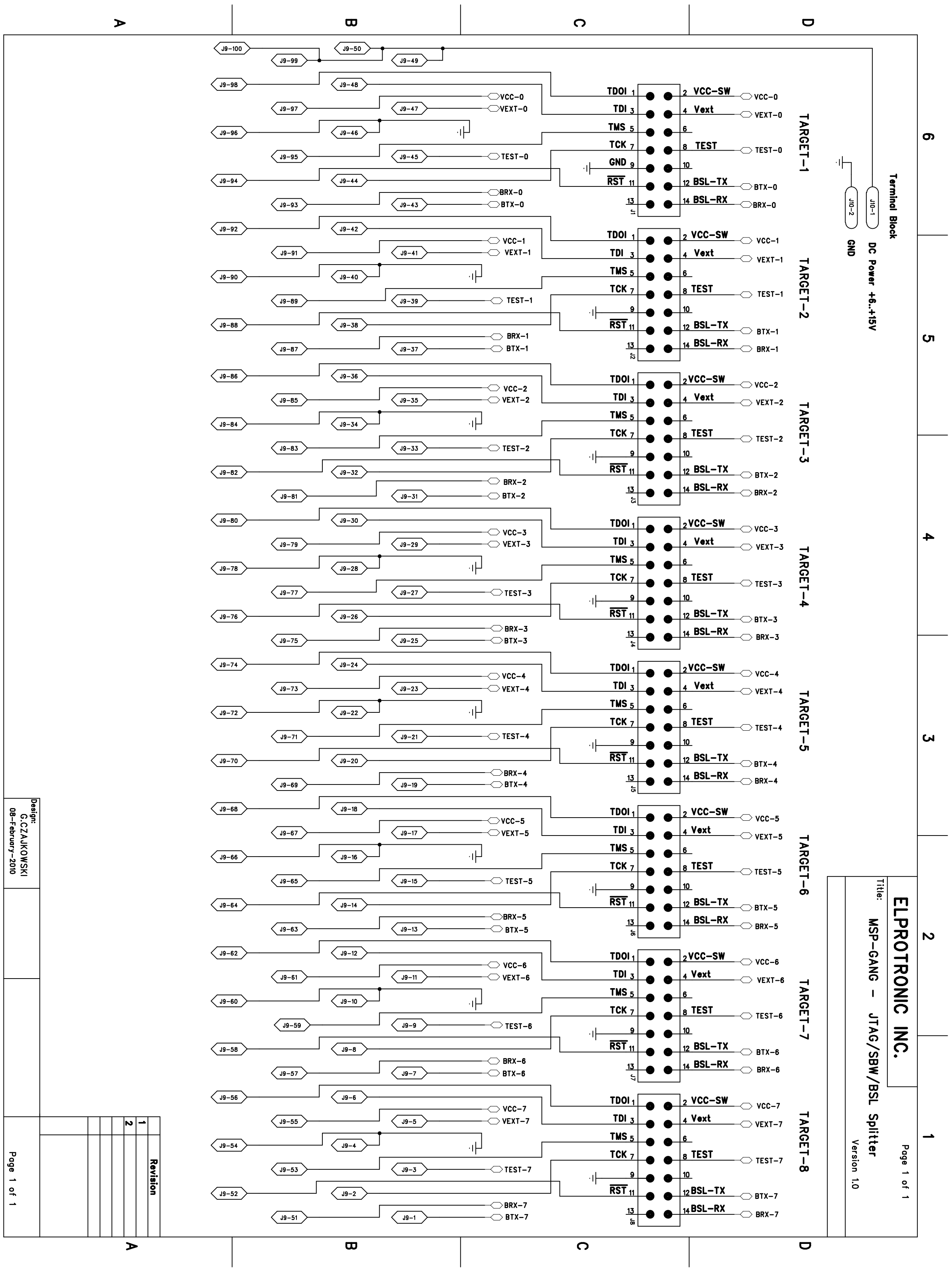


Figure 5-5. Gang Splitter Schematic

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	35-514-0	1	14-pins Header Connector (Mode Electronics, Electrosonic)
2	Connector	35-514-0	1	14-pins Header Connector (Mode Electronics, Electrosonic)
3	Connector	35-514-0	1	14-pins Header Connector (Mode Electronics, Electrosonic)
4	Connector	35-514-0	1	14-pins Header Connector (Mode Electronics, Electrosonic)
5	Connector	35-514-0	1	14-pins Header Connector (Mode Electronics, Electrosonic)
6	Connector	35-514-0	1	14-pins Header Connector (Mode Electronics, Electrosonic)
7	Connector	35-514-0	1	14-pins Header Connector (Mode Electronics, Electrosonic)
8	Connector	35-514-0	1	14-pins Header Connector (Mode Electronics, Electrosonic)
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

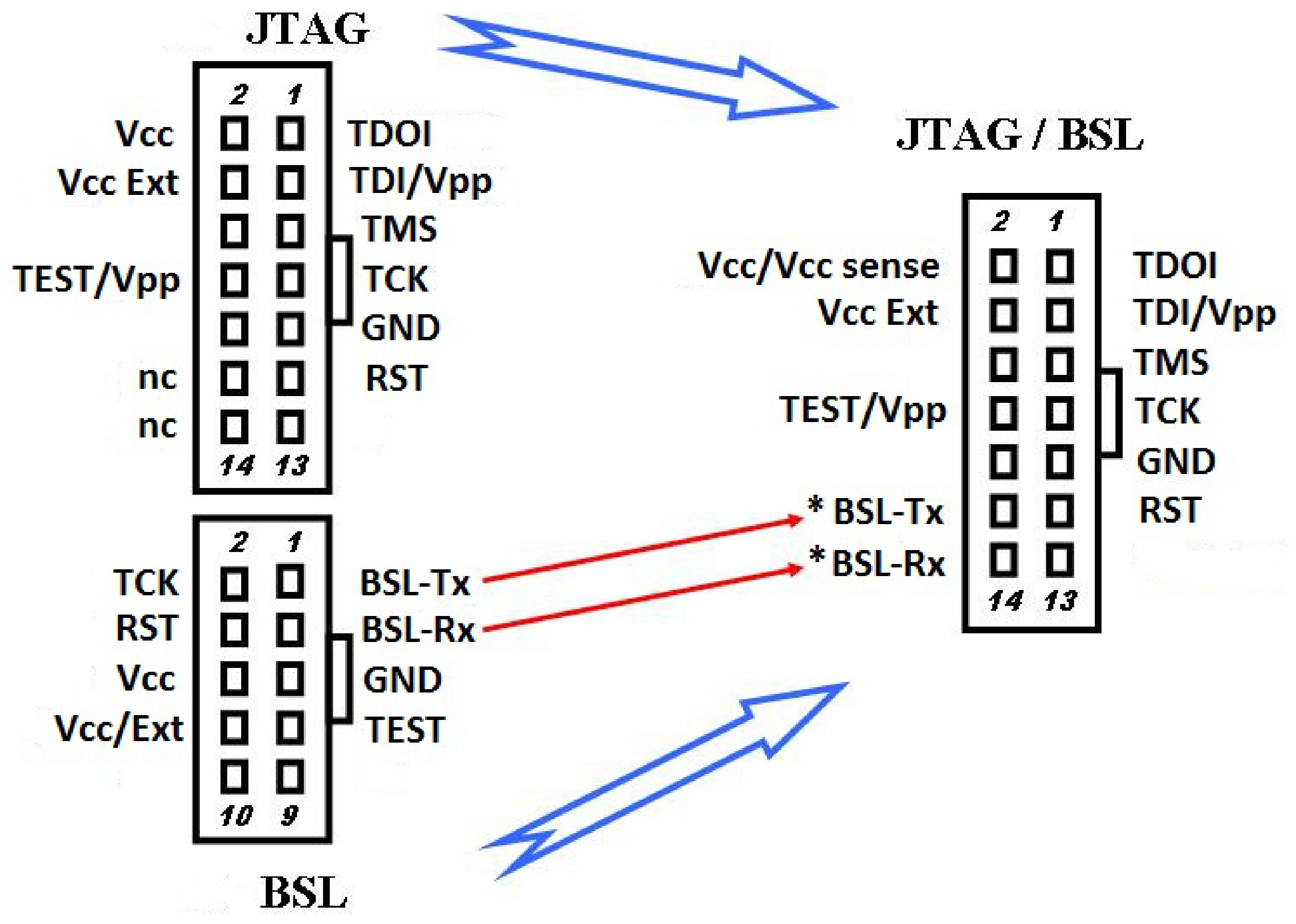


Figure 5-6. BSL Connection Schematic

Detailed description of the BSL connection can be found in *MSP430 Programming Via the Bootstrap Loader (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 µs (a clock pulse duration of 2 µs 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.

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 Ω 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 MSP430 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 MSP430 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-Ω 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 MSP430 families 1xx, 2xx, and 4xx. For blowing the security fuse in these devices, the programmer provides Vpp 6.5 V / 100 mA on the TEST or TDI lines to MSP430 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 Vcc and start the 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 date (if modified), use this option from the pulldown menu:

- View > Compare Code File and Flash Data

Revision History

Changes from G Revision (October 2014) to H Revision **Page**

- Added Section 2.6 52
- Added the "Data Format" section to the description of MSPGANG_ReadImageBlock in Section 4.2.31........ 86
- Added Chapter 6 115

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

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