

Generic FPA DLL
and
Command-Line

for Flash and Gang Programmers
User's Manual

Software version 1.61

PM037A01 Rev.1.6
September-2-2016

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

Copyright ©Elprotronic Inc. All rights reserved.

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 ENFORCEABLE 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.*

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

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

Warning: Changes or modifications not expressly approved by 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 appareil numerique de la classe B respecte toutes les exigences du Reglement sur le material brouilleur du Canada.

Contents

1	How To Use The Generic DLL	1
1.1	Introduction	1
1.2	Functionality	1
1.3	Portability	4
1.4	Generic-FPA DLLSetup	5
1.5	Function Calls to the Generic-FPA DLLand to API DLLs	7
1.6	Integrating Function Calls to Different API DLLs	8
1.7	Return Value upon Calling a Generic-FPA DLLfunction	9
1.8	Timeouts	9
1.9	Access Configuration Values by Name	10
1.10	Using the Generic-FPA-TS DLLand Managing Your Own Thread Pool	11
1.11	Generic-FPA-TS DLLSetup	14
1.12	config.ini conflict	15
2	Commandline Interface	16
2.1	Introduction	16
2.2	Functionality	16
2.3	Server Configuration Flags	17
2.3.1	DISCOVER	17
2.3.2	Setupfile(required)	17
2.3.3	-configtest	19
2.3.4	-p <pipename>	19
2.3.5	-b	19
2.3.6	-f <level>	19
2.3.7	-v <level>	20
2.3.8	-of <filename>	20
2.3.9	-ofr <filename>	21
2.3.10	-of_fpa <i> <filename>	21
2.3.11	-ofr_fpa <i> <filename>	21

2.3.12	-ofq	21
2.4	Getting Started and Example Usage	21
2.5	Script Mode	26
2.5.1	Client Configuration Flags	29
2.5.2	-configtest	29
2.5.3	-p <pipename>	29
2.5.4	-i <i>	29
2.5.5	-m <command + params>	29
3	DLL Functions	30

List of Figures

1.1	Multi-FPA DLL overview	2
1.2	Generic-FPA DLL overview	3
1.3	Generic-FPA-TS DLLoverview	12
2.1	Generic-Command-line overview	18
2.2	Generic-Command-line server process	23
2.3	Generic-Command-line client process	24
2.4	Generic-Command-line server process	25
2.5	Generic-Command-line client process	27
2.6	Generic-Command-line server process	28

List of Tables

3.1	Generic-FPA DLL function compatibility (part 1/4)	31
3.2	Generic-FPA DLL function compatibility (part 2/4)	32
3.3	Generic-FPA DLL function compatibility (part 3/4)	33
3.4	Generic-FPA DLL function compatibility (part 4/4)	34
3.5	Generic-FPA-TS DLL function compatibility (part 1/4)	35
3.6	Generic-FPA-TS DLL function compatibility (part 2/4)	36
3.7	Generic-FPA-TS DLL function compatibility (part 3/4)	37
3.8	Generic-FPA-TS DLL function compatibility (part 4/4)	38

Chapter 1

How To Use The Generic DLL

1.1 Introduction

This chapter describes the Generic-FPA DLL, and explains how to use it. The Generic-FPA DLL is a shared library that can control different types of Flash Programming Adapters (FPAs) simultaneously. It is intended for users that need to use different types of FPAs to program a board with multiple MCUs from different families and vendors. The Generic-FPA DLL can simultaneously initialize, configure, program, and perform other operations on multiple, and different, FPAs such as the MSP430, C2000, ARM, CC, MSP430-GP (Gang Pro), CC-GP (Gang Pro), ARM-GP (Gang Pro) and future Elprotronic FPAs.

In addition to the Generic-FPA DLL, this document also describes a lower-level Generic-FPA-TS DLL (a thread-safe DLL). The latter is intended for advanced users that wish to manage their own thread pool, and control all FPAs in a totally independent fashion (execute different functions on different FPAs in parallel). The Generic-FPA-TS DLL is described in greater detail in Section 1.10.

The Generic-FPA DLL is an alternative to already existing Multi-FPA DLLs provided by Elprotronic. It is meant to ease programming effort for customers wishing to: control different types of FPAs together, by providing one top-level DLL, or to control FPAs independently, by providing a thread-safe DLL. Similarly to Multi-FPA DLLs, the Generic-FPA DLL is multi-threaded and can control up to 64 FPAs simultaneously; however, only in a broadcast-style fashion, where all FPAs are executing the same command at the same time. Only the Generic-FPA-TS DLL allows each FPA to be controlled independently by each thread, where each FPA is executing a different command at the same time.

1.2 Functionality

The Generic-FPA DLL was created to overcome a shortcoming of traditional Multi-FPA DLLs demonstrated in the following example. If the programmer wanted to control FlashPro430 adapters

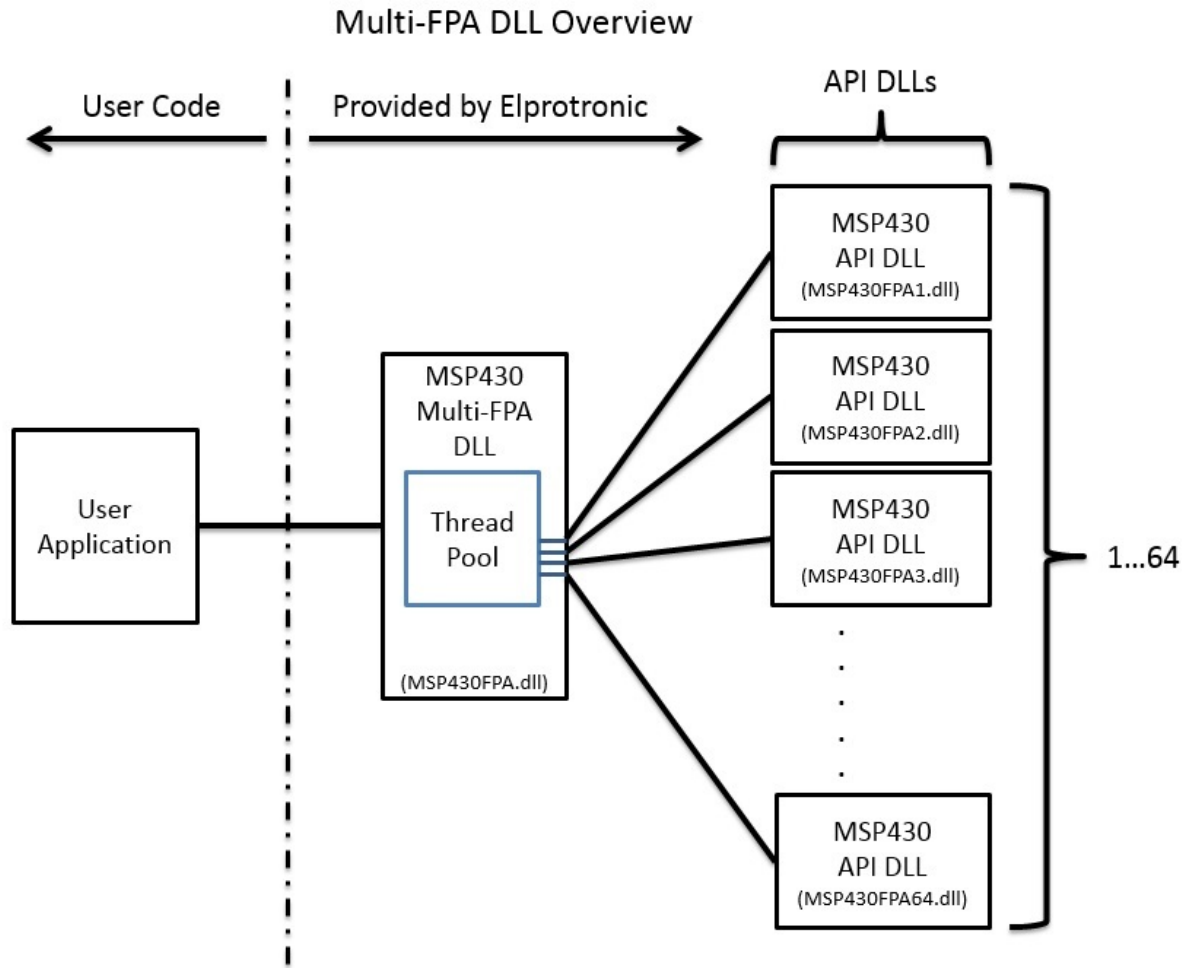


Figure 1.1: Multi-FPA DLL setup used to control multiple adapters of the same type.

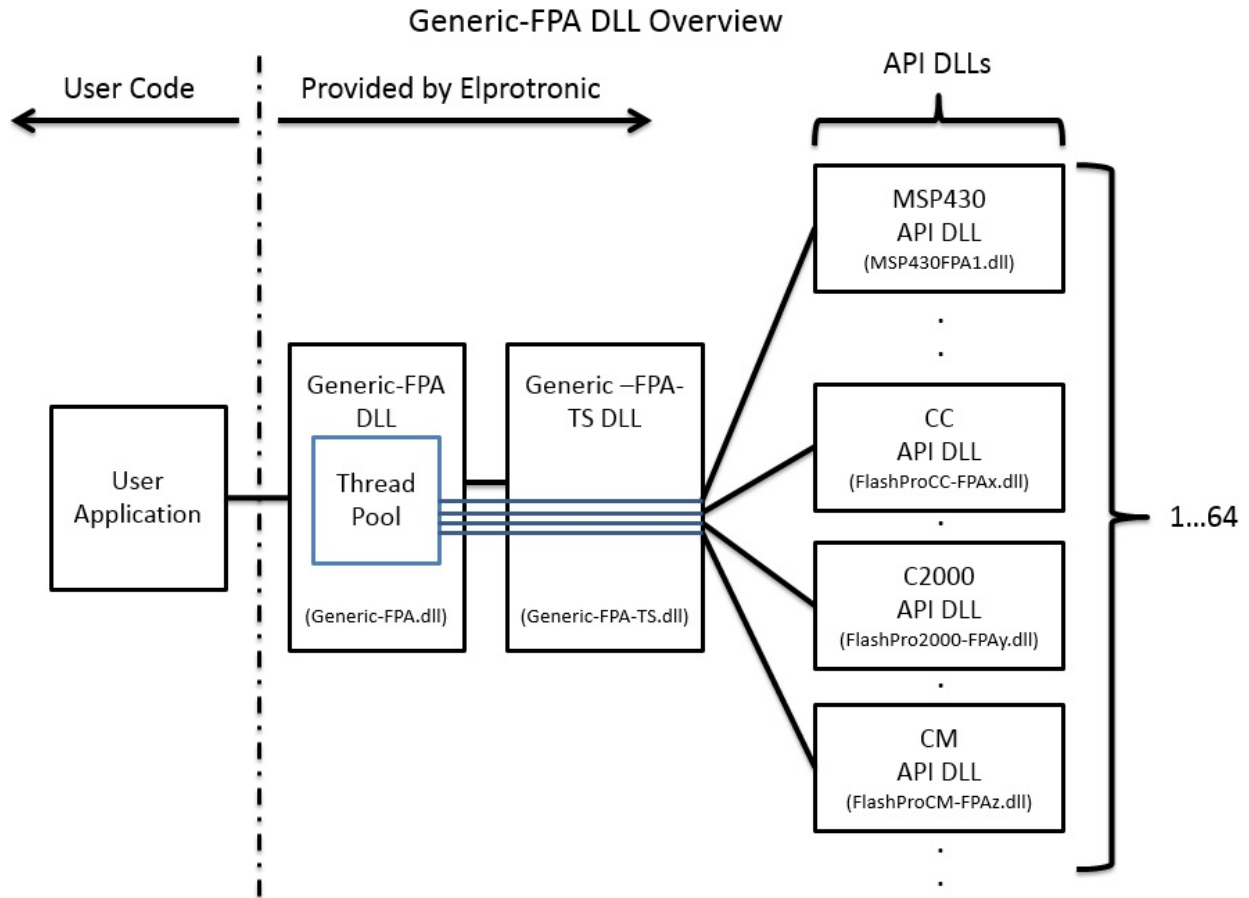


Figure 1.2: Generic-FPA DLL setup used to control adapters of different types. Threads are managed by the Generic-FPA DLL and allow synchronous control of up to 64 adapters.

they could use the Multi-FPA DLL (shown in Figure 1.1) for the MSP430 (MSP430FPA.dll). In order to control another type of adapter, like the FlashPro2000, it was necessary to load another Multi-FPA DLL for the C2000 (FlashPro2000-FPA.dll). This was a messy solution due to sometimes conflicting prototypes, and having to issue the same commands to different Multi-FPA DLLs. Rather than dealing with many different Multi-FPA DLLs, the Generic-FPA DLL (shown in Figure 1.2) combines all their functionality and provides a unified shared library for controlling all types of FPAs, synchronously.

Figures 1.1 and 1.2 provide an overview of how the Multi-FPA DLL and the Generic-FPA DLL control FPAs, respectively. Shown in Figure 1.1, a single Multi-FPA DLL controls multiple API DLLs, each of which is responsible for communicating with one adapter. When the user application calls a function from the Multi-FPA DLL, it in turns uses a thread pool to issue simultaneous calls to the same function in all the API DLLs. Using this scheme the user application can control up to 64 FPAs synchronously without managing its own thread pool or the usual complications associated with multi-threaded programming. The Generic-FPA DLL shown in Figure 1.2 expands upon the Multi-FPA DLL concept and provides additional capabilities without compromising previous features. The Generic-FPA DLL can also control up to 64 FPAs and manages its own thread pool allowing for synchronous control of all adapters, just like the Multi-FPA DLL. Additional features of the Generic-FPA DLL include:

- API DLLs can be of any type. The API DLLs currently supported are:
 - MSP430FPA1.dll
 - GangPro430FPA1.dll
 - FlashPro2000-FPA1.dll
 - FlashProCC-FPA1.dll
 - GangProCC-FPA1.dll
 - FlashProARM-FPA1.dll
 - GangProARM-FPA1.dll
- A new thread-safe Generic-FPA-TS DLL (shown in Figure 1.3) is provided for advanced users. Users can use it and ignore the top-level Generic-FPA DLL to manage their own thread pool and control FPAs in a totally independent fashion.

1.3 Portability

Any existing application that uses a Multi-FPA DLL (sometimes referred to as a Selector in other User Guides) for any of the specific FPAs listed above can be ported to use the Generic-FPA DLL with minimal effort. This is because at its core, the Generic-FPA DLL relays commands to the appropriate API DLL depending on which adapter is being accessed. The Generic-FPA

DLL combines all functions from the different API DLLs, usually with the same name and parameters, except in cases where there are conflicting prototypes for identically named functions. In cases of conflict, a prefix is added to distinguish special cases.

For example, the function `F_Set_PC_and_RUN`, takes different parameters for different adapters. Such as:

```
ChipCon: F_CC_Set_PC_and_RUN( INT_X xram_en, INT_X PC_addr)
Others:  F_Set_PC_and_RUN( INT_X PC_addr )
```

The behaviour of each call to this function is dependent on the API DLL being invoked. This means that, when the target adapter is the FlashPro430, the call to `F_Set_PC_and_RUN` goes to the `MSP430FPA1.dll`, when the target adapter is the FlashPro-ARM, the call goes to the `FlashProARM-FPA1.dll`, and so on. In summary, the Generic-FPA DLL forwards function calls and parameters to the proper API DLL, as determined during initialization.

1.4 Generic-FPA DLLSetup

Similarly to the Multi-FPA DLL, the Generic-FPA DLL should be initialized using a setup file or setup string. Below is an example of a setup file where the user wishes to use 4 different adapters:

```
FPA-1  20130001  TYPE-MSP
FPA-2  20130002  TYPE-C2000
FPA-4  20130004  TYPE-ARM
FPA-5  20130005  TYPE-CC
```

A setup string for the same adapters:

```
*# FPA-1 20130001 TYPE-MSP FPA-2 20130002 TYPE-C2000 FPA-4 20130004 TYPE-ARM FPA-5
20130005 TYPE-CC
```

The DLL accepts both formats, but only the setup file can be used for Generic-CommandLine-Server initialization.

The first column specifies the FPA index, which can be an arbitrary number between 1 and 64. An FPA index is used to identify a particular adapter when using the Generic-FPA DLL. The second column lists the specific serial number (SN) that the adapter should have if it is to be initialized properly. That number can be read from the adapter's sticker label on the actual physical product. If the SN between the setup file and the adapter label doesn't match, then initialization for that specific adapter will fail and the user won't be able to control it. The previous two columns were already present in setup files used for the Multi-FPA DLL. A setup file for the Generic-FPA DLL requires an additional third column which identifies the adapter type. Currently, seven types are supported:

- "TYPE-MSP" - uses `MSP430FPA1.dll`

- "TYPE-MSP-GP" - uses GangPro430FPA1.dll
- "TYPE-C2000" - uses FlashPro2000-FPA1.dll
- "TYPE-CC" - uses FlashProCC-FPA1.dll
- "TYPE-CC-GP" - uses GangProCC-FPA1.dll
- "TYPE-ARM" - uses FlashProARM-FPA1.dll
- "TYPE-ARM-GP" - uses GangProARM-FPA1.dll

To initialize the Generic-FPA DLL, load it and invoke the function:

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

As input provide the setup file or setup string, and the function will return the number of successfully initialized adapters. For the setup file provided here, if the proper adapters with the listed SNs are connected to the computer, then the function will return 4. This action will also create 4 worker threads inside the Generic-FPA DLL ready to invoke functions on API DLLs in parallel.

Adapter Type in Setup File is not Explicitly Verified During Initialization

If in the setup file the user enters proper SNs, but incorrect adapter types (TYPE-CC instead of TYPE-MSP), F_OpenInstancesAndFPAs will still succeed for that adapter. However, the incorrectly identified adapter will return errors on API DLL calls.

If you don't know what FPAs are connected, you can call the function:

```
char *   F_DiscoverFPAs(char *type)
```

As input provide one of the types, "MSP", "CC", etc. The type parameter specifies which API-DLL you want to use to query for connected FPAs. If your working directory contains all API-DLLs then it doesn't matter which type you select here. The function will return a parsable string that can be used to extract FPA types and serial numbers. An example string is shown below:

```
1 SN=20130005
  HW ID=6, HW sub-ID=1
  Full Access=1, Interface Type=3
  Access Key=1
  HW PN=USB-FPA-6.1
  Adapter Desc=FlashPro-ARM
```

Once initialized, individual FPAs can be enabled or disabled using the following functions in a manner identical to previous Multi-FPA DLLs:

```
void    F_Enable_FPA_index( BYTE fpa )
void    F_Disable_FPA_index( BYTE fpa )
```

The input parameter for these functions is the FPA index, which for the chosen setup file would be indices 1, 2, 4, and 5. Once enabled, an FPA can accept commands. After calling `F_OpenInstancesAndFPAs`, all successfully initialized FPAs are enabled by default. To double-check at run-time the assigned type of each FPA, call the function:

```
char * F_Get_FPA_type( BYTE fpa )
```

For the specified FPA index, between 1 and 64, this function will return the string identifier, "MSP", "MSP-GP", "C2000", "CC", "CC-GP", "ARM", or "ARM-GP".

Before invoking other functions on the Generic-FPA DLL the user should first select which FPAs they want commands sent to, using the following functions:

```
INT_X   F_Set_FPA_index( BYTE fpa )  
BOOL    F_Get_FPA_index( BYTE fpa )
```

Set the desired FPA index, from 1 to 64, or 0 (zero) to send commands to all enabled FPAs. Verify your selection with the latter function. This might seem redundant with the enable/disable option but it is useful if some FPAs encounter errors. If, for example, 1 out of 4 FPAs fails for a Write or Verify function, it can be disabled from receiving further commands using the `F_Disable_FPA_index` function. While the `fpa` index is set to 0 (zero) the user can still issue commands to the other 3 enabled FPAs simultaneously. In summary, the Generic-FPA DLL will forward function calls only to enabled and selected FPAs.

1.5 Function Calls to the Generic-FPA DLL and to API DLLs

Some Generic-FPA DLL functions are not forwarded to API DLLs because they only read/write metadata within the top-level DLL. In general, these functions handle initialization for the Generic-FPA DLL and some bookkeeping. These functions are:

```
F_OpenInstancesAndFPAs  
F_CloseInstances  
F_Enable_FPA_index  
F_Disable_FPA_index  
F_Set_FPA_index  
F_Get_FPA_index  
F_Get_FPA_type  
F_LastStatus  
F_Set_Timeout_Short  
F_Set_Timeout_Long  
F_Trace_ON  
F_Trace_OFF  
F_Debug_Enable  
F_Debug_Disable
```


F_GenericDLLVer
F_GenericTSDLLVer

Other Generic-FPA DLL functions are forwarded to selected API DLLs. Individual functions are described in greater detail in Chapter 3.

1.6 Integrating Function Calls to Different API DLLs

Since the Generic-FPA DLL exports all functions from supported API DLLs, the user application can call functions that are not supported by some specific adapter types. In these cases the function call is simply ignored for adapters which don't support it, without interrupting calls to other adapters that do. This feature will be demonstrated using the example of calling `CC_Set_PC_and_RUN`, and `Set_PC_and_RUN`.

Recall the setup introduced in Section 1.4. The user selects all FPAs as enabled and intends to issue commands simultaneously to 4 FPAs by selecting FPA index to 0 (zero) using `F_Set_FPA_index`. Looking at individual API DLL headers, the MSP430 and ARM adapter types (FPAs 1, and 4 respectively) support the function:

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

and the CC adapter type (FPA 5) supports the expanded function:

```
INT_X F_CC_Set_PC_and_RUN( INT_X xram_en, INT_X PC_addr )
```

and the C2000 adapter type (FPA 2) doesn't support any of these two functions.

Invoking `F_Set_PC_and_RUN` on the Generic-FPA DLL will invoke this function on API DLLs for FPAs 1, and 4, whereas API DLLs for FPAs 2, and 5 will not be called. Conversely invoking `F_CC_Set_PC_and_RUN` on the Generic-FPA DLL will invoke this function on the API DLL for FPA 5, whereas API DLLs for FPAs 1, 2, and 4 will not be called. When a function call is ignored for a specific FPA, that adapter's API DLL is not called and the Generic-FPA DLL will instead set the return value as `ELP_ENOTIMP` for that FPA. The same is true for functions only supported by GP FPAs, and vice-versa. Most core functions such as `AutoProgram`, `Memory_Erase`, `Memory_Write`, `Memory_Verify`, etc. are supported by all FPA types with identical prototypes therefore no alternatives are necessary.

Rather than guessing which functions are supported by which FPA types, the user can reference the tables shown in Chapter 3 or run the Generic-FPA DLL in debug mode with the following functions:

```
INT_X F_Debug_Enable();  
INT_X F_Debug_Disable();
```

These functions return the new debug state, therefore the first function should return 1 and the second should return 0(zero). Once debug is enabled, function calls are no longer forwarded to API

DLLs, but instead return either `ELP_IMPL`, to signify that the function is implemented for that FPA type, or return `ELP_ENOTIMP` to signify the opposite. This is useful during development, to double-check that the desired function is supported for sepecific adapters. An example of how to use debug to iterate over functions is shown in the Generic-FPA DLL Demo.

1.7 Return Value upon Calling a Generic-FPA DLLfunction

As is shown in Figure 1.2, a single function invocation of the Generic-FPA DLL is translated into multiple individual calls to API DLLs for all selected FPAs. If 4 FPAs are selected then invoking `F_Initialization` on Generic-FPA DLL will invoke `F_Initialization` on 4 API DLLs. Two scenarios are possible:

- All API DLL calls return the same value,
- At least one API DLL call returns a different value.

In the first scenario the return value from the Generic-FPA DLL is simply the return value from one of the API DLLs for that function invocation. Since all return values are the same, no information is lost. In the second scenario, where return values are different, the return value from the Generic-FPA DLL will be `FPA_UNMATCHED_RESULTS` for that function invocation (with exception to functions that return void). This behaviour is identical to Multi DLLs. To retrieve the result for each FPA use the function:

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

As input the user should provide the FPA index of the adapter whose return value is desired, and as output this function will give the return value of the last invoked function on the API DLL for that adapter.

1.8 Timeouts

Due to unforeseen events, some Generic-FPA DLL calls can stall or worker threads responsible for calling API DLLs can die. The most common cause of these stalls is that the corresponding adapter is accidentally disconnected from the computer. To deal with these events, the Generic-FPA DLL maintains a timer per API DLL call, and a timeout per API DLL call type. After the timeout expires, the responsible worker thread is restarted, and the respective API DLL is reloaded. For timed-out calls, the function will return:

- `FPA_TIMEOUT_RESTART_SUCCESS`
- `FPA_TIMEOUT_RESTART_FAILURE`

The former means that the worker thread for that FPA is successfully restarted, and the API DLL is reloaded. More specifically, a timeout restart involves killing the thread, creating a new thread, and opening the FPA instance anew. Consequently, the API DLL state is not restored, meaning that the API DLL state after restart is the same as after `F_OpenInstancesAndFPAs`. The program that picks up that return value should proceed to run `F_Initialization`, `F_ConfigFileLoad` and `F_ReadCodeFile` on that FPA. If the latter return value is received, the worker thread for the timedout FPA was not restarted, most likely because the timedout FPA was not detected.

There are two timeout categories, short and long. The only functions with long timeouts are:

- `F_AutoProgram`, and
- `F_Clear_Locked_Device`

The short timeout is 30 seconds, and the long timeout is 60 seconds. These timeouts can be changed with the following functions:

```
INT_X F_Set_Timeout_Short( int new_short_timeout )
INT_X F_Set_Timeout_Long( int new_long_timeout )
```

The input is the new timeout in seconds. The functions will return the new timeout value. The minimum value that each timeout type can be set to is 1, but there is no maximum (beyond `INT_MAX` ofcourse). Setting an excessively large timeout effectively disables this feature.

1.9 Access Configuration Values by Name

There are two ways to configure an adapter, either with a prepared file or at run-time. Using the function `F_ConfigFileLoad` is a reliable way to obtain the same configuration on multiple reruns; however, sometimes run-time reconfiguration is necessary, usually when tied to a GUI. Run-time reconfiguration was done using the following functions:

```
INT_X F_GetConfig( INT_X index )
INT_X F_SetConfig( INT_X index, INT_X data )
```

To use these functions effectively it is necessary to have a list of indices for all the configuration parameters. However, since the Generic-FPA DLL allows the user to connect multiple different FPA types, each FPA type will have its own list of indices for configuration parameters. The same index will refer to different configuration parameters depending on which adapter type is being configured.

To avoid this confusion, several new functions were added to reference configuration parameters by name:

```
char *      F_Get_Config_Name_List( INT_X index );
unsigned int F_Get_Config_Value_By_Name( char *name, int type );
INT_X      F_Set_Config_Value_By_Name( char *name, unsigned int newValue );
```

The first function can be used to list all available configuration parameters for a selected FPA (an example of how to do this is given in the Generic-FPA DLLDemo). Starting from index 0(zero), the function will return strings denoting parameter names, until the function returns "0" after the index exceeds the length of the configuration parameter list.

The second function, `F_Get_Config_Value_By_Name`, accepts a string as input, usually obtained by iteratively running `F_Get_Config_Name_List` or reading a config file, and a second input parameter int type to specify one of the following actions:

- `CONFSEL_VALIDATE`, returns 1 if name was a valid config parameter, 0(zero) otherwise.
- `CONFSEL_VALUE`, returns value for configuration parameter given by name, or 0(zero) if name wasn't found.
- `CONFSEL_MIN`, returns the minimum value for configuration parameter given by name, or 0(zero) if name wasn't found.
- `CONFSEL_MAX`, returns the maximum value for configuration parameter given by name, or 0(zero) if name wasn't found.
- `CONFSEL_DEFAULT`, returns the default value for configuration parameter given by name if it is not set explicitly, or 0(zero) if name wasn't found.

Since this function will return 0(zero) for an invalid name string regardless of type parameter, it is best to validate a configuration parameter string first, before proceeding to obtain value or ranges.

The third function, `F_Set_Config_Value_By_Name` changes the configuration parameter given by name to the new value. Returns 1 if name was found and changed, and 0(zero) otherwise. This function automatically trims the new value to fit within the min/max ranges. If the new value for a configuration parameter is out of min/max range, it is set to DEFAULT value. The default value is chosen when input is out of range because it is usually safe, and this is especially true when setting read/write protection bits.

The aforementioned functions can be invoked on an FPA before it is configured, with the exception of the ARM adapter type. Since the ARM adapter type can support multiple MCU vendors with different configuration lists, at least one must be selected before proceeding. Use the function `F_Set_MCU_Family_Group` to set a vendor first, otherwise the aforementioned functions will always return 0 regardless of the name.

1.10 Using the Generic-FPA-TS DLL and Managing Your Own Thread Pool

Advanced users can manage their own thread pool and control adapters using the thread-safe Generic-FPA-TS DLL. An overview of this DLL is shown in Figure 1.3. Rather than relying on the Generic-FPA DLL to create and manage a pool of threads, the user can create their own threads and access adapters through the Generic-FPA-TS DLL in parallel in a safe, mutually exclusive

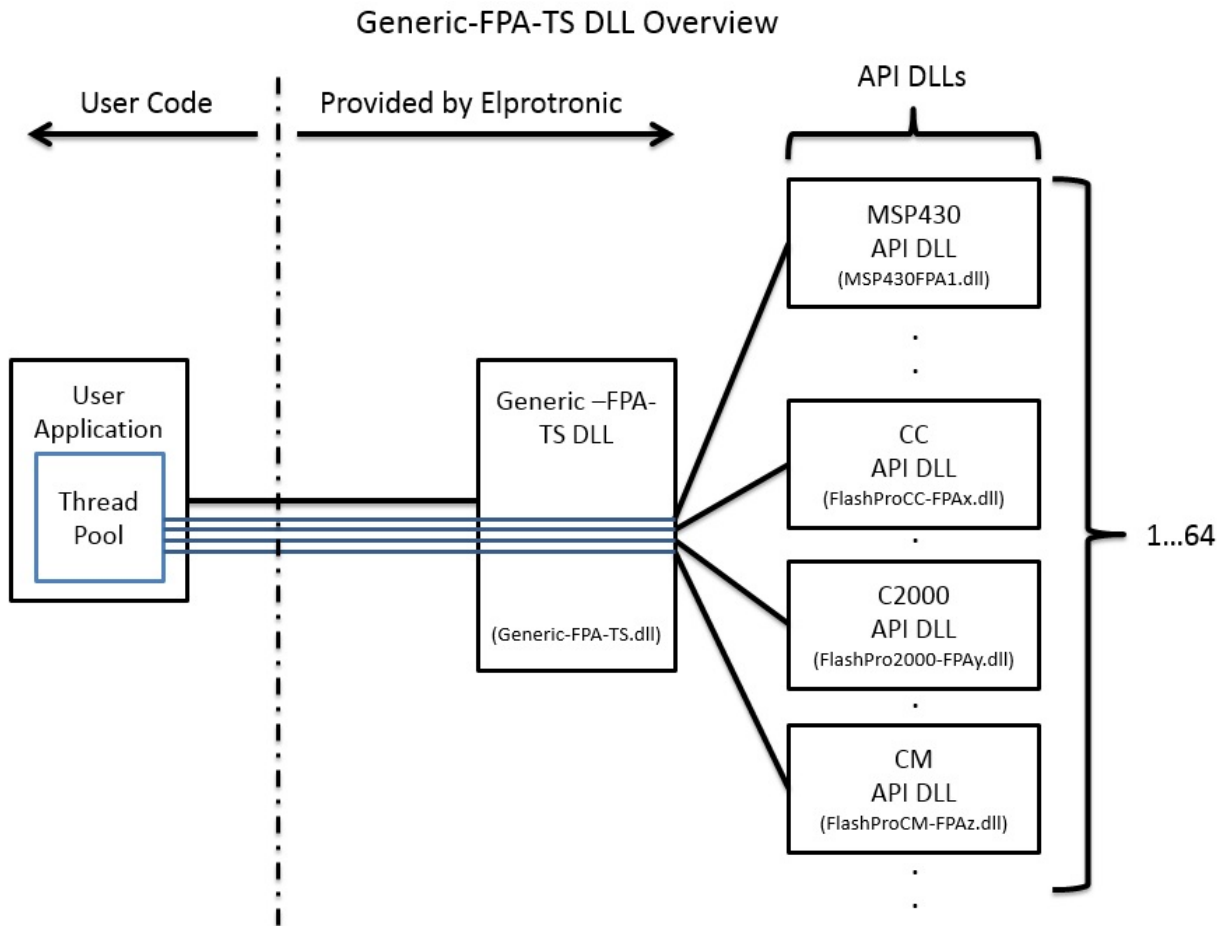


Figure 1.3: Generic-FPA-TS DLL setup where the user application can create its own thread pool and control up to 64 different adapters simultaneously.

fashion. Mutexes inside the Generic-FPA-TS DLL prevent two threads from issuing commands to the same adapter simultaneously; however, separate adapters can be controlled independently. All functions that control FPAs support this functionality. Using this DLL the user can call different functions for different adapters at the same time. The return value for each function call is only the return value from that chosen adapter's API DLL. Generic-FPA-TS DLL function calls that can only be accessed by one thread at a time are metadata functions that handle init/deinit and don't call API DLLs. These functions are:

```
TS_AssignFPAs      (new) - replaces F_OpenInstancesAndFPAs
TS_OpenOneInstance (new)
TS_CloseOneInstance (new)
TS_CloseInstances
```

All these aforementioned functions are protected by a single mutex. The exception is the function:

```
TS_GenericTSDLLVer
```

Which is a metadata function but can be accessed in parallel by all threads. Generic-FPA DLL functions that do not exist in the Generic-FPA-TS DLL are:

```
F_OpenInstancesAndFPAs - replaced by TS_AssignFPAs
F_Enable_FPA_index
F_Disable_FPA_index
F_Get_FPA_index
F_Set_FPA_index
F_Set_Timeout_Short
F_Set_Timeout_Long
F_Trace_ON
F_Trace_OFF
F_Debug_Enable
F_Debug_Disable
F_LastStatus
F_GenericDLLVer
```

Most of the aforementioned functions are no longer necessary because each adapter is controlled by a separate user thread. The Generic-FPA-TS DLL also does not support the timeout functionality described earlier, since worker threads are no longer controlled explicitly by the DLL.

Function prototypes for the Generic-FPA-TS DLL differ only from the Generic-FPA DLL in that an extra parameter, a `thread_struct_t *ti`, is necessary and the prefix TS is added. The extra parameter is defined as follows:

```
typedef struct
{
    int fpa_index;
```

```
    int debug_mode;
    int dll_call_status;
} thread_struct_t;
```

A pointer to this initialized structure needs to be passed every time a call is being made to most Generic-FPA-TS DLL functions. The `fpa_index`, between 1 and 64, specifies which adapter and the corresponding API DLL is to be invoked.

Fpa Index 0 is Invalid - Each Thread Controls One FPA

The `debug_mode` can be set to 1 if the user wishes to test calls to the Generic-FPA-TS DLL without actually calling API DLLs (otherwise set `debug` to 0). The `dll_call_status` returns the status of the call from the Generic-FPA-TS DLL to the application for debugging and error-checking purposes. The values for `dll_call_status` can be:

- `ELP_CALLED` - specific DLL function is supported by the API DLL of selected adapter, and was called.
- `ELP_BUSY` - DLL is busy serving another thread (two or more simultaneous calls were made with same `fpa_index`), non-blocking.
- `ELP_IMPL` - specific DLL function is supported by the API DLL of selected adapter (returned instead of `ELP_CALLED` if `debug_mode == 1`).
- `ELP_ENOTIMP` - specific DLL function is not supported by the API DLL of selected adapter.
- `ELP_ERANGE` - `fpa_index` out of range. Must be between 1 and 64.
- `ELP_ENOINIT` - API DLL for selected adapter did not initialize, or failed to load (API DLL not present, etc.).

These status values can aid the developer in debugging applications using the Generic-FPA-TS DLL. The return value of a function invocation for each thread is the result of calling the API DLL for the selected adapter, or `ELP_ENOTIMP`, if the selected adapter type does not support the invoked function (calling `TS_CC_Set_PC_and_RUN` with the MSP430 adapter selected would return such a value). For functions that are not supported by the API DLL, the specific API DLL is not actually called. Instead, metadata within the Generic-FPA-TS DLL keeps track of which functions are supported by each API DLL and acts accordingly.

1.11 Generic-FPA-TS DLLSetup

The Generic-FPA-TS DLL differs in several ways from the Generic-FPA DLL, one of which is the process of initialization. Whereas in the Generic-FPA DLL it is sufficient to call `F_OpenInstancesAndFPAs`, the Generic-FPA-TS DLL doesn't support that function. Instead, initialization is split into two steps:

1. One thread to call TS_AssignFPAs with the same setup file or setup string used to call F_OpenInstancesAndFPAs described in Section 1.4:

```
INT_X TS_AssignFPAs(char *FileName)
```

The number of successfully initialized FPAs will be returned. For the given setup file this number will be 4. At this stage, only metadata for each FPA is created and adapters can't be controlled yet.

2. Each worker thread calls TS_OpenOneInstance with the thread_struct_t set to its FPA index:

```
INT_X TS_OpenOneInstance( thread_struct_t *ti )
```

Function will return 1 if successful, 0 (zero) otherwise.

If successful, each thread can use its thread_struct_t block to control different adapters independently in parallel.

1.12 config.ini conflict

Multi-FPA DLLs for current adapters types, such as MSP430, CC, etc., implicitly use a local config.ini file when calling F_Initialization. The Generic-FPA DLL doesn't support using config.ini for the F_Initialization function because each adapter type uses a different config.ini file. Instead, delete any existing config.ini file in the current working directory, or invoke the following new function on all FPAs:

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

Select 0(zero) as input to disable automatic loading of config.ini during initialization. Since the config will not be loaded automatically from config.ini, instead invoke the following function to explicitly load the desired config file:

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

Select the proper FPA index using F_Set_FPA_index for each adapter and load the config files separately.

Chapter 2

Commandline Interface

2.1 Introduction

This chapter describes an alternative way of utilizing the capabilities of the Generic-FPA-TS DLL. Rather than incorporating the Generic-FPA-TS DLL into a custom application, the user can use the standalone Generic-Command-line tool. This tool executes the same functions as are available with the Generic-FPA-TS DLL, by parsing string input from the user and invoking the appropriate API-DLLs.

To support all the necessary features, the Generic-Command-line tool is composed of two applications, a server application that controls programming, and a client application that issues commands. Once started, the server is a persistent process that initializes a named pipe, initializes FPAs connected to the computer, maintains state for initialized FPAs, and invokes the Generic-FPA-TS DLL and associated API-DLLs for each FPA in use. The client application is stateless with respect to programming, and can be executed on demand to send one or many commands to the server.

The client application can be repeatedly executed from a script file with a few basic parameters to send one command at a time. Alternatively, the client application can be kept running to be used as a console for typing multiple commands over the course of the entire programming process. The former is optimal for the production environment, whereas the latter is better for testing and development. Multiple clients can connect to the same server at the same time, where the maximum number is 64. Each client can choose which FPA to control independently.

There is no requirement to use our client application. The user can copy our client code and integrate it into their own tester or production application.

2.2 Functionality

The Generic-Command-line relies on the Generic-FPA-TS DLL and associated API-DLLs. The directory that contains the server application executable must also contain the Generic-FPA-

TS DLL and any API-DLLs such as the MSP430FPA1.dll, GangPro430FPA1.dll, FlashPro2000-FPA1.dll, FlashProCC-FPA1.dll, GangProCC-FPA1.dll, FlashProARM-FPA1.dll, and GangProARM-FPA1.dll. From a functional point of view, the server application is a basic addition on top of the Generic-FPA-TS DLL. The server implements a thread-pool that allows it to handle commands and replies to and from all FPAs asynchronously. Figure 2.1 shows the relationship between the components that make up the command-line interface. A useful feature provided by this command-line interface is that it allows control of production over the network using already provided executables, for computers within the same workgroup as the server.

The client application code is provided as a demo in the installation package.

2.3 Server Configuration Flags

The server application is started by running the Generic-CommandLine-Server.exe executable. The only mandatory parameter is the setup file that contains FPA serial numbers (SNs) and types used for initialization. Multiple optional flags let you customize the server's behavior. The server executable can also be run in discovery mode, where it will print out metadata about connected FPAs and exit.

2.3.1 DISCOVER

Information about FPAs connected to the computer running this executable will be printed out, up to 64 FPAs. Use this to automatically generate a setup file.

```
>Generic-CommandLine-Server.exe DISCOVER
1 SN=20130005
  HW ID=6, HW sub-ID=1
  Full Access=1, Interface Type=3
  Access Key=1
  HW PN=USB-FPA-6.1
  Adapter Desc=FlashPro-ARM
2 ...
```

2.3.2 Setupfile(required)

The file path, relative to the server process, that specifies the location of the setup file. The setup file contains FPA SNs and adapter types to be used during initialization. An example setup file, called FPAs-setup.ini is shown below:

```
FPA-1 20140001 TYPE-ARM
FPA-2 20140002 TYPE-ARM-GP
FPA-4 20140003 TYPE-C2000
FPA-5 20140004 TYPE-MSP
```

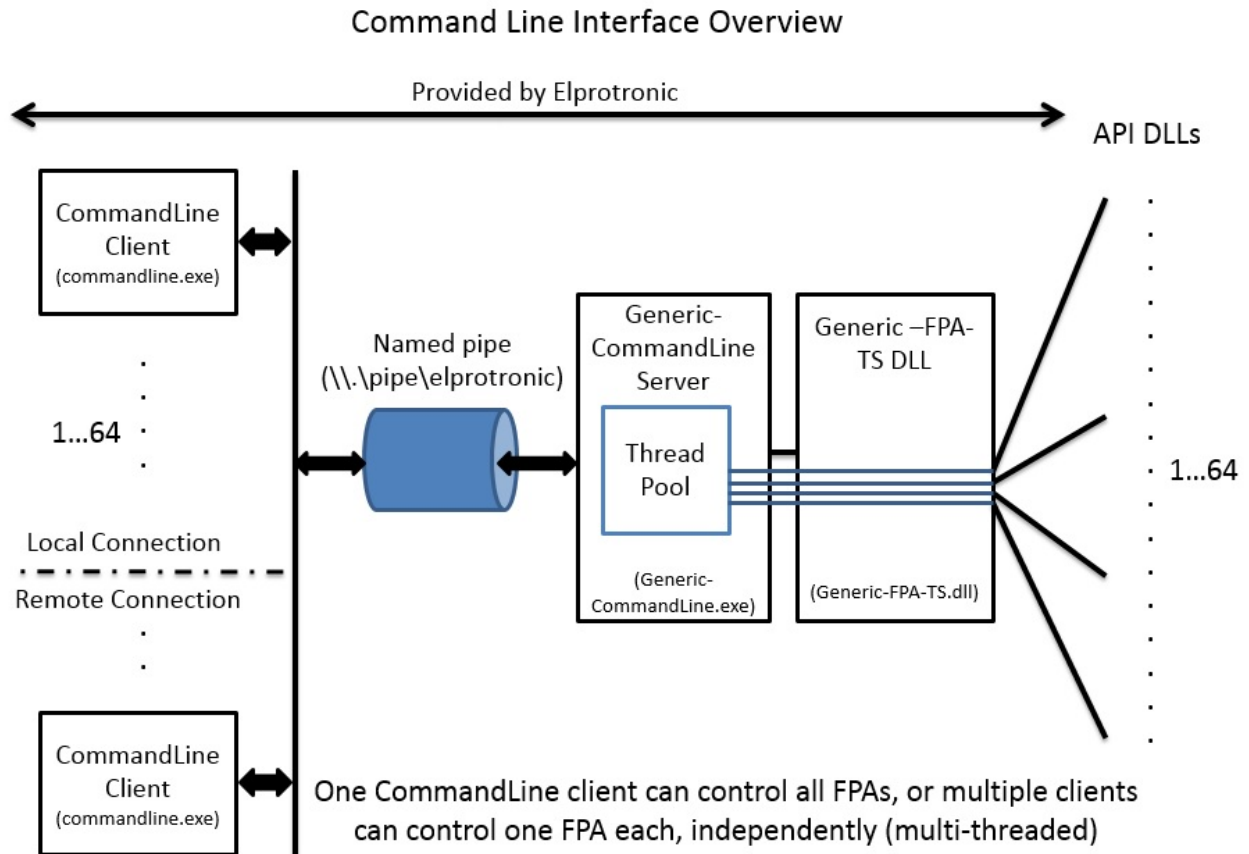


Figure 2.1: Generic-Command-line setup where the client application can control one, or up to 64 different adapters simultaneously. Multiple clients can connect simultaneously to control each FPA independently.

FPA-6 20140005 TYPE-MSP-GP
FPA-3 20140006 TYPE-CC
FPA-7 20140007 TYPE-CC-GP

In the example above, the computer where the server is running should have seven (7) FPAs connected via USB (for that many you would usually need a hub). Since the command-line server is running on top of the Generic-FPA-TS DLL, it can control different FPA types simultaneously. A FlashPro-X adapter can be configured using any type, but still must be set to one type at a time during initialization.

2.3.3 -configtest

Test configuration options, touch log files (open for append), and detect any errors in syntax. After the configuration setup has been parsed the process will exit. Does not perform FPA initialization.

2.3.4 -p <pipename>

Custom pipe name that the server should listen on. By default this value is:

```
\\.\pipe\elprotronic
```

2.3.5 -b

Client commands to server will be blocking. Server will execute command and return result to client that issued the command. No other commands can be executed until the server is finished with an outstanding command. This is the easiest way to use the command-line interface because it basically behaves as a single threaded application that performs one task at a time.

When blocking commands are disabled, client commands that require a call to the API-DLL will be issued asynchronously and delegated to a worker thread inside the server process. The server will respond to the client right away with a BUSY return value instead of the actual command return value. This is because the command's return value is not available yet. To obtain the command's return value, the client has to repeatedly poll the server using the *LastStatus < fpa >* command. The *LastStatus < fpa >* command will return BUSY until the last command for that FPA has finished executing, after which it will return the last command's actual return value.

There is a separate worker thread for each initialized FPA, so one or multiple clients can issue long running commands (like AutoProgram) to many FPAs at once. This allows asynchronous and independent control of all programming adapters, which is useful for running different tests or failure recovery when one or more target devices fail.

2.3.6 -f <level>

Client feedback level, between 0 and 3. The server responds to clients with strings that contain a lot of useful information. However, once the user has optimized the programming flow, a lot of

feedback might be unnecessary and clutter output. Therefore, the feedback level specifies what information is returned to the client after each command.

The higher the level, the more feedback is given. By lowering the feedback level you will exclude feedback associated with a higher number. The options for feedback are:

- echo back original command (3),
- information related to issued command (3),
- errors related to issued command (2),
- return value from issued command (1), and
- busy status (1).

When level 0 is specified, the server will return an empty string. Clients have to request command return values with the *LastStatus < fpa >* command. Naturally, level 0 option does not apply to the LastStatus command.

2.3.7 -v <level>

Server verbose level, between 0 and 3. The server can print status messages to stdout/stderr (default) or log file(s). This option specifies how much information should be printed.

The higher the level, the more information is printed. By lowering the verbose level you will exclude information associated with a higher number. The options for information printed are:

- echo client command (3),
- information related to issued command (3),
- initialization results, number of FPAs, SNs, (2),
- return value from issued command (2), and
- errors related to issued command (1),

At level 0 the server produces no output, except if the initial configuration parameters are incorrect.

2.3.8 -of <filename>

Send server non-error output to file with given filename. If this option is not specified, then server output messages (depends on -v option) will go to stdout. If -of.fpa option is specified then this log file will not store FPA specific output.

2.3.9 -ofr <filename>

Send server error output to file with given filename. If this option is not specified, then server output messages (depends on -v option) will go to stderr. If -ofr_fpa option is specified then this log file will not store FPA specific output. Errors during server initialization will always go to stderr.

2.3.10 -of_fpa <i> <filename>

Send server non-error output exclusive to FPA index i to given filename. If this option is not specified, the output is combined with general server non-error output given by option -of.

2.3.11 -ofr_fpa <i> <filename>

Send server error output exclusive to FPA index i to given filename. If this option is not specified, the output is combined with general server error output given by option -ofr. Errors during server initialization will always go to stderr.

2.3.12 -ofq

Stop server if write to any log file fails for any reason. Logging error will be dumped to -ofr <filename>, or if that fails then to stderr before exit. The server will exit regardless of this setting if it fails to open the provided files during initialization.

2.4 Getting Started and Example Usage

To setup the command-line interface follow these steps:

1. Download and install the Generic-DLL package from the website (under Download section).
2. Move the Generic-FPA-DLLs directory to a location where you can work freely without administrator privileges (not *Program Files* directory).
3. Copy your config and code files to the *bin* sub-directory.
4. Your *bin* sub-directory should have:
 - Generic-CommandLine-Server.exe - server process will control everything on machine with FPAs
 - CommandLine-Client.exe - client executable reads your commands and sends them to server via named pipe
5. Source code for client is provided in the *CommandLine-Client* directory, it can be copied into your tester for direct connection to Generic-CommandLine-Server process.

6. Connect your FPAs to the computer that will run the server executable.
7. Create a list file, i.e. FPAs-setup.ini, that will list the serial numbers (SNs) and types of your FPAs. For example:

```
FPA-1 20130005 TYPE-ARM
FPA-2 20100565 TYPE-ARM
```

8. Open a command console and run server executable using the following example parameters:

```
Generic-CommandLine-Server.exe -f 3 -v 3 FPAs-setup.ini -b
```

You can list the available options and their description by simply running the executable with no parameters. To widen the command console for easier viewing, right click the top bar of the window, go to properties – >layout – >window size and increase that number.

9. Now the server is running in verbose mode and you should get lots of feedback. An example of what you should see after initialization is shown in Figure 2.2. An important option here is *-b* which will cause calls to the server to be blocking. In this mode the server will execute commands synchronously and only respond to the client when the command has finished. This mode is the easiest to start using the command-line tool. If you wish to connect multiple clients in a production setup, and control all FPAs independently of each other, you would disable the blocking option.
10. Run client executable with no parameters. The actual location of the client executable is irrelevant because all file paths read are relative to the server process. The client can also run remotely if the two computers are in the same work group. For a remote connection you should modify the pipe name in the client to reflect the correct path to the server computer.

```
CommandLine-Client.exe
```

11. When the client has started, it will automatically connect to the pipe specified in the source code. For a network connection, enter the server's computer name instead of the dot. The two computers should be in the same work group. You might also have to log into the server computer from the client computer using the Windows Explorer network browser first (double click the server computer and enter credentials), before the remote pipe connection is allowed.
12. Once connected to the server, you can begin typing commands or type HELP for a list of available commands. Commands are not case sensitive. Listed commands are identical to DLL commands, except that there is no F_ or TS_ prefix. Command parameters are separated by spaces instead of contained inside brackets. When the client first connects, its designated FPA index is -1, meaning it cannot control any FPA, but it can still issue metadata commands. For

```

C:\Windows\system32\cmd.exe - Generic-CommandLine-Server.exe -v 3 -f 3 FPAs-setup.ini -b
E:\Tmp\Generic-FPA-DLLs\bin>
E:\Tmp\Generic-FPA-DLLs\bin>ls
ARM-128k.txt          FlashPro2000-FPA1.dll  FlashProCC-FPA1.dll  Generic-CommandLine-Server.exe
CommandLine-Client.exe FlashProARM-FPA1.dll  GangPro430FPA1.dll  Generic-FPA-Demo.exe
FPAs-setup.ini       FlashProARM-FPA2.dll  GangProCC-FPA1.dll  Generic-FPA-TS.dll
E:\Tmp\Generic-FPA-DLLs\bin>Generic-CommandLine-Server.exe -v 3 -f 3 FPAs-setup.ini -b
[05/Jun/2015:15:32:53] INIT: opened file FPAs-setup.ini
[05/Jun/2015:15:32:54] INIT: Found 2 adapters
[05/Jun/2015:15:32:54] INIT: 1: Get_FPA_Label, SN=20130005
[05/Jun/2015:15:32:54] INIT: HW ID=6, HW sub-ID=1
[05/Jun/2015:15:32:54] INIT: Full Access=1, Interface Type=3
[05/Jun/2015:15:32:54] INIT: Access Key=1
[05/Jun/2015:15:32:54] INIT: HW PN=USB-FPA-6.1
[05/Jun/2015:15:32:54] INIT: Adapter Desc=FlashPro-X
[05/Jun/2015:15:32:54] INIT: 2: Get_FPA_Label, SN=20100565
[05/Jun/2015:15:32:54] INIT: HW ID=6, HW sub-ID=0
[05/Jun/2015:15:32:54] INIT: Full Access=1, Interface Type=3
[05/Jun/2015:15:32:54] INIT: Access Key=1
[05/Jun/2015:15:32:54] INIT: HW PN=USB-FPA-6.0
[05/Jun/2015:15:32:54] INIT: Adapter Desc=FlashPro-X

```

Figure 2.2: Generic-Command-line server process started with the FPA setup file as input (required). Two FPAs specified in the setup file were successfully initialized, and the server is ready to accept client connections.

example, it can change its FPA index, or ask for the LastStatus from the FPA index provided as parameter. The HELP command will list all commands for all FPA types when the FPA index is -1. When the FPA index is changed to an initialized FPA, the HELP command will list commands only for that FPA type (ARM, MSP, etc.)

- The basic commands required to program a target MCU are given in the example below:

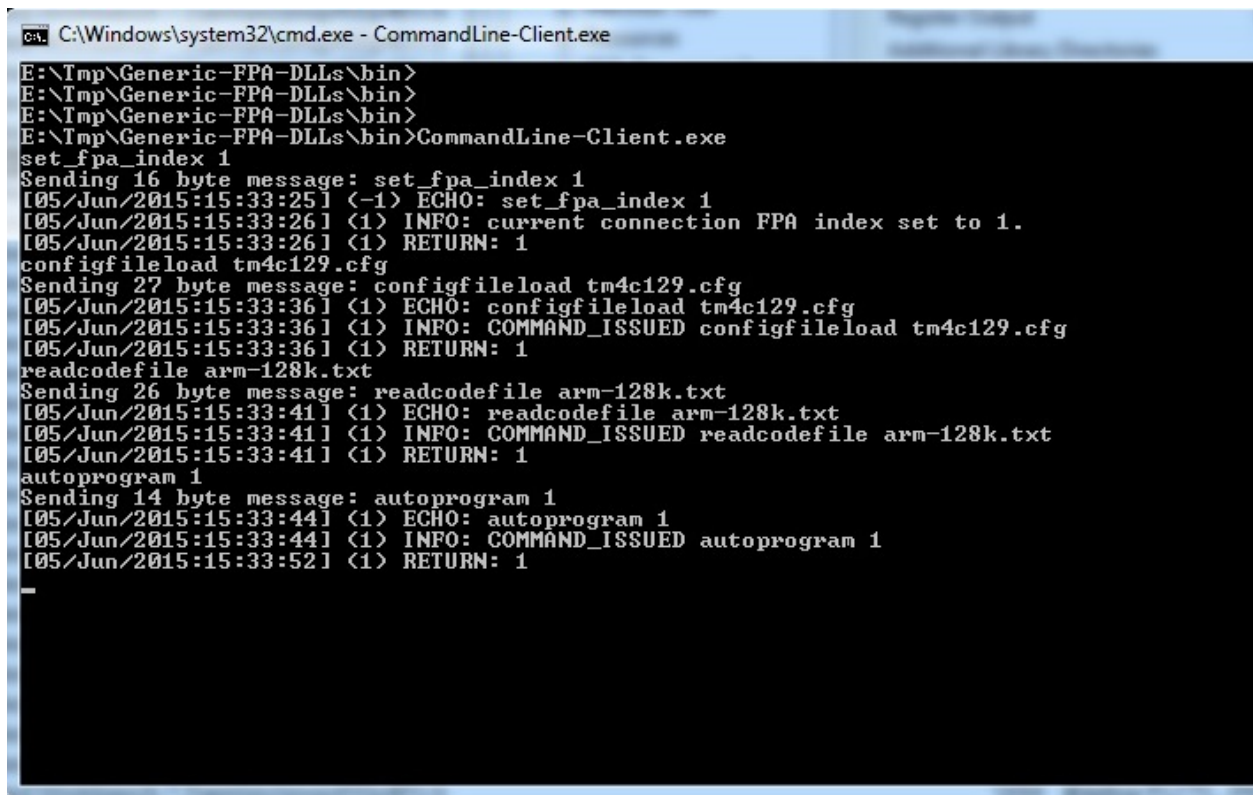
```

set_fpa_index 1
configfileload tm4c129.cfg
readcodefile arm-128k.txt
autoprogram 1

```

Figure 2.3 shows the result in the client console.

- After the client has issued the aforementioned commands, the result in the server console is shown in Figure 2.4.
- To terminate the server process using the client type the CloseInstances command.
- If you want the server to keep running and program more devices later on, simply close the client application and reconnect later.



```
C:\Windows\system32\cmd.exe - CommandLine-Client.exe
E:\Tmp\Generic-FPA-DLLs\bin>
E:\Tmp\Generic-FPA-DLLs\bin>
E:\Tmp\Generic-FPA-DLLs\bin>
E:\Tmp\Generic-FPA-DLLs\bin>CommandLine-Client.exe
set_fpa_index 1
Sending 16 byte message: set_fpa_index 1
[05/Jun/2015:15:33:25] (-) ECHO: set_fpa_index 1
[05/Jun/2015:15:33:26] (1) INFO: current connection FPA index set to 1.
[05/Jun/2015:15:33:26] (1) RETURN: 1
configfileload tm4c129.cfg
Sending 27 byte message: configfileload tm4c129.cfg
[05/Jun/2015:15:33:36] (1) ECHO: configfileload tm4c129.cfg
[05/Jun/2015:15:33:36] (1) INFO: COMMAND_ISSUED configfileload tm4c129.cfg
[05/Jun/2015:15:33:36] (1) RETURN: 1
readcodefile arm-128k.txt
Sending 26 byte message: readcodefile arm-128k.txt
[05/Jun/2015:15:33:41] (1) ECHO: readcodefile arm-128k.txt
[05/Jun/2015:15:33:41] (1) INFO: COMMAND_ISSUED readcodefile arm-128k.txt
[05/Jun/2015:15:33:41] (1) RETURN: 1
autoprogram 1
Sending 14 byte message: autoprogram 1
[05/Jun/2015:15:33:44] (1) ECHO: autoprogram 1
[05/Jun/2015:15:33:44] (1) INFO: COMMAND_ISSUED autoprogram 1
[05/Jun/2015:15:33:52] (1) RETURN: 1
-
```

Figure 2.3: Generic-Command-line client process connected to the server. This application's source code can be integrated into your tester to communicate with the command-line server directly.

```
ca. C:\Windows\system32\cmd.exe - Generic-CommandLine-Server.exe -v 3 -f 3 FPAs-setup.ini -b
[05/Jun/2015:15:32:54] INIT: 1: Get_FPA_Label, SN=20130005
[05/Jun/2015:15:32:54] INIT: HW ID=6, HW sub-ID=1
[05/Jun/2015:15:32:54] INIT: Full Access=1, Interface Type=3
[05/Jun/2015:15:32:54] INIT: Access Key=1
[05/Jun/2015:15:32:54] INIT: HW PN=USB-FPA-6.1
[05/Jun/2015:15:32:54] INIT: Adapter Desc=FlashPro-X
[05/Jun/2015:15:32:54] INIT: 2: Get_FPA_Label, SN=20100565
[05/Jun/2015:15:32:54] INIT: HW ID=6, HW sub-ID=0
[05/Jun/2015:15:32:54] INIT: Full Access=1, Interface Type=3
[05/Jun/2015:15:32:54] INIT: Access Key=1
[05/Jun/2015:15:32:54] INIT: HW PN=USB-FPA-6.0
[05/Jun/2015:15:32:54] INIT: Adapter Desc=FlashPro-X
[05/Jun/2015:15:33:25] (-1) ECHO: set_fpa_index 1
[05/Jun/2015:15:33:26] (1) INFO: current connection FPA index set to 1.
[05/Jun/2015:15:33:26] (1) RETURN: 1
[05/Jun/2015:15:33:36] (1) ECHO: configfileload tm4c129.cfg
[05/Jun/2015:15:33:36] (1) INFO: COMMAND_ISSUED configfileload tm4c129.cfg
[05/Jun/2015:15:33:36] (1) RETURN: 1
[05/Jun/2015:15:33:41] (1) ECHO: readcodefile arm-128k.txt
[05/Jun/2015:15:33:41] (1) INFO: COMMAND_ISSUED readcodefile arm-128k.txt
[05/Jun/2015:15:33:41] (1) RETURN: 1
[05/Jun/2015:15:33:44] (1) ECHO: autoprogram 1
[05/Jun/2015:15:33:44] (1) INFO: COMMAND_ISSUED autoprogram 1
[05/Jun/2015:15:33:52] (1) RETURN: 1
```

Figure 2.4: Generic-Command-line server process has been given the command to select FPA 1, load a config file, load a code file, and finally the command to AutoProgram. When a new client connects, its default FPA index is -1. The client must issue the set_fpa_index command to control a specific FPA.

2.5 Script Mode

Often times the advantage of a command-line tool is that it can be easily integrated into a script or batch file. The command-line client demo provided already comes with a few options that make it ideal for script usage. Since the server process is the one that maintains all the state, the client executable can be invoked and closed at will.

Figure 2.5 shows the client executing the same programming sequence as shown in the previous section using a few optional flags. The client connects to the specified pipe name, selects the chosen FPA index, and invokes the desired command. The client executable is ran three times to execute each of the commands separately. Figure 2.6 shows the server output for this programming sequence.

```
C:\Windows\system32\cmd.exe
E:\Tmp\Generic-FPA-DLLs\bin>
E:\Tmp\Generic-FPA-DLLs\bin>
E:\Tmp\Generic-FPA-DLLs\bin>
E:\Tmp\Generic-FPA-DLLs\bin>CommandLine-Client.exe -p \\.\pipe\elprotronic -i 1 -m configfileload tm4c129.cfg
Sending 16 byte message: set_fpa_index 1
[05/Jun/2015:15:50:32] (-1) ECHO: set_fpa_index 1
[05/Jun/2015:15:50:32] (1) INFO: current connection FPA index set to 1.
[05/Jun/2015:15:50:32] (1) RETURN: 1
Sending 27 byte message: configfileload tm4c129.cfg
[05/Jun/2015:15:50:32] (1) ECHO: configfileload tm4c129.cfg
[05/Jun/2015:15:50:32] (1) INFO: COMMAND_ISSUED configfileload tm4c129.cfg
[05/Jun/2015:15:50:32] (1) RETURN: 1
E:\Tmp\Generic-FPA-DLLs\bin>CommandLine-Client.exe -p \\.\pipe\elprotronic -i 1 -m readcodefile arm-128k.txt
Sending 16 byte message: set_fpa_index 1
[05/Jun/2015:15:50:35] (-1) ECHO: set_fpa_index 1
[05/Jun/2015:15:50:35] (1) INFO: current connection FPA index set to 1.
[05/Jun/2015:15:50:35] (1) RETURN: 1
Sending 26 byte message: readcodefile arm-128k.txt
[05/Jun/2015:15:50:35] (1) ECHO: readcodefile arm-128k.txt
[05/Jun/2015:15:50:35] (1) INFO: COMMAND_ISSUED readcodefile arm-128k.txt
[05/Jun/2015:15:50:35] (1) RETURN: 1
E:\Tmp\Generic-FPA-DLLs\bin>CommandLine-Client.exe -p \\.\pipe\elprotronic -i 1 -m autoprogram 1
Sending 16 byte message: set_fpa_index 1
[05/Jun/2015:15:50:39] (-1) ECHO: set_fpa_index 1
[05/Jun/2015:15:50:39] (1) INFO: current connection FPA index set to 1.
[05/Jun/2015:15:50:39] (1) RETURN: 1
Sending 14 byte message: autoprogram 1
[05/Jun/2015:15:50:39] (1) ECHO: autoprogram 1
[05/Jun/2015:15:50:39] (1) INFO: COMMAND_ISSUED autoprogram 1
[05/Jun/2015:15:50:48] (1) RETURN: 1
E:\Tmp\Generic-FPA-DLLs\bin>_
```

Figure 2.5: Generic-Command-line client process has been given the options to connect to a desired pipe name, select FPA 1, and invoke a given command. After the reply is received, the client exits and can be ran again to invoke another command.

```

C:\Windows\system32\cmd.exe - Generic-CommandLine-Server.exe -v 3 -f 3 FPAs-setup.ini -b
E:\Tmp\Generic-FPA-DLLs\bin>Generic-CommandLine-Server.exe -v 3 -f 3 FPAs-setup.ini -b
[05/Jun/2015:15:50:24] INIT: opened file FPAs-setup.ini
[05/Jun/2015:15:50:25] INIT: Found 2 adapters
[05/Jun/2015:15:50:25] INIT: 1: Get_FPA_Label, SM=20130005
[05/Jun/2015:15:50:25] INIT:   HW ID=6, HW sub-ID=1
[05/Jun/2015:15:50:25] INIT:   Full Access=1, Interface Type=3
[05/Jun/2015:15:50:25] INIT:   Access Key=1
[05/Jun/2015:15:50:25] INIT:   HW PN=USB-FPA-6.1
[05/Jun/2015:15:50:25] INIT:   Adapter Desc=FlashPro-X
[05/Jun/2015:15:50:26] INIT: 2: Get_FPA_Label, SM=20100565
[05/Jun/2015:15:50:26] INIT:   HW ID=6, HW sub-ID=0
[05/Jun/2015:15:50:26] INIT:   Full Access=1, Interface Type=3
[05/Jun/2015:15:50:26] INIT:   Access Key=1
[05/Jun/2015:15:50:26] INIT:   HW PN=USB-FPA-6.0
[05/Jun/2015:15:50:26] INIT:   Adapter Desc=FlashPro-X
[05/Jun/2015:15:50:32] (-) ECHO: set_fpa_index 1
[05/Jun/2015:15:50:32] (1) INFO: current connection FPA index set to 1.
[05/Jun/2015:15:50:32] (1) RETURN: 1
[05/Jun/2015:15:50:32] (1) ECHO: configfileload tm4c129.cfg
[05/Jun/2015:15:50:32] (1) INFO: COMMAND_ISSUED configfileload tm4c129.cfg
[05/Jun/2015:15:50:32] (1) RETURN: 1
[05/Jun/2015:15:50:35] (-) ECHO: set_fpa_index 1
[05/Jun/2015:15:50:35] (1) INFO: current connection FPA index set to 1.
[05/Jun/2015:15:50:35] (1) RETURN: 1
[05/Jun/2015:15:50:35] (1) ECHO: readcodefile arm-128k.txt
[05/Jun/2015:15:50:35] (1) INFO: COMMAND_ISSUED readcodefile arm-128k.txt
[05/Jun/2015:15:50:35] (1) RETURN: 1
[05/Jun/2015:15:50:39] (-) ECHO: set_fpa_index 1
[05/Jun/2015:15:50:39] (1) INFO: current connection FPA index set to 1.
[05/Jun/2015:15:50:39] (1) RETURN: 1
[05/Jun/2015:15:50:39] (1) ECHO: autoprogram 1
[05/Jun/2015:15:50:39] (1) INFO: COMMAND_ISSUED autoprogram 1
[05/Jun/2015:15:50:48] (1) RETURN: 1

```

Figure 2.6: Generic-Command-line server process output when client process is invoking one command at a time. The client flags select the appropriate FPA index and command to be executed.

2.5.1 Client Configuration Flags

The client application source code is provided in the installation package. Nevertheless, a few useful flags were added to this demo application to allow it to function easily in scripts and batch files. Feel free to modify the client application as you wish, or integrate it into your tester.

2.5.2 -configtest

Test configuration options, and detect any errors in syntax. After the configuration setup has been parsed the process will exit.

2.5.3 -p <pipename>

Custom pipe name that the client should connect to. By default this value is:

```
\\.\pipe\elprotronic
```

If the server process is on a remote computer, an example pipe name would be:

```
\\SERVER-PC\pipe\elprotronic
```

If you want to connect over the network, the two computers should be in the same work group. You might also have to log into the server computer from the client computer using the Windows Explorer network browser first (double click the server computer and enter credentials), before the remote pipe connection is allowed.

2.5.4 -i <i>

When client connects, change to specified FPA index right away. Choose between 1 and 64.

2.5.5 -m <command + params>

Issue one shot command and exit, useful with -i option for scripts. Everything after -m flag will be formed into one string as a command with parameters.

Chapter 3

DLL Functions

This chapter lists all Generic-FPA DLL and Generic-FPA-TS DLL functions, their API DLL equivalents, and which Flash Programming Adapters (FPAs) support them. Tables 3.1, 3.2, 3.3, and 3.4 list Generic-FPA DLL functions, whereas Tables 3.5, 3.6, 3.7, and 3.8 list Generic-FPA-TS DLL functions.

The first table of each series starts off by listing metadata functions that do not explicitly call API DLLs, but instead are used to configure the Generic-FPA DLL or Generic-FPA-TS DLL. These functions do not have an API DLL equivalent because they are used to setup and control the environment of the top-level DLL so that it can work properly. Calls to the remaining functions are forwarded to selected and enabled FPAs and their respective API DLLs. The table columns provide the following information:

1. Generic-FPA DLL or Generic-FPA-TS DLL Function Name,
2. Equivalent API DLL Function Name (not applicable to metadata functions),
3. Function supported by MSP FPAs (MSP430FPA1.dll).
4. Function supported by MSP-GP FPAs (GangPro430FPA1.dll).
5. Function supported by C2000 FPAs (FlashPro2000-FPA1.dll).
6. Function supported by CC FPAs (FlashProCC-FPA1.dll).
7. Function supported by CC-GP FPAs (GangProCC-FPA1.dll).
8. Function supported by ARM FPAs (FlashProARM-FPA1.dll).
9. Function supported by ARM-GP FPAs (GangProARM-FPA1.dll).

Columns 3-8 denote support for individual FPA types and their respective API DLLs. If the function is supported then a 1 (one) is entered in the corresponding cell; conversely, if the function is not supported then a 0 (zero) is entered in that cell.

Generic-FPA DLLFunction Name	API DLL Function Name	MSP	MSP-GP	C2000	CC	CC-GP	ARM	ARM-GP
F_DiscoverFPAs	N/A	1	1	1	1	1	1	1
F_OpenInstancesAndFPAs	N/A	1	1	1	1	1	1	1
F_CloseInstances	N/A	1	1	1	1	1	1	1
F_Enable_FPA_index	N/A	1	1	1	1	1	1	1
F_Disable_FPA_index	N/A	1	1	1	1	1	1	1
F_Set_FPA_index	N/A	1	1	1	1	1	1	1
F_Get_FPA_index	N/A	1	1	1	1	1	1	1
F_Get_FPA_type	N/A	1	1	1	1	1	1	1
F_LastStatus	N/A	1	1	1	1	1	1	1
F_Set_Timeout_Short	N/A	1	1	1	1	1	1	1
F_Set_Timeout_Long	N/A	1	1	1	1	1	1	1
F_Trace_ON	N/A	1	1	1	1	1	1	1
F_Trace_OFF	N/A	1	1	1	1	1	1	1
F_Debug_Enable	N/A	1	1	1	1	1	1	1
F_Debug_Disable	N/A	1	1	1	1	1	1	1
F_GenericDLLVer	N/A	1	1	1	1	1	1	1
F_GenericTSDLLVer	N/A	1	1	1	1	1	1	1
F_DLLTypeVer	F_DLLTypeVer	1	1	1	1	1	1	1
F_Check_FPA_access	F_Check_FPA_access	1	1	1	1	1	1	1
F_Get_FPA_SN	F_Get_FPA_SN	1	1	1	1	1	1	1
F_Get_FPA_Label	F_Get_FPA_Label	1	0	1	1	0	1	1
F_Initialization	F_Initialization	1	1	1	1	1	1	1
F_Use_Config_INI	F_Use_Config_INI	1	1	1	1	1	1	1
F_Close_All	F_Close_All	1	1	1	1	1	1	1
F_Power_Target	F_Power_Target	1	1	0	1	1	1	1
F_Reset_Target	F_Reset_Target	1	1	1	1	1	1	1
F_ReportMessage	F_ReportMessage	1	1	1	1	1	1	1
F_Report_Message	F_Report_Message	1	1	1	1	1	1	1
F_ReadCodeFile	F_ReadCodeFile	0	0	1	0	1	1	1
F_MSP_ReadCodeFile	F_ReadCodeFile	1	1	0	0	0	0	0
F_CC_ReadCodeFile	F_ReadCodeFile	0	0	0	1	0	0	0
F_AppendCodeFile	F_AppendCodeFile	0	0	0	0	0	1	1
F_ReadPasswFile	F_ReadPasswFile	0	0	1	0	0	0	0
F_MSP_ReadPasswFile	F_ReadPasswFile	1	1	0	0	0	0	0
F_ConfigFileLoad	F_ConfigFileLoad	1	1	1	1	1	1	1
F_SetConfig	F_SetConfig	1	1	1	1	1	0	0
F_GetConfig	F_GetConfig	1	1	1	1	1	0	0
F_Get_Config_Name_List	F_Get_Config_Name_List	1	1	1	1	1	1	1
F_Get_Config_Value_By_Name	F_Get_Config_Value_By_Name	1	1	1	1	1	1	1
F_Set_Config_Value_By_Name	F_Set_Config_Value_By_Name	1	1	1	1	1	1	1
F_Put_Byte_to_Buffer	F_Put_Byte_to_Buffer	1	0	0	1	0	1	1
F_Get_Byte_from_Buffer	F_Get_Byte_from_Buffer	1	0	0	1	0	1	0
F_GetReportMessageChar	F_GetReportMessageChar	1	1	1	1	1	1	1
F_Clr_Code_Buffer	F_Clr_Code_Buffer	1	1	1	1	1	1	1
F_Put_Byte_to_Code_Buffer	F_Put_Byte_to_Code_Buffer	1	1	0	1	1	1	1
F_Get_Byte_from_Code_Buffer	F_Get_Byte_from_Code_Buffer	1	1	0	1	1	1	1
F_Put_IEEEAddr64_to_Buffer	F_Put_IEEEAddr64_to_Buffer	0	0	0	1	0	0	0

Table 3.1: The following table lists Generic-FPA DLL functions, their API DLL equivalents and which FPAs support them (part 1/4).

Generic-FPA DLLFunction Name	API DLL Function Name	MSP	MSP-GP	C2000	CC	CC-GP	ARM	ARM-GP
F_Get_IEEEAddr64_from_Buffer	F_Get_IEEEAddr64_from_Buffer	0	0	0	1	0	0	0
F_Put_IEEEAddr_Byte_to_Buffer	F_Put_IEEEAddr_Byte_to_Buffer	0	0	0	1	0	0	0
F_Get_IEEEAddr_Byte_from_Buffer	F_Get_IEEEAddr_Byte_from_Buffer	0	0	0	1	0	0	0
F_Put_Byte_to_Password_Buffer	F_Put_Byte_to_Password_Buffer	1	1	0	0	0	0	0
F_Get_Byte_from_Password_Buffer	F_Get_Byte_from_Password_Buffer	1	1	0	0	0	0	0
F_Get_Targets.Result	F_Get_Targets.Result	0	1	0	0	1	0	1
F_Get_Active_Targets_Mask	F_Get_Active_Targets_Mask	0	1	0	0	1	0	1
F_Get_DCO_constant	F_Get_DCO_constant	0	1	0	0	0	0	0
F_Set_DCO_constant	F_Set_DCO_constant	0	1	0	0	0	0	0
F_GP_Test_DCO_Frequency	F_Test_DCO_Frequency	0	1	0	0	0	0	0
F_Get_DCO_Freq_result	F_Get_DCO_Freq_result	0	1	0	0	0	0	0
F_AutoProgram	F_AutoProgram	1	1	1	1	1	1	1
F_VerifyFuseOrPassword	F_VerifyFuseOrPassword	1	1	0	0	0	0	0
F_Memory_Erase	F_Memory_Erase	1	1	1	1	1	1	1
F_Memory_Blank_Check	F_Memory_Blank_Check	1	1	1	1	1	1	1
F_Memory_Write	F_Memory_Write	1	1	1	1	1	1	1
F_Memory_Verify	F_Memory_Verify	1	1	1	1	1	1	1
F_Memory_Read	F_Memory_Read	0	0	1	1	0	1	1
F_MSP_Memory_Read	F_Memory_Read	1	0	0	0	0	0	0
F_Gang_Flash_Read	F_Gang_Flash_Read	0	1	0	0	1	0	0
F_Memory_Write_Data	F_Memory_Write_Data	1	0	0	0	0	0	0
F_Memory_Read_Data	F_Memory_Read_Data	1	0	0	0	0	0	0
F_Write_IEEE_Address	F_Write_IEEE_Address	0	0	0	1	1	0	0
F_Read_IEEE_Address	F_Read_IEEE_Address	0	0	0	1	1	0	0
F_Write_Lock_Bits	F_Write_Lock_Bits	0	0	0	1	1	0	0
F_Open_Target_Device	F_Open_Target_Device	1	1	1	1	1	1	1
F_Close_Target_Device	F_Close_Target_Device	1	1	1	1	1	1	1
F_Segment_Erase	F_Segment_Erase	1	1	1	1	1	1	1
F_Get_Sector_Size	F_Get_Sector_Size	1	1	1	1	1	1	1
F_Sectors_Blank_Check	F_Sectors_Blank_Check	1	1	1	1	1	1	1
F_Copy_Buffer_to_Flash	F_Copy_Buffer_to_Flash	1	1	1	1	1	1	1
F_Copy_Flash_to_Buffer	F_Copy_Flash_to_Buffer	1	0	1	1	0	1	0
F_Copy_All_Flash_to_Buffer	F_Copy_All_Flash_to_Buffer	1	0	0	0	0	0	0
F_Write_Word	F_Write_Word	1	1	0	0	0	0	0
F_Read_Word	F_Read_Word	1	0	1	0	0	0	0
F_Write_Byte	F_Write_Byte	1	1	0	0	0	0	0
F_Read_Byte	F_Read_Byte	1	0	0	0	0	0	0
F_Copy_Buffer_to_RAM	F_Copy_Buffer_to_RAM	1	1	0	0	0	0	0
F_Copy_RAM_to_Buffer	F_Copy_RAM_to_Buffer	1	0	0	0	0	0	0
F_Write_Byte_to_XRAM	F_Write_Byte_to_XRAM	0	0	0	1	1	0	0
F_Read_Byte_from_XRAM	F_Read_Byte_from_XRAM	0	0	0	1	0	0	0
F_Copy_Buffer_to_XRAM	F_Copy_Buffer_to_XRAM	0	0	0	1	1	0	0
F_Copy_XRAM_to_Buffer	F_Copy_XRAM_to_Buffer	0	0	0	1	0	0	0
F_Write_Byte_to_direct_RAM	F_Write_Byte_to_direct_RAM	0	0	0	1	1	0	0
F_Read_Byte_from_direct_RAM	F_Read_Byte_from_direct_RAM	0	0	0	1	0	0	0
F_Copy_Buffer_to_direct_RAM	F_Copy_Buffer_to_direct_RAM	0	0	0	1	1	0	0
F_Copy_direct_RAM_to_Buffer	F_Copy_direct_RAM_to_Buffer	0	0	0	1	0	0	0

Table 3.2: The following table lists Generic-FPA DLL functions, their API DLL equivalents and which FPAs support them (part 2/4).

Generic-FPA DLLFunction Name	API DLL Function Name	MSP	MSP-GP	C2000	CC	CC-GP	ARM	ARM-GP
F_Set_PC_and_RUN	F_Set_PC_and_RUN	1	1	0	0	0	1	1
F_CC_Set_PC_and_RUN	F_Set_PC_and_RUN	0	0	0	1	1	0	0
F_Get_MCU_Data	F_Get_MCU_Data	0	0	1	1	0	1	1
F_Get_Targets_Vcc	F_Get_Targets_Vcc	1	1	1	1	1	1	1
F_Get_CodeCS	F_Get_CodeCS	1	1	1	1	1	1	1
F_Get_Device_Info	F_Get_Device_Info	1	1	1	1	1	1	1
F_Set_MCU_Name	F_Set_MCU_Name	1	1	1	1	1	1	1
F_Set_fpa_io_state	F_Set_fpa_io_state	1	1	0	1	1	1	1
F_Blow_Fuse	F_Blow_Fuse	1	1	0	0	0	0	0
F_Restore_JTAG_Security_Fuse	F_Restore_JTAG_Security_Fuse	1	0	0	0	0	0	0
F_Capture_PC_Addr	F_Capture_PC_Addr	1	0	0	0	0	1	0
F_GP_Capture_PC_Addr	F_Capture_PC_Addr	0	1	0	0	0	0	1
F_Synch_CPU_JTAG	F_Synch_CPU_JTAG	1	1	0	0	0	1	1
F_Adj_DCO_Frequency	F_Adj_DCO_Frequency	1	1	0	0	0	0	0
F_Test_DCO_Frequency	F_Test_DCO_Frequency	1	0	0	0	0	0	0
F_Customize	F_Customize	1	1	0	0	0	0	0
F_init_custom_jtag	F_init_custom_jtag	1	0	0	0	0	0	0
F_custom_jtag_stream	F_custom_jtag_stream	1	0	0	0	0	0	0
F_Custom_Function	F_Custom_Function	1	0	0	0	0	0	0
F_Put_Word_to_Buffer	F_Put_Word_to_Buffer	0	0	1	0	0	0	0
F_Get_Word_from_Buffer	F_Get_Word_from_Buffer	0	0	1	0	0	0	0
F_Put_Word_to_Code_Buffer	F_Put_Word_to_Code_Buffer	0	0	1	0	0	0	0
F_Get_Word_from_Code_Buffer	F_Get_Word_from_Code_Buffer	0	0	1	0	0	0	0
F_Put_Word_to_CSM_Buffer	F_Put_Word_to_CSM_Buffer	0	0	1	0	0	0	0
F_Get_Word_from_CSM_Buffer	F_Get_Word_from_CSM_Buffer	0	0	1	0	0	0	0
F_Write_CSM_Password	F_Write_CSM_Password	0	0	1	0	0	0	0
F_Verify_CSM_Password	F_Verify_CSM_Password	0	0	1	0	0	0	0
F_Verify_Access_to_MCU	F_Verify_Access_to_MCU	0	0	0	0	0	1	1
F_Write_Word_to_RAM	F_Write_Word_to_RAM	0	0	1	0	0	0	0
F_Set_fpa_IO	F_Set_fpa_IO	0	0	1	0	0	0	0
F_Write_Debug_Register	F_Write_Debug_Register	0	0	0	0	0	1	1
F_Write_Locking_Registers	F_Write_Locking_Registers	0	0	0	0	0	1	1
F_Clear_Locked_Device	F_Clear_Locked_Device	0	0	0	0	0	1	1
F_Lock_MCU	F_Lock_MCU	0	0	0	0	0	1	1
F_ARM_Write_Byte_to_RAM	F_Write_Byte_to_RAM	0	0	0	0	0	1	1
F_ARM_Write_Word16_to_RAM	F_Write_Word16_to_RAM	0	0	0	0	0	1	1
F_ARM_Write_Word32_to_RAM	F_Write_Word32_to_RAM	0	0	0	0	0	1	1
F_ARM_Read_Byte	F_Read_Byte	0	0	0	0	0	1	0
F_ARM_Read_Word16	F_Read_Word16	0	0	0	0	0	1	0
F_ARM_Read_Word32	F_Read_Word32	0	0	0	0	0	1	0
F_ARM_Read_Bytes_Block	F_Read_Bytes_Block	0	0	0	0	0	1	0
F_ARM_Write_Bytes_Block_to_RAM	F_Write_Bytes_Block_to_RAM	0	0	0	0	0	1	1
F_Gang_Read_Byte	F_Gang_Read_Byte	0	0	0	0	0	0	1
F_Gang_Read_Word16	F_Gang_Read_Word16	0	0	0	0	0	0	1
F_Gang_Read_Word32	F_Gang_Read_Word32	0	0	0	0	0	0	1
F_Gang_Read_Bytes_Block	F_Gang_Read_Bytes_Block	0	0	0	0	0	0	1
F_ARM_Get_MCU_Name_list	F_Get_MCU_Name_list	0	0	0	0	0	1	1

Table 3.3: The following table lists Generic-FPA DLL functions, their API DLL equivalents and which FPAs support them (part 3/4).

Generic-FPA DLLFunction Name	API DLL Function Name	MSP	MSP-GP	C2000	CC	CC-GP	ARM	ARM-GP
F_Put_IEEEAddr64_to_Gang_Buffer	F_Put_IEEEAddr64_to_Gang_Buffer	0	0	0	0	1	0	0
F_Get_IEEEAddr64_from_Gang_Buffer	F_Get_IEEEAddr64_from_Gang_Buffer	0	0	0	0	1	0	0
F_Put_IEEEAddr_Byte_to_Gang_Buffer	F_Put_IEEEAddr_Byte_to_Gang_Buffer	0	0	0	0	1	0	0
F_Get_IEEEAddr_Byte_from_Gang_Buffer	F_Get_IEEEAddr_Byte_from_Gang_Buffer	0	0	0	0	1	0	0
F_ARM_Set_MCU_Family_Group	F_Set_MCU_Family_Group	0	0	0	0	0	1	1
F_Verify_Lock_Bits	F_Verify_Lock_Bits	0	0	0	1	1	0	0
F_Put_Byte_to_Gang_Buffer	F_Put_Byte_to_Gang_Buffer	0	1	0	0	1	0	1
F_Get_Byte_from_Gang_Buffer	F_Get_Byte_from_Gang_Buffer	0	1	0	0	1	0	1
F_Copy_Gang_Buffer_to_Flash	F_Copy_Gang_Buffer_to_Flash	0	1	0	0	1	0	1
F_Copy_Flash_to_Gang_Buffer	F_Copy_Flash_to_Gang_Buffer	0	1	0	0	1	0	1
F_Copy_Gang_Buffer_to_RAM	F_Copy_Gang_Buffer_to_RAM	0	1	0	0	0	0	0
F_Copy_RAM_to_Gang_Buffer	F_Copy_RAM_to_Gang_Buffer	0	1	0	0	0	0	0
F_Get_Lock_Bits	F_Get_Lock_Bits	0	0	0	0	1	0	0
F_Copy_Gang_Buffer_to_XRAM	F_Copy_Gang_Buffer_to_XRAM	0	0	0	0	1	0	0
F_Copy_XRAM_to_Gang_Buffer	F_Copy_XRAM_to_Gang_Buffer	0	0	0	0	1	0	0
F_Copy_Gang_Buffer_to_direct_RAM	F_Copy_Gang_Buffer_to_direct_RAM	0	0	0	0	1	0	0
F_Copy_direct_RAM_to_Gang_Buffer	F_Copy_direct_RAM_to_Gang_Buffer	0	0	0	0	1	0	0
F_Copy_MCU_Data_to_Buffer	F_Copy_MCU_Data_to_Buffer	0	0	0	0	1	0	0
F_Get_MCU_Data_from_Buffer	F_Get_MCU_Data_from_Buffer	0	0	0	0	1	0	0
F_GetProgressBar	F_GetProgressBar	1	1	1	1	1	1	1
F_GetLastOpCode	F_GetLastOpCode	1	1	1	1	1	1	1
F_Get_Power_Results	F_Get_Power_Results	1	1	1	1	1	1	1

Table 3.4: The following table lists Generic-FPA DLL functions, their API DLL equivalents and which FPAs support them (part 4/4).

Generic-FPA-TS DLLFunction Name	API DLL Function Name	MSP	MSP-GP	C2000	CC	CC-GP	ARM	ARM-GP
TS_DiscoverFPAs	N/A	1	1	1	1	1	1	1
TS_AssignFPAs	N/A	1	1	1	1	1	1	1
TS_OpenOneInstance	N/A	1	1	1	1	1	1	1
TS_CloseOneInstance	N/A	1	1	1	1	1	1	1
TS_CloseInstances	N/A	1	1	1	1	1	1	1
TS_Get_FPA_Type	N/A	1	1	1	1	1	1	1
TS_GenericTSDLLVer	N/A	1	1	1	1	1	1	1
TS_DLLTypeVer	F_DLLTypeVer	1	1	1	1	1	1	1
TS_Check_FPA_access	F_Check_FPA_access	1	1	1	1	1	1	1
TS_Get_FPA_SN	F_Get_FPA_SN	1	1	1	1	1	1	1
TS_Get_FPA_Label	F_Get_FPA_Label	1	1	1	1	1	1	1
TS_Initialization	F_Initialization	1	1	1	1	1	1	1
TS_Use_Config_INI	F_Use_Config_INI	1	1	1	1	1	1	1
TS_Close_All	F_Close_All	1	1	1	1	1	1	1
TS_Power_Target	F_Power_Target	1	1	0	1	1	1	1
TS_Reset_Target	F_Reset_Target	1	1	1	1	1	1	1
TS_ReportMessage	F_ReportMessage	1	1	1	1	1	1	1
TS_Report_Message	F_Report_Message	1	1	1	1	1	1	1
TS_ReadCodeFile	F_ReadCodeFile	0	0	1	0	0	1	1
TS_MSP_ReadCodeFile	F_ReadCodeFile	1	1	0	0	0	0	0
TS_CC_ReadCodeFile	F_ReadCodeFile	0	0	0	1	1	0	0
TS_AppendCodeFile	F_AppendCodeFile	0	0	0	0	0	1	1
TS_ReadPasswFile	F_ReadPasswFile	0	0	1	0	0	0	0
TS_MSP_ReadPasswFile	F_ReadPasswFile	1	1	0	0	0	0	0
TS_ConfigFileLoad	F_ConfigFileLoad	1	1	1	1	1	1	1
TS_SetConfig	F_SetConfig	1	1	1	1	1	0	0
TS_GetConfig	F_GetConfig	1	1	1	1	1	0	0
TS_Get_Config_Name_List	F_Get_Config_Name_List	1	1	1	1	1	1	1
TS_Get_Config_Value_By_Name	F_Get_Config_Value_By_Name	1	1	1	1	1	1	1
TS_Set_Config_Value_By_Name	F_Set_Config_Value_By_Name	1	1	1	1	1	1	1
TS_Put_Byte_to_Buffer	F_Put_Byte_to_Buffer	1	0	0	1	0	1	1
TS_Get_Byte_from_Buffer	F_Get_Byte_from_Buffer	1	0	0	1	0	1	0
TS_GetReportMessageChar	F_GetReportMessageChar	1	1	1	1	1	1	1
TS_Clr_Code_Buffer	F_Clr_Code_Buffer	1	1	1	1	1	1	1
TS_Put_Byte_to_Code_Buffer	F_Put_Byte_to_Code_Buffer	1	1	0	1	1	1	1
TS_Get_Byte_from_Code_Buffer	F_Get_Byte_from_Code_Buffer	1	1	0	1	1	1	1
TS_Put_IEEEAddr64_to_Buffer	F_Put_IEEEAddr64_to_Buffer	0	0	0	1	0	0	0
TS_Get_IEEEAddr64_from_Buffer	F_Get_IEEEAddr64_from_Buffer	0	0	0	1	0	0	0
TS_Put_IEEEAddr_Byte_to_Buffer	F_Put_IEEEAddr_Byte_to_Buffer	0	0	0	1	0	0	0
TS_Get_IEEEAddr_Byte_from_Buffer	F_Get_IEEEAddr_Byte_from_Buffer	0	0	0	1	0	0	0
TS_Put_Byte_to_Password_Buffer	F_Put_Byte_to_Password_Buffer	1	1	0	0	0	0	0
TS_Get_Byte_from_Password_Buffer	F_Get_Byte_from_Password_Buffer	1	1	0	0	0	0	0
TS_Get_Targets_Result	F_Get_Targets_Result	0	1	0	0	1	0	1
TS_Get_Active_Targets_Mask	F_Get_Active_Targets_Mask	0	1	0	0	1	0	1
TS_Get_DCO_constant	F_Get_DCO_constant	0	1	0	0	0	0	0
TS_Set_DCO_constant	F_Set_DCO_constant	0	1	0	0	0	0	0
TS_GP_Test_DCO_Frequency	F_Test_DCO_Frequency	0	1	0	0	0	0	0

Table 3.5: The following table lists Generic-FPA-TS DLL functions, their API DLL equivalents and which FPAs support them (part 1/4).

Generic-FPA-TS DLLFunction Name	API DLL Function Name	MSP	MSP-GP	C2000	CC	CC-GP	ARM	ARM-GP
TS_Get_DCO_Freq_result	F_Get_DCO_Freq_result	0	1	0	0	0	0	0
TS_AutoProgram	F_AutoProgram	1	1	1	1	1	1	1
TS_VerifyFuseOrPassword	F_VerifyFuseOrPassword	1	1	0	0	0	0	0
TS_Memory_Erase	F_Memory_Erase	1	1	1	1	1	1	1
TS_Memory_Blank_Check	F_Memory_Blank_Check	1	1	1	1	1	1	1
TS_Memory_Write	F_Memory_Write	1	1	1	1	1	1	1
TS_Memory_Verify	F_Memory_Verify	1	1	1	1	1	1	1
TS_Memory_Read	F_Memory_Read	0	0	1	1	0	1	1
TS_MSP_Memory_Read	F_Memory_Read	1	0	0	0	0	0	0
TS_Gang_Flash_Read	F_Gang_Flash_Read	0	1	0	0	1	0	0
TS_Memory_Write_Data	F_Memory_Write_Data	1	0	0	0	0	0	0
TS_Memory_Read_Data	F_Memory_Read_Data	1	0	0	0	0	0	0
TS_Write_IEEE_Address	F_Write_IEEE_Address	0	0	0	1	1	0	0
TS_Read_IEEE_Address	F_Read_IEEE_Address	0	0	0	1	1	0	0
TS_Write_Lock_Bits	F_Write_Lock_Bits	0	0	0	1	1	0	0
TS_Open_Target_Device	F_Open_Target_Device	1	1	1	1	1	1	1
TS_Close_Target_Device	F_Close_Target_Device	1	1	1	1	1	1	1
TS_Segment_Erase	F_Segment_Erase	1	1	1	1	1	1	1
TS_Get_Sector_Size	F_Get_Sector_Size	1	1	1	1	1	1	1
TS_Sectors_Blank_Check	F_Sectors_Blank_Check	1	1	1	1	1	1	1
TS_Copy_Buffer_to_Flash	F_Copy_Buffer_to_Flash	1	1	1	1	1	1	1
TS_Copy_Flash_to_Buffer	F_Copy_Flash_to_Buffer	1	0	1	1	0	1	0
TS_Copy_All_Flash_to_Buffer	F_Copy_All_Flash_to_Buffer	1	0	0	0	0	0	0
TS_Write_Word	F_Write_Word	1	1	0	0	0	0	0
TS_Read_Word	F_Read_Word	1	0	1	0	0	0	0
TS_Write_Byte	F_Write_Byte	1	1	0	0	0	0	0
TS_Read_Byte	F_Read_Byte	1	0	0	0	0	0	0
TS_Copy_Buffer_to_RAM	F_Copy_Buffer_to_RAM	1	1	0	0	0	0	0
TS_Copy_RAM_to_Buffer	F_Copy_RAM_to_Buffer	1	0	0	0	0	0	0
TS_Write_Byte_to_XRAM	F_Write_Byte_to_XRAM	0	0	0	1	1	0	0
TS_Read_Byte_from_XRAM	F_Read_Byte_from_XRAM	0	0	0	1	0	0	0
TS_Copy_Buffer_to_XRAM	F_Copy_Buffer_to_XRAM	0	0	0	1	1	0	0
TS_Copy_XRAM_to_Buffer	F_Copy_XRAM_to_Buffer	0	0	0	1	0	0	0
TS_Write_Byte_to_direct_RAM	F_Write_Byte_to_direct_RAM	0	0	0	1	1	0	0
TS_Read_Byte_from_direct_RAM	F_Read_Byte_from_direct_RAM	0	0	0	1	0	0	0
TS_Copy_Buffer_to_direct_RAM	F_Copy_Buffer_to_direct_RAM	0	0	0	1	1	0	0
TS_Copy_direct_RAM_to_Buffer	F_Copy_direct_RAM_to_Buffer	0	0	0	1	0	0	0
TS_Set_PC_and_RUN	F_Set_PC_and_RUN	1	1	0	0	0	1	1
TS_CC_Set_PC_and_RUN	F_Set_PC_and_RUN	0	0	0	1	1	0	0
TS_Get_MCU_Data	F_Get_MCU_Data	0	0	1	1	0	1	1
TS_Get_Targets_Vcc	F_Get_Targets_Vcc	1	1	1	1	1	1	1
TS_Get_CodeCS	F_Get_CodeCS	1	1	1	1	1	1	1
TS_Get_Device_Info	F_Get_Device_Info	1	1	1	1	1	1	1
TS_Set_MCU_Name	F_Set_MCU_Name	1	1	1	1	1	1	1
TS_Set_fpa_io_state	F_Set_fpa_io_state	1	1	0	1	1	1	1

Table 3.6: The following table lists Generic-FPA-TS DLL functions, their API DLL equivalents and which FPAs support them (part 2/4).

Generic-FPA-TS DLLFunction Name	API DLL Function Name	MSP	MSP-GP	C2000	CC	CC-GP	ARM	ARM-GP
TS_Blow_Fuse	F_Blow_Fuse	1	1	0	0	0	0	0
TS_Restore_JTAG_Security_Fuse	F_Restore_JTAG_Security_Fuse	1	0	0	0	0	0	0
TS_Capture_PC_Addr	F_Capture_PC_Addr	1	0	0	0	0	1	0
TS_GP_Capture_PC_Addr	F_Capture_PC_Addr	0	1	0	0	0	0	1
TS_Synch_CPU_JTAG	F_Synch_CPU_JTAG	1	1	0	0	0	1	1
TS_Adj_DCO_Frequency	F_Adj_DCO_Frequency	1	1	0	0	0	0	0
TS_Test_DCO_Frequency	F_Test_DCO_Frequency	1	0	0	0	0	0	0
TS_Customize	F_Customize	1	1	0	0	0	0	0
TS_init_custom_jtag	F_init_custom_jtag	1	0	0	0	0	0	0
TS_custom_jtag_stream	F_custom_jtag_stream	1	0	0	0	0	0	0
TS_Custom_Function	F_Custom_Function	1	0	0	0	0	0	0
TS_Put_Word_to_Buffer	F_Put_Word_to_Buffer	0	0	1	0	0	0	0
TS_Get_Word_from_Buffer	F_Get_Word_from_Buffer	0	0	1	0	0	0	0
TS_Put_Word_to_Code_Buffer	F_Put_Word_to_Code_Buffer	0	0	1	0	0	0	0
TS_Get_Word_from_Code_Buffer	F_Get_Word_from_Code_Buffer	0	0	1	0	0	0	0
TS_Put_Word_to_CSM_Buffer	F_Put_Word_to_CSM_Buffer	0	0	1	0	0	0	0
TS_Get_Word_from_CSM_Buffer	F_Get_Word_from_CSM_Buffer	0	0	1	0	0	0	0
TS_Write_CSM_Password	F_Write_CSM_Password	0	0	1	0	0	0	0
TS_Verify_CSM_Password	F_Verify_CSM_Password	0	0	1	0	0	0	0
TS_Verify_Access_to_MCU	F_Verify_Access_to_MCU	0	0	0	0	0	1	1
TS_Write_Word_to_RAM	F_Write_Word_to_RAM	0	0	1	0	0	0	0
TS_Set_fpa_IO	F_Set_fpa_IO	0	0	1	0	0	0	0
TS_Write_Debug_Register	F_Write_Debug_Register	0	0	0	0	0	1	1
TS_Write_Locking_Registers	F_Write_Locking_Registers	0	0	0	0	0	1	1
TS_Clear_Locked_Device	F_Clear_Locked_Device	0	0	0	0	0	1	1
TS_Lock_MCU	F_Lock_MCU	0	0	0	0	0	1	1
TS_ARM_Write_Byte_to_RAM	F_Write_Byte_to_RAM	0	0	0	0	0	1	1
TS_ARM_Write_Word16_to_RAM	F_Write_Word16_to_RAM	0	0	0	0	0	1	1
TS_ARM_Write_Word32_to_RAM	F_Write_Word32_to_RAM	0	0	0	0	0	1	1
TS_ARM_Read_Byte	F_Read_Byte	0	0	0	0	0	1	0
TS_ARM_Read_Word16	F_Read_Word16	0	0	0	0	0	1	0
TS_ARM_Read_Word32	F_Read_Word32	0	0	0	0	0	1	0
TS_ARM_Read_Bytes_Block	F_Read_Bytes_Block	0	0	0	0	0	1	0
TS_ARM_Write_Bytes_Block_to_RAM	F_Write_Bytes_Block_to_RAM	0	0	0	0	0	1	1
TS_Gang_Read_Byte	F_Gang_Read_Byte	0	0	0	0	0	0	1
TS_Gang_Read_Word16	F_Gang_Read_Word16	0	0	0	0	0	0	1
TS_Gang_Read_Word32	F_Gang_Read_Word32	0	0	0	0	0	0	1
TS_Gang_Read_Bytes_Block	F_Gang_Read_Bytes_Block	0	0	0	0	0	0	1
TS_ARM_Get_MCU_Name_list	F_Get_MCU_Name_list	0	0	0	0	0	1	1
TS_ARM_Set_MCU_Family_Group	F_Set_MCU_Family_Group	0	0	0	0	0	1	1
TS_Verify_Lock_Bits	F_Verify_Lock_Bits	0	0	0	1	1	0	0
TS_Put_Byte_to_Gang_Buffer	F_Put_Byte_to_Gang_Buffer	0	1	0	0	1	0	1
TS_Get_Byte_from_Gang_Buffer	F_Get_Byte_from_Gang_Buffer	0	1	0	0	1	0	1
TS_Copy_Gang_Buffer_to_Flash	F_Copy_Gang_Buffer_to_Flash	0	1	0	0	1	0	1
TS_Copy_Flash_to_Gang_Buffer	F_Copy_Flash_to_Gang_Buffer	0	1	0	0	1	0	1
TS_Copy_Gang_Buffer_to_RAM	F_Copy_Gang_Buffer_to_RAM	0	1	0	0	0	0	0
TS_Copy_RAM_to_Gang_Buffer	F_Copy_RAM_to_Gang_Buffer	0	1	0	0	0	0	0

Table 3.7: The following table lists Generic-FPA-TS DLL functions, their API DLL equivalents and which FPAs support them (part 3/4).

Generic-FPA-TS DLLFunction Name	API DLL Function Name	MSP	MSP-GP	C2000	CC	CC-GP	ARM	ARM-GP
TS_Put_IEEEAddr64_to_Gang_Buffer	F_Put_IEEEAddr64_to_Gang_Buffer	0	0	0	0	1	0	0
TS_Get_IEEEAddr64_from_Gang_Buffer	F_Get_IEEEAddr64_from_Gang_Buffer	0	0	0	0	1	0	0
TS_Put_IEEEAddr_Byte_to_Gang_Buffer	F_Put_IEEEAddr_Byte_to_Gang_Buffer	0	0	0	0	1	0	0
TS_Get_IEEEAddr_Byte_from_Gang_Buffer	F_Get_IEEEAddr_Byte_from_Gang_Buffer	0	0	0	0	1	0	0
TS_Get_Lock_Bits	F_Get_Lock_Bits	0	0	0	0	1	0	0
TS_Copy_Gang_Buffer_to_XRAM	F_Copy_Gang_Buffer_to_XRAM	0	0	0	0	1	0	0
TS_Copy_XRAM_to_Gang_Buffer	F_Copy_XRAM_to_Gang_Buffer	0	0	0	0	1	0	0
TS_Copy_Gang_Buffer_to_direct_RAM	F_Copy_Gang_Buffer_to_direct_RAM	0	0	0	0	1	0	0
TS_Copy_direct_RAM_to_Gang_Buffer	F_Copy_direct_RAM_to_Gang_Buffer	0	0	0	0	1	0	0
TS_Copy_MCU_Data_to_Buffer	F_Copy_MCU_Data_to_Buffer	0	0	0	0	1	0	0
TS_Get_MCU_Data_from_Buffer	F_Get_MCU_Data_from_Buffer	0	0	0	0	1	0	0
TS_GetProgressBar	F_GetProgressBar	1	1	1	1	1	1	1
TS_GetLastOpCode	F_GetLastOpCode	1	1	1	1	1	1	1
TS_Get_Power_Results	F_Get_Power_Results	1	1	1	1	1	1	1

Table 3.8: The following table lists Generic-FPA-TS DLL functions, their API DLL equivalents and which FPAs support them (part 4/4).