FlashPro-ARM API-DLL

for Flash Programmers

User's Manual

Software version 1.90

PM036A02 Rev.2.4
May-9-2016

Elprotronic Inc.

16 Crossroads Drive
Richmond Hill,
Ontario, L4E-5C9
CANADA

Web site: www.elprotronic.com
E-mail: info@elprotronic.com
Fax: 905-780-2414
Voice: 905-780-5789

Disclaimer:

No part of this document may be reproduced without the prior written consent of Elprotronic Inc. The information in this document is subject to change without notice and does not represent a commitment on any part of Elprotronic Inc. While the information contained herein is assumed to be accurate, Elprotronic Inc. assumes no responsibility for any errors or omissions.

In no event shall Elprotronic Inc, its employees or authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claims for lost profits, fees, or expenses of any nature or kind.

The software described in this document is furnished under a licence and may only be used or copied in accordance with the terms of such a licence.

Disclaimer of warranties: You agree that Elprotronic Inc. has made no express warranties to You regarding the software, hardware, firmware and related documentation. The software, hardware, firmware and related documentation being provided to You "AS IS" without warranty or support of any kind. Elprotronic Inc. disclaims all warranties with regard to the software, express or implied, including, without limitation, any implied warranties of fitness for a particular purpose, merchantability, merchantable quality or noninfringement of third-party rights.

Limit of liability: In no event will Elprotronic Inc. be liable to you for any loss of use, interruption of business, or any direct, indirect, special incidental or consequential damages of any kind (including lost profits) regardless of the form of action whether in contract, tort (including negligence), strict product liability or otherwise, even if Elprotronic Inc. has been advised of the possibility of such damages.

END USER LICENSE AGREEMENT

PLEASE READ THIS DOCUMENT CAREFULLY BEFORE USING THE SOFTWARE AND THE ASSOCIATED HARDWARE. ELPROTRONIC INC. AND/OR ITS SUBSIDIARIES ("ELPROTRONIC") IS WILLING TO LICENSE THE SOFTWARE TO YOU AS AN INDIVIDUAL, THE COMPANY, OR LEGAL ENTITY THAT WILL BE USING THE SOFTWARE (REFERENCED BELOW AS "YOU" OR "YOUR") ONLY ON THE CONDITION THAT YOU AGREE TO ALL TERMS OF THIS LICENSE AGREEMENT. THIS IS A LEGAL AND ENFORCABLE CONTRACT BETWEEN YOU AND ELPROTRONIC. BY OPENING THIS PACKAGE, BREAKING THE SEAL, CLICKING I AGREE BUTTON OR OTHERWISE INDICATING ASSENT ELECTRONICALLY, OR LOADING THE SOFTWARE YOU AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, CLICK ON THE I DO NOT AGREE BUTTON OR OTHERWISE INDICATE REFUSAL, MAKE NO FURTHER USE OF THE FULL PRODUCT AND RETURN IT WITH THE PROOF OF PURCHASE TO THE DEALER FROM WHOM IT WAS ACQUIRED WITHIN THIRTY (30) DAYS OF PURCHASE AND YOUR MONEY WILL BE REFUNDED.

1. License.
The software, firmware and related documentation (collectively the "Product") is the property of Elprotronic or its licensors and is protected by copyright law. While Elprotronic continues to own the Product, You will have certain rights to use the Product after Your acceptance of this license. This license governs any releases, revisions, or enhancements to the Product that Elprotronic may furnish to You. Your rights and obligations with respect to the use of this Product are as follows:

YOU MAY:

- A. use this Product on many computers;
- B. make one copy of the software for archival purposes, or copy the software onto the hard disk of Your computer and retain the original for archival purposes;
- C. use the software on a network

YOU MAY NOT:

- A. sublicense, reverse engineer, decompile, disassemble, modify, translate, make any attempt to discover the Source Code of the Product; or create derivative works from the Product;
- B. redistribute, in whole or in part, any part of the software component of this Product;
- C. use this software with a programming adapter (hardware) that is not a product of Elprotronic Inc.

2. Copyright
All rights, title, and copyrights in and to the Product and any copies of the Product are owned by Elprotronic. The Product is protected by copyright laws and international treaty provisions. Therefore, you must treat the Product like any other copyrighted material.

3. Limitation of liability.
In no event shall Elprotronic be liable to you for any loss of use, interruption of business, or any direct, indirect, special, incidental or consequential damages of any kind (including lost profits) regardless of the form of action whether in contract, tort (including negligence), strict product liability or otherwise, even if Elprotronic has been advised of the possibility of such damages.

4. DISCLAIMER OF WARRANTIES.
You agree that Elprotronic has made no express warranties to You regarding the software, hardware, firmware and related documentation. The software, hardware, firmware and related documentation being provided to You "AS IS" without warranty or support of any kind. Elprotronic disclaims all warranties with regard to the software and hardware, express or implied, including, without limitation, any implied warranties of fitness for a particular purpose, merchantability, merchantable quality or noninfringement of third-party rights.

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.

*N*OTE: 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.

*W*arning: Changes or modifications not expressly approved by Elprotronic 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.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document describes the FlashPro-ARM API-DLL, and explains how to use it. The FlashPro-ARM API-DLL is a dynamic library that can control different types of Flash Programming Adapters (FPAs) simultaneously. It is intended for users that need to use multiple ARM FPAs to program a board with multiple ARM microcontroller units (MCUs). An example hardware setup is shown in Figure 1.1.

The FlashPro-ARM API-DLL is actually implemented using two DLLs, the Multi API-DLL and the API-DLL. One instance of the top-level Multi API-DLL controls multiple instances of the API-DLL. The Multi API-DLL manages all global meta-data and control flow while the API-DLL actually performs programming for each MCU. The user only directly interacts with the Multi API-DLL.

Initializing multiple FPAs using the Multi API-DLL is easy. Only specify the index and serial numbers (SNs) of the FPAs you wish to control, and the Multi API-DLL will create enough API-DLL instances to accomplish the task automatically. Using the Multi API-DLL, select which FPA you wish to control (or all at once), and issue commands that will be automatically forwarded to the corresponding API-DLL. Currently, up to 64 individual FPAs can be controlled using the Multi API-DLL (these would normally be connected using USB hubs connected to the host computer's USB port).

1.1 Getting Started

Install the package provided from Elprotronic or download the latest version from www.elprotronic.com

After installation (default C:\ Program Files(x86)\ Elprotronic\ ARM\ FlashPro-ARM) the directory will contain the FlashPro-ARM executable, Multi API-DLL and API-DLL files, a C++ and C# code example, and several documents. The easiest way to get started using the DLL is to run the code examples. Copy the code examples out of the *ProgramFiles* directory to a place where files can be freely modified (or run Visual Studio using administrator privileges).

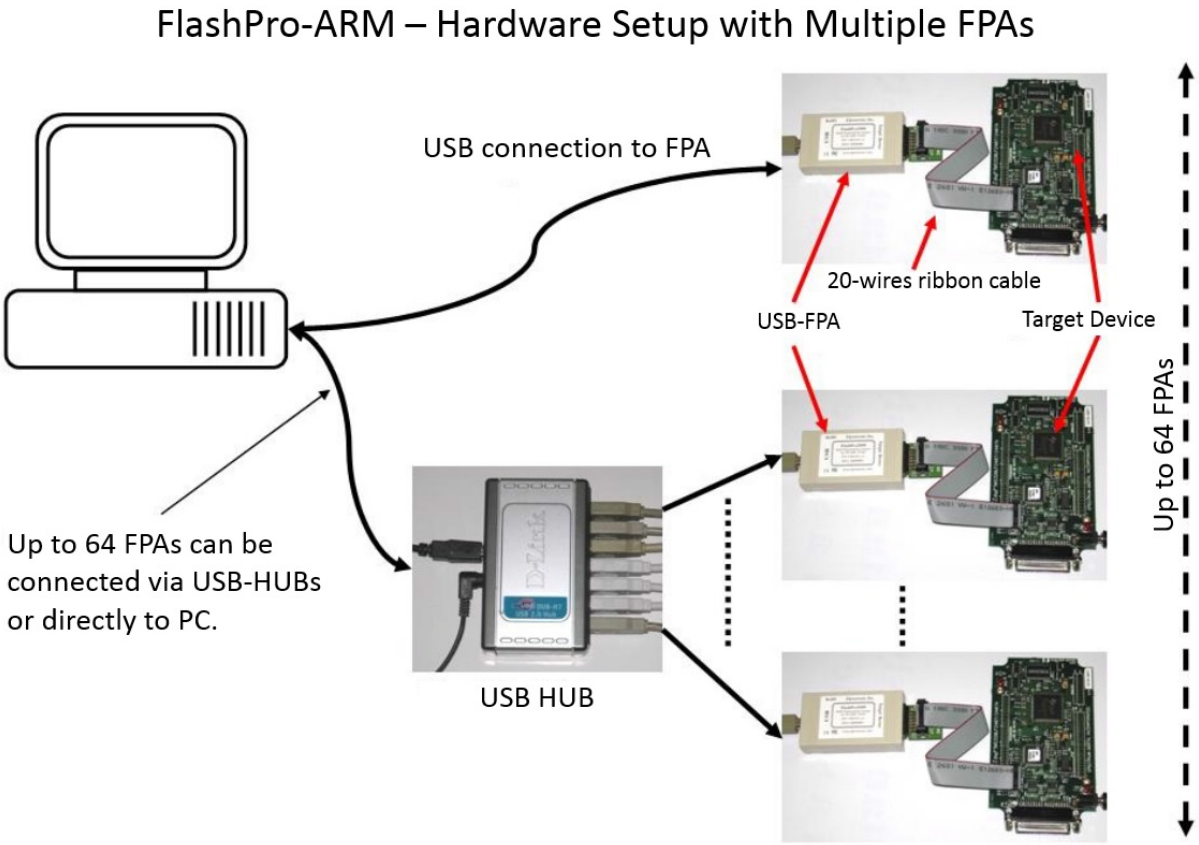

Figure 1.1: Illustration of a typical hardware setup when programming with multiple FPAs.

If converting from the current Visual Studio 2005 project, to newer versions, sometimes incorrect conversions can break intended GUI dimensions.

When the MS VC++ application is created, the following files should be added to the project:

- FlashProARM-Dll.h - header file for C++
- FlashProARM-FPAsel.lib - lib file for C++
- ARM-Errors-list.cpp - (Optional) Errors list description
- ARM-Errors-list.h - Errors list definitions

When the MS VC# application is created, the following file should be added to the project:

- FLASHPROARM_API.cs - DLL Import definitions
- FLASHPROARM_DLL.cs - Definitions used by DLL

The following files should be placed within the current working directory of the application executable:

- FlashProARM-FPAsel.dll - Multi API-DLL selection/distribution
- FlashProARM-FPA1.dll - API-DLL for FPA adapter

The code examples provided illustrate how to use most, but not all available DLL functions. The pull-down menus that list MCU vendors, families, groups, and names are intended to be used for each individual FPA separately (index 1 to 64), not for FPA index 0 (ALL_FPAs) since each FPA can support different vendors depending on purchased model.

1.2 Data and Control Flow

The Multi API-DLL(FlashProARM-FPAsel.dll) forwards calls coming from application software to individual API-DLL instances (FlashProARM-FPA1.dll to FlashProARM-FPA64.dll). A representation of the control and data flow is shown in Figure 1.2. The desired destination FPA can be selected using the function F_Set_FPA_index(fpa) where indices 1 to 64 select only one desired FPA. Select index 0 when ALL FPAs should be selected.

The selected FPA index modified by the F_Set_FPA_index(FPA) function can be modified at any time. By default, the FPA index is 1 and if only one FPA is used then FPA index does not need to be modified. When FPA index 1 to 64 is used, then the result of any forwarded function call will be directly returned to application software from the single API-DLL instance being invoked transparently. When FPA index 0 is used (ALL-FPAs) and results are the same from all FPAs, then the same result is passed back to application software. If results are not the same, then the Multi API-DLL will return value FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF). To obtain individual results per API-DLL instance use the function F_LastStatus(FPA). This function will

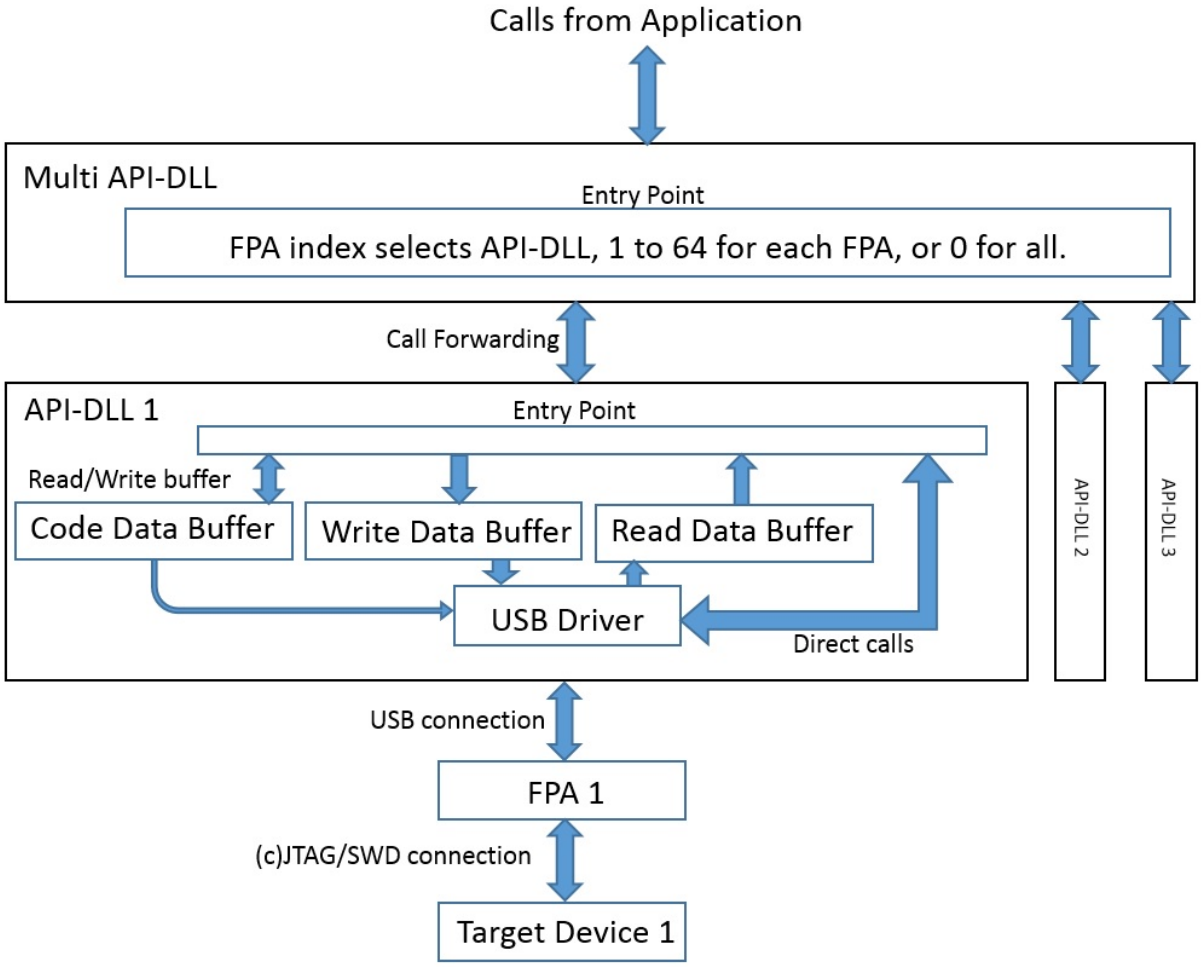

Figure 1.2: Illustration of how the Multi API-DLL controls the API-DLL instances. Calls to the Multi API-DLL are forwarded based on the FPA index.

read back the results of the previously called API-DLL function, which was stored internally in the Multi API-DLL's result buffer. This temporary result buffer will be overwritten on the next call to each API-DLL instance.

When the FPA index is 0 (ALL-FPAs) then almost all functions are executed simultaneously. Less critical functions are executed sequentially from FPA-1 up to FPA-64 but this distinction cannot be seen from the application software. The longest running programming functions, F_AutoProgram, etc., listed in Section 3.4, are all executed fully in parallel. When different configuration are used for each FPA (FPA-1 programs 10kB of data and FPA-2 programs 20kB of data) then the Multi API-DLL function call will return when all API-DLL function calls have finished.

When an inactive FPA index is selected (FPA not initialized), then return value from selected function is FPA_INVALID_NO (-2 or 0xFFFFFFFE). When all FPA has been selected (FPA index = 0) then only active FPAs will be called. For example if only one FPA is active and FPA index = 0, then only one FPA will be used. It is optimal to prepare application software that allows to remote control up to 64 FPAs and on startup activate only desired number of FPAs.

All API-DLL instances used by the Multi API-DLL are fully independent with respect to each other. Data transferred to one FPA does not have to be the same as the data transferred to other FPAs. For example, code data programmed using FPA-1 will use the buffers from API-DLL 1, and code data programmed using FPA-2 will use buffers from API-DLL 2. Once the different code buffers are setup, programming can be done simultaneously using FPA index 0 and the function F_AutoProgram. To configure the data buffers shown in Figure 1.2 use functions from Section 3.3.

The major features supported by the FlashPro-ARM API-DLL are:

- Initialize and terminate communication with FPA and target device,
- Configure settings from file or using functions,
- Program code data from file or modify code buffer using functions,
- Return string report message generated for most actions,
- Reset target device,
- Program target device (erase, blank check, program, verify, lock, and unlock),
  - Erase all or selected part of memory,
  - Blank check all or selected part of memory,
  - Write all or selected part of memory,
  - Verify all or selected part of memory,
  - Read all or selected part of memory,
  - Lock memory protection bits and debug access (if supported on device),
  - Unlock device to return memory to factory settings (if supported on device).

Chapter 2

Configuration

Configuration options described in this chapter are used to configure how each FPA will program its target device. Currently the FlashPro-ARM API-DLL supports vendors: Texas Instruments - ARM, Texas Instruments - Chipcon, ST Microelectronics, Silicon Labs, Nordic Semiconductor, Maxim, Freescale, Atmel, and Active-Semi. Most configuration options described in this chapter can be used by all vendors, except for Memory Protection options, which are specific to each vendor. Memory Protection options are described in the last section and divided on a per vendor basis. If a configuration file supplied to F_ConfigFileLoad contains protection options from a different vendor than the MCU currently being programmed, they will be ignored, and default values will be substituted instead (no protection).

The configuration parameter names listed in this chapter are the exact string names that can be used as inputs to the functions F_Set_Config_Value_By_Name and F_Get_Config_Value_By_Name or specified in the configuration file for use by the function F_ConfigFileLoad. To obtain these string names from the Multi API-DLL, select the desired API-DLL instance (using F_Set_FPA_index) and use the function F_Get_Config_Name_List. All the values specified here are parsed in as 32-bit unsigned integers (hex values can be entered using the prefix "0x").

2.1 Communication

Configuration option specified in this section can be configured visually using the main GUI window, Interface tab. This option affects all communication via FPA to the target device.

2.1.1 Interface

This configuration option specifies the communication type and speed. The type and speed settings are OR-ed together to create the final value specified for this parameter.

Communication type:

- COMM_JTAG (0x10) : Joint Test Action Group (JTAG) communication.
- COMM_SWD (0x20) : Serial Wire Debug (SWD) communication.

Communication speed:

- COMM_SLOW (0) : Slow communication frequency. For testing unreliable connection.
- COMM_MED (1) : Medium communication frequency.
- COMM_FAST (2) : Fast communication frequency. For maximum performance with a good connection.

Configuration option end value:

- COMM_JTAG_SLOW (COMM_JTAG | COMM_SLOW) : Slow JTAG communication.
- COMM_JTAG_MED (COMM_JTAG | COMM_MEDIUM) : Medium JTAG communication.
- COMM_JTAG_FAST (COMM_JTAG | COMM_FAST) : Fast JTAG communication.
- COMM_SWD_SLOW (COMM_SWD | COMM_SLOW) : Slow SWD communication.
- COMM_SWD_MED (COMM_SWD | COMM_MEDIUM) : Medium SWD communication.
- COMM_SWD_FAST (COMM_SWD | COMM_FAST) : Fast SWD communication.

2.2 Power Source

The configuration option in this section specifies the power source to the target device and can be configured visually using the main GUI window, MCU Vcc tab. This option is used by all functions that Encapsulated Functions listed in Section 3.4 and several Sequential Functions listed in Section 3.5.

2.2.1 PowerFromFpaEn

This configuration option enables the FPA to supply power to the target device for programming.

Logically grouped together with VccFromFPAin_mV.

- 1 - enabled
- 0 - disabled

2.2.2 VccFromFPAin_mV

Voltage in millivolts to be supplied from the FPA to the target device if PowerFromFpaEn is enabled. Ignored otherwise.

2.3 Code File

The configuration option in this section specifies if the code file chosen for programming using the function F_ReadCodeFile should be reloaded before programming every time. This option can be configured visually using the main GUI window, Device Action, Reload Code File check box.

2.3.1 CodeFileReload

This configuration option will force the code file to be reloaded every time before the F_AutoProgram operation is performed. This will erase any manual changes to the Code Data Buffer if enabled.

- 1 : enabled
- 0 : disabled

2.4 System Clock

The configuration option in this section specifies the frequency to which the System Clock on the target device has been set. This setting is only used if the processor does not have an internal clock source, and requires an external oscillator.

Currently, all supported MCUs have an internal oscillator, therefore this option has no effect.

2.4.1 CLK_frequency

System clock frequency that the FPA will expect to read when communication with the target is established, in MHz.

2.5 Memory Options

Configuration options specified in this section can be configured visually using the GUI dialog under Setup, Memory Options. These options are only used when calling Encapsulated Functions listed in Section 3.4.

2.5.1 FlashEraseModeIndex

This configuration parameter chooses which memory segments (or all) will be used for the F_AutoProgram, F_Memory_Erase, F_Memory_Write, and F_Memory_Verify operations.

- ERASE_NONE_MEM_INDEX (0) : Update only. no Erase operation.
- ERASE_ALL_MEM_INDEX (1) : OTP and Flash Memory. Erase all Flash Memory, Write and Verify all Code File contents.
- ERASE_PRG_ONLY_MEM_INDEX (2) : Flash Memory only. OTP Memory is not touched even if included in Code File.
- ERASE_INFILE_MEM_INDEX (3) : Used by Code File. Erase only memory needed by Code File, Write and Verify all Code File contents.
- ERASE_DEF_CM_INDEX (4) : User defined. Specify Flash Memory and OTP Memory range for programming.
- WRITE_OTP_MEM_ONLY_INDEX (5) : OTP Memory only. Flash Memory is not touched even if included in Code File.

2.5.2 EraseMemIfLockedEn

During the Auto Program procedure, if blank check fails after erase, the FPA can attempt to perform the F_Clear_Locked_Device procedure to return the device to factory settings. If this option is enabled F_AutoProgram will do this automatically.

- 1 : enabled
- 0 : disabled

2.5.3 NotEraseSegmentIfBlank

Before each memory segment is erased, a quick checksum is calculated to see if the memory segment is already blank. If this option is enabled then memory segments whose checksum indicates a blank segment will not be erased. This can speed up programming times for mostly empty target devices.

- 1 : enabled
- 0 : disabled

2.5.4 EraseDefMainMemEn

When programming code using the "User defined" setting, it is possible to specify the address range for Flash Memory, and OTP Memory to be programmed. This configuration option enables Flash Memory programming in "User defined" mode for the Erase, Write, and Verify operations.

Logically grouped together with DefEraseFlashStart and DefEraseFlashStop.

- 1 - enabled
- 0 - disabled

2.5.5 DefEraseFlashStart

Start address for Flash Memory programming if EraseDefMainMemEn is enabled (inclusive). Ignored otherwise. Granularity for the Erase procedure is one memory segment.

2.5.6 DefEraseFlashStop

End address for Flash Memory programming if EraseDefMainMemEn is enabled (inclusive). Ignored otherwise. Granularity for the Erase procedure is one memory segment.

2.5.7 DefOTPWriteEnable

When programming code using the "User defined" setting, it is possible to specify the address range for Flash Memory, and OTP Memory to be programmed. This configuration option enables OTP Memory programming in "User defined" mode for the Write, and Verify operations. OTP Memory cannot be erased.
Logically grouped together with DefOTPWriteStart and DefOTPWriteStop.

- 1 : enabled
- 0 : disabled

2.5.8 DefOTPWriteStart

Start address for OTP Memory programming if DefOTPWriteEnable is enabled (inclusive). Ignored otherwise.

2.5.9 DefOTPWriteStop

End address for OTP Memory programming if DefOTPWriteEnable is enabled (inclusive). Ignored otherwise.

2.5.10 FlashReadModeIndex

This configuration parameter chooses which memory contents will be read using the F_Memory_Read operation into the Read Data Buffer.

- READ_ALL_MEM_INDEX (0) : OTP and Flash Memory. All memory, including protection bits will be read.
- READ_PRGMEM_ONLY_INDEX (1) : Flash Memory only.
- READ_INFOMEM_ONLY_INDEX (2) : OTP Memory only.
- READ_DEF_MEM_INDEX (3) : User defined. Specify Flash Memory and OTP Memory range for reading.

2.5.11 ReadDefMainMemEn

When reading memory using the "User defined" setting, it is possible to specify the address range for Flash Memory, and OTP Memory to be read. This configuration option enables Flash Memory reading in "User defined" mode for the Read operation.

Logically grouped together with DefReadStartAddr and DefReadStopAddr.

- 1 - enabled
- 0 - disabled

2.5.12 DefReadStartAddr

Start address for reading Flash Memory if ReadDefMainMemEn is enabled (inclusive). Ignored otherwise.

2.5.13 DefReadStopAddr

End address for reading Flash Memory if ReadDefMainMemEn is enabled (inclusive). Ignored otherwise.

2.5.14 ReadDefInfoMemEn

When reading memory using the "User defined" setting, it is possible to specify the address range for Flash Memory, and OTP Memory to be read. This configuration option enables OTP Memory reading in "User defined" mode for the Read operation.

Logically grouped together with DefOTPStartAddr and DefOTPStopAddr.

- 1 : enabled
- 0 : disabled

2.5.15 DefOTPStartAddr

Start address for reading OTP Memory if ReadDefInfoMemEn is enabled (inclusive). Ignored otherwise.

2.5.16 DefOTPStopAddr

End address for reading OTP Memory if ReadDefInfoMemEn is enabled (inclusive). Ignored otherwise.

2.5.17 RetainDefinedData

When performing the F_AutoProgram and F_Memory_Erase operations it is possible to exempt some Flash Memory from being modified. This is useful for maintaining calibration data. Because the retain data range will most likely not span whole segments of memory (whereas erasing is done on whole segments), this operation is implemented in three steps. The Flash Memory contents are read first, the underlying segments are erased, and saved contents are written back to Flash.

The most common usage of this feature is when performing two calls to F_AutoProgram during production. First call writes calibration data during testing, second call writes final application code into the target device. The intention is to prevent programming of application code from erasing calibration data.

Logically grouped together with RetainDataStart and RetainDataStop.

- 1 : enabled
- 0 : disabled

2.5.18 RetainDataStart

Start address for Flash Memory to be preserved if RetainDefinedData is enabled (inclusive). Ignore otherwise.

2.5.19 RetainDataStop

End address for Flash Memory to be preserved if RetainDefinedData is enabled (inclusive). Ignore otherwise.

2.5.20 VerifyModeIndex

This configuration option chooses the verification method for the F_Memory_Verify operation.

- VERIFY_NONE_INDEX (0) : None.
- VERIFY_STD_INDEX (1) : Standard. Verify first using checksum, and PSA, and then perform a full read and byte-by-byte verification.
- VERIFY_FAST_INDEX (2) : Fast. Verify using checksum and PSA.

2.6 Check Sum Options

Configuration options specified in this section can be configured visually using the GUI dialog under Setup, Check Sum Options. In addition to the internal check sum calculation done for programming verification in the procedure F_AutoProgram and F_Memory_Verify, the user can also calculate and save a check sum for specific code segments. This is useful for internal verification by application code at startup to ensure that the internal flash has not become corrupted before executing the rest of the application code. Up to four separate check sum results can be calculated and stored in different addresses.

2.6.1 CS_type_index

This configuration parameter specifies how the check sum result is to be calculated. If none is selected, then the other options in this section are ignored. If a defined polynomial is selected, then the CS_Polynomial parameter will be used for Check Sum calculations.

Logically grouped together with all parameters in this section.

- CS_TYPE_NONE_INDEX (0) : None.
- CS_TYPE_ARYTHMETIC_8r16 (1) : Arithmetic sum (8b / 16b).
- CS_TYPE_ARYTHMETIC_8r32 (2) : Arithmetic sum (8b / 32b).
- CS_TYPE_ARYTHMETIC_16r16 (3) : Arithmetic sum (16b / 16b).
- CS_TYPE_ARYTHMETIC_16r32 (4) : Arithmetic sum (16b / 32b)
- CS_TYPE_CRC16_P11021 (5) : CRC16 (Poly = 0x11021) (8b / 16b).
- CS_TYPE_CRC16_DEF (6) : CRC16 defined polynomial (8b / 16b).
- CS_TYPE_CRC32_STD (7) : CRC32 (Poly = 0x04C11DB7) (8b / 32b).
- CS_TYPE_CRC32_DEF (8) : CRC32 defined polynomial (8b / 32b).

2.6.2 CS_Init_index

Initial value that will be modified as the check sum is calculated.

- CS_INIT_VALUE_0_INDEX (0) : Initial value is 0.
- CS_INIT_VALUE_1_INDEX (1) : Initial value is 0xFFFFFFFF.
- CS_INIT_VALUE_ADDR_INDEX (2) : Initial value is the start address.

2.6.3 CS_Result_index

The end result can be stored as is or inverted.

- CS_RESULT_ASIS_INDEX (0) : Result stored as calculated.
- CS_RESULT_INVERTED_INDEX (1) : Result is inverted and then stored.

2.6.4 CS_Polynominal

If a defined, polynomial is selected to calculate the check sum, then this parameter will specify this polynomial. For legacy reasons, the spelling error in the name is left uncorrected.

2.6.5 CS_code_overwrite_en

It is possible that the address chosen for storing the check sum result will conflict with the Code Data Buffer. If this occurs, this parameter will specify if code contents should be overwritten with the check sum result.

- 1 : overwrite.
- 0 : do not overwrite.

2.6.6 CS1_Enable

This configuration parameter specifies whether check sum one should be calculated and stored. The parameter value should be the OR-ed result of enabled and saved if the intention is to calculate the check sum and save it in Flash Memory.

Logically grouped together with CS1_StartAddress, CS1_EndAddress and CS1_ResultAddress.

- 0 : disabled.
- CS_CALC_ENABLE_BIT (1) : Calculate check sum result.

- CS_SAVE_ENABLE_BIT (2) : Save check sum result.
- CS_CALC_ENABLE_BIT | CS_SAVE_ENABLE_BIT (3) : Calculate and save check sum result.

2.6.7 CS1_StartAddress

Data at this start address within the Code Data Buffer will be the first data used to calculate the check sum (inclusive).

2.6.8 CS1_EndAddress

Data at this end address within the Code Data Buffer will be the last data used to calculate the check sum (inclusive).

2.6.9 CS1_ResultAddress

The check sum result will be stored at this address. The size of the check sum depends on the CS_type_index.

2.6.10 CS2_Enable

This configuration parameter specifies whether check sum two should be calculated and stored. The parameter value should be the OR-ed result of enabled and saved if the intention is to calculate the check sum and save it in Flash Memory.

Logically grouped together with CS2_StartAddress, CS2_EndAddress and CS2_ResultAddress.

- 0 : disabled.
- CS_CALC_ENABLE_BIT (1) : Calculate check sum result.
- CS_SAVE_ENABLE_BIT (2) : Save check sum result.
- CS_CALC_ENABLE_BIT | CS_SAVE_ENABLE_BIT (3) : Calculate and save check sum result.

2.6.11 CS2_StartAddress

Data at this start address within the Code Data Buffer will be the first data used to calculate the check sum (inclusive).

2.6.12 CS2_EndAddress

Data at this end address within the Code Data Buffer will be the last data used to calculate the check sum (inclusive).

2.6.13 CS2_ResultAddress

The check sum result will be stored at this address. The size of the check sum depends on the CS_type_index.

2.6.14 CS3_Enable

This configuration parameter specifies whether check sum three should be calculated and stored. The parameter value should be the OR-ed result of enabled and saved if the intention is to calculate the check sum and save it in Flash Memory.

Logically grouped together with CS3_StartAddress, CS3_EndAddress and CS3_ResultAddress.

- 0 : disabled.
- CS_CALC_ENABLE_BIT (1) : Calculate check sum result.
- CS_SAVE_ENABLE_BIT (2) : Save check sum result.
- CS_CALC_ENABLE_BIT | CS_SAVE_ENABLE_BIT (3) : Calculate and save check sum result.

2.6.15 CS3_StartAddress

Data at this start address within the Code Data Buffer will be the first data used to calculate the check sum (inclusive).

2.6.16 CS3_EndAddress

Data at this end address within the Code Data Buffer will be the last data used to calculate the check sum (inclusive).

2.6.17 CS3_ResultAddress

The check sum result will be stored at this address. The size of the check sum depends on the CS_type_index.

2.6.18 CS4_Enable

This configuration parameter specifies whether check sum four should be calculated and stored. The parameter value should be the OR-ed result of enabled and saved if the intention is to calculate the check sum and save it in Flash Memory.

Logically grouped together with CS4_StartAddress, CS4_EndAddress and CS4_ResultAddress.

- 0 : disabled.
- CS_CALC_ENABLE_BIT (1) : Calculate check sum result.
- CS_SAVE_ENABLE_BIT (2) : Save check sum result.
- CS_CALC_ENABLE_BIT | CS_SAVE_ENABLE_BIT (3) : Calculate and save check sum result.

2.6.19 CS4_StartAddress

Data at this start address within the Code Data Buffer will be the first data used to calculate the check sum (inclusive).

2.6.20 CS4_EndAddress

Data at this end address within the Code Data Buffer will be the last data used to calculate the check sum (inclusive).

2.6.21 CS4_ResultAddress

The check sum result will be stored at this address. The size of the check sum depends on the CS_type_index.

2.7 Device Reset

Configuration options specified in this section can be configured visually using the GUI dialog under Setup, Device Reset. MCU Hardware Reset mode selection is used before communication initialization (a.k.a. device open) (ResetTimeIndex), and whenever a reset is required during programming (usually to verify protection bits). After the device is disconnected (a.k.a. device close) (ReleaseJtagState) JTAG lines are set to a finished state. Sequential Functions listed in Section 3.5 and Encapsulated Functions listed in Section 3.4 both use these settings.

The Final Target Device action (ApplicationStartEnable) is only used by F_AutoProgram, an Encapsulated Function listed Section 3.4.

2.7.1 ResetTimeIndex

This configuration option specifies the duration of a hardware reset done by the FPA on the RST line before initiating communication, or whenever a reset is required during programming (usually to verify protection bits).

Logically grouped together with ResetPulseTime and ResetIdleTime.

- RESET_10MS_INDEX (0) : 10 ms reset pulse.
- RESET_100MS_INDEX (1) : 100 ms reset pulse.
- RESET_200MS_INDEX (2) : 200 ms reset pulse.
- RESET_500MS_INDEX (3) : 500 ms reset pulse.
- RESET_CUSTOM_INDEX (4) : Custom length reset pulse (see related parameters).

2.7.2 ResetPulseTime

Time in milliseconds that hardware reset line (RST line) is low if custom length is selected for ResetTimeIndex. Ignored otherwise.

2.7.3 ResetIdleTime

Time in milliseconds that FPA will wait before performing any actions after the reset pulse if custom length is selected for ResetTimeIndex. Ignored otherwise.

2.7.4 ReleaseJtagState

The state of the JTAG lines (TMS, TCK, and TDI) when target device is closed (communication has been terminated).

- DEFAULT_JTAG_3ST (0) : Tri-stated.
- DEFAULT_JTAG_HI (1) : High.
- DEFAULT_JTAG_LO (2) : Low.

2.7.5 ApplicationStartEnable

This configuration option specifies the action taken by the FPA after a successful invocation of F_AutoProgram. Usually this means that the target device has all uploaded firmware code that has been positively verified. If the selection is to allow the application program to start, the maximum runtime can be selected using the ApplProgramRunTime setting.

Logically grouped together with ApplProgramRunTime

- APPLICATION_KEEP_RESET (0) : Keep hardware reset line active - on low level (default).
- APPLICATION_TOGGLE_RESET (1) : Hardware reset (RST line) and start the application program.
- APPLICATION_NOT_RESET (2) : Do not reset the target device.
- APPLICATION_JTAG_RESET (3) : Soft reset (over JTAG/SWD) and start the application program.

2.7.6 ApplProgramRunTime

The time to allow the application to run on the target device after F_AutoProgram has successfully completed. Select 0 for unlimited, and between 1-120 for time in seconds. After the time expires, the RST line will be brought to low level. Ignored if ApplicationStartEnable setting does not run the application program.

2.8 JTAG chain

This configuration option specifies the IR size, TAPs, and position for each device in the JTAG chain. Currently JTAG chain support will be added in the future. These settings should be kept at default value for each vendor, therefore they do not need to be specified.

2.8.1 JTAG_Chain_pos

Position of device currently being programmed within the chain.

2.8.2 JTAG_Chain_size

Total number of devices in JTAG chain.

2.8.3 JtagIRsizeDevice-1

Number of bits in Instruction Register (IR) for device one.

2.8.4 JtagIRsizeDevice-2

Number of bits in Instruction Register (IR) for device two.

2.8.5 JtagIRsizeDevice-3

Number of bits in Instruction Register (IR) for device three.

2.8.6 JtagIRsizeDevice-4

Number of bits in Instruction Register (IR) for device four.

2.8.7 JtagIRsizeDevice-5

Number of bits in Instruction Register (IR) for device five.

2.8.8 JtagIRsizeDevice-6

Number of bits in Instruction Register (IR) for device six.

2.8.9 JtagTAPsDevice-1

Number of TAPs for device one.

2.8.10 JtagTAPsDevice-2

Number of TAPs for device two.

2.8.11 JtagTAPsDevice-3

Number of TAPs for device three.

2.8.12 JtagTAPsDevice-4

Number of TAPs for device four.

2.8.13 JtagTAPsDevice-5

Number of TAPs for device five.

2.8.14 JtagTAPsDevice-6

Number of TAPs for device six.

2.8.15 JtagSelDevice-1

Index of MCU on position one within the JTAG chain. This is an index for the FPA's processor list that identifies a particular MCU.

2.8.16 JtagSelDevice-2

Index of MCU on position two within the JTAG chain. This is an index for the FPA's processor list that identifies a particular MCU.

2.8.17 JtagSelDevice-3

Index of MCU on position three within the JTAG chain. This is an index for the FPA's processor list that identifies a particular MCU.

2.8.18 JtagSelDevice-4

Index of MCU on position four within the JTAG chain. This is an index for the FPA's processor list that identifies a particular MCU.

2.8.19 JtagSelDevice-5

Index of MCU on position five within the JTAG chain. This is an index for the FPA's processor list that identifies a particular MCU.

2.8.20 JtagSelDevice-6

Index of MCU on position six within the JTAG chain. This is an index for the FPA's processor list that identifies a particular MCU.

2.9 Memory Protection for Texas Instruments-ARM

Configuration options specified in this section can be configured visually using the GUI dialog under Setup, Memory Protection. None of the options in this section will be used unless WriteLocking-BitsEn is enabled. The WriteLockingBitsEn can be visually set in the main GUI window, Memory Protection, Enable check box.

2.9.1 WriteLockingBitsEn

This option specifies whether memory protection and debug access locations should be modified. If disabled, functions such as F_AutoProgram will only program code to Flash and OTP Memory. If disabled, the functions F_Lock_MCU, F_Write_Locking_Registers and F_Write_Debug_Register will do nothing.

Logically grouped together with all parameters in this section.

- 1 : enabled
- 0 : disabled

2.9.2 LM_MemoryProtSource

This configuration option specifies whether the configuration file should be used as the source of memory protection bit values, or if the code file should be used as the source. The options specified in the main GUI are the same ones saved in the configuration file (User-defined option). If the code file is used as the source, the protection bits must be specified at the correct address within the code file. Correct addresses can be obtained from device manuals. The GUI provides an option to inspect memory protection bits as read from the code file (From File option).

If the FILE_SOURCE value is chosen for this parameter, the only parameters in this section that will also be read are LM_USER_REG_WrEn and LM_USER_DBG_WrEn. The former enables writes to USER_REGs and the latter must be enabled to write to the register that controls communication debug access.

Logically grouped together will all _REG and Data parameters in this section.

- USER_SOURCE(0) : Protection settings from the configuration file will be used (can be user defined in GUI).
- FILE_SOURCE(1) : Protection settings from code file will be used (code file must include protection bits at proper addresses).

2.9.3 LM_USER_DBG_WrEn

This configuration parameter enables writes to the register that controls communication debug access.

Logically grouped together with LM-UserDebugData.

- 1 : enabled
- 0 : disabled

2.9.4 LM-UserDebugData

Value that will be written to the BOOTCFG register if LM_USER_DBG_WrEn is enabled, and LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.5 LM_USER_REG_WrEn

This configuration parameter enables writes to USER REGs.

Logically grouped together with LM-USER_REG0-3

- 1 : enabled
- 0 : disabled

2.9.6 LM-USER_REG0

Value that will be written to the USER_REG0 register if LM_USER_REG_WrEn is enabled, and LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.7 LM-USER_REG1

Value that will be written to the USER_REG1 register if LM_USER_REG_WrEn is enabled, and LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.8 LM-USER_REG2

Value that will be written to the USER_REG2 register if LM_USER_REG_WrEn is enabled, and LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.9 LM-USER_REG3

Value that will be written to the USER_REG3 register if LM_USER_REG_WrEn is enabled, and LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.10 LM-FMPReadEn0-Data

Value that will be written to the FMPRE0 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.11 LM-FMPReadEn1-Data

Value that will be written to the FMPRE1 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.12 LM-FMPReadEn2-Data

Value that will be written to the FMPRE2 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.13 LM-FMPReadEn3-Data

Value that will be written to the FMPRE3 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.14 LM-FMPReadEn4-Data

Value that will be written to the FMPRE4 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.15 LM-FMPReadEn5-Data

Value that will be written to the FMPRE5 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.16 LM-FMPReadEn6-Data

Value that will be written to the FMPRE6 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.17 LM-FMPReadEn7-Data

Value that will be written to the FMPRE7 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.18 LM-FMPReadEn8-Data

Value that will be written to the FMPRE8 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.19 LM-FMPReadEn9-Data

Value that will be written to the FMPRE9 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.20 LM-FMPReadEn10-Data

Value that will be written to the FMPRE10 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.21 LM-FMPReadEn11-Data

Value that will be written to the FMPRE11 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.22 LM-FMPReadEn12-Data

Value that will be written to the FMPRE12 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.23 LM-FMPReadEn13-Data

Value that will be written to the FMPRE13 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.24 LM-FMPReadEn14-Data

Value that will be written to the FMPRE14 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.25 LM-FMPReadEn15-Data

Value that will be written to the FMPRE15 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.26 LM-FMPPrgEn0-Data

Value that will be written to the FMPPE0 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.27 LM-FMPPrgEn1-Data

Value that will be written to the FMPPE1 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.28 LM-FMPPrgEn2-Data

Value that will be written to the FMPPE2 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.29 LM-FMPPrgEn3-Data

Value that will be written to the FMPPE3 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.30 LM-FMPPrgEn4-Data

Value that will be written to the FMPPE4 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.31 LM-FMPPrgEn5-Data

Value that will be written to the FMPPE5 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.32 LM-FMPPrgEn6-Data

Value that will be written to the FMPPE6 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.33 LM-FMPPrgEn7-Data

Value that will be written to the FMPPE7 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.34 LM-FMPPrgEn8-Data

Value that will be written to the FMPPE8 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.35 LM-FMPPrgEn9-Data

Value that will be written to the FMPPE9 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.36 LM-FMPPrgEn10-Data

Value that will be written to the FMPPE10 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.37 LM-FMPPrgEn11-Data

Value that will be written to the FMPPE11 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.38 LM-FMPPrgEn12-Data

Value that will be written to the FMPPE12 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.39 LM-FMPPrgEn13-Data

Value that will be written to the FMPPE13 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.40 LM-FMPPrgEn14-Data

Value that will be written to the FMPPE14 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.9.41 LM-FMPPrgEn15-Data

Value that will be written to the FMPPE15 register if LM_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10 Memory Protection for Texas Instruments-CC

Configuration options specified in this section can be configured visually using the GUI dialog under Setup, Memory Protection. None of the options in this section will be used unless WriteLocking-BitsEn is enabled. The WriteLockingBitsEn can be visually set in the main GUI window, Memory Protection, Enable check box.

2.10.1 WriteLockingBitsEn

This option specifies whether memory protection and debug access locations should be modified. If disabled, functions such as F_AutoProgram will only program code to Flash and OTP Memory. If disabled, the functions F_Lock_MCU, F_Write_Locking_Registers and F_Write_Debug_Register will do nothing.

Logically grouped with all parameters in this section.

- 1 : enabled
- 0 : disabled

2.10.2 CC_MemoryProtSource

This configuration option specifies whether the configuration file should be used as the source of memory protection bit values, or if the code file should be used as the source. This parameter only applies to CC 25xx MCUs. The options specified in the main GUI are the same ones saved in the configuration file (User-defined option). If the code file is used as the source, the protection bits must be specified at the correct address within the code file. Correct addresses can be obtained from device manuals. The GUI provides an option to inspect memory protection bits as read from the code file (From File option).

If the FILE_SOURCE value is chosen for this parameter, the only parameter in this section that will also be read is CC_USER_DBG_WrEn. That parameter must be enabled to write to the register that controls communication debug access.

Logically grouped together will all -Data parameters in this section.

- USER_SOURCE(0) : Protection settings from the configuration file will be used (can be user defined in GUI).
- FILE_SOURCE(1) : Protection settings from code file will be used (code file must include protection bits at proper addresses).

2.10.3 CC_USER_DBG_WrEn

This configuration parameter enables writes to LOCKPAGE bit 255 that controls communication debug access.

Logically grouped together with CC-LOCKPAGE7-Data.

- 1 : enabled
- 0 : disabled

2.10.4 CC-LOCKPAGE0-Data

Value that will be written to LOCKPAGE bits 0-31 if CC_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.5 CC-LOCKPAGE1-Data

Value that will be written to LOCKPAGE bits 32-63 if CC_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.6 CC-LOCKPAGE2-Data

Value that will be written to LOCKPAGE bits 64-95 if CC_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.7 CC-LOCKPAGE3-Data

Value that will be written to LOCKPAGE bits 96-127 if CC_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.8 CC-LOCKPAGE4-Data

Value that will be written to LOCKPAGE bits 128-159 if CC_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.9 CC-LOCKPAGE5-Data

Value that will be written to LOCKPAGE bits 160-191 if CC_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.10 CC-LOCKPAGE6-Data

Value that will be written to LOCKPAGE bits 192-223 if CC_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.11 CC-LOCKPAGE7-Data

Value that will be written to LOCKPAGE bits 224-255 if CC_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

Bit 255 controls communication debug access to the MCU. Will be written if CC_USER_DBG_WrEn is enabled. Ignored otherwise.

2.10.12 CC_26xx_MemoryProtSource

This configuration option specifies whether the configuration file should be used as the source of memory protection bit values, or if the code file should be used as the source. This parameter only applies to CC 26xx MCUs. The options specified in the main GUI are the same ones saved in the configuration file (User-defined option). If the code file is used as the source, the protection bits must be specified at the correct address within the code file. Correct addresses can be obtained from device manuals. The GUI provides an option to inspect memory protection bits as read from the code file (From File option).

If the FILE_SOURCE value is chosen for this parameter, the only parameters in this section that will also be read are CC_26xx_GenOpt_WrEn and CC_26xx_USER_DBG_WrEn. Those parameters must be enabled to write to general configuration registers and the registers that control communication debug access.

Logically grouped together will all -CCFG- parameters in this section.

- USER_SOURCE(0) : Protection settings from the configuration file will be used (can be user defined in GUI).
- FILE_SOURCE(1) : Protection settings from code file will be used (code file must include protection bits at proper addresses).

2.10.13 CC_26xx_GenOpt_WrEn

This configuration parameter enables writes to IEEE_MAC_0, IEEE_MAC_1, IEEE_BLE_0, IEEE_BLE_1, BL_CONFIG, and ERASE_CONF registers.

Logically grouped together with CC-26xx-CCFG-IEEE-MAC-0, CC-26xx-CCFG-IEEE-MAC-1, CC-26xx-CCFG-IEEE-BLE-0, CC-26xx-CCFG-IEEE-BLE-1, CC-26xx-CCFG-BL-CONFIG, CC-26xx-CCFG-ERASE-CONF.

- 1 : enabled
- 0 : disabled

2.10.14 CC_26xx_USER_DBG_WrEn

This configuration parameter enables writes to CCFG_TI_BACKDOOR, CCFG_TAP_DAP_0, and CCFG_TAP_DAP_1 registers.

Logically grouped together with CC-26xx-CCFG-TI-BACKDOOR, CC-26xx-CCFG-TAP-DAP-0, CC-26xx-CCFG-TAP-DAP-1.

- 1 : enabled
- 0 : disabled

2.10.15 CC-26xx-CCFG-31-0

Value that will be written to the CCFG_PROT_31_0 register if CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.16 CC-26xx-CCFG-63-32

Value that will be written to the CCFG_PROT_63_32 register if CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.17 CC-26xx-CCFG-95-64

Value that will be written to the CCFG_PROT_95_64 register if CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.18 CC-26xx-CCFG-127-96

Value that will be written to the CCFG_PROT_127_96 register if CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.19 CC-26xx-CCFG-IEEE-MAC-0

Value that will be written to the IEEE_MAC_0 register if CC_26xx_GenOpt_WrEn is enabled, and CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.20 CC-26xx-CCFG-IEEE-MAC-1

Value that will be written to the IEEE_MAC_1 register if CC_26xx_GenOpt_WrEn is enabled, and CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.21 CC-26xx-CCFG-IEEE-BLE-0

Value that will be written to the IEEE_BLE_0 register if CC_26xx_GenOpt_WrEn is enabled, and CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.22 CC-26xx-CCFG-IEEE-BLE-1

Value that will be written to the IEEE_BLE_1 register if CC_26xx_GenOpt_WrEn is enabled, and CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.23 CC-26xx-CCFG-BL-CONFIG

Value that will be written to the BL_CONFIG register if CC_26xx_GenOpt_WrEn is enabled, and CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.24 CC-26xx-CCFG-ERASE-CONF

Value that will be written to the ERASE_CONF register if CC_26xx_GenOpt_WrEn is enabled, and CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.25 CC-26xx-CCFG-TI-BACKDOOR

Value that will be written to the CCFG_TI_BACKDOOR register if CC_26xx_USER_DBG_WrEn is enabled, and CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.26 CC-26xx-CCFG-TAP-DAP-0

Value that will be written to the CCFG_TAP_DAP_0 register if CC_26xx_USER_DBG_WrEn is enabled, and CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.10.27 CC-26xx-CCFG-TAP-DAP-1

Value that will be written to the CCFG_TAP_DAP_1 register if CC_26xx_USER_DBG_WrEn is enabled, and CC_26xx_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.11 Memory Protection for ST Microelectronics

Configuration options specified in this section can be configured visually using the GUI dialog under Setup, Memory Protection. None of the options in this section will be used unless WriteLockingBitsEn is enabled. The WriteLockingBitsEn can be visually set in the main GUI window, Memory Protection, Enable check box.

2.11.1 WriteLockingBitsEn

This option specifies whether memory protection and debug access locations should be modified. If disabled, functions such as F_AutoProgram will only program code to Flash and OTP Memory. If disabled, the functions F_Lock_MCU, F_Write_Locking_Registers and F_Write_Debug_Register will do nothing.

Logically grouped with all parameters in this section.

- 1 : enabled
- 0 : disabled

2.11.2 STM32_MemoryProtSource

This configuration option specifies whether the configuration file should be used as the source of memory protection bit values, or if the code file should be used as the source. The options specified in the main GUI are the same ones saved in the configuration file (User-defined option). If the code file is used as the source, the protection bits must be specified at the correct address within the code file. Correct addresses can be obtained from device manuals. The GUI provides an option to inspect memory protection bits as read from the code file (From File option).

If the FILE_SOURCE value is chosen for this parameter, the only parameters in this section that will also be read are STM32_USER_WrEn and STM32_RDP_WrEn. The former enables writes to USER_REGs and the latter must be enabled to write to the register that controls communication debug access.

Logically grouped together will all _Data parameters in this section.

- USER_SOURCE(0) : Protection settings from the configuration file will be used (can be user defined in GUI).
- FILE_SOURCE(1) : Protection settings from code file will be used (code file must include protection bits at proper addresses).

2.11.3 STM32_USER_WrEn

This configuration parameter enables writes to option bytes and data registers.

Logically grouped together with STM32-USERREG_Data, STM32_Data0, and STM32_Data1.

2.11.4 STM32-USERREG_Data

Value that will be written to the USER register if STM32_USER_WrEn is enabled, and STM32_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.11.5 STM32_Data0

Value that will be written to the DATA0 register if STM32_USER_WrEn is enabled, and STM32_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.11.6 STM32_Data1

Value that will be written to the USER register if STM32_USER_WrEn is enabled, and STM32_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.11.7 STM32_RDP_WrEn

This configuration parameter enables writes to the register that controls reading of Flash Memory and communication debug access.

Logically grouped together with STM32_RDP_Data.

- 1 : enabled
- 0 : disabled

2.11.8 STM32_RDP_Data

Value that will be written to the RDP register if STM32_RDP_WrEn is enabled, and STM32_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.11.9 STM32_WRP0_Data

Value that will be written to the WRP0 or nWRP (depends on family) register if STM32_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.11.10 STM32_WRP1_Data

Value that will be written to the WRP1 register if STM32_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.11.11 STM32_WRP2_Data

Value that will be written to the WRP2 register if STM32_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.11.12 STM32_WRP3_Data

Value that will be written to the WRP3 register if STM32_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.12 Memory Protection for Silicon Labs

Configuration options specified in this section can be configured visually using the GUI dialog under Setup, Memory Protection. None of the options in this section will be used unless WriteLocking-BitsEn is enabled. The WriteLockingBitsEn can be visually set in the main GUI window, Memory Protection, Enable check box.

2.12.1 WriteLockingBitsEn

This option specifies whether memory protection and debug access locations should be modified. If disabled, functions such as F_AutoProgram will only program code to Flash and OTP Memory. If disabled, the functions F_Lock_MCU, F_Write_Locking_Registers and F_Write_Debug_Register will do nothing.

Logically grouped with all parameters in this section.

- 1 : enabled
- 0 : disabled

2.12.2 SL_MemoryProtSource

This configuration option specifies whether the configuration file should be used as the source of memory protection bit values, or if the code file should be used as the source. The options specified in the main GUI are the same ones saved in the configuration file (User-defined option). If the code file is used as the source, the protection bits must be specified at the correct address within the code file. Correct addresses can be obtained from device manuals. The GUI provides an option to inspect memory protection bits as read from the code file (From File option).

If the FILE_SOURCE value is chosen for this parameter, the only parameter in this section that will also be read is SL_USER_DBG_WrEn. That parameter must be enabled to write to the register that controls communication debug access.

Logically grouped together will all -xLW and -Data parameters in this section.

- USER_SOURCE(0) : Protection settings from the configuration file will be used (can be user defined in GUI).
- FILE_SOURCE(1) : Protection settings from code file will be used (code file must include protection bits at proper addresses).

2.12.3 SL_USER_DBG_WrEn

This configuration parameter enables writes to the register that controls communication debug access.

Logically grouped together with SL-DLW.

- 1 : enabled
- 0 : disabled

2.12.4 SL-DLW

Value that will be written to the DLW location on the protection page if SL_USER_DBG_WrEn is enabled, and SL_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.12.5 SL-ULW

Value that will be written to the ULW location on the protection page if SL_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.12.6 SL-MLW

Value that will be written to the MLW location on the protection page if SL_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.12.7 SL-FMPReadEn0-Data

Value that will be written to bits 0:31 of the PLW location on the protection page if SL_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.12.8 SL-FMPReadEn1-Data

Value that will be written to bits 32:63 of the PLW location on the protection page if SL_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.12.9 SL-FMPReadEn2-Data

Value that will be written to bits 64:95 of the PLW location on the protection page if SL_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.12.10 SL-FMPReadEn3-Data

Value that will be written to bits 96:127 of the PLW location on the protection page if SL_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.12.11 SL-FMPReadEn4-Data

Value that will be written to bits 128:159 of the PLW location on the protection page if SL_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.12.12 SL-FMPReadEn5-Data

Value that will be written to bits 160:191 of the PLW location on the protection page if SL_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.12.13 SL-FMPReadEn6-Data

Value that will be written to bits 192:223 of the PLW location on the protection page if SL_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.12.14 SL-FMPReadEn7-Data

Value that will be written to bits 224:255 of the PLW location on the protection page if SL_MemoryProtSource is set to USER_SOURCE. Ignored otherwise.

2.13 Memory Protection for Active-Semi

Configuration options specified in this section can be configured visually using the GUI dialog under Setup, Memory Protection. None of the options in this section will be used unless WriteLocking-BitsEn is enabled. The WriteLockingBitsEn can be visually set in the main GUI window, Memory Protection, Enable check box.

2.13.1 WriteLockingBitsEn

This option specifies whether memory protection and debug access locations should be modified. If disabled, functions such as F_AutoProgram will only program code to Flash and OTP Memory. If disabled, the functions F_Lock_MCU, F_Write_Locking_Registers and F_Write_Debug_Register will do nothing.

Logically grouped together with all parameters in this section.

- 1 : enabled
- 0 : disabled

2.13.2 AS_DBG_WrEn

This configuration parameter enables writes to the register that controls communication debug access.

Logically grouped together with AS_SWD_Protection.

- 1 : enabled
- 0 : disabled

2.13.3 AS_SWD_Protection

If this option is enabled, and AS_DBG_WrEn is enabled, communication debug access to the MCU will be permanently disabled.

- 1 : enabled
- 0 : disabled

Chapter 3

DLL functions

3.1 Multi API-DLL Functions

These functions operate on meta-data within the Multi API-DLL and help manage the underlying API-DLL instances. These functions do not directly program target devices. Use these functions to initialize the desired number of FPAs, selected all or one FPA(s), read FPA serial numbers, or clean-up resources used by Elprotronic's DLLs.

Once the desired FPAs have been successfully opened, use Generic Functions (Section 3.2) to initialize the FPAs, and configure them for the target device(s) being programmed.

3.1.1 F_OpenInstancesAndFPAs

General Description

Multi API-DLL scans USB ports for connected FPAs listed in the setup file, or input string. When an FPA listed in the setup file is found, the corresponding API-DLL is copied on disk if necessary, and loaded. F_Initialization should be called for each FPA after this function succeeds.

IMPORTANT

Do not invoke F_Check_FPA_access after this function has assigned FPAs to USB ports. To check simple communication with FPA, use the F_Get_FPA_SN function.

Syntax

```
INT_X F_OpenInstancesAndFPAs( char * FileName )
```

Input

char * FileName : path to setup file, or list of serial numbers.

Setup file: Should contain a list of FPA and SN pairs.

```
FPA-1 20090123
FPA-3 20090234
--or--
FPA-1 20090123
FPA-3 *   //any serial number (can only be done at last line)

--INVALID--
FPA-1 *  //this line will be read
FPA-3 *  //this line will be ignored
```

List of serial numbers: A string with serial numbers, automatically assigned to FPA-1, 2, 3, etc.

```
input string: "*# 20090123 20090346"  translates to:
FPA-1 20090123
FPA-2 20090345
input string: "*# *" translates to:
FPA-1 *
```

In both cases, for the setup file and the input string, if a specified FPA is missing then it will not affect subsequent entries. Therefore, a setup file with these contents:

```
FPA-1 20090123
FPA-2 20090346
FPA-3 20090222
FPA-4 20090245
```

and with FPA-3 missing (not connected, etc.) will initialize the Multi API-DLL to:

```
FPA-1 20090123
FPA-2 20090346
FPA-3 empty
FPA-4 20090245
```

Output

INT_X : number of instances opened successfully

3.1.2 F_CloseInstances

General Description

All FPAs terminate communication with target devices and close target devices according to configuration settings (power setting, adapter line states, etc.). Finally, the USB connections to the FPAs are terminated and API-DLL instances are freed. The Multi API-DLL can now be used to open a new set of FPAs using the function F_OpenInstancesAndFPAs.

Syntax

```
INT_X F_CloseInstances ( void );
```

Input

none.

Output

INT_X : success

- TRUE (1)

3.1.3 F_Set_FPA_index

General Description

Select desired FPA index to perform specific tasks (access specific API-DLL instance).

Syntax

```
INT_X F_Set_FPA_index ( BYTE fpa );
```

Input

BYTE fpa : desired FPA index, 1 to 64, or 0 for all

Output

INT_X : success or error

- TRUE (1), used FPA index is valid
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

IMPORTANT

Other functions that try to access FPAs will also return FPA_INVALID_NO if this function was not called with a proper parameter.

3.1.4 F_Get_FPA_index

General Description

Get current FPA index.

Syntax

```
BYTE F_Get_FPA_index( void );
```

Input

none.

Output

BYTE : current FPA index as entered using F_Set_FPA_index

3.1.5 F_Check_FPA_index

General Description

Get current FPA index and check if index is valid. A valid index corresponds to an individual FPA that has been opened with F_OpenInstancesAndFPAs. If FPA index was set to 0 (all), then this function will only return 0, even if no FPAs are open. Does not indicate whether FPA index is enabled or disabled.

Syntax

```
INT_X F_Check_FPA_index ( void );
```

Input

none.

Output

INT_X : current fpa index if valid

- current FPA index as entered using F_Set_FPA_index if selected FPA has been opened with F_OpenInstancesAndFPAs
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.1.6 F_Enable_FPA_index

General Description

Enable desired FPA index to accept commands. An FPA index is enabled by default. A disabled FPA index will ignore commands when the FPA index is set to it, or 0 (all FPAs).

Syntax

```
void F_Enable_FPA_index ( BYTE fpa );
```

Input

BYTE fpa : desired FPA index, 1 to 64

Output

none.

3.1.7 F_Disable_FPA_index

General Description

Disable desired FPA index to ignore commands. An FPA index is enabled by default. A disabled FPA index will ignore commands when the FPA index is set to it, or 0 (all FPAs).

Syntax

```
void F_Disable_FPA_index ( BYTE fpa );
```

Input

BYTE fpa : desired FPA index, 1 to 64

Output

none.

3.1.8 F_LastStatus

General Description

Return value from the last function call issued to the specified FPA (API-DLL instance). This function is useful when multiple FPAs are being programmed using FPA index 0 (all), but the return value was not the same for all FPAs (i.e. 4 FPAs succeeded, 1 FPA failed). When the return value is not the same for all FPAs then, then the actual return value from the Multi API-DLL will be FPA_UNMATCHED_RESULTS. To find out per FPA return values, select each FPA index individually using F_Set_FPA_index, and call F_LastStatus for each index.

Syntax

```
INT_X F_LastStatus ( BYTE fpa );
```

Input

BYTE fpa : desired FPA index, 1 to 64.

Output

INT_X : Last return value from the selected FPA (API-DLL instance).

- INT_X type return value from last function call on selected FPA
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.1.9 F_Multi_DLLTypeVer

General Description

Get Multi API-DLL software version number.

Syntax

```
INT_X F_Multi_DLLTypeVer( void );
```

Input

none.

Output

INT_X : (DLL_ID) — (0x0FFF & Version)

- DLL_ID = 0x06000, FlashPro430
- DLL_ID = 0x07000, GangPro430
- DLL_ID = 0x08000, FlashPro-CC
- DLL_ID = 0x09000, GangPro-CC
- DLL_ID = 0x0C000, FlashPro2000
- DLL_ID = 0x0D000, GangPro2000
- DLL_ID = 0x10000, FlashPro-ARM
- DLL_ID = 0x11000, GangPro-ARM

- Version = 0x0xyz, version x.yz (i.e. 0x0190 = v 1.90)
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.1.10 F_Get_FPA_SN

General Description

Get serial number (SN) of FPA assigned to specified index.

Syntax

```
INT_X F_Get_FPA_SN ( BYTE fpa );
```

Input

BYTE fpa : desired FPA index, 1 to 64

Output

INT_X : serial number

- SN of selected FPA
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.1.11 F_Get_FPA_Label

General Description

Get detailed information of FPA assigned to specified index.

Syntax

```
INT_X F_Get_FPA_Label ( BYTE fpa, BYTE *label );
```

Input

BYTE fpa : desired FPA index, 1 to 64.
BYTE *label : pointer to byte array, of at least FPA_LABEL_SIZE (80 bytes)

Output

INT_X : serial number, hardware ID, meta-data

- updated FPA_LABEL structure pointed to by label input parameter
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

```
#define FPA_LABEL_SIZE 80
#define ADAPTER_HARDWARE_PN_SIZE 16
#define ADAPTER_DESCRIPTION_SIZE 32
union FPA_LABEL
{
  BYTE bytes[ FPA_LABEL_SIZE ];
  struct
  {
    long SN;
    BYTE hardware_ID;
    BYTE hardware_SubID;
    BYTE full_access;
    BYTE Interface_Type;
    BYTE Valid_access_key;
    BYTE spare3;
    BYTE spare4;
    BYTE spare5;
    BYTE spare6;
    BYTE spare7;
    BYTE spare8;
    BYTE spare9;
    char adapter_hardware_PN[ ADAPTER_HARDWARE_PN_SIZE ];
    char adapter_description[ ADAPTER_DESCRIPTION_SIZE ];
  }x;
};
```

3.1.12 F_GetProgressBar

General Description

Get internal progress counter value for operations done inside Encapsulated Functions (Section 3.4). When used in conjunction with F_GetLastOpCode it allows the user application to keep track of progress in the same way that the GangPro-ARM GUI does.

Will return value between 0 and 100 for current sub-operation being performed inside encapsulated function. Depending on the complexity of the encapsulated function, the number of

sub-operations will vary. Use F_GetLastOpCode to get currently running sub-operation opcode. When transitioning from one sub-operation that reached a progress value of 100, to another sub-operation, the progress value will restart at 0. This will repeat until the last sub-operation upon which the progress value will remain at 100. A new invocation of an encapsulated function will restart the progress value at 0.

This function is thread-safe, therefore it can be called while the Multi API-DLL is busy, for example running F_AutoProgram. Intended usage is to call an encapsulated function with one thread, and repeatedly call this function and F_GetLastOpCode with another thread. When simultaneously programming using multiple FPAs (fpa index set to 0 (ALL_FPAs), the thread monitoring progress can iterate different input parameters (fpa=1, fpa=2, fpa=3, etc.) to monitor the progress of each FPA individually. This function cannot be called with parameter 0 (ALL_FPAs).

Syntax

```
INT_X  MSPPRG_API F_GetProgressBar( BYTE fpa );
```

Input

BYTE fpa : desired FPA index, 1 to 64.

Output

INT_X : progress indicator

- value between 0 and 100
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

Example

Refer to GetLastOpCode.

3.1.13 F_GetLastOpCode

General Description

Read internal opcode value for sub-operations done inside Encapsulated Functions (Section 3.4). When used in conjunction with F_GetProgressBar it allows the user application to keep track of progress in the same way that the GangPro-ARM GUI does.

Will return opcode for current sub-operation being performed inside encapsulated function. Depending on the complexity of the encapsulated function, the number of sub-operations will vary. Use F_GetProgressBar to get progress value between 0 and 100, of currently running sub-operation. When transitioning from one sub-operation that reached a progress value of 100, to

another sub-operation, the progress value will restart at 0 and the opcode will be set to the new sub-operation, currently being run. This will repeat until the last sub-operation upon which the opcode will not change. A new invocation of an encapsulated function will reset the opcode to the first sub-operation of the new encapsulated function.

This function is thread-safe, therefore it can be called while the Multi API-DLL is busy, for example running F_AutoProgram. Intended usage is to call an encapsulated function with one thread, and repeatedly call this function and F_GetProgressBar with another thread. When simultaneously programming using multiple FPAs (fpa index set to 0 (ALL_FPAs), the thread monitoring progress can iterate different input parameters (fpa=1, fpa=2, fpa=3, etc.) to monitor the progress of each FPA individually. This function cannot be called with parameter 0 (ALL_FPAs).

Syntax

```
INT_X  MSPPRG_API F_GetLastOpCode( BYTE fpa );
```

Input

BYTE fpa : desired FPA index, 1 to 64.

Output

INT_X : opcode of currently running sub-operation

- PROG_OPCODE_VERIFY_ACCESS (1) : FPA is attempting to perform action described in F_Verify_Access_to_MCU.
- PROG_OPCODE_FLASH_ERASE (2) : FPA is attempting to perform action described in F_Memory_Erase.
- PROG_OPCODE_FLASH_BLANK_CHECK (3) : FPA is attempting to perform action described in F_Memory_Blank_Check.
- PROG_OPCODE_FLASH_SELECTED_BLANK_CHECK (4) : FPA is attempting to perform action described in F_Memory_Blank_Check when a subset of the memory space is selected in configuration settings.
- PROG_OPCODE_FLASH_WRITE (5) : FPA is attempting to perform action described in F_Memory_Write.
- PROG_OPCODE_FLASH_VERIFY (6) : FPA is attempting to perform action described in F_Memory_Verify.
- PROG_OPCODE_FLASH_READ (7) : FPA is attempting to perform action described in F_Memory_Read.

- PROG_OPCODE_LOCK_MCU (8) : FPA is attempting to perform action described in F_Lock_MCU.
- PROG_OPCODE_UNLOCK_MCU (9) : FPA is attempting to perform action described in F_Clear_Locked_Device.
- PROG_OPCODE_START_APP (10) : FPA is attempting to run application programmed onto target device (if enabled to be ran after F_AutoProgram in configuration settings).
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

Example

```
void ProgDemoDlg::UpdateProgress()
{
  //This function works a bit different than all the others.
  //Usually the Multi-API DLL is not thread safe and
  //only one thread can enter the Multi-API DLL.
  //This function is thread-safe because it is read-only. FPA index != 0.
  if(fpa == 0)
  {
    //You can average out the results here,
    //or do separate bars per FPA.
    //For AutoProgram the progress bar will go from 0 to 100 many
    //times during one run because of sub-opcodes.
    return;
  }

  INT_X progValue = F_GetProgressBar(fpa);
  prog->SetPos(progValue);

  INT_X lastOpCode = F_GetLastOpCode(fpa);
  CString opCodeUpdate = "Unknown";
  switch(lastOpCode)
  {
    case PROG_OPCODE_VERIFY_ACCESS:
      opCodeUpdate = "Verify Access To MCU";
      break;
    case PROG_OPCODE_FLASH_ERASE:
      opCodeUpdate = "Flash Erase";
      break;
    case PROG_OPCODE_FLASH_BLANK_CHECK:
```

```
      opCodeUpdate = "Flash Blank Check";
      break;
    case PROG_OPCODE_FLASH_SELECTED_BLANK_CHECK:
      opCodeUpdate = "Selected Memory Blank Check";
      break;
    case PROG_OPCODE_FLASH_WRITE:
      opCodeUpdate = "Flash Write";
      break;
    case PROG_OPCODE_FLASH_VERIFY:
      opCodeUpdate = "Flash Verify";
      break;
    case PROG_OPCODE_FLASH_READ:
      opCodeUpdate = "Flash Read";
      break;
    case PROG_OPCODE_LOCK_MCU:
      opCodeUpdate = "Lock MCU";
      break;
    case PROG_OPCODE_UNLOCK_MCU:
      opCodeUpdate = "Unlock MCU";
      break;
    case PROG_OPCODE_START_APP:
      opCodeUpdate = "Start Application";
      break;
  }
  CString currentOpCode = "";
  GetDlgItemText(IDC_OPCODE, currentOpCode);

  //Don't update if it's the same text (prevents flashing)
  if(currentOpCode.Compare(opCodeUpdate) != 0)
    SetDlgItemText(IDC_OPCODE, opCodeUpdate);
}
```

3.1.14 Trace_ON

General Description

Activate tracing for subsequent calls to the Multi API-DLL. Log is saved in DLLtrace.txt located in the Multi API-DLL directory. When activated, records all API-DLL function calls from the application software invoked via the Multi API-DLL. The DLLtrace.txt is overwritten for each new session.

Syntax

```
void F_Trace_ON( void );
```

Input

none.

Output

none.

3.1.15 Trace_OFF

General Description

Disable tracing.

Syntax

```
void F_Trace_OFF( void );
```

Input

none.

Output

none.

3.2 Generic Functions

Having correctly configured the top-level Multi API-DLL using Multi API-DLL Functions (Section 3.1), the functions in this section can be used to initialize and configure each FPA individually (and it's API-DLL instance).

Generic functions read and write the API-DLL's configuration, access status messages and can instruct the FPA to power or reset the target device. The functions described in this section require that each FPA already be opened using F_OpenInstancesAndFPAs.

After the following functions have been used to configure each FPA correctly for its target device, use Data Buffer Functions (Section 3.3) to determine what code should be written during programming. Finally, Encapsulated functions can perform the process of programming, writing, erasing, verifying, etc., according to the aforementioned settings.

3.2.1 F_Initialization

General Description

Initialize FPA after it has been successfully opened using F_OpenInstancesAndFPAs. F_Initialization performs the following tasks:

- all internal data is cleared and set to default values,
- USB driver is initialized if has not been initialized before.

Select which FPA to initialize, using the function F_Set_FPA_index with desired parameter (0-all, 1-64 for individual FPA). After successful initialization, call F_ConfigFileLoad to read settings from file, or configure settings manually using the function F_Set_Config_Value_By_Name.

For backwards compatibility, F_Initialization will call F_OpenInstancesAndFPAs(*# *) if no instances are currently opened, and perform initialization on FPA 1. If F_OpenInstancesAdFPAs was already invoked successfully then initialization is performed on the selected FPA index from the function F_Set_FPA_index.

Default config file used to perform initialization is config.ini (Use F_Use_Config_INI(FALSE) to disable automatic load of config.ini).

Syntax

```
INT_X F_Initialization( void );
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- 4 : Programming adapter not detected
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.2.2 F_Use_Config_INI

General Description

Configure F_Initialization to use (default) or skip config.ini file. When skipping config.ini it is necessary to use ConfigFileLoad or F_Set_Config_Value_By_Name functions to configure the adapter.

Syntax

```
INT_X F_Use_Config_INI(BYTE use);
```

Input

BYTE use : 1 to use config.ini, 0 to skip

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.2.3 F_Get_Config_Name_List

General Description

Iterate over configuration parameter names, starting from index 0. Increase index until return is null character. Will return a pointer to a character array with name of parameter or null character once index is too high.

Requires the target FPA to be opened using F_OpenInstancesAndFPAs and initialized using F_Initialization.

Syntax

```
char * F_Get_Config_Name_List( INT_X index );
```

Input

INT_X : configuration parameter index, starting from zero. Increment until return value is null character.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.

Output

char * : configuration parameter name

- a valid character array pointer containing name of configuration parameter (do not free after use)
- null character : end of parameter list
- "" : invalid fpa index, cannot be zero

3.2.4 F_Get_Config_Value_By_Name

General Description

Access the current FPA's configuration. Use F_Get_Config_Name_List to get configuration parameter names, then use them as input to this function. Select input type to validate name, and get current, minimum, maximum, and default values.

Because normal return values from this function can be any value, from 0 to 0xFFFFFFFF, it is important to pick a correct fpa index (use F_Check_FPA_index to validate) and validate the configuration parameter name first (using CONFSEL_VALIDATE), before retrieving the current, minimum, maximum or default values. Unless all FPAs are to be configured identically, avoid using FPA index 0 to prevent confusion in results (when results don't match using FPA index 0 (all FPAs) the result is simply FPA_UNMATCHED_RESULTS (-1)).

Requires the target FPA to be opened using F_OpenInstancesAndFPAs and initialized using F_Initialization.

Syntax

```
unsigned int F_Get_Config_Value_By_Name(char *name, INT_X type);
```

Input

char * name : parameter name

INT_X type : select action

- CONFSEL_VALIDATE (0) : returns TRUE if name was found in configuration list

- CONFSEL_VALUE (1) : returns current value of configuration parameter
- CONFSEL_MIN (2) : returns minimum value that this configuration parameter can have
- CONFSEL_MAX (3) : returns maximum value that this configuration parameter can have
- CONFSEL_DEFAULT (4) : returns default value for this configuration parameter if not set by user/file

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

unsigned int : depends on input

- FALSE (0) : failed name validation (CONFSEL_VALIDATE)
- TRUE (1) : successful name validation (CONFSEL_VALIDATE)
- any : actual parameter value, minimum and maximum bounds, and default value if not set by user/file (CONFSEL_VALUE, CONFSEL_MIN, CONFSEL_MAX, CONFSEL_DEFAULT)
- -1 or 0xFFFFFFFF : name not found (avoid this by using CONFSEL_VALIDATE first)
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus(avoid this by not using FPA index 0)
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range(avoid this by using F_Check_FPA_index first)

3.2.5 F_Set_Config_Value_By_Name

General Description

Set the current FPA's configuration. Use F_Get_Config_Name_List to get configuration parameter names, then use them as input to this function. New value for configuration parameter will be trimmed by minimum and maximum values allowed for this parameter. If not initialized, this parameter will have a default value. Double check parameters after configuration using F_Get_Config_Value_By_Name.

Requires the target FPA to be opened using F_OpenInstancesAndFPAs and initialized using F_Initialization.

Syntax

```
INT_X F_Set_Config_Value_By_Name(char *name, unsigned int newValue);
```

Input

char * name : parameter name
unsigned int newValue : assign new configuration parameter value (will be trimmed by min/max if necessary)

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : parameter name not found
- TRUE (1) : name found and parameter value changed
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.2.6 F_Get_Device_Info

General Description

Get information about selected microcontroller.

Syntax

```
INT_X F_Get_Device_Info( INT_X index );
```

Input

INT_X : select type of information to receive

- DEVICE_NAME (0) : get character 0, increase index up to 19 to get last character
- DEVICE_VENDOR_INDEX (30) : Vendor Index (superset of all supported MCUs, i.e. company name)
- DEVICE_FAMILY_INDEX (20) : Family Index (subset of MCU vendor, index 30)
- DEVICE_GROUP_INDEX (21) : Group Index (subset of MCU family, index 20)

- DEVICE_NAME_INDEX (22) : Name Index (subset of MCU group, index 21)
- DEVICE_GROUP (23) : Obsolete
- DEVICE_FLASH_START_ADDR (24) : Flash Start Address, i.e. 0x00000
- DEVICE_FLASH_END_ADDR (25) : Flash End Address, i.e. 0x1FFFF
- DEVICE_OTP_START_ADDR (26) : OTP Start Address, i.e. 0x400FE1E0
- DEVICE_OTP_END_ADDR (27) : OTP End Address, i.e. 0x400FE1EF
- DEVICE_RAM_START_ADDR (28) : RAM Start Address, i.e. 0x20000000
- DEVICE_RAM_END_ADDR (29) : RAM Start Address, i.e. 0x20010000

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.

Output

INT_X : value dependent on input parameter

- any : based on input parameter
- -1 (0xFFFFFFFF) : invalid input parameter
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.2.7 F_Set_MCU_Name

General Description

Set target microcontroller name. A list of acceptable names can be obtained using the functions F_Set_MCU_Family_Group and F_Get_MCU_Name_list.

Syntax

```
INT_X F_Set_MCU_Name( char *MCU_name );
```

Input

char * : Exact name of target microcontroller.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : microcontroller index if successful

- microcontroller index if input name was found and set, index is greater than or equal to zero (0)
- -1 (0xFFFFFFFF) : name not found (avoid by getting list of proper names using F_Get_MCU_Name_list)
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.2.8 F_Get_MCU_Name_list

General Description

Obtain a list of supported vendors, families, groups, and MCUs for the specified FPA. Use this function in conjunction with F_Set_MCU_Family_Group to get all supported MCU names. First obtain the names of supported vendors, then select one vendor. Then, for the selected vendor obtain the names of supported families, then select one family, etc.

The MCU names obtained from this function are the exact names used to configure the FPA using the function F_Set_MCU_Name for the target device you want to program.

Syntax

```
char *  F_Get_MCU_Name_list( INT_X type, INT_X index );
```

Input

INT_X type : specify which list to get names from

- MCU_VENDOR_LIST (0) : get name of vendors enabled on this FPA
- MCU_FAMILY_LIST (1) : get name of MCU families supported from the selected vendor
- MCU_GROUP_LIST (2) : get name of MCU groups supported from the selected MCU family
- MCU_NAME_LIST (3) : get name of MCU supported from the selected MCU group

INT_X index : list index starting from zero, increment until returned string is null character.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.

Output

char * : Name of vendor, family, group, or MCU for the given index.

- valid name : if correct parameters provided
- "" : incorrect FPA index provided (avoid using F_Check_FPA_index)

Example

Refer to F_Set_MCU_Family_Group.

3.2.9 F_Set_MCU_Family_Group

General Description

Select vendor, family, group and MCU index. Use this function in conjunction with F_Get_MCU_Name_list to get all supported MCU names. First obtain the names of supported vendors, then select one vendor. Then, for the selected vendor obtain the names of supported families, then select one family, etc.

Use MCU names to configure the FPA using the function F_Set_MCU_Name. Use this function only to obtain the list of supported MCU names, not to configure the FPA. When NEW MCUs are supported then these indices will change.

Syntax

```
INT_X F_Set_MCU_Family_Group( INT_X type, INT_X index );
```

Input

INT_X type : specify which list to set index for

- MCU_VENDOR_LIST (0) : set vendor index on this FPA
- MCU_FAMILY_LIST (1) : set family index from the selected vendor
- MCU_GROUP_LIST (2) : set group index from selected MCU family
- MCU_NAME_LIST (3) : set MCU index from selected MCU group (use actual

INT_X index : set list index
Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : index parameter out of range
- TRUE (1) : index parameter set correctly
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

Example

```
//C++ and C# are available in the installation package
//that implement this code
//Example from demo included in installation package
//Select one FPA index to perform this sequence on.
//If all FPA are the same then iterating over other FPA indices is not necessary.

//Iterate over supported vendors, then select vendor zero in this case
//Pull-down menu in demo allows user to change selected vendor index
void ProgDemoDlg::Init_MCU_Vendor_ComboBox( void )
{
  int k;
  char *name;
  SelMCU_Vendor->ResetContent();//Reset list of vendors
  k = 0;
  Vendor_list_size = 0;
  do
  {
    //-----------------------------------------------------
    name = F_Get_MCU_Name_list( MCU_VENDOR_LIST, k );//get name of vendor at index k
    if( strlen( name ) == 0 )  break; //reached end of list of vendors
    //-----------------------------------------------------
    if(SelMCU_Vendor)
    {
      //-----------------------------------------------------
      SelMCU_Vendor->AddString( name ); //add newly acquired vendor name to GUI list
      //-----------------------------------------------------
      Vendor_list_size++;
    }
```

```
    k++;
  }while(1);

  //Select vendor zero in demo GUI (will not affect DLL yet)
  SelMCU_Vendor->SetCurSel(0);

  //Select chosen vendor and read supported MCU families
  OnCbnSelchangeMcuVendor();
}

//Initialize list of supported families for selected vendor
void ProgDemoDlg::OnCbnSelchangeMcuVendor( void )
{
  Vendor_list_index = SelMCU_Vendor->GetCurSel();
  //-----------------------------------------------------
  //Set vendor to selected index (during initialization this is usually zero)
  F_Set_MCU_Family_Group( MCU_VENDOR_LIST, Vendor_list_index );
  //-----------------------------------------------------
  Init_MCU_Family_ComboBox();
}

//Iterate over supported families for selected vendor, then select family zero in this case
//Pull-down menu in demo allows user to change selected family index
void ProgDemoDlg::Init_MCU_Family_ComboBox( void )
{
  int k;
  char *name;
  SelMCU_Family->ResetContent();
  k = 0;
  Family_list_size = 0;
  do
  {
    //-----------------------------------------------------
    name = F_Get_MCU_Name_list( MCU_FAMILY_LIST, k ); //get name of family at index k
    if( strlen( name ) == 0 )  break;  //reached end of list of families
    //-----------------------------------------------------

    if(SelMCU_Family)
    {
      //-----------------------------------------------------
      SelMCU_Family->AddString( name ); //add newly acquired family name to GUI list
```

```
      //-----------------------------------------------------
      Family_list_size++;
    }
    k++;
  }while(1);

  //Select family zero in demo GUI (will not affect DLL yet)
  SelMCU_Family->SetCurSel(0);

  //Select chosen family and read supported MCU groups
  OnCbnSelchangeMcuFamily();
}

//Initialize list of supported groups for selected family
void ProgDemoDlg::OnCbnSelchangeMcuFamily( void )
{
  Family_list_index = SelMCU_Family->GetCurSel();
  //-----------------------------------------------------
  //Set family to selected index (during initialization this is usually zero)
  F_Set_MCU_Family_Group( MCU_FAMILY_LIST, Family_list_index );
  //-----------------------------------------------------
  Init_MCU_Group_ComboBox();
}
//Iterate over supported groups for selected family, then select group zero in this case
//Pull-down menu in demo allows user to change selected group index
void ProgDemoDlg::Init_MCU_Group_ComboBox( void )
{
  int k;
  char *name;
  SelMCU_Group->ResetContent();
  MCU_Group_list_size = 0;
  k = 0;
  do
  {
    //-----------------------------------------------------
    name = F_Get_MCU_Name_list( MCU_GROUP_LIST, k ); //get name of group at index k
    if( strlen( name ) == 0 ) break; //reached end of list of groups
    //-----------------------------------------------------

    if(SelMCU_Group)
    {
```

```
      //-----------------------------------------------------
      SelMCU_Group->AddString( name );  //add newly acquired group name to GUI list
      //-----------------------------------------------------
      MCU_Group_list_size++;
    }
    k++;
  }while(1);

  //Select group zero in demo GUI (will not affect DLL yet)
  SelMCU_Group->SetCurSel(0);

  //Select chosen group and read supported MCUs
  OnCbnSelchangeMcuGroup();
}

//Initialize list of supported MCUs for selected group
void ProgDemoDlg::OnCbnSelchangeMcuGroup( void )
{
  MCU_Group_list_index = SelMCU_Group->GetCurSel();
  //-----------------------------------------------------
  //Set group to selected index (during initialization this is usually zero)
  F_Set_MCU_Family_Group( MCU_GROUP_LIST, MCU_Group_list_index );
  //-----------------------------------------------------
  Init_MCU_List_ComboBox();
}

//Iterate over supported MCUs for selected group, then select MCU zero in this case
//Pull-down menu in demo allows user to change selected MCU index
void ProgDemoDlg::Init_MCU_List_ComboBox( void )
{
  int k;
  char *name;
  SelMCU_List->ResetContent();
  MCU_Name_list_size = 0;
  k = 0;
  do
  {
    //-----------------------------------------------------
    name = F_Get_MCU_Name_list( MCU_NAME_LIST, k ); //get name of MCU at index k
    if( strlen( name ) == 0 ) break; //reached end of list of MCUs
    //-----------------------------------------------------
```

```
    if(SelMCU_List)
    {
      //-----------------------------------------------------
      SelMCU_List->AddString( name ); //add newly acquired MCU name to GUI list
      //-----------------------------------------------------
      MCU_Name_list_size++;
    }
    k++;
  }while(1);

  //Select MCU zero in demo GUI (will not affect DLL yet)
  SelMCU_List->SetCurSel(0);

  //At this point all the MCU names for selected vendor, family, and group have been read.
  //Iterate over all vendors, families, groups, and MCU lists to obtain names of all
  //supported MCUs. Use the actual MCU names to call function F_Set_MCU_name and configure
  //the FPA.

  //NOTE: DO NOT RELY ON MCU INDEX TO CONFIGURE ADAPTER, USE ACTUAL MCU NAMES.
  //Although calling F_Set_MCU_Family_Group( MCU_NAME_LIST, MCU_index ) will work at this
  //point to select the correct MCU the input MCU_index is only referring to the desired
  //MCU if vendor, family, and group index are also set correctly. An MCU_index of 5 will
  //be two different MCUs depending on the aforementioned parameters. Most importantly,
  //when NEW MCUs are added to the list of supported MCUs, these indices will no longer
  //be referring to the same MCUs.
  //For this reason, it is preferable to use actual MCU names to configure the FPA,
  //because it will be more reliable.
}
```

3.2.10 F_ReportMessage, F_Report_Message

General Description

Get the last report message from the programmer. When any of the DLL functions are called, an output message is usually created. The GUI uses these messages to print to the dialog box. The user application can use these messages to populate its own dialog box. The last report message can be read by user application using these functions.

When these functions are called, then a report message of up to REPORT_MESSAGE_MAX_SIZE characters is copied to the character array passed as input parameter, or in the case of the latter function, a pointer to a statically allocated character array is returned with the output message text.

Calling F_ReportMessage clears the message buffer. If F_ReportMessage is not called, then the report message will collect all reported information up to REPORT_MESSAGE_MAX_SIZE most recent characters.

Syntax

```
void F_ReportMessage( char * text );
char* F_Report_Message( void );
```

Input

char * text : F_ReportMessage, allocated text buffer of up to REPORT_MESSAGE_MAX_SIZE (2000) characters.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.

Output

char * : F_Report_Message, pointer to statically allocated character array containing output message text

3.2.11 F_GetReportMessageChar

General Description

Get one character from the report message buffer.

Syntax

```
char F_GetReportMessageChar( INT_X index );
```

Input

INT_X index : character index, starting from 0 to REPORT_MESSAGE_MAX_SIZE - 1.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.

Output

char : character from report message buffer.

3.2.12 F_DLLTypeVer

General Description

Get information about DLL software type and version. This function returns integer number with DLL ID and software revision version and copies a text message to the report message buffer about DLL ID and software revision. Text content can read using F_ReportMessage, F_Report_Message, or F_GetReportMessageChar functions.

Syntax

```
INT_X F_DLLTypeVer( void );
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.

Output

INT_X : (DLL_ID) — (0x0FFF & Version)

- DLL_ID = 0x1000, FlashPro430 - Parallel Port
- DLL_ID = 0x2000, FlashPro430 - USB
- DLL_ID = 0x3000, GangPro430 - USB
- DLL_ID = 0x4000, FlashPro-CC - USB
- DLL_ID = 0x5000, GangPro-CC - USB
- DLL_ID = 0xA000, FlashPro2000 - USB
- DLL_ID = 0xB000, GangPro2000 - USB
- DLL_ID = 0xE000, FlashPro-ARM - USB
- DLL_ID = 0xF000, GangPro-ARM - USB
- Version = 0x0xyz, version x.yz (i.e. 0x0190 = v 1.90)
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.2.13 F_ConfigFileLoad

General Description

Modify programmer's configuration according to data taken from specified configuration file. This setup will override previous values and leave omitted parameters untouched. When loaded for the first time, unspecified configuration parameters take on default values as given by F_Get_Config_Value_By_Name. When calling this function multiple times with different configuration files, make sure to specify all relevant configuration parameters.

A configuration file can be created using the FlashPro GUI software, or use the functions F_Get_Config_Name_List, F_Get_Config_Value_By_Name to extract a list of configuration names and values. The F_Set_Config_Value_By_Name function can be used to set configuration parameters instead of F_ConfigFileLoad.

Requires the target FPA to be opened using F_OpenInstancesAndFPAs and initialized using F_Initialization.

Syntax

```
INT_X F_ConfigFileLoad( char * filename );
```

Input

char * filename : path to configuration file including filename and extension

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- STATUS_OPEN_FILE_ERROR (535) : could not open file
- STATUS_CORRUPT_CONFIG_ERROR (554) : malformed configuration file (missing header, etc.)
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.2.14 F_Power_Target

General Description

Enable power from FPA to target device. For this function to have its desired effect, configuration settings PowerFromFpaEn must be set to 1. If PowerFromFpaEn is set to 0, then power from FPA is always off regardless of this function's input parameter.

Syntax

INT_X F_Power_Target(INT_X OnOff);

Input

INT_X : power setting

- FALSE (0) : power off
- TRUE (1) : power on

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.2.15 F_Reset_Target

General Description

Generate short reset pulse on target device's reset line. The length of the reset pulse can be selected using configuration options. The ResetTimeIndex is used to select pre-set times, or for custom setup select the last index and specify ResetPulseTime, and ResetIdleTime.

Syntax

```
INT_X F_Reset_Target( void );
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.2.16 F_Get_Targets_Vcc

General Description

Get V_{cc} in millivolts as read from target device.

Syntax

```
INT_X F_Get_Targets_Vcc( void )
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : returns target voltage in millivolts

- value of zero or greater : voltage in millivolts
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.2.17 F_Set_fpa_io_state

General Description

Set state of the Reset, V_{cc} and JTAG lines after programming is finished.

Syntax

```
INT_X F_Set_fpa_io_state( BYTE jtag, BYTE reset, BYTE VccOn )
```

Input

BYTE jtag : set JTAG lines (TMS, TCK, TDI)

- DEFAULT_JTAG_3ST (0) : JTAG lines set to tri-state
- DEFAULT_JTAG_HI (1) : JTAG lines set to V_{cc}
- DEFAULT_JTAG_LO (2) : JTAG lines set to GND

BYTE reset : set reset line

- 0 : reset line set to GND
- 1 : reset line set to V_{cc}

BYTE VccOn : set V_{cc} line from adapter

- 0 : set to tri-state
- 1 : set to configured V_{cc} (configuration parameter VccFromFPAin_mV)

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.2.18 F_Get_Sector_Size

General Description

Will return the segment/sector size for memory region that contains given address. This function will access the FPA's meta-data for the target device specified in configuration options. No communication with target device actually takes place.

Syntax

```
INT_X  F_Get_Sector_Size( INT_X address );
```

Input

INT_X address : address contained by segment/sector

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : address out of range
- positive number : actual sector size
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus(avoid this by not using FPA index 0)
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range(avoid this by using F_Check_FPA_index first)

3.3 Data Buffer Functions

Having setup the Multi API-DLL and API-DLL instances for the correct target devices using functions from previous sections (Sections 3.1 and 3.2). Use the functions in this section to specify what code should be written by each FPA to their target device.

All data coming to and from the target device is saved in temporary buffers (see Figure 1.2) located inside each API-DLL instance. In summary, these buffers are the Code Data Buffer, Write Data Buffer, and Read Data Buffer. During normal programming using Encapsulated Functions, such as F_AutoProgram (Section 3.4), the Code Data Buffer is used to program the target device. For custom modifications the Write Data Buffer can be used to write to the target device without disturbing the Code Data Buffer using Sequential Functions, such as F_Copy_Buffer_to_Flash (Section 3.5).

The Read Data Buffer is used for reading from the target device by the F_Copy_Flash_to_Buffer and F_Memory_Read functions, the contents of which can be accessed using F_Get_Byte_from_Buffer.

3.3.1 F_ReadCodeFile

General Description

Read code from file and store in internal FPA buffer to be used in programming. Only code data that fits within the target MCUs memory space will be read, and the rest will be discarded. Therefore, it is necessary to configure the FPA for the correct MCU first, then load the code file. Unspecified locations within the code file are set to the target MCU's default empty value (for most MCUs it is 0xFF, for some it is 0x00).

Syntax

```
INT_X F_ReadCodeFile( char * FileName )
```

Input

char * FileName : path to code file including filename and extension

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded

- STATUS_OPEN_FILE_ERROR (535): could not open file
- STATUS_FILE_NAME_ERROR (536): format not supported
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.3.2 F_AppendCodeFile

General Description

Append code from selected file to internal FPA buffer to be used in programming. Only code data that fits within the target MCUs memory space will be read, and the rest will be discarded.

Already read code data is not overwritten by newly appended file. Existing code data locations that contain the default empty value (for most MCUs it is 0xFF, for some it is 0x00) can be overwritten if the OverwriteEmptyValues option is enabled in the Preferences file, prefer.ini (read from current working directory). By default, no overwrites are performed.

Syntax

```
INT_X F_AppendCodeFile( char * FileName )
```

Input

char * FileName : path to code file including filename and extension

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- TRUE (1) : succeeded
- STATUS_OPEN_FILE_ERROR (535): could not open file
- STATUS_MAX_FILE_COUNT (557): total number of files exceeds MAX_FILE_INDEX (20)
- STATUS_DUPLICATE_FILE_PATH (558): appending already existing file
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus

- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.3.3 F_Get_CodeCS

General Description

Read code from selected buffer and calculate check sum.

Syntax

```
INT_X F_Get_CodeCS( INT_X dest )
```

Input

INT_X dest : choose operation

- 1 : Calculate checksum of code from internal Code Buffer
- 2 : Calculate checksum of code used in last F_AutoProgram or F_Memory_Write operation.
- 3 : Calculate checksum of flash memory read after last F_AutoProgram or F_Memory_Verify operation.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- any : checksum
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus(avoid this by not using FPA index 0)
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range(avoid this by using F_Check_FPA_index first)

3.3.4 F_Clr_Code_Buffer

General Description

Clear contents of internal code buffer.

Syntax

```
INT_X F_Clr_Code_Buffer( void )
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.3.5 F_Put_Byte_to_Code_Buffer

General Description

Write to internal code buffer. Can be used instead of or in conjunction with the F_ReadCodeFile function. When starting from scratch, use the F_Clr_Code_Buffer function to clear the internal code buffer.

Syntax

```
INT_X F_Put_Byte_to_Code_Buffer( INT_X  addr, BYTE data )
```

Input

INT_X addr : valid flash address for target MCU
BYTE data : new byte to be written to internal code buffer

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.3.6 F_Get_Byte_from_Code_Buffer

General Description

Read from internal code buffer. Can be used in conjunction with F_Put_Byte_to_Code_Buffer to verify writes to internal code buffer.

Syntax

```
INT_XF_Get_Byte_from_Code_Buffer( INT_X  addr )
```

Input

INT_X addr : valid flash address for target MCU

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- BYTE (0 to 0xFF) : byte from internal code buffer
- -1 : addr parameter out of flash range
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.3.7 F_Put_Byte_to_Buffer

General Description

Write byte to temporary Write Data Buffer (see Figure 1.2).

Syntax

```
INT_X F_Put_Byte_to_Buffer( INT_X  addr, BYTE data )
```

Input

INT_X addr : valid flash address for target MCU
BYTE data : new byte to be written to temporary Write Data Buffer

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.3.8 F_Get_Byte_from_Buffer

General Description

Read from temporary Read Data Buffer (see Figure 1.2).

Syntax

```
INT_X F_Get_Byte_from_Buffer( INT_X  addr )
```

Input

INT_X addr : valid flash address for target MCU

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- BYTE (0 to 0xFF) : byte from temporary Write Data Buffer
- -1 : addr parameter out of flash range
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.4 Encapsulated Functions

Encapsulated functions are powerful and easy to use. When called, these functions perform all device actions from beginning to end automatically.

These functions require that all FPAs were opened using the Multi API-DLL function F_OpenInstancesAndFPAs (Section 3.1) and each API-DLL instance was correctly initialized using Generic Functions (Section 3.2). If not already included in configuration settings, the code being programmed by each FPA to its target device can be set using Data Buffer Functions (Section 3.3).

Encapsulated functions use the following sequence:

- FPA opens target device according to configuration settings (power source, reset, etc.),
- FPA establishes communication with target device (JTAG/SWD), checks ID, calibrates clock,
- FPA performs selected encapsulated function,
- FPA terminates communication with target device,
- FPA closes target device according to configuration settings (power setting, app. start for F_AutoProgram only, adapter line states, etc.)

3.4.1 F_AutoProgram

General Description

Perform full programming sequence on selected FPA. Power on (optional), open communication, erase, blank check, program, verify and lock device (optional). Exact behavior of each action is controlled through configuration settings (i.e. PowerFromFpaEn has to be enabled to power target device from FPA, otherwise outside power source is required).

Syntax

```
INT_X F_AutoProgram( INT_X mode )
```

Input

INT_X mode : for future use (currently has no effect)

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.4.2 F_Verify_Access_to_MCU

General Description

Determine if FPA has successfully established communication with the target device. This step requires a working connection, a read of the processor ID, memory size, and flash size checked against internal records (except when target MCU doesn't have this meta-data available), and calibration of internal CPU clock. If these steps match expected values the operation will succeed.

This function is internally called by all other encapsulated functions at startup before they proceed to perform any other operations (except one). Therefore if this function fails, other encapsulated functions will invariably fail as well. The exception is F_Clear_Locked_Device which uses a custom procedure to unlock the MCU (if supported by vendor).

Syntax

```
INT_X F_Verify_Access_to_MCU( void )
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.4.3 F_Memory_Erase

General Description

Erase memory specified in configuration options. Depending on settings, this operation will erase all or part of target device's memory contents. Use Retain Data settings to preserve calibration or custom data. Retain Data settings only apply to encapsulated functions (F_AutoProgram, this function, etc.) custom writes using Data Buffer or Sequential instructions will not retain data.

Syntax

```
INT_X F_Memory_Erase( INT_X mode )
```

Input

INT_X mode : type of memory erase operation

- 0 : erase memory specified by configuration option FlashEraseModeIndex
- 1 : erase all flash memory, regardless of configuration options

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.4.4 F_Memory_Blank_Check

Blank check memory specified in configuration options. Depending on settings, this operation will blank check all or part of target device's memory contents.

Syntax

```
INT_X F_Memory_Blank_Check( void )
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.4.5 F_Memory_Write

General Description

Write contents of Code Buffer to target device's memory. The Code Buffer is usually written from a code file using F_ReadCodeFile, but can be modified with custom instructions such as F_Put_Byte_to_Code_Buffer. An enabled configuration option, CodeFileReload, will cause this function to always reload code file before proceeding. This setting can be useful if code file is frequently modified, but it can cause issues if custom instructions are used to modify the code buffer before programming, as their modifications will be discarded.

Syntax

```
INT_X F_Memory_Write( INT_X mode )
```

Input

INT_X mode : for future use (currently has no effect)

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.4.6 F_Memory_Verify

General Description

Verify target device's memory contents against Code Buffer. The sections compared depend on configuration settings, and are identical to sections of memory being programmed using F_Memory_Write. For common behavior, do not modify settings between calling F_Memory_Write and this function.

Syntax

```
INT_X F_Memory_Verify( INT_X mode )
```

Input

INT_X mode : for future use (currently has no effect)

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.4.7 F_Memory_Read

Read target device's memory contents and store in Read Data Buffer (see Figure 1.2). The sections of memory read depend on configuration settings, FlashReadModeIndex. Use F_Get_Byte_from_Buffer to access the contents of this buffer after this function has completed.

Syntax

```
INT_X F_Memory_Read( void )
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.4.8 F_Lock_MCU

This function can write memory protection bits and disable debug access. Performs the functions F_Write_Locking_Registers and F_Write_Debug_Register together. If the former fails, the latter will not execute (locking bits are verified first before debug access is disabled).

This procedure is not supported by all target devices. Consult individual device manuals or the FlashPro GUI, Setup, Memory Protection window which will indicate which bits can be set.

Syntax

```
INT_X F_Lock_MCU( void )
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.4.9 F_Clear_Locked_Device

General Description

Will attempt to return all memory to factory settings. Usually requires a mass erase of the memory contents. This procedure can be used to unlock memory protection or enable debug access after the F_Lock_MCU procedure.

This procedure is not supported by all target devices. Consult individual device manuals or the FlashPro GUI, Setup, Memory Protection window which will indicate which bits can be reset.

Syntax

```
INT_X F_Clear_Locked_Device( void )
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5 Sequential Functions

Sequential functions allow access to the target device in any combination of small instructions like erase, read, write sector, modify part of memory, etc. Sequential functions work correctly after communication between target device and programming adapter has been initialized. This requires the target FPA to be opened using F_OpenInstancesAndFPAs and F_Initialization, and for proper configuration options to be set using F_ConfigFileLoad or F_Set_Config_Value_by_Name.

Once the FPA has been initialized using the aforementioned functions, open the target device using the function F_Open_Target_Device. When communication is established, then any

of the sequential instructions can be called. When finished, conclude your task by calling F_Close_Target_Device.

```
F_OpenInstancesAndFPAs
F_Set_FPA_index
F_Initialization
F_ConfigFileLoad
F_Open_Target_Device
... < sequential functions here > ...
F_Close_Target_Device
```

3.5.1 F_Open_Target_Device

General Description

The FPA will establish communication with the target device. This step requires a working connection, a read of the processor ID, memory size, and flash size checked against internal records (except when target MCU doesn't have this meta-data available), and calibration of internal CPU clock. If these steps match expected values the operation will succeed.

This function is identical to F_Verify_Access_to_MCU except that the target device is not closed when this function concludes (F_Open_Target_Device keeps the target device open to accept further instructions listed below).

Syntax

```
INT_X F_Open_Target_Device( void );
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus

- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.2 F_Close_Target_Device

General Description

Terminates communication between FPA and target device. Sets reset line to GND and communication lines (TMS, TCK, TDI) to configuration setting ReleaseJtagState.

Syntax

```
INT_X F_Close_Target_Device( void );
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.3 F_Segment_Erase

General Description

Send command to erase target device's memory segment.

Syntax

```
INT_X F_Segment_Erase( INT_X address );
```

Input

INT_X address : address of segment/sector to be erased
Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- STATUS_ADDRESS_OUT_OF_FLASH_SPACE_ERR (534): address out of range
- STATUS_ERASE_SEGMENT_FAILED (549): erase operation failed
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.4 F_Sectors_Blank_Check

General Description

Blank check part or all of the target device's memory.

Syntax

```
INT_X F_Sectors_Blank_Check( INT_X  start_addr, INT_X  stop_addr );
```

Input

INT_X start_addr : first memory address to be included in blank check (inclusive)
INT_X stop_addr : last memory address to be included in blank check (inclusive)

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.5 F_Copy_Buffer_to_Flash

General Description

Write specified number of bytes from temporary Write Data Buffer (see Figure 1.2) to MCU flash or OTP.

Syntax

```
INT_X F_Copy_Buffer_to_Flash( INT_X  start_addr, INT_X  size )
```

Input

INT_X start_addr : valid flash or OTP start address
INT_X size : number of bytes to copy beginning at start address

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.6 F_Copy_Flash_to_Buffer

General Description

Read specified number of bytes from MCU flash or OTP to temporary Read Data Buffer (see Figure 1.2).

Syntax

```
INT_X F_Copy_Flash_to_Buffer( INT_X  start_addr, INT_X  size )
```

Input

INT_X start_addr : valid flash or OTP start address
INT_X size : number of bytes to copy beginning at start address

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.7 F_Write_Byte_to_RAM

General Description

Use the FPA to write a byte to the target device's RAM. This write is not verified (no read performed). Use F_Read_Byte to verify.

Syntax

```
INT_X F_Write_Byte_to_RAM( INT_X  addr, BYTE data )
```

Input

INT_X addr : address of byte to be written (byte addressable)
BYTE data : byte to be written

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.8 F_Write_Word16_to_RAM

General Description

Use the FPA to write two bytes to the target device's RAM. This write is not verified (no read performed). Use F_Read_Word16 to verify.

Syntax

```
INT_X F_Write_Word16_to_RAM( INT_X  addr, INT_X data )
```

Input

INT_X addr : address of 16-bit data to be written (byte addressable)
INT_X data : 16-bit data to be written

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded

- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.9 F_Write_Word32_to_RAM

General Description

Use the FPA to write four bytes to the target device's RAM. This write is not verified (no read performed). Use F_Read_Word32 to verify.

Syntax

```
INT_X F_Write_Word32_to_RAM( INT_X  addr, INT_X data )
```

Input

INT_X addr : address of 32-bit data to be written (byte addressable)
INT_X data : 32-bit data to be written

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.10 F_Write_Bytes_Block_to_RAM

General Description

Use the FPA to write a block of bytes to RAM. Currently this function only writes a block of bytes with a size that is a multiple of four, to a 32-bit word aligned address. These writes are not verified (no reads performed). Use F_Read_Bytes_Block to verify.

Syntax

```
INT_X F_Write_Bytes_Block_to_RAM( INT_X addr, INT_X size, BYTE *data )
```

Input

INT_X addr : address of 32-bit data to be written (byte addressable)
INT_X size : number of bytes to be written
BYTE *data : array of bytes to be written (not verified)

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- STATUS_ADDR_ALIGNMENT_ERROR (543) : not 32-bit word aligned address
- STATUS_SIZE_ALIGNMENT_ERROR (544) : number of bytes not a multiple of four
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.11 F_Read_Byte

General Description

Use the FPA to read one byte from any address (RAM, flash, etc.).

Syntax

```
INT_X F_Read_Byte( INT_X  addr )
```

Input

INT_X addr : address of byte to be read

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : data read

- 0-0xFF : byte value
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus(avoid this by not using FPA index 0)
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range(avoid this by using F_Check_FPA_index first)

3.5.12 F_Read_Word16

General Description

Use the FPA to read two bytes from any address (RAM, flash, etc.).

Syntax

```
INT_X F_Read_Word16( INT_X  addr )
```

Input

INT_X addr : address of 16-bit word to be read

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : data read

- 0-0xFFFF : 16-bit word value
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus(avoid this by not using FPA index 0)
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range(avoid this by using F_Check_FPA_index first)

3.5.13 F_Read_Word32

General Description

Use the FPA to read four bytes from any address (RAM, flash, etc.).

Syntax

```
INT_X F_Read_Word32( INT_X  addr )
```

Input

INT_X addr : address of 32-bit word to be read

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : data read

- 0-0xFFFFFFFF : 32-bit word value
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus(avoid this by not using FPA index 0)
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range(avoid this by using F_Check_FPA_index first)

3.5.14 F_Read_Bytes_Block

General Description

Use the FPA to read a block of bytes from any address (RAM, flash, etc.).

Syntax

```
INT_X F_Read_Bytes_Block( INT_X addr, INT_X size, BYTE *data );
```

Input

INT_X addr : first byte address to be read (byte addressable)
INT_X size : number of bytes to be read
BYTE *data : array of bytes that will be written with memory contents

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.15 F_Set_PC_and_RUN

General Description

Start target MCU from specified program counter (PC). There should be a valid instruction at the target address.

Syntax

```
INT_X F_Set_PC_and_RUN( INT_X  PC_addr )
```

Input

INT_X PC_addr : set program counter to this address and start processor

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- TRUE (1) : succeeded
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.16 F_Write_Locking_Registers

General Description

Write to non-volatile memory protection bits and/or persistent user-data registers (if supported). Does not block communication over debug interface, but can disable writing and/or reading from target device's memory. These bits can be reset to factory settings using the F_Clear_Locked_Device function if the device supports it. To write protection bits, enable the WriteLockingBitsEn configuration option.

Syntax

```
INT_X F_Write_Locking_Registers( void )
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.17 F_Write_Debug_Register

General Description

Write to non-volatile protection bits to disable communication over debug interface. These bits can be reset to factory settings using the F_Clear_Locked_Device function if the device supports it. To disable debug access, enable WriteLockingBitsEn and (vendor)_USER_DBG_WrEn configuration options.

Syntax

```
INT_X F_Write_Debug_Register( void )
```

Input

none.

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed
- TRUE (1) : succeeded
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range

3.5.18 F_Get_MCU_Data

General Description

Return the MCU ID, flash size, or RAM size read by the FPA to identify a specific target device, if supported by vendor.

Syntax

```
INT_X F_Get_MCU_Data( INT_X  type )
```

Input

INT_X type : select type of information to receive

- GET_MCU_ID (1) : connected MCU package identifier (MCU family and group, etc.), not a unique silicon ID
- GET_MCU_FLASH_SIZE (2) : connected MCU flash size, if available
- GET_MCU_RAM_SIZE (3) : connected MCU RAM size, if available

Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed. No communication or incorrect parameters.
- any postive value: MCU ID, flash size, RAM size

- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus(avoid this by not using FPA index 0)
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range(avoid this by using F_Check_FPA_index first)

3.5.19 F_Capture_PC_Addr

General Description

Read program counter (PC) register using debug interface.

Syntax

```
INT_X F_Capture_PC_Addr( void );
```

Input

none.
Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed.
- any postive value: program counter
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus(avoid this by not using FPA index 0)
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range(avoid this by using F_Check_FPA_index first)

3.5.20 F_Synch_CPU_JTAG

General Description

Stop target device (debug enable and halt).

Syntax

```
INT_X F_Synch_CPU_JTAG();
```

Input

none.
Select FPA to perform operation on using F_Set_FPA_index, index 1 to 64.Use index 0 to perform operation on all FPAs (if results differ, use F_LastStatus to get individual results).

Output

INT_X : result of operation

- FALSE (0) : failed.
- TRUE (1) : succeeded. MCU stopped.
- FPA_UNMATCHED_RESULTS (-1 or 0xFFFFFFFF) : Result of operation inconsistent across all selected FPAs, refer to F_LastStatus(avoid this by not using FPA index 0)
- FPA_INVALID_NO (-2 or 0xFFFFFFFE) : FPA not opened with F_OpenInstancesAndFPAs or index out of range(avoid this by using F_Check_FPA_index first)