PSoC Programmer

Command-Line Interface (CLI) Guide

Document Number: 001-16300 Rev. *V

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
www.cypress.com

Copyrights

Contents

1. Overview

PSoC® Programmer is a flexible, integrated programming application that connects to hardware to program and configure PSoC devices. PSoC Programmer supports PSoC Creator™ and PSoC Designer™ applications as well as secondary applications such as the Bridge Control Panel and the Clock Programmer.

This document describes the command-line interface (CLI) for PSoC Programmer. Although we provide PPCOM APIs to control PSoC Programmer, you can still use CLI from a DOS shell. A legacy PERL module (*PSoC_Programmer.pm*) is included in the installation folder. Do not update it with new CLI APIs; it is recommended to migrate all your Perl CLI scripts to the Perl COM interface. PPCOM is a more powerful interface, which provides access to all PSoC Programmer APIs and is always up to date.

The Cypress engineering programmers are not recommended for mass production programming environments due to their mechanical design, plastic enclosures, and plastic headers. The Miniprog3 provides over current protection and over voltage protections, but due to the manufacturing environments the engineering programmers might be subjected to damaging conditions. It is recommended that you use mass production programmers for manufacturing. You can find the complete list of recommended vendors here.

1.1 Starting the CLI

To invoke the CLI, type the following command:

```
PPCli --command
```

To generate timing information, use the `-t` option:

```
PPCli -t --command
```

For example, to execute PPCLI script from the file, call PPCLI as shown below. It runs PPCLI and executes the "runfile" API with the "script.cli" parameter:

```
PPCLI "--runfile script.cli"
```

Perl:

```
my $pp = new PSoC_Programmer;
$pp->start(“ppcli.exe”);
```

Note that not all CLI APIs are exposed in the *PSoC_Programmer.pm* module. Only the APIs that have the Perl example are implemented in the Perl module.

1.2 Basic Usage

The PSoC Programmer CLI uses the symbol **>** as a prompt. The command output has the following syntax:

< *output* **OK**

For example, if you submit the GetPorts command to get a list of hardware ports, your screen will look as follows:

```
>GetPorts
<LPT1
LPT2
LPT3
USB/0526C018 MINIProg1/84A467CE070B
0.030303 OK
>
```

When a command fails, the command output has the following syntax:

E *output* **OK**

A failed command looks similar to this:

```
>Acquire
E
Acquire returned –3
OK
>
```

1.3 Command Syntax

The following is an example of the syntax used in this guide:

Command(IN parameter, OUT parameter)

where

IN indicates an input parameter

OUT indicates an output parameter

Input and output parameters are hexadecimal or decimal numbers, unless otherwise noted:

string = string argument

svector = vector of strings

nvector = vector of numbers

Commands are described in the Command Descriptions chapter on page 18. The descriptions include examples in CLI and Perl.

1.4 Return Values

Most PSoc Programmer CLI functions return a status that shows the function name and a string representation of the status or error. Other functions return a string or void value.

It is recommended that you analyze the return values before processing the returned parameters or moving to the next function. If a function fails, some parameters may remain in an uninitialized state. Processing these parameters may cause an exception or runtime error.

All CLI APIs return either a negative or positive integer to indicate success or failure:

- 0 or greater = Success

Example:
```
Verify
<
7.57662 0 OK
```

- Negative integer = Failed

Example:
```
E
Acquire returned –3
OK
```

1.5 Command Overview

Table 1-1 shows all PSoC Programmer CLI APIs and the basic categories to which they belong.

A number of APIs begin with a prefix that indicates their basic function. For example, APIs used for the hex file start with “HEX_”, and those that provide access to a chip’s debug access port start with “DAP_”.

Table 1-1. PSoC Programmer APIs

APIs	Chip APIs						Port APIs	Programmer APIs	Protocols	Misc
	Common	PSoC 1	PSoC 3/5	PSoC 4	FM0	PSoC 6				
Acquire(...)		X								
AcquireChip(...)		X								
AcquireChipWithDelay(...)		X								
Calibrate(...)		X								
Calibrate0(...)		X								
Calibrate1(...)		X								
Calibrate1General(...)		X								
Calibrate1M8S8(...)		X								
CheckSum(...)		X								
CheckSum1(...)		X								
ClosePort(...)							X			
DAP_Acquire(...)			X	X	X					
DAP_AcquireChip(...)			X	X	X					
DAP_GetJtagID(...)			X	X	X					
DAP_PollIO(...)			X	X	X					
DAP_PollIO1(...)			X	X	X					
DAP_ReadIO(...)			X	X	X					
DAP_ReadRaw(...)			X	X	X					
DAP_ReleaseChip(...)			X	X	X					
DAP_WriteIO(...)			X	X	X					
DAP_WriteRaw(...)			X	X	X					
Erase(...)		X								
EraseAll(...)		X								
EraseBlock(...)		X								
EraseBlock1(...)		X								
GetAcquireMode(...)								X		
GetFlashBanks()		X								
GetFlashBlocksPerBank()		X								

Table 1-1. PSoC Programmer APIs *(continued)*

APIs	Chip APIs						Port APIs	Programmer APIs	Protocols	Misc
	Common	PSoC 1	PSoC 3/5	PSoC 4	FM0	PSoC 6				
GetFlashSectorSize()		X								
GetJtagId(...)			X							
GetPorts(...)							X			
GetPower(...)								X		
GetPower1(...)								X		
GetPower2 (...)								X		
GetPowerVoltage(...)								X		
GetProgrammerCapabilities(...)								X		
GetProgrammerVersion(...)								X		
GetRowsPerArrayInFlash(...)				X	X					
HEX_ReadChecksum(...)	X									
HEX_ReadChipProtection(...)	X									
HEX_ReadConfig(...)	X									
HEX_ReadData(...)	X									
HEX_ReadEEPROM(...)	X									
HEX_ReadExtra(...)	X									
HEX_ReadFile(...)	X									
HEX_ReadImageSize(...)	X									
HEX_ReadJtagID(...)	X									
HEX_ReadNvlCustom(...)	X									
HEX_ReadNvlWo(...)	X									
HEX_ReadProtection(...)	X									
HEX_WriteChipProtection(...)	X									
HEX_WriteData(...)	X									
HEX_WriteFile(...)	X									
HEX_WriteProtection(...)	X									
HEX_WriteEEPROM(...)	X									
HEX_GetDataInRange (...)	X									
HEX_GetDataSizeIn-Range (...)	X									
I2C_DataTransfer(...)									X	
I2C_GetDeviceList(...)									X	
I2C_GetSpeed(...)									X	
I2C_ReadData(...)									X	
I2C_ResetBus()									X	
I2C_SendData(...)									X	
I2C_SetSpeed(...)									X	
IsPortOpen(...)							X			
JTAG_EnumerateDevices(...)									X	
JTAG_SetChainConfig(...)									X	
JTAG_SetIR(...)									X	

Table 1-1. PSoC Programmer APIs *(continued)*

APIs	Chip APIs						Port APIs	Programmer APIs	Protocols	Misc
	Common	PSoC 1	PSoC 3/5	PSoC 4	FM0	PSoC 6				
JTAG_ShiftDR(...)									X	
JTAG_ShiftIR(...)									X	
OpenPort(...)							X			
PSoC3_CheckSum(...)			X							
PSoC3_DebugPortConfig(...)			X							
PSoC3_EraseAll(...)			X							
PSoC3_GetEccStatus(...)			X							
PSoC3_GetTemp(...)			X							
PSoC3_ProgramRow(...)			X							
PSoC3_ProgramRowFromHex(...)			X							
PSoC3_ProtectAll(...)			X							
PSoC3_ProtectArray(...)			X							
PSoC3_ReadNvlArray(...)			X							
PSoC3_ReadProtection(...)			X							
PSoC3_ReadRow(...)			X							
PSoC3_VerifyProtect(...)			X							
PSoC3_VerifyRowFromHex(...)			X							
PSoC3_WriteNvlArray(...)			X							
PSoC3_WriteRow(...)			X							
PSoC3_GetFlashArrayInfo (…)			X							
PSoC3_GetEepromArrayInfo (…)			X							
PSoC3_EraseSector (…)			X							
PSoC3_EraseRow (…).			X							
PSoC4_CheckSum(...)				X						
PSoC4_EraseAll(...)				X						
PSoC4_GetFlashInfo(...)				X						
PSoC4_GetSiliconID(...)				X						
PSoC4_ProgramRow(...)				X						
PSoC4_ProgramRowFromHex(...)				X						
PSoC4_ProtectAll(...)				X						
PSoC4_ReadProtection(...)				X						
PSoC4_ReadRow(...)				X						
PSoC4_VerifyProtect(...)				X						
PSoC4_VerifyRowFromHex(...)				X						
PSoC4_WriteProtection(...)				X						
PSoC4_WriteRow(...)				X						
FM0_VerifyRowFromHexOneRead (...)					X					
FM0_ProgramRow (...)					X					

Table 1-1. PSoC Programmer APIs *(continued)*

APIs	Chip APIs						Port APIs	Programmer APIs	Protocols	Misc
	Common	PSoC 1	PSoC 3/5	PSoC 4	FM0	PSoC 6				
FM0_EraseAll (...)					X					
FM0_CheckSum (...)					X					
FM0_ReadRow (...)					X					
FM0_GetFlashInfo (...)					X					
FM0_ProgramRowFrom-Hex (...)					X					
FM0_VerifyRowFrom-Hex (...)					X					
FM0_GetSiliconID (...)					X					
FM0_EraseSector (...)					X					
FL_Init (...)					X					
FL_UnInit (...)					X					
FM0_FL_ProgramRow-FromHex (...)					X					
FL_ProgramPage (...)					X					
FL_ProgramSector (...)					X					
FL_LoadElf						X				
FL_ExecAPI						X				
FL_IsApiExists						X				
FL_SetRamForAlgo-rithms						X				
FL_PrepareTarget						X				
FL_ExecCmsisApiInit						X				
FL_ExecCmsisApiUnInit						X				
FL_ExecCmsisApiErase-Sector						X				
FL_ExecCmsisApiPro-gramPage						X				
FL_ExecCmsisApiVerify						X				
FL_ReleaseTarget						X				
PSoC6_WriteRow(...)						X				
PSoC6_ProgramRow (...)						X				
PSoC6_EraseAll (...)						X				
PSoC6_CheckSum (...)						X				
PSoC6_WriteProtection (...)						X				
PSoC6_ReadRow (...)						X				
PSoC6_ReadProtection (...)						X				
PSoC6_ProtectAll(...)						X				
PSoC6_GetFlashInfo(...)						X				
PSoC6_ProgramRow-FromHex(...)						X				
PSoC6_VerifyRowFrom-Hex(...)						X				
PSoC6_GetSiliconID(...)						X				
PSoC6_WriteRowFrom-Hex(...)						X				
HEX_GetRowAd-dress(...)						X				

Table 1-1. PSoC Programmer APIs *(continued)*

APIs	Chip APIs						Port APIs	Programmer APIs	Protocols	Misc
	Common	PSoC 1	PSoC 3/5	PSoC 4	FM0	PSoC 6				
HEX_GetRowsCount(...)						X				
DAP_JTAGtoSWD(...)	X									
DAP_SWDtoJTAG(...)	X									
DAP_JTAGtoDS(...)	X									
DAP_SWDtoDS(...)	X									
DAP_DStoSWD(...)	X									
DAP_DStoJTAG(...)	X									
PowerOff(...)								X		
PowerOn(...)								X		
Program(...)		X								
Protect(...)		X								
ProtectBlocks(...)		X								
ReadBlock(...)		X								
ReadBlock1(...)		X								
ReadIO(...)		X								
ReadProgram(...)		X								
ReadRam(...)		X								
ReleaseChip(...)		X								
SetAcquireMode(...)								X		
SetAutoReset(...)								X		
SetBank(...)		X								
SetChipType(...)	X									
SetLedState(...)								X		
SetPowerVoltage(...)								X		
SetProtocol(...)								X		
SetProtocolClock(...)								X		
SetProtocolConnector(...)								X		
SPI_ConfigureBus (...)									X	
SPI_DataTransfer (...)									X	
SPI_GetSupportedFreq (...)									X	
TableRead(...)		X								
TestClock(...)		X								
ToggleReset(...)									X	
UpdateProgrammer(...)								X		
Verify(...)		X								
VerifyProtect(...)		X								
WriteBlock(...)		X								
WriteBlock1(...)		X								
WriteIO(...)		X								
WriteRam(...)		X								
jtagio(...)								X		
jtagior(...)								X		
jtagiow(...)								X		
help										X
quit										X

Table 1-1. PSoC Programmer APIs *(continued)*

APIs	Chip APIs						Port APIs	Programmer APIs	Protocols	Misc
	Common	PSoC 1	PSoC 3/5	PSoC 4	FM0	PSoC 6				
runfile										X
swdior(...)								X		
swdior_raw(...)								X		
swdiow(...)								X		
swdiow_raw(...)								X		
swd_LineReset(...)								X		
SWV_ReadData(...)									X	
SWV_Setup(...)									X	

1.6 Obsolete APIs

Table 1-2 shows obsolete APIs and their current equivalent function. Although these obsolete functions are still supported, they are not recommended for use in new designs.

Table 1-2. Obsolete APIs and Replacements

Obsolete API	Replaced with
PSoC3_Acquire(...)	DAP_Acquire(...)
PSoC3_AcquireChip(...)	DAP_AcquireChip(...)
PSoC3_GetJtagID(...)	DAP_GetJtagID(...)
PSoC3_PollIO(...)	DAP_PollIO(...)
PSoC3_PollIO1(...)	DAP_PollIO1(...)
PSoC3_ReadIO(...)	DAP_ReadIO(...)
PSoC3_ReleaseChip(...)	DAP_ReleaseChip(...)
PSoC3_WriteIO(...)	DAP_WriteIO(...)
ReadHexChecksum(...)	HEX_ReadChecksum(...)
PSoC3_ReadHexConfig(...)	HEX_ReadConfig(...)
ReadHexData(...)	HEX_ReadData(...)
PSoC3_ReadHexEEPROM(...)	HEX_ReadEEPROM(...)
PSoC3_ReadHexExtra(...)	HEX_ReadExtra(...)
ReadHexFile(...)/SetHexFile(...)	HEX_ReadFile(...)
PSoC3_ReadHexImageSize(...)	HEX_ReadImageSize(...)
PSoC3_ReadHexJtagID(...)	HEX_ReadJtagID(...)
PSoC3_ReadHexNvlCustom(...)	HEX_ReadNvlCustom(...)
PSoC3_ReadHexNvlWo(...)	HEX_ReadNvlWo(...)
ReadHexProtection(...)	HEX_ReadProtection(...)
WriteHexData(...)	HEX_WriteData(...)
WriteHexFile(...)	HEX_WriteFile(...)
WriteHexProtection(...)	HEX_WriteProtection(...)
PSoC3_SetJtagChainConfig	JTAG_SetChainConfig(...)

1.7 Documentation Conventions

Table 1-3. Document Conventions for Guides

Convention	Usage
`Courier New`	Displays file locations, user-entered text, and source code: `C:\ ...cd\icc\`
Italics	Displays file names and reference documentation: Read about the *sourcefile.hex* file in the *PSoC Designer User Guide*.
[**Bracketed, Bold**]	Displays keyboard commands in procedures: [**Enter**] or [**Ctrl**] [**C**]
File > Open	Represents menu paths: **File > Open > New Project**
Bold	Displays commands, menu paths, and icon names in procedures: Click the **File** icon and then click **Open**.
Times New Roman	Displays an equation: 2 + 2 = 4
Text in gray boxes	Describes cautions or unique functionality of the product.

2. Command Descriptions

This chapter includes all commands along with descriptions and usage examples.

Acquire()

This API calls AcquireChip and then GetSiliconId.

Example:

```
>Acquire
<
0 OK
>
```

Perl:

```
$pp->Acquire()
```

AcquireChip()

This API calls AcquireChipWithDelay() using a delay from 1 to 20 ms until acquired.

Example:

```
>AcquireChip
<
0 OK
>
```

AcquireChipWithDelay(IN delay)

This API acquires the chip after *delay*/2 ms after power-on or reset. For example, if *delay* = 2, the acquire sequence starts after 2/2 ms (1 ms).

Example:

```
>AcquireChipWithDelay 0x0A
<
0 OK
>
```

Calibrate()

This API calls the Calibrate0 supervisory operation (see the Technical Reference Manual).

Example:

```
>Calibrate
<
1 OK
>
```

Perl:

```
$pp->Calibrate0();
```

Calibrate0(OUT result)

This API calls the Calibrate0 SROM operation of the PSoC 1 chip.

Example:

```
>Calibrate0
<0x00000001
1 OK
>
```

Calibrate1(OUT result)

This API calls the Calibrate1 SROM operation of the PSoC 1 chip (but not for M8S8/Indium devices).

Example:

```
>Calibrate1
<0x00000000
1 OK
>
```

Calibrate1General(OUT result)

This API calls the Calibrate1 SROM operation of any PSoC 1 chip.

Example:

```
>Calibrate1General
<0x00000000
1 OK
>
```

Calibrate1M8S8(OUT result)

This API calls the Calibrate1 SROM operation of the M8S8/Indium PSoC 1 chip.

Example:

```
>Calibrate1M8S8
<0x00000000
0 OK
>
```

CheckSum(IN blockid, OUT result)

This API calls the Checksum Supervisory operation (see the TRM) and returns the checksum of the first *blockid* blocks.

Example:

```
>CheckSum 0
<0x00006e8a
0 OK
>
```

CheckSum1(IN blockid, IN bank, OUT result)

This API calls the Checksum Supervisory operation (see the TRM); sets the bank to *bank* and returns the checksum of the first *blockid* blocks.

Example:

```
>CheckSum 128 0
<0x00001e8a
0 OK
>
```

Perl:

```
my @cs = $pp->CheckSum(128, 0);
```

ClosePort()

This API closes the port opened with OpenPort.

Example:

```
>ClosePort
<
0 OK
>
```

DAP_Acquire()

This API calls DAP_AcquireChip() and then DAP_GetJtagID(), which configures the software for the acquired device. Before calling this function, make sure that all modes are set correctly.

Example:

```
>OpenPort MiniProg3/300000000000 .
<
0 OK
>SetAcquireMode “Reset”
<
0 OK
>SetProtocol 8
<
0 OK
>SetProtocolClock 224
<
0 OK
>SetProtocolConnector 1
<
0 OK
>SetPowerVoltage 5.0
<
0 OK
>DAP_Acquire
<
0 OK
>
```

Perl:

```
$pp->OpenPort(“MiniProg3/300000000000”);
$pp->SetAcquireMode(“Reset”);
$pp->SetProtocol(8);
$pp->SetProtocolClock(224);
$pp->SetProtocolConnector(1);
$pp->SetPowerVoltage(“5.0”);
$pp->DAP_Acquire();
```

DAP_AcquireChip()

This API tries to acquire a PSoC 3 or PSoC 5 device using the SWD protocol. Before calling this function, make sure that all modes are set correctly.

Example:

```
>OpenPort MiniProg3/300000000000 .
<
0 OK
>SetAcquireMode “Reset”
<
0 OK
>SetProtocol 8
<
0 OK
>SetProtocolClock 224
<
0 OK
>SetProtocolConnector 1
<
0 OK
>SetPowerVoltage 5.0
<
0 OK
>DAP_AcquireChip
<
0 OK
>
```

DAP_GetJtagID (nvector OUT jtagID)

This API reads the JTAG ID of the acquired PSoC device and then returns a 4-byte array. This method is used only for devices that work using a debug access port (DAP). Although the function is used for PSoC 3, PSoC 4, and PSoC 5 devices, for PSoC 4 it returns the same ID - 0x0BB11477, which is the ID of ARM's CM0 core.

Example:

```
>DAP_GetJtagID
<1e 02 80 69
0 OK
>
```

DAP_PollIO (IN baseAddr, IN expectedValue, IN timeout)

This API polls the I/O register of the acquired device until it is assigned to *expectedValue* or until *timeout* elapses.

Example:

```
>DAP_PollIO 0x00000005 0x12 2000
<
0 OK
>
```

DAP_PollIO1 (IN baseAddr, IN expectedMask, IN timeout)

This API polls the I/O register of the acquired device until it is masked by *expectedMask* or *timeout* elapses.

Example:

```
>DAP_PollIO1 0x00000005 0x80 2000
<
0 OK
>
```

DAP_ReadIO (IN address, OUT data)

This API reads data located at the address. The data parameter is 4 bytes long, but actually can be 1, 2, or 4 bytes depending on the last call of PSoC3_DebugPortConfig(). By default, the chip is acquired in the 1-byte transfer mode.

Example:

```
>PSoC3_DebugPortConfig 0x04
<
0 OK
>DAP_ReadIO 0x05
<0x1ffff400
0 OK
>
```

Perl:

```
$pp->PSoC3_DebugPortConfig(0x04);
my $data = $pp->DAP_ReadIO(0x05);
```

DAP_ReleaseChip ()

This API releases the acquired chip. If PowerCycle was used to acquire the device, this function powers off the chip. If the Reset (or PowerDetect) mode was used, the chip is reset using the XRES pin (if present). The SetAutoReset() function can be used to disable reset if necessary.

Example:

```
>DAP_ReleaseChip
<
0 OK
>
```

Perl:

```
$pp->DAP_ReleaseChip();
```

DAP_ReadRaw (unsigned char IN address, OUT data, unsigned char OUT status)

This API reads 32-bit data from the DAP's register. This API works in the SWD and JTAG modes. Parameters in this API is the same as in DAP_WriteRaw() API.

Refer to the example for DAP_WriteRaw() API to see how to use DAP_ReadRaw() API.

DAP_WriteIO (IN address, IN data)

This API writes the given data to the addressed cell. The *data* parameter may be 1, 2, or 4 bytes depending on the last call of PSoC3_DebugPortConfig(). By default, the chip is acquired in 1-byte transfer mode.

Example:

```
>DAP_WriteIO 0x00050220 0xB3
<
0 OK
>
```

Perl:

```
$pp->DAP_WriteIO(0x00050220, 0xB3);
```

DAP_WriteRaw (unsigned char IN address, IN data, unsigned char OUT status)

This API writes 32-bit data in the Debug Access Port (DAP) register. This API works in SWD and JTAG modes. DAP is available in an ARM-based silicon (for example, PSoC 4, PSoC 3, and PSoC 5). The "dap_addr" parameter specifies DAP register (three LSB bits):

- bit 2 (mask 0x04) - defines APACC (1) or DPACC (0) access.
- bits 1-0 (mask 0x03) - defines register in selected access port (APACC or DPACC).

The returned parameter "status" contains a 3-bit status of previous DAP transaction. This status is different for SWD and JTAG protocols. Only three LSB bits should be considered.

Table 2-1. Status for SWD Protocol

SWD	
001	ACK
010	WAIT
100	FAULT
OTHER	NACK

Table 2-2. Status for JTAG Protocol

JTAG	
010	ACK
001	WAIT
100	FAULT
OTHER	NACK

For detail information about the status, refer to ARM's specification.

Example:

```
>JTAG_SetChainConfig 1 4 4
```

```
<
0 OK
>DAP_Acquire
<
0 OK
>DAP_WriteRaw 0x05 0x20000004
<0x02
0 OK
>DAP_WriteRaw 0x07 0x23242526
<0x02
0 OK
>DAP_ReadRaw 0x05
<0x2e000000
0x02
0 OK
>DAP_ReadRaw 0x07
<0x20000004
0x02
0 OK
>DAP_ReadRaw 0x07
<0x23242526
0x02
0 OK
>
```

Perl:

```
my @data = (4,4);
my $dt;
my $status;
$val = $pp->JTAG_SetChainConfig(1, @data);
$val = $pp->DAP_Acquire();
$val = DAP_WriteRaw(0x05, 0x20000004, $status);
$val = DAP_WriteRaw(0x07, 0x23242526, $status);
$val = DAP_ReadRaw(0x05, @dt, $status);
$val = DAP_ReadRaw(0x07, @dt, $status);
$val = DAP_ReadRaw(0x07, @dt, $status);
```

Erase()

This API calls the EraseAll SROM operation of the PSoC 1 chip. This is a duplicate of the EraseAll() operation.

Example:

```
>Erase
<
1 OK
>
```

Perl:

```
$pp->Erase();
```

EraseAll()

This API calls the Erase Supervisory operation (see TRM); erases the entire chip and protection blocks.

Example:

```
>EraseAll
<
1 OK
>
```

EraseBlock(IN blockid, OUT result)

This API calls the EraseBlock Supervisory operation. Erases *blockid* in current bank.

Example:

```
>EraseBlock 1
<
1 OK
>
```

Perl:

```
$pp->EraseBlock($block);
```

EraseBlock1(IN blockid, IN bank, OUT result)

This API calls the EraseBlock Supervisory operation. Erases *blockid* in *bank*.

Example:

```
>EraseBlock 1, 0
<
1 OK
>
```

GetAcquireMode(string OUT mode)

This API returns the current value of AcquireMode. Possible values are:

- Power
- Reset

Example:

```
>GetAcquireMode
<Reset
0 OK
>
```

GetFlashBanks()

Number of banks.

Example:

```
>GetFlashBanks
```

```
<
1 OK
>
```

GetFlashBlocksPerBank()

Number of blocks for each bank.

Example:

```
>GetFlashBlocksPerBank
<
100 OK
>
```

GetFlashSectorSize()

Number of bytes for each block.

Example:

```
>GetFlashSectorSize
<
40 OK
>
```

GetJtagID(nvector OUT jtagID)

This API returns the JTAG ID of a PSoC 3 or PSoC 5 chip and configures the software to work with this device. This function should be called after the device is acquired by the AcquireChip() or AcquireChipWithDelay() functions. (The Acquire() function automatically calls GetJtagID().)

Example:

```
>GetJtagId
<0e 02 80 69
0 OK
>
```

GetPorts(svector OUT ports)

This API returns a list of ports, one for each line.

Example:

```
>GetPorts
<LPT1
LPT2
LPT3
MINIProg1/848E4756090B
0 OK
```

Perl:

```
my @ports = $pp->GetPorts();
```

GetPower(OUT power)

This API returns the power status as read by the programmer.

Example:

```
>GetPower
<0x00000003
0 OK
>
```

GetPower1(OUT power, OUT voltage)

This API is an extension of the GetPower() function, which also returns voltage (mV) measured on the target board by the programmer. The *power* parameter is described in the GetPower() function. The *voltage* parameter is expressed in mV units and is meaningful only for programmers that are set to CAN_MEASURE_POWER (see the GetProgrammerCapabilities() function).

Example:

```
>GetPower1
<0x00000003
0x0012fcd0
0 OK
>
```

GetPower2 (OUT power, OUT voltage1, OUT voltage2)

This API returns the current power status of the programmer. The *power* parameter has the same meaning as the GetPower() function. The *voltage1* and *voltage 2* parameters are measured voltages on V_{COM} and V_{AUX} sources of TrueTouchBridge. For MiniProg3, only *voltage1* is used. For other programmers, these parameters are not applicable.

Example:

```
>GetPower2
<0x00000003
0x00001041
0x00000f3c
0 OK
>
```

Perl:

```
my @arr = $pp->GetPower2();
```

GetPowerVoltage(string OUT voltage)

This API returns the currently selected internal voltage source of the programmer. (It does not mean that power is applied to the target board.) The possible string values are: 5.0, 3.3, 2.5, and 1.8.

Example:

```
>GetPowerVoltage
<5.0
0 OK
>
```

GetProgrammerCapabilities(CProgrammerCapabilities OUT capabilities)

This API returns CProgrammerCapabilities data (vector of 6 bytes), which describes capabilities of the programmer connected to the opened port. Each byte of the array is defined as follows:

Capabilities[0] – The set of allowed acquire modes for the current programmer. This characteristic can be a union of the following flags.

CAN_RESET_ACQUIRE	0x01
CAN_POWER_CYCLE_ACQUIRE	0x02
CAN_POWER_DETECT_ACQUIRE	0x04

Capabilities[1] – The power abilities of the programmer. It is a union of the following flags.

CAN_POWER_DEVICE	0x01
CAN_READ_POWER	0x02
CAN_MEASURE_POWER	0x04
CAN_MEASURE_POWER_2	0x10

Capabilities[2] – This value is 0 if the programmer firmware cannot be updated.

Capabilities[3] – A set of PSoC families that are supported by this programmer. It is a union of the following flags.

CAN_PROGRAM_CY8C25xxx_CY8C2 6xxx	0x01
CAN_PROGRAM_ENCORE	0x02

Capabilities[4] – A set of protocols (interfaces) supported by the programmer.

JTAG	0x01
ISSP	0x02
I2C	0x04
SWD	0x08
SPI	0x10

Capabilities[5] – A set of internal voltage sources of the programmer that can be supplied to the target board.

VOLT_50V	0x01
VOLT_33V	0x02
VOLT_25V	0x04
VOLT_18V	0x08

Example:

```
>GetProgrammerCapabilities
<0x0007-0x0007-0x0001-0x0002-0x002f-0x000f
0 OK
>
```

GetProgrammerVersion(string OUT versionstring)

This API returns a string containing the programmer version.

Example:

```
>GetProgrammerVersion
<MINI Version 1.74
0 OK
>
```

Perl:

```
my $programmerversion = $pp->GetProgrammerVersion();
```

GetRowsPerArrayInFlash(string OUT rowsPerArray)

This API returns rows per array in flash of the acquired device.

Example:

```
>GetRowsPerArrayInFlash
<0x00000200
0
```

Perl:

```
my $rowsperarrayinflash = $pp->GetRowsPerArrayInFlash();
```

HEX_ReadChecksum(OUT checksum)

This API reads the checksum from the hexadecimal file.

Example:

```
>HEX_ReadChecksum
<0x3c01
0 OK
>
```

HEX_ReadChipProtection(nvector OUT data)

This API reads the chip-level protection record from the hex file.

Example:

```
>HEX_ReadChipProtection
<02
0 OK
```

```
>
```

HEX_ReadConfig(IN address, IN size, nvector OUT data)

This API reads data from the section in the active hex file with configuration data. It reads *size* bytes starting at *address*.

Example:

```
>HEX_ReadConfig 0x00 0x100
<00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
0 OK
>
```

HEX_ReadData(IN address, IN size,nvector OUT data)

This API reads *size* bytes starting at *address* and returns *size* bytes.

Example:

```
>HEX_ReadData 0x00 0x40
<02 05 06 00 8f 82 8e 83 e0 54 01 00 00 00 70 f8 22 78 ff e4 f6 d8 fd 02 05
9c bb 01 06 89 82 8a 83 e0 22 50 02 e7 22 bb fe 02 e3 22 89 82 8a 83 e4 93
22 bb 01 06 89 82 8a 83 f0 22 50 02 f7 22
0 OK
>
```

HEX_ReadEEPROM(IN address, IN size, nvector OUT data)

This API reads data in the active hex file from the section with EEPROM data. It reads *size* bytes starting at *address*.

Example:

```
>HEX_ReadEEPROM 0x00 0x00
<
0 OK
>
```

HEX_ReadExtra(nvector OUT extra)

This API returns all metadata from the hex file of a PSoC 3 or PSoC 5 device.

Example:

```
>HEX_ReadExtra
<00 01 1e 02 80 69 04 01 1e 09 55 dd
0 OK
```

```
>
```

HEX_ReadFile(string IN filename)

This API reads the hex file from *filename*.

Example:

```
>HEX_ReadFile “c:\\file.hex”
<
0 OK
>
```

HEX_ReadImageSizes(OUT flashSize, OUT configSize, OUT eepromSize, OUT nvlUserSize, OUT nvlWoSize)

This API returns corresponding sizes of direct memory blocks from the hex file of a PSoC 3 or PSoC 5 device.

Example:

```
>HEX_ReadImageSizes
<0x00040000
0x00008000
0x00000000
0x00000004
0x00000004
0 OK
>
```

HEX_ReadJtagID(nvector OUT data)

This API reads the JTAG ID from the active hex file. This function is applicable only to PSoC 3 hex files. This ID should be used to check compatibility of the acquired device and hex file.

Example:

```
>HEX_ReadJtagID
<1e 02 80 69
0 OK
>
```

HEX_ReadNvlCustom(IN address, IN size, nvector OUT data)

This API reads data from the section in the active hex file with custom NVL data. It reads *size* bytes starting at *address*.

Example:

```
>HEX_ReadNvlCustom 0x00 0x04
<00 00 00 05
0 OK
>
```

HEX_ReadNvlWo(IN address, IN size, nvector OUT data)

This API reads data from the section in the active hex file with Write-Once NVL data. It reads *size* bytes starting at *address*.

Example:

```
>HEX_ReadNvlWo 0x00 0x04
<af ac 90 bc
0 OK
>
```

HEX_ReadProtection(IN address, IN size,nvector OUT data)

This API reads *size* bytes starting at *address* from the protection record.

Example:

```
>HEX_ReadProtection 0x00 0x20
<00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
0 OK
>
```

HEX_WriteChipProtection(nvector IN data)

This API writes to the chip-level protection record in the hex file.

Example:

```
>HEX_WriteChipProtection 0x01
<
0 OK
>
```

HEX_WriteData(IN address,nvector IN data)

This API writes (sets) data starting at *address* in the hex file.

Example:

```
>HEX_WriteData 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
<
0 OK
>
```

HEX_WriteEEPROM(IN address,nvector IN data)

This API writes a block of bytes into the EEPROM area of the active hex starting from *address*. Before calling this function, you must load a hex with the HEX_ReadFile() function.

Example:

```
>HEX_ReadEEPROM 0x00 5
<00 00 00 00 00
0 OK
>HEX_WriteEEPROM 0x00 0x01 0x02 0x03 0x04 0x05
```

```
<
0 OK
>HEX_ReadEEPROM 0x00 5
<01 02 03 04 05
0 OK
>
```

HEX_WriteFile(string IN filename)

This API writes the contents of the hex file to *filename*.

Example:

```
>HEX_WriteFile "c:\\WriteFile.hex"
<
0 OK
>
```

HEX_WriteProtection(IN address,nvector IN data)

This API writes (sets) the protection data in the hex file.

Example:

```
>HEX_WriteProtection 0x00 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
<
0 OK
>
```

HEX_GetDataInRange(IN startAddr, IN endAddr, nvector OUT data, string OUT strError)

Retrieves *data* from hex file within the given address [*startAddr*, *endAddr*] range.

Example:

```
> HEX_GetDataInRange 0x10000000 0x10000200
<01 e0 fa 20 c0 01 00 2b 27 d1 b2 23 b1 21 03 22 db 00 c9 00 e5 58 63 58 66
58 1b 0f 36 0f 13 40 16 42 05 d0 36 4e 71 58 09 0f 0a 40 01 2a 01 d1 ea 07
01 d4 03 2b 3a d1 b0 23 34 4a db 00 e5 58 a1 58 e6 58 c9 04 c9 0c 00 f0 fe
fb ad 03 f1 01 ad 0b c9 0f 68 43 01 31 27 e0 01 2b 27 d1 c8 23 c0 21 03 22
db 00 c9 00 e5 58 63 58 66 58 36 0f 16 42 05 d0 22 4e 71 58 09 0f 0a 40 01
2a 01 d1 ea 07 03 d4 9b 00 9b 0f 03 2b 10 d1 c0 22 7f …
0 OK
>
```

HEX_GetDataSizeInRange(IN startAddr, IN endAddr, OUT size, string OUT strError)

This API gets the *size* of data in hex file within the given address [*startAddr*, *endAddr*] range.

Example:

```
> HEX_GetDataSizeInRange 0x10000000 0x10010000
< 0x00001e00
0 OK
>
```

I2C_DataTransfer(IN deviceAddr, IN mode, IN readLen, nvector IN dataIN, nvector OUT dataOUT)

This low-level protocol function is used as the sole communication function for all other I2C_xxx functions. This section explains how to use it for I²C transactions. The parameters are:

- *deviceAddr* – address of I²C slave device
- *mode* – a union of bits that define I²C bus protocol signals, which will be generated with the current packet

Table 2-3. Bits and Description

Bits	Description
0	Write/Read I²C operation
1	Make “Start” condition, send address byte
2	Make “ReStart” condition, send address byte
3	Make “Stop” condition after data transfer
4	Reinitialize I²C bus
5	Make I²C reconfiguration

- *readLen* – number of bytes to be read from the slave device. This parameter is used only if the *mode* parameter’s bit 0 is set to 1 (Read).
- *dataIN* – data to be sent to the I²C slave device. This parameter is used only if the *mode* parameter’s bit 0 is set to 0 (Write). For a read operation, this array is disregarded and its size can be set to 0.
- *dataOUT* – data returned as a result of the transaction. It has different meanings for Read and Write operations. The only common meaning is the first byte—the acknowledgement result of the address byte (1-ACK, 0 - NACK).
  - Read – dataOUT.size = 1 + readLen (bytes), where *readLen* is the bytes actually read from the device. They are meaningful if the address byte is ACKed.
  - Write – dataOUT.size = 1 + dataIN.size (bytes). The second part of the packet contains the ACK/NACK result for every sent byte. Normally, when the first NACK byte occurs, all others are NACKed, too.

This function can be used for find control of the I²C bus. For example, it can be used for communication with a slave device when one transaction is divided into several:

1. mode = 0x02 (Start + Write, No Stop) – bus is busy after this transaction
2. mode = 0x05 (Restart + Read, No Stop) – bus is still busy
3. mode = 0x0D (Restart + Read + Stop) – bus is released after this transaction

All other high-level I²C communication functions have the Start and Stop bits set. They are all executed at one time.

Example:

```
>I2C_DataTransfer 00 11 4
<01 00 00 00 00
0 OK
>
```

I2C_GetDeviceList(nvector OUT deviceList)

This API scans the I²C bus for connected slave devices. This function uses the current setting of bus frequency. It returns an array with addresses that ACKed their address.

Example:

```
>I2C_GetDeviceList
<05
0 OK
>
```

I2C_GetSpeed(enumI2Cspeed OUT speed)

This API reads the current speed of the I²C bus. The possible values of speed enumeration are:

- CLK_100K 0x01
- CLK_400K 0x02
- CLK_50K 0x04
- CLK_1000K 0x05

1 MHz speed is only supported by MiniProg3 and TrueTouch Bridge devices.

Example:

```
>I2C_GetSpeed
<0x00000001
0 OK
>
```

I2C_ReadData(IN deviceAddr, IN readSize, nvector OUT data)

This API reads *readSize* bytes from the given slave device.

Example:

```
>I2C_ReadData 00 4
<00 b6 01 1f
0 OK
>
```

I2C_ResetBus()

This API resets the I²C bus. This is a hardware reinitialization of the I²C bus in the event of bus hang-up.

Example:

```
>I2C_ResetBus
<
0 OK
>
```

I2C_SendData(IN deviceAddr, nvector IN data)

This API sends an array of data to the given device on the bus.

Example:

```
>I2C_SendData 00 00 00 00 00 00 01 00 01
```

```
<
0 OK
>
```

I2C_SetSpeed(enumI2Cspeed IN speed)

This API sets the speed of the I^2C bus. The possible values of speed enumeration are the same as for the I2C_GetSpeed() function.

Example:

```
>I2C_SetSpeed 1
<
0 OK
>
```

IsPortOpen(OUT isOpen)

This API returns the port status. A response of 0 means the port is closed; otherwise, the port is open.

Example:

```
>IsPortOpen
<0x00000001
0 OK
>
```

JTAG_EnumerateDevices(svector OUT devices)

This API returns the JTAG IDs of devices found on the JTAG chain. This function is implemented only for devices that support SWD or JTAG protocols.

Example:

```
>JTAG_EnumerateDevices
<0E028069
0E028069
0 OK
>
```

Perl:

```
my @val = $pp->JTAG_EnumerateDevices()
```

JTAG_SetChainConfig(IN deviceAddr, nvector IN Ir)

This function sets the parameters of the JTAG chain of which PSoC 3, PSoC 5, or PSoC 5LP is a part. It is necessary to pass the address of the target chip “deviceAddr” in the chain and size of each instruction. The following example demonstrates how to use this API.

Example:

```
>OpenPort MiniProg3/1041DD000007 .
<
0 OK
```

```
>SetAcquireMode “Reset”
<
0 OK
>SetProtocol 1
<
0 OK
>SetProtocolClock 152
<
0 OK
>SetProtocolConnector 1
<
0 OK
>SetPowerVoltage 5.0
<
0 OK
>PowerOn
<
0 OK
>JTAG_SetChainConfig 1 4 4
<
0 OK
>DAP_Acquire
<
0 OK
>
```

JTAG_SetIR (IN IR)

This API sets the instruction register on the selected device on the JTAG chain. The length of the IR and address of the selected device must be set by JTAG_SetChainConfig() API before using this API. This JTAG transaction ends in IDLE state and can start from any TAP’s state.

Example:

```
>JTAG_SetChainConfig 1 4 4
<
0 OK
>DAP_Acquire
<
0 OK
>JTAG_SetIR 0x0B
<
0 OK
```

JTAG_ShiftDR (IN drSize, nvector IN drIn, nvector OUT drOut)

This API shifts in DR “drSize” bits from “drIn” array, and at the same time “drOut” array receives “size” bits shifted out from DR. Bits in “drIn” and “drOut” arrays come in the little ending format. This transaction ends in IDLE and will start from IDLE. Therefore, it is recommended to call JTAG_SetIR() to initialize IR and move TAP to IDLE, before using JTAG_ShiftDR() API. It operates on the device selected in the last call to JTAG_SetChainConfig().

Example:

```
>JTAG_SetChainConfig 1 4 4
<
0 OK
>DAP_Acquire
<
0 OK
>JTAG_SetIR 0x0B
<
0 OK
>JTAG_ShiftDR 0x23 0x42 0x00 0x00 0x00 0x01
<32 29 21 19 01
0 OK
>JTAG_ShiftDR 2x23 0xbe 0xb1 0xa9 0xa1 0x01
<32 29 21 19 01
0 OK
```

JTAG_ShiftIR (IN irSize, nvector IN irIn, nvector OUT irOut)

This API shifts in IR “irSize” bits from “irIn” array, and at the same time “irOut” array receives “size” bits shifted out from IR. Bits in “irIn” and “irOut” arrays come in the little ending format. This transaction ends in IDLE and will also start from IDLE. Therefore, it is recommended to call JTAG_SetIR() to initialize IR and move TAP to IDLE, before using JTAG_ShiftIR() API. It operates on the device selected in the last call to JTAG_SetChainConfig().

Example:

```
>JTAG_SetChainConfig 1 4 4
<
0 OK
>DAP_Acquire
<
0 OK
>JTAG_SetIR 0x0B
<
0 OK
>JTAG_ShiftIR 0x04 0x0E                  // JTAG ID Code Instruction
<01
0 OK
>JTAG_ShiftDR 0x20 0x00 0x00 0x00 0x00   //Shift Out 32 bit JTAG ID
<77 04 a0 4b
0 OK
>
```

JTAGIO (IN read, nvector IN tdi_tms, nvector OUT tdo)

This API sends TMS (four higher bits) + TDI (four lower bits) bits to the JTAG chain. All TMS and TDI bits are sent to pass for every clock. If *read* is not 0, then *tdo* returns readout TDO bits.

Example:

```
>jtagio 1 0xF0 0xF0 0xF0 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x30
0x00
<fe ff 69 80 02 0e fe
0 OK
```

```
>
```

Perl:

```
my $val = $pp->jtagio(1, "0xF0 0xF0 0xF0 0x20 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x30 0x00");
```

JTAGIOR (IN Addr, OUT data)

This API reads content of the register of a PSoC 3 or PSoC 5 device using the JTAG interface. Note that this function works only in 1:1 configuration of the bus.

Example:

```
>jtagior 0
<0x0000007b
0 OK
>
```

Perl:

```
my $val = $pp->jtagior(0);
```

JTAGIOW (IN Addr, IN data)

This API writes *data* value to the register of the PSoC 3 or PSoC 5 device using the JTAG interface. Note that this function works only in 1:1 configuration of the bus.

Example:

```
>jtagiow 0 123
<
0 OK
>
```

Perl:

```
$pp->jtagiow(0, 123);
```

OpenPort(string IN port, string IN homedir)

This API opens the port. The port is one of the strings returned in GetPorts. *homedir* is the installation directory for PSoC Programmer. It contains the following files:

- minifirmware.hex
- cougar.vxd
- badger.vxd
- IceCubefirmware.fwm

Note: Run the OpenPort command an additional time if you receive an error the first time.

Example:

```
>OpenPort MINIProg1/068900863114 .
<
0 OK
```

```
>
```

Perl:

```
$pp->OpenPort($ports[3],$homedir);
```

PowerOff()

This API tells the programmer to stop powering the device.

Example:

```
>PowerOff
<
0 OK
>
```

Perl:

```
$pp->PowerOff();
```

PowerOn()

This API applies power to the V_{DD} pin.

Example:

```
>PowerOn
<
0 OK
>
```

Perl:

```
$pp->PowerOn();
```

Program()

This API programs the part using the current hex file.

Example:

```
>Program
<
6 OK
>
```

Protect()

This API protects the part according to the current hex file.

Example:

```
>HEX_ReadFile “c:\\PSoC1.hex”
<
0 OK
```

```
>HEX_WriteProtection 0 01 02 03 04 05
<
0 OK
>HEX_WriteFile “c:\\PSoC1.hex”
<
0 OK
>Protect
<
1 OK
>
```

ProtectBlocks(IN bank, nvector IN block, OUT result)

This API protects the part according to the current hex file with flash bank setting.

Example:

```
>SetAcquireMode “Power”
<
0 OK
>Acquire
<
0 OK
>Calibrate0
<
1 OK
>EraseAll
<0x00000001
1 OK
>ProtectBlocks 0 1 1 1
<0x0042cf34
1 OK
>
```

PSoC3_CheckSum(IN arrayID, IN startRowID, IN noOfRows, OUT checksum)

This API calculates the checksum of the *noOfRows* rows starting at *startRowID* in the array *arrayID*. To find the checksum of the entire flash (all arrays), you should calculate the checksum of each array and then total them. There is no method to calculate the checksum of all flash in the PSoC 5 silicon. For PSoC 3, which has one array only, it can be done with one API call (see the example below).

Example:

```
>PSoC3_CheckSum 0 0 256
<0x0006f165
0 OK
>
```

Perl:

```
$pp->PSoC3_CheckSum(0, 0, 256, $arrayChecksum);
```

PSoC3_DebugPortConfig(IN value)

This API sets the *value* (input parameter) of the Debug Port Configuration (DBGPRT_CFG) register of the test controller. This register allows you to adjust the auto-incGetPowerVoltagerement setting and transfer size. The transfer size options are 8, 16, and 32 bits. This register affects functionality of the DAP_WriteIO() and DAP_ReadIO() functions and any others that depend on I/O communication with PSoC 3 or PSoC 5 devices in Test Mode. Use the following values to set up the transfer size:

0 - 8 bits;

2 - 16 bits;

4 - 32 bits;

Example:

```
>PSoC3_DebugPortConfig 4
<
0 OK
>
```

Perl:

```
$pp->PSoC3_DebugPortConfig(4);
```

PSoC3_EraseAll()

This API erases all user locations in flash for all flash arrays on the chip, then erases all flash protection rows for all flash arrays on the chip. For flash arrays with ECC capability, this also erases all ECC bytes.

Example:

```
>PSoC3_EraseAll
<
0 OK
>
```

PSoC3_GetEccStatus (OUT eccStatus)

For PSoC 3 devices that support an Error Correction Code (ECC) in flash, the returned eccStatus (bit in custom NVL) determines whether the ECC status is enabled. Possible returned values are: 0 - disabled, 1 - enabled.

Example:

```
>PSoC3_GetEccStatus
<0x00000000
0 OK
>
```

Perl:

```
my @status = $pp->PSoC3_GetEccStatus();
```

PSoC3_GetTemp (IN sampleNumber, IN timerPeriod, IN clockDivider OUT tempSign, OUT tempMagnitude)

This API calls the SPC_GET_TEMP operation of the SPC block of the PSoC 3 or PSoC 5 chip. It returns the internal die temperature. Refer to the chip's TRM for usage information.

Example:

```
>PSoC3_GetTemp 0x1 0xFFF 0x4
<0x0
0x0
0 OK
>
```

Perl:

```
my @val = $pp->PSoC3_GetTemp(0x1, 0xfff, 0x4);
```

PSoC3_ProgramRow (IN arrayID, IN rowID, nvector IN data)

This API programs the addressed row of the flash array with the given data. If the flash array supports ECC and it is disabled, then *data* must contain user and config data of the row. For example, if the first 256 bytes are user data and following 32 bytes are config data, the requirements are:

- Array ID must be within flash or EEPROM range
- Row must not be write protected

Example:

```
>PSoC3_ProgramRow 0x00 0xFF 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08
<
0 OK
>
```

PSoC3_ProgramRowFromHex (IN arrayID, IN rowID, IN eccOption)

This API programs the addressed row of the flash array with the data from the active hex file. The *eccOption* parameter determines whether the function should add configuration data to the User block. The *eccOption* should be set to ‘0’ if the flash array has no ECC capability, or it is disabled. Otherwise, it must be set to ‘1’.

Example:

```
>HEX_ReadFile "c:\\PSoC3_ES3.hex"
<
0 OK
>PSoC3_ProgramRowFromHex 0x00 0xFE 0x01
<
0 OK
>
```

Perl:

```
$pp->HEX_ReadFile($file);
$pp->PSoC3_ProgramRowFromHex($arrayID, $rowID, $eccOption
```

PSoC3_ProtectAll ()

This API programs the flash protection row in each flash array using data from active hex file. Can only be executed if none of the protection bits are currently set.

Example:

```
>HEX_ReadFile “c:\\PSoC3.hex”
<
0 OK
>HEX_WriteProtection 0 01 02 03 04 05
<
0 OK
>HEX_WriteFile “c:\\PSoC3.hex”
<
0 OK
>PSoC3_ProtectAll
<
1 OK
```

PSoC3_ProtectArray (IN arrayID, nvector IN data)

This API programs the flash protection row with the given data. Size of the data array must correspond to the number of rows in the addressed array. Can only be executed if none of the protection bits are currently set.

Example:

```
>PSoC3_ProtectArray 0x00 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
<
0 OK
>
```

PSoC3_ReadNvlArray (IN arrayID, nvector OUT data)

This API reads the NVL array from the acquired chip. The *arrayID* parameter must address a valid NVL array (Custom NVL = 0x80, Write-Once = 0xF8, Factory = 0xC0).

Example:

```
>PSoC3_ReadNvlArray 0x80
<00 00 00 05
0 OK
>
```

PSoC3_ReadProtection (IN arrayID, nvector OUT data)

This API reads all protection bytes for *arrayID*. The *arrayID* must be within a valid flash range.

Example:

```
>PSoC3_ReadProtection 0x00
<ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

```
ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0 OK
>
```

PSoC3_ReadRow (IN arrayID, IN rowID, IN eccOption, nvector OUT data)

This API reads the entire flash row addressed by *arrayID* and *rowID*. The *eccOption* parameter specifies whether the user (0) or configuration (1) area of the row is read out. The row must not be in a read-protected state.

Example:

```
>PSoC3_ReadRow 0x00 0x00 0x00
<02 05 06 00 8f 82 8e 83 e0 54 01 00 00 00 70 f8 22 78 ff e4 f6 d8 fd 02 05
9c bb 01 06 89 82 8a 83 e0 22 50 02 e7 22 bb fe 02 e3 22 89 82 8a 83 e4 93
22 bb 01 06 89 82 8a 83 f0 22 50 02 f7 22 bb fe 01 f3 22 bc 00 0b be 00 29
ef 8d f0 84 ff ad f0 22 e4 cc f8 75 f0 08 ef 2f ff ee 33 fe ec 33 fc ee 9d
ec 98 40 05 fc ee 9d fe 0f d5 f0 e9 e4 ce fd 22 ed f8 f5 f0 ee 84 20 d2 1c
fe ad f0 75 f0 08 ef 2f ff ed 33 fd 40 07 98 50 06 d5 f0 f2 22 c3 98 fd 0f
d5 f0 ea 22 c5 f0 f8 a3 e0 28 f0 c5 f0 f8 e5 82 15 82 70 02 15 83 e0 38 f0
22 bb 01 0a 89 82 8a 83 f0 e5 f0 a3 f0 22 50 06 f7 09 a7 f0 19 22 bb fe 06
f3 e5 f0 09 f3 19 22 ef 2b ff ee 3a fe ed 39 fd ec 38 fc 22 e8 8f f0 a4 cc
8b f0 a4 2c fc e9 8e f0 a4 2c fc 8a f0 ed a4 2c fc ea 8e f0 a4 cd a8 f0 8b
f0 a4 2d cc 38 25
0 OK
>
```

PSoC3_VerifyProtect ()

This API verifies the protection area of all flash arrays against the active hex file.

Example:

```
>PSoC3_VerifyProtect
<
0 OK
>
```

PSoC3_VerifyRowFromHex (IN arrayID, IN rowID, IN eccOption, OUT verResult)

This API verifies the row specified by *arrayID* and *rowID* against the active hex file. The *eccOption* parameter specifies whether the user (0) or configuration (1) area of the row is read out. The *verResult* output parameter takes the value 0 if verification failed and 1 if passed.

Example:

```
>PSoC3_VerifyRowFromHex 0x00 0x00 0x00
<0x00000001
0 OK
>
```

PSoC3_WriteNvlArray (IN arrayID, nvector IN data)

This API writes the entire NVL array specified by *arrayID*. The size of the *data* parameter must be equal to the size of the array. For this operation to succeed, no flash protection bits should be set. The possible values for *arrayID* can be found in the PSoC3_ReadNvlArray() method. Note that Factory NVLs are not allowed for programming in this field; it will break the silicon irreversibly.

Example:

```
>PSoC3_WriteNvlArray 0x80 0 0 0 5
<
0 OK
>
```

Perl:

```
$pp->PSoC3_WriteNvlArray(0x80, $dataIN);
```

PSoC3_WriteRow (IN arrayID, IN rowID, nvector IN data)

This API erases the addressed row and then programs it with the given data (both user data and ECC/Config data, if supported by the flash array). In contrast to PSoC3_ProgramRow(), this function does not fail if the row is already programmed.

Requirements:

- *arrayID* must be within the flash or EEPROM range
- Row must not be externally write-protected

Example:

```
>PSoC3_WriteRow 0x00 0xFF 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08
<
0 OK
>
```

PSoC3_GetEepromArrayInfo (IN arrayID, OUT rowSize, OUT rowsPerArray)

This API retrieves the characteristics of a given EEPROM array. It returns *rowSize*, which is the size of the EEPROM row in bytes, and *rowsPerArray*, which is the number of rows in this array. Note that for the correct use of this function, the device must either be acquired using DAP_Acquire() or DAP_GetJtagID() must be called once after the device was acquired.

Example:

```
>PSoC3_GetEepromArrayInfo 0x40
<0x00000010
0x00000080
0 OK
>
```

Perl:

```
my @o;
@o = $pp->PSoC3_GetEepromArrayInfo(0x40);
```

PSoC3_GetFlashArrayInfo (IN arrayID, OUT rowSize, OUT rowsPerArray, OUT eccPresence)

This API retrieves characteristics of a given flash array. It returns *rowSize*, which is the size of the flash row in bytes; *rowsPerArray,* which is the number of rows in this array; and *eccPresence*, which tells if the array supports ECC capability (not whether ECC is enabled or disabled, which is returned by PSoC3_GetEccStatus()). The device must either be acquired using DAP_Acquire() or DAP_GetJtagID() must be called once after the device is acquired.

Example:

```
>PSoC3_GetFlashArrayInfo 0
<0x00000100
0x00000100
0x00000001
0 OK
>
```

Perl:

```
my @o;
@o = $pp->PSoC3_GetFlashArrayInfo(0x00);
```

PSoC3_EraseSector(IN arrayID, IN sectorID)

This API erases a given sector of flash or EEPROM array. A sector is a block of 64 contiguous rows that is set on a 64-row boundary. For flash arrays with ECC capability, this also erases the associated ECC bytes. This API is extremely useful when it is necessary to erase a whole EEPROM array.

Requirements:

- *ArrayID* must be within flash or EEPROM range
- Row must be not be write-protected (flash only)

Example:

```
>DAP_WriteIO 0x43AC 0x11
<
0 OK
>PSoC3_EraseSector 0x40 0x00
<
0 OK
>PSoC3_EraseSector 0x40 0x01
<
0 OK
>
```

Perl:

```
#Power-on EEPROM
$pp->DAP_WriteIO(0x43AC, 0x11);
#Erase All EEPROM - Sector 0 and 1
$pp->PSoC3_EraseSector(0x40, 0x00);
$pp->PSoC3_EraseSector(0x40, 0x01);
```

PSoC3_EraseRow(IN arrayID, IN rowID)

This API erases the given flash or EEPROM array. This API works only for PSoC 3 ES3 (Production) and PSoC 5LP.

Requirements:

- *ArrayID* must be within flash or EEPROM range
- Row must be not be write-protected (flash only)

Example:

```
>DAP_WriteIO 0x43AC 0x11
<
0 OK
>PSoC3_EraseRow 0x40 0x00
<
0
>
```

Perl:

```
#Erase EEPROM Row
$pp->DAP_WriteIO(0x43AC, 0x11); #power-on EEPROM for PSoC3
#$pp->DAP_WriteIO(0x400043AC, 0x11); #power-on EEPROM for PSoC5
$pp->PSoC3_EraseRow(0x40, 0x00);
```

PSoC4_CheckSum(IN rowID, OUT checksum)

This API calculates the checksum for the given row. If *rowID* = 0x8000, then *checksum* is calculated for the whole flash (privileged and user rows).

Example:

```
>OpenPort MiniProg3/300000000000 .
<
0 OK
>SetAcquireMode “Reset”
<
0 OK
>
>SetProtocol 8
<
0 OK
>SetProtocolClock 224
<
0 OK
```

```
>SetProtocolConnector 1
<
0 OK
>SetPowerVoltage 5.0
<
0 OK
}
>PowerOn
<
0 OK
>DAP_Acquire
<
0 OK
>PSoC4_CheckSum 1
<0x00002a28
0 OK
>
```

PSoC4_EraseAll()

This API erases all flash content, including security rows and chip-level protection.

Example:

```
>PSoC4_EraseAll
<
0 OK
>
```

PSoC4_GetFlashInfo(string OUT rowsPerFlash, OUT rowSize)

This API returns flash characteristics of the acquired device.

Note that the `rowsPerFlash` parameter returns the maximum possible number of rows for the family, but the acquired part number can have lesser flash rows. See the Ordering Information in the datasheet for the actual flash size for the part number of the interest and divide the flash size by the row size to get the actual number of rows.

Example:

```
>PSoC4_GetFlashInfo
<0x00000200
0x00000080
0 OK
>
```

PSoC4_GetSiliconID(nvector IN siliconID)

This API reads the silicon ID from the PSoC 4 device in the SWD mode. This function can be called after the device is successfully acquired by the DAP_Acquire() function. The returned vector is 4 bytes.

- siliconID[0] - high byte of PSoC silicon ID
- siliconID[1] - low byte of PSoC silicon ID
- siliconID[2] - Revision ID of the PSoC device
- siliconID[3] - Family ID of the PSoC device

Example:

```
>PSoC4_GetSiliconID
<00 02 12 90
0 OK
>
```

PSoC4_ProgramRow(IN rowID, nvector IN data)

This API programs data to the given row. The row must be unprotected and erased.

Example:

```
>PSoC4_ProgramRow 0x01 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04
<
0 OK
>
```

PSoC4_ProgramRowFromHex(IN rowID)

This API programs the given row of flash using data from the hex file.

Example:

```
>PSoC4_ProgramRowFromHex 0x00
<
0 OK
>
```

PSoC4_ProtectAll()

This API protects all flash arrays and writes chip-level protection using data from the hex file.

Example:

```
>HEX_ReadFile “c:\\PSoC4.hex”
<
0 OK
>
WriteProtection 0 01 02 03 04 05
<
0 OK
>HEX_WriteFile “c:\\PSoC4.hex”
<
0 OK
>PSoC4_ProtectAll
<
1 OK
```

```
>
```

PSoC4_ReadProtection(BYTE OUT chipProtect,nvector OUT flashProtect)

This API reads flash and chip-level protection. Chip protection must be in OPEN state.

Example:

```
>OpenPort MiniProg3/300000000000 .
<
0 OK
>SetAcquireMode “Reset”
<
0 OK
>
>SetProtocol 8
<
0 OK
>SetProtocolClock 224
<
0 OK
>SetProtocolConnector 1
<
0 OK
>SetPowerVoltage 5.0
<
0 OK
>PowerOn
<
0 OK
>DAP_Acquire
<
0 OK
>PSoC4_EraseAll
<
0 OK
>PSoC4_WriteProtection 2 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 34 5
6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
<
0 OK
>PSoC4_ReadProtection
<0x02
01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05
06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00
01 02 03 0 4 05 06 07 08 09 00 01 02 03 04
0 OK
>
```

PSoC4_ReadRow(IN rowID, nvector OUT data)

This API reads content of the given flash row. The row must be unprotected and chip-level protection must be in the OPEN state.

Example:

```
>PSoC4_ReadRow 0x01
<01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05
06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a
01 02 0304 05 06 07 08 09 0a 01 02 03 04 01 02 03 04 05 06 07 08 09 0a 01
02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06
07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01
02 03 04
0 OK
>
```

PSoC4_VerifyProtect()

This API verifies flash and chip-level protection using data from the currently active hex file.

Example:

```
>PSoC4_ProtectAll
<
0 OK
>PSoC4_VerifyProtect
<
0 OK
>
```

PSoC4_VerifyRowFromHex(IN rowID, OUT verResult)

This API verifies a row of flash using data from the hex file. Flash must be unprotected.

Example:

```
>PSoC4_VerifyRowFromHex 0x00
<0x00000001
0 OK
>
```

PSoC4_WriteProtection(BYTE IN chipProtect, nvector IN flashProtect)

This API writes flash and chip-level protection.

Example:

```
>PSoC4_WriteProtection 0x02 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04
<
0 OK
>
```

PSoC4_WriteRow(IN rowID, nvector IN data)

This API writes data to a given row of flash array. Row must be unprotected and may contain any data.

Example:

```
>PSoC4_WriteRow 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
```

```
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04
<
0 OK
>
```

FM0_VerifyRowFromHexOneRead (IN rowID, IN size, nvector IN chipData, OUT verResult, string OUT strError)

This API verifies a row of flash using data from the hex file and data, which will be read by the FM0_ReadRow API. Flash must be unprotected.

Note: This API can check data in all sectors for FM0+ devices.

Example:

```
> FM0_VerifyRowFromHexOneRead 0x00 0x256 0x01 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x08
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04
<0x00000001
0 OK

>
```

FM0_ProgramRow (IN rowID, nvector IN data, string OUT strError)

This API programs data to row. The row must be unprotected and erased.

Example:

```
>FM0_ProgramRow 0x01 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04
<
0 OK
>
```

FM0_EraseAll(OUT strError)

This API erases all flash content including security rows and chip-level protection. Refer to the FM0_EraseSector API to erase one sector.

Example:

```
>FM0_EraseAll
<
0 OK
 >
```

FM0_CheckSum (IN rowID, OUT checksum, string OUT strError)

This API calculates the checksum for rowID.

Example:

```
>OpenPort MiniProg3/300000000000 .
<
0 OK
>SetAcquireMode "Reset"
<
0 OK
>
>SetProtocol 8
<
0 OK
>SetProtocolClock 224
<
0 OK
>SetProtocolConnector 1
<
0 OK
>SetPowerVoltage 5.0
<
0 OK
}
>PowerOn
<
0 OK
>DAP_Acquire
<
0 OK
>FM0_CheckSum 1
<0x00002c32
0 OK
>
```

FM0_ReadRow (IN rowID, nvector OUT data, string OUT strError)

This API reads the content of a flash row. The row must be unprotected and chip-level protection must be in the OPEN state.

Example:

```
>FM0_ReadRow 0x01
```

```
<01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05
06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a
01 02 0304 05 06 07 08 09 0a 01 02 03 04 01 02 03 04 05 06 07 08 09 0a 01
02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06
07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01
02 03 04
0 OK
>
```

FM0_GetFlashInfo(string OUT rowsPerFlash, OUT rowSize, string strError)

This API returns flash characteristics of the acquired device.

Example:

```
 >FM0_GetFlashInfo
<0x00000200
0x00000080
0 OK
>
```

FM0_ProgramRowFromHex (IN rowID, string OUT strError)

This API programs row of flash using data from the hex file. The row must be unprotected and erased.

Example:

```
>FM0_ProgramRowFromHex 0x00
<
0 OK
>
```

FM0_VerifyRowFromHex (IN rowID, OUT verResult, string OUT strError)

This API verifies a row of flash using data from the hex file. Flash must be unprotected.

Example:

```
>FM0_VerifyRowFromHex 0x00
<0x00000001
0 OK
>
```

FM0_GetSiliconID (nvector IN siliconID, string OUT strError)

This API reads the silicon ID from a FM0+ device in SWD mode. This function can be called after the device is successfully acquired by the DAP_Acquire() functions.

Example:

```
>FM0_GetSiliconID
< E1 20 30 C3
0 OK
>
FM0_EraseSector (IN rowID, string OUT strError)
This API erased full sector, which contain current row id. Flash must be
unprotected.
Example:
>FM0_ EraseSector 0x00
```

```
<
0 OK
>
```

FL_Init(nvector IN data, string OUT strError)

This API loads the flash loader in RAM. When the load is complete, execute the configure CPU and run Init functions from flash loader.

Note: Use the correct flash loader to correct devices with correct flash memory. For more information, refer to the datasheet with the specific chip name. Use the FL_UnInit API after this API. You can use the FL_ProgramPage, FL_ProgramSector, and FM0_FL_ProgramRowFromHex APIs between the FL_Init and FL_UnInit APIs.

Example:

```
>FM0_ Init 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01
02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06
07 08 09 0a 01 02 0304 05 06 07 08 09 0a 01 02 03 04 01 02 03 04 05 06 07
08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02
03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07
08 09 0a 01 02 03 04
<
0 OK
>
```

FL_UnInit(string OUT strError)

This API executes the UnInit function from RAM and returns the MCU in normal state.

Note: Use the FL_Init API before this API. You can use the FL_ProgramPage, FL_ProgramSector, and FM0_FL_ProgramRowFromHex APIs between the FL_Init and FL_UnInit APIs.

Example:

```
>FM0_ UnInit
<
0 OK
>
```

FM0_FL_ProgramRowFromHex(IN rowId, IN rowSize, OUT m_sLastError)

This API programs rows of flash using data from the hex file. The row must be unprotected and erased.

Note: You can use the FM0_FL_ProgramRowFromHex API between FL_Init and FL_UnInit APIs.

Example:

```
> FM0_FL_ProgramRowFromHex 0x00 0x256
 <
0 OK
 >
```

FL_ProgramSector (IN address, nvector IN data, out m_sLastError);

This API programs data in sector use flash loader functions. The sector must be unprotected and erased.

Note: Use correct flash loader to correct devices with correct flash memory. For more information, refer to the datasheet with the specific chip name. Use the FL_UnInit API after this API. You can use the FL_ProgramPage and FL_ProgramSector APIs between FL_Init and FL_UnInit APIs.

Example:

```
>FM0_ ProgramSector 00 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08
09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03
04 05 06 07 08 09 0a 01 02 0304 05 06 07 08 09 0a 01 02 03 04 01 02 03 04
05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09
0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04
05 06 07 08 09 0a 01 02 03 04
<
0 OK
>
```

FL_ProgramPage(IN rowID, nvector IN data, out m_sLastError);

This API programs data in row use flash loader functions. The row must be unprotected and erased.

Note: Use correct flash loader to correct devices with correct flash memory. For more information, refer to the datasheet with the specific chip name. Use the FL_UnInit API after this API. You can use the FL_ProgramPage API between the FL_Init and FL_UnInit APIs.

Example:

```
>FM0_ ProgramPage 00 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09
0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04
05 06 07 08 09 0a 01 02 0304 05 06 07 08 09 0a 01 02 03 04 01 02 03 04 05
06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a
01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05
06 07 08 09 0a 01 02 03 04
<
0 OK
>
```

FL_LoadElf(string IN algoPath, string OUT strError)

Loads and parses algorithms from ELF (*.FLM) file. Only 32 bit and little endian ELF files are supported. *algoPath* represents path to the ELF file.

Example:

```
>FL_LoadElf "C:\\Program Files (x86)\\Cypress\\Programmer\\3rd_Party_Con-
figuration_Files\\CY8C6xxx\\Prog_Algorithm\\CY8C6xx7.FLM"
<
0 OK
>
```

FL_IsApiExists(string IN apiName, bool OUT exists, string OUT strError)

Checks whether *apiName* exists in the ELF file being loaded. It can be used to check optional(implementation is not mandatory) flashloader functions (e.g. BlankCheck, EraseChip, Verify). Returns *exists*=true in case of existence, otherwise *exists*=false.

Example:

```
>FL_IsApiExists Verify
<0x1
```

```
0 OK
>
```

FL_ExecAPI(string IN apiName, IN r0, IN r1, IN r2, IN r3, IN r4, IN r5, IN r6, IN r7, IN dataBufferAddr, IN timeout, nvector IN dataBuffer, OUT apiResult, string OUT strError)

Executes *apiName* from the loaded ELF file within provided parameters. These parameters are:

- *r0 - r7* – R0-R7 ARM general purpose registers
- *dataBufferAddr* – address of data to be written
- *timeout* – time to wait for API complete until termination
- *dataBuffer* – data to be written
- *apiResult* – API execution result

Example:

```
>FL_ExecAPI EraseSector 0x10000000 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
5000 00
<
0 OK
>
```

FL_SetRamForAlgorithms(IN baseAddr, IN maxSize, string OUT strError)

Sets RAM area for algorithms usage to a *baseAddr* with a size of *maxSize*.

Example:

```
>FL_SetRamForAlgorithms 0x08002400 0x8000
<
0 OK
>
```

FL_PrepareTarget(bool IN cacheRAM, OUT strError)

Loads programming algorithm into the RAM area and prepares CPU for algorithm usage. *cacheRAM* parameter is reserved, not implemented yet.

Note: before this API usage MCU should be configured by using the following APIs: FL_LoadElf, FL_SetRamForAlgorithms.

Example:

```
>FL_PrepareTarget 0
<
0 OK
>
```

FL_ExecCmsisApiInit(IN adr, IN clk, IN fnc, IN timeout, OUT apiResult, string OUT strError)

Initializes the microcontroller for Flash programming. *adr* is the device's base address, *clk* – clock frequency used for device programming. Each `FL_ExecCmsisApiInit` has to end up with `FL_ExecCmsisApiUnInit` once the work is done.

fnc should be one of the following:

- Erase – 1

- Program – 2
- Verify – 3

Note: Before Flashloader API execution MCU should be configured by using the following APIs: `FL_LoadElf, FL_SetRamForAlgorithms, FL_PrepareTarget.`

Example:

```
>FL_ExecCmsisApiInit 0x10000000 0 1 5000
< 0x00000000
0 OK
>
```

FL_ExecCmsisApiUnInit(IN fnc, IN timeout, out apiResult, string OUT strError)

De-initializes the MCU configured by `FL_ExecCmsisApiInit` and is invoked at the end of an erasing, programming, or verification step which should be specified by *fnc* parameter. Returns *apiResult*=0 on success, *apiResult*=1 on failure.

Example:

```
>FL_ExecCmsisApiUnInit 1 5000
< 0x00000000
0 OK
>
```

FL_ExecCmsisApiEraseSector(IN adr, IN timeout, OUT apiResult, string OUT strError)

Deletes the content of the sector starting at the address specified by the *adr* parameter. Returns *apiResult*=0 if erasing succeeds, otherwise erasing fails.

Note: First, the MCU must be prepared for flash algorithm execution by using: `FL_LoadElf, FL_SetRamForAlgorithms, FL_PrepareTarget, FL_ExecCmsisApiInit.`

Example:

```
>FL_ExecCmsisApiEraseSector 0x10000000 5000
< 0x00000000
0 OK
>
```

FL_ExecCmsisApiProgramPage(IN adr, IN sz, IN timeout, nvector IN dataBuffer, OUT apiResult, string OUT strError)

Writes code into the Flash memory. Code that being written - *dataBuffer* at the specified address=*adr* with the size=*sz*. Returns: status information: 0 on success, 1 on failure.

Note: First, the MCU must be prepared for flash algorithm execution by using: `FL_LoadElf, FL_SetRamForAlgorithms, FL_PrepareTarget, FL_ExecCmsisApiInit.`

Example:

```
>FL_ExecCmsisApiProgramPage 0x10000000 0x200 5000 00 01 02 03 04 05 06 07
08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02
03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07
08 09 00 01 02 03 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08
09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03
```

```
04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 00 01 02 03 04
05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04
05 06 07 08 09 00 01 02 03 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05
06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00
01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 00 01
02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06
07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01
02 03 04 05 06 07 08 09 00 01 02 03 00 01 02 03 04 05 06 07 08 09 00 01 02
03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07
08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02
03 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03
04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08
09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04
05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09
00 01 02 03
< 0x00000000
0 OK
>
```

FL_ExecCmsisApiVerify(IN adr, IN sz, IN timeout, nvector IN dataBuffer, OUT apiResult, string OUT strError)

Compares the content of the Flash memory at *adr* with *sz* bytes of *dataBuffer*. Returns *apiResult=adr+sz* - on success and any other number - on failure which represents the failing address.

Note: First, the MCU must be prepared for flash algorithm execution by using: `FL_LoadElf, FL_SetRamForAlgorithms, FL_PrepareTarget, FL_ExecCmsisApiInit.`

Example:

```
>FL_ExecCmsisApiVerify 0x10000000 0x200 5000 00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04
05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09
00 01 02 03 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00
01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05
06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 00 01 02 03 04 05 06
07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01
02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06
07 08 09 00 01 02 03 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07
08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02
03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 00 01 02 03
04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08
09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03
04 05 06 07 08 09 00 01 02 03 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04
05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 00
01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05
06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00
01 02 03 04 05 06 07 08 09 00 01 02 03 00 01 02 03 04 05 06 07 08 09 00 01
02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06
07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01
02 03
```

```
< 0x10000200
0 OK
>
```

FL_ReleaseTarget(bool IN restoreRAM, string OUT strError)

Releases CPU after flash algorithms execution. Uses as the final step in flashloader APIs sequence. *restoreRAM* parameter is reserved, not implemented yet.

Example:

```
>FL_ReleaseTarget 0
<
0 OK
>
```

PSoC6_WriteRow(IN rowID, nvector IN data)

Writes data to a given row of flash array. The row must be unprotected and may contain any data.

Example:

```
>PSoC6_WriteRow 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04
<
0 OK
```

PSoC6_ProgramRow (IN rowID, nvector IN data, string OUT strError)

Programs data to the given row. The row must be unprotected and erased.

Example:

```
>PSoC6_ProgramRow 0x01 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04
<
0 OK
>
```

PSoC6_EraseAll()

Erases all flash content, including security rows and chip-level protection.

Example:

```
>PSoC6_EraseAll
```

```
<
0
>
```

PSoC6_CheckSum(IN rowID, OUT checksum)

Calculates the checksum for the given row. If rowID = 0xFFFFFFFF then checksum is calculated for the whole flash (privileged and user rows).

Example:

```
>OpenPort MiniProg3/300000000000 .
<
0 OK
>SetAcquireMode "Reset"
<
0 OK
>
>SetProtocol 8
<
0 OK
>SetProtocolClock 224
<
0 OK
>SetProtocolConnector 1
<
0 OK
>SetP<
owerVoltage 5.0
0 OK

}
>PowerOn
<
0 OK
>DAP_Acquire
<
0 OK
>PSoC6_CheckSum 1
<0x00002a28
0 OK
>
```

PSoC6_WriteProtection(BYTE IN lifeCycle, nvector IN secureRestrict, nvector IN deadRestrict, BOOL IN voltageVerification)

This API has Write Chip Protection, Secure, and Dead Access Restriction properties. For programming eFuse, you need 2.5 V. If voltageVerification is true(1), voltage verification will be executed; if false (0), it is ignored.

Each byte of *secureRestrict* and *deadRestrict* arrays maps to a single eFuse bit.

- 0x01 - blow eFuse
- 0x00 - do not blow eFuse
- 0xFF - ignore

The lifeCycle stages are controlled using four bits:

- b'0001 - NORMAL stage
- b'001x - SECURE WITH DEBUG stage
- b'01xx - SECURE stage
- b'1xxx - RMA stage

Use bit shifting to set the correct lifecycle stage:

lifecycle = (0x01 << 0) ---> NORMAL

lifecycle = (0x01 << 1) ----> SECURE WITH DEBUG

lifecycle = (0x01 << 2) ---> SECURE

lifecycle = (0x01 << 3) ---> RMA

Note that transitioning to RMA stage is not supported.

Example:

```
>PSoC6_WriteProtection 01 [0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF] [0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF] 01
<
0 OK
>
```

PSoC6_ReadRow(IN rowID, nvector OUT data)

Reads content of the given flash row. The row must be unprotected and chip-level protection must be in the OPEN state.

Example:

```
>PSoC6_ReadRow 0x01
<01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05
06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a
01 02 0304 05 06 07 08 09 0a 01 02 03 04 01 02 03 04 05 06 07 08 09 0a 01
02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06
07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01 02 03 04 05 06 07 08 09 0a 01
02 03 04
0>
```

PSoC6_ReadProtection(BYTE OUT chipProtect)

Reads chip-level protection.

Example:

```
>OpenPort MiniProg3/300000000000 .
<
0 OK
>SetAcquireMode "Reset"
<
0 OK
>
>SetProtocol 8
<
0 OK
>SetProtocolClock 224
<
0 OK
>SetProtocolConnector 1
<
0 OK
>SetPowerVoltage 5.0
<
0 OK
>PowerOn
<
0 OK
>DAP_Acquire
<
0 OK
>PSoC6_EraseAll
<
0 OK
>PSoC6_WriteProtection 2 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 34 5
6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
<
0 OK
>PSoC6_ReadProtection
<0x00
0 OK
```

```
>
```

PSoC6_ProtectAll(BOOL IN voltageVerification)

Writes chip-level protection using data from hex file. EFuse should be programmed with 2.5 V. If voltageVerification is set to true, voltage check will be done. Use 1 as true, 0 as false.

Example:

```
>PSoC6_ProtectAll 1
<
1 OK
>
```

PSoC6_GetFlashInfo(string OUT rowsPerFlash, OUT rowSize)

Returns flash characteristics of the acquired device.

Example:

```
>PSoC6_GetFlashInfo
<0x00000800
0x00000200
0 OK
>
```

PSoC6_ProgramRowFromHex(IN hexRowID)

Programs data from the row id in hex file into flash array. The flash must be unprotected and erased.

Example:

```
>PSoC6_ProgramRowFromHex 0x00
<
0 OK

>
```

PSoC6_VerifyRowFromHex(IN hexRowID, OUT verResult)

Verifies a row of flash using data from the hex file. Flash must be unprotected.

Example:

```
>PSoC6_VerifyRowFromHex 0x00
<0x00000001
0 OK
>
```

PSoC6_GetSiliconID(nvector IN siliconID, IN familyIdHi, IN familyIdLo, IN revisionIdMaj, IN revisionIdMin, IN siliconIdHi, IN siliconIdLo, IN sromFmVersionMaj, IN sromFmVersionMin, IN protectState, string OUT strError)

Reads the silicon ID from the PSoC 6 device in SWD mode. This function can be called after the device is successfully acquired by the DAP_Acquire() function. The returned vector is 4 bytes.

- siliconID[0] - high byte of PSoC silicon ID
- siliconID[1] - low byte of PSoC silicon ID
- siliconID[2] - Revision ID of the PSoC device
- siliconID[3] - Family ID of the PSoC device

Example:

```
>PSoC6_GetSiliconID
<E2 00 00 00
0x00000001
0x00000000
0x00000000
0x00000000
0x000000E2
0x00000000
0x00000000
0x00000001
0x00000002
0 OK
>
```

PSoC6_WriteRowFromHex (IN hexRowID)

Writes data from row id in the hex file into flash array. The row must be unprotected and may contain any data.

Example:

```
> PSoC6_WriteRowFromHex 0x00
<0 OK
>
```

PSoC6_EraseRow(int IN rowAddr, string OUT strError)

Erases row at given address.

Example:

```
>PSoC6_EraseRow 0x10000000
<
0 OK
>
```

HEX_GetRowAddress (IN hexRowID, OUT rowAddr)

Gets the address in flash for the current row in the hex file.

Example:

```
> HEX_GetRowAddress 0x00
<0x10001000
0 OK
>
```

HEX_GetRowsCount (OUT rowsPerHex)

Gets the count of rows in hex file.

Example:

```
> HEX_GetRowsCount
<20
0 OK
>
```

DAP_JTAGtoSWD()

Switches DAP with JTAG to SWD.

Example:

```
>DAP_JTAGtoSWD
<
0
>
```

DAP_SWDtoJTAG()

Switches DAP with SWD to JTAG.

Example:

```
>DAP_SWDtoJTAG
<
0
>
```

DAP_JTAGtoDS()

Switches DAP with JTAG to DS.

Example:

```
>DAP_JTAGtoDS
<
0
>
```

DAP_SWDtoDS()

Switches DAP with SWD to DS.

Example:

```
>DAP_SWDtoDS
<
0
>
```

DAP_DStoSWD()

Switches DAP with DS to SWD.

Example:

```
>DAP_DStoSWD
<
0
>
```

DAP_DStoJTAG()

Switches DAP with DS to JTAG.

Example:

```
>DAP_DStoJTAG
<
0
>
```

ReadBlock(IN blockid,nvector OUT block, OUT result)

Reads *blockid* and returns 64 bytes of data.

Example:

```
>ReadBlock 1
<30 30 30 ... 30
0x00000001
8 OK
>
```

Perl:

```
@block = $pp->$ReadBlock($blockid);
```

ReadBlock1(IN bank, IN blockid,nvector OUT block, OUT result)

Reads *blockid* from *bank* and returns 64 bytes of data.

Example:

```
>ReadBlock1 0x00 0x01
<30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 7d 02 6c 30 7d 09 e4 30 30 30 30 30 30 30 30 30 30
30 30 7d 00 92 30 7d 01 f4 30 7d 04 2b 30
0x00000000
8 OK
>
```

ReadIO(IN addr, IN size, nvector OUT data)

Reads *size* bytes from the I/O registers of the PSoC device, starting at the *addr* address. The device must have previously been acquired using the Acquire function.

Example:

```
>ReadIO 0 10
<00 00 00 ff 00 00 00 ff 00 00
1 OK
```

```
>
```

ReadProgram(IN start, IN number, nvector OUT data, nvector OUT protData)

Returns *data* from *number* blocks, starting at *start* block. *protData* is protection data for the acquired device. If the block is protected, the high byte will be nonzero. Data with a nonzero high byte is not valid because data can be moved from the previous block to the current block.

Example:

```
>ReadProgram 0 4

<80 67 30 30 30 30 30 30 7e 30 30 30 7d 06 cd 7e 30 30 30 30 30 30 30 7e
 30 30 30 30 30 30 30 7e 30 30 30 7e 30 30 30 7d 03 d3 7e 7e 30 30 30 30 30
 30 30 30 30 30 30 30 30 30 30 30 30 30 7e 30 30 30 7e 30 30 30 7e 30 30
 30 7e 30 30 30 7e 30 30 30 7e 30 30 30 7e 30 30 30 7e 30 30 30
 7e 30 30 30 71 10 62 fa 00 62 e3 07 70 ef 50 20 28 41 fe fb 50 80 4e 55 f8
 00 55 f9 17e 130 130 130 17e 130 130 130 17e 130 130 130 17e 130 130 130
17e 130 130 130 17e 130 130 130 17e 130 130 130 17e 130 130 130 17e 130 130
130 17e 130 130 130 171 110 162 1fa 100 162 1e3 107 170 1ef 150 120 128 141
1fe 1fb 150 180 14e 155 1f8 100 155 1f9 17e 130 130 130 17e 130 130 130 17e
130 130 130 17e 130 130 130 17e 130 130 130 17e 130 130 130 17e 130 130 130
17e 130 130 130 17e 130 130 130 17e 130 130 130 171 110 162 1fa 100 162 1e3
107 170 1ef 150 120 128 141 1fe 1fb 150 180 14e 155 1f8 100 155 1f9

f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00

8 OK
>
```

Perl:

```
my @u = $pp->ReadProgram(0,2);
my @protection = @{$u[1]};
my @data = @{$u[0]};
```

Here the first four blocks are protected.

ReadRAM(IN addr, IN size, nvector OUT data)

Reads *size* bytes starting at *address* from the SRAM of the PSoC device. To get correct data from PSoC, you need to acquire device of the same name functions. Note that this function operates on the current SRAM bank.

Example:

```
>ReadRam 0x00 0x20
<00 05 00 00 69 02 82 00 00 00 00 16 ad bd 16 b9 14 db 01 dd 0d 14 00 00 00
00 00 00 00 00 00 00
0 OK
>
```

ReleaseChip()

Powers the chip off, leaves the PSoC in Test mode. If the chip was acquired in Reset mode, it is reset using XRES pin. SetAutoReset() function can be used to disable reset if necessary.

Example:

```
>ReleaseChip
```

```
<
0 OK
<
```

Perl:

```
$pp->ReleaseChip();
```

runfile(IN fNamePath)

Executes the indicated file with a list of PPCLi commands. *fNamePath* is the path to the PPCLi script. See the script example below:

```
GetPorts
OpenPort MiniProg3dP/868C72CF030D .
SetAcquireMode “Power”
Acquire
SetProtocol 2
SetPowerVoltage 5.0
HEX_ReadFile "C:\\I2C_USB_StoreData.hex"
Program
ReleaseChip
ClosePort
```

Example:

```
>runfile "c:\\Script.txt"
>GetPorts
<MiniProg3dP/868C72CF030D
0 OK
>OpenPort MiniProg3dP/868C72CF030D .
<
0 OK
>SetAcquireMode “Power”
<
0 OK
>Acquire
<
0 OK
>SetProtocol 2
<
0 OK
>SetPowerVoltage 5.0
<
0 OK
>HEX_ReadFile "C:\\test.hex"
<
0 OK
>Program
<
6 OK
>ReleaseChip
<
1 OK
>ClosePort
<
```

```
0 OK
>
```

Perl:

```
$pp->runfile("C://Script.txt");
```

SetAcquireMode(string IN mode)

Sets the acquire model: Power, Reset, or PowerDetect.

Example:

```
>SetAcquireMode Power
<
0 OK
>
```

SetAutoReset(IN mode)

Defines whether the chip will be reset during the release operation by ReleaseChip() or DAP_ReleaseChip() functions. This option is only applicable if the chip was acquired in Reset mode. The *mode* parameter can be one of: 0x00 – autoreset disabled, 0x01 – autoreset enabled.

Example:

```
>SetAutoReset 0x01
<
0 OK
>
```

Perl:

```
$pp->SetAutoReset(0x01);
```

SetBank(IN bank)

Sets the bank for the next operation.

Example:

```
>SetBank 0x01
<
1 OK
>
```

SetChipType(IN familyID)

This function is used before device acquisition. It is not necessary in most cases. However, to acquire devices from certain families with the ICE programmer (CY7C602xx, CY7C633xx, CY7C638xx, and CY7C639xx), you must set this parameter to 4. Set it to 0x00-0x03 for any other PSoC 1 family. For other programmers (not ICE-cube), this API can be ignored.

Example:

```
>SetChipType 0x04
<
0 OK
>
```

Perl:

```
$pp->SetChipType($familyID); // enCoRe
```

SetLedState(IN ledNo, IN ledState)

Sets the programmer's onboard LED in the given state. Currently this function is supported only by MiniProg3 device, which has four LEDs that can be in one of the four states. Also, if l*edNo* = –1, then the identify feature of MiniProg3 is accessible; *ledState* defines mode (0 – Normal, 1 – Chaser, 2 – All On, 3 – All Off).

Example:

```
>SetLedState -1 1
<
0 OK
>
```

SetPowerVoltage(string IN voltage)

Sets the voltage applied when PowerOn is called or when acquiring with power cycle.

Example:

```
>SetPowerVoltage 5.0
<
0 OK
>
```

Perl:

```
$pp->SetPowerVoltage($voltage);
```

SetProtocol(enumInterfaces IN protocol)

Activates a given protocol of the programmer. The possible protocols are described here. The MiniProg3 can be configured for TX8 (UART) mode data receiving. SWD_SWV mode must be set and the SWV_Setup() API must be used for configuration.

```
public enum enumInterfaces
{
JTAG = 1,
ISSP = 2,
I2C = 4,
SWD = 8,
SPI = 16,
SWD_SWV = 32
}
```

Example:

```
>SetProtocol 8
<
0 OK
>
```

SetProtocolClock(enumFrequencies IN clock)

Sets bus frequency for the active protocol. Currently, this function is applicable only for the MiniProg3 device in SWD/JTAG mode. The possible frequency values are:

```
public enum enumFrequencies
{
      FREQ_48_0 = 0,
      FREQ_24_0 = 4,
      FREQ_16_0 = 16,
      FREQ_03_2 = 24,
      FREQ_06_0 = 96,
      FREQ_12_0 = 132,
      FREQ_08_0 = 144,
      FREQ_01_6 = 152,
      FREQ_01_5 = 192,
      FREQ_03_0 = 224,
      FREQ_RESET = 252,
}
```

Example:

```
>SetProtocolClock 224
<
0 OK
>
```

SetProtocolConnector(IN connector)

Activates *connector* of the programmer. This function is applicable to the MiniProg3 device in SWD mode and for TrueTouch Bridge in I^2C or SWD mode. The possible values for MiniProg3 in SWD mode are:

- 0 – 5-pin connector
- 1 – 10-pin connector

In I^2C mode for TrueTouch Bridge, this function selects a pair of I^2C pins on the bridge header. The possible values are:

- 0 – original I^2C - Pins 6, 8
- 1 – ISSP: Clk, Data - Pins 9, 7
- 2 – SPI: Clk, Data - Pins 13,15

In SWD mode for TrueTouch Bridge, this function selects a trio of SWD pins on the bridge header. The function is intended for the programming of the CY8CTMA12xx/CY8CTMA17xx devices in SWD mode. The possible values are:

- 1 – original SWD – Pins 7, 8, 5: SWDCK/ISSP_CLK, SWDIO/ISSP_DATA, SWD_XRES/ISSP_X-RES;
- 2 – SWD1 – Pins 8, 6, 14: SWDCK1/SCL_nSS, SWDIO1/SDA_MISO, SWD_XRES1/TX_Host

Example:

```
>SetProtocolConnector 1
<
0 OK
```

>

SPI_ConfigureBus (IN bitOrder, IN mode, IN freq, nvector extra)

Sets the configuration of the SPI master. This function should be called after SPI mode is set by SetProtocol(). The possible values of *bitOrder* and *mode* enumerations appear below. The *frequency* parameter (Hz) must be one of the frequencies returned by SPI_GetSupportedFreq(). The *extra* parameter is an array of bytes. The zero element of this array is used to set the EzI2C SPI interface. The possible values are: Gen3 - 0 and Gen4 - 1. If the *extra* parameter is empty, the Gen3 value is used.

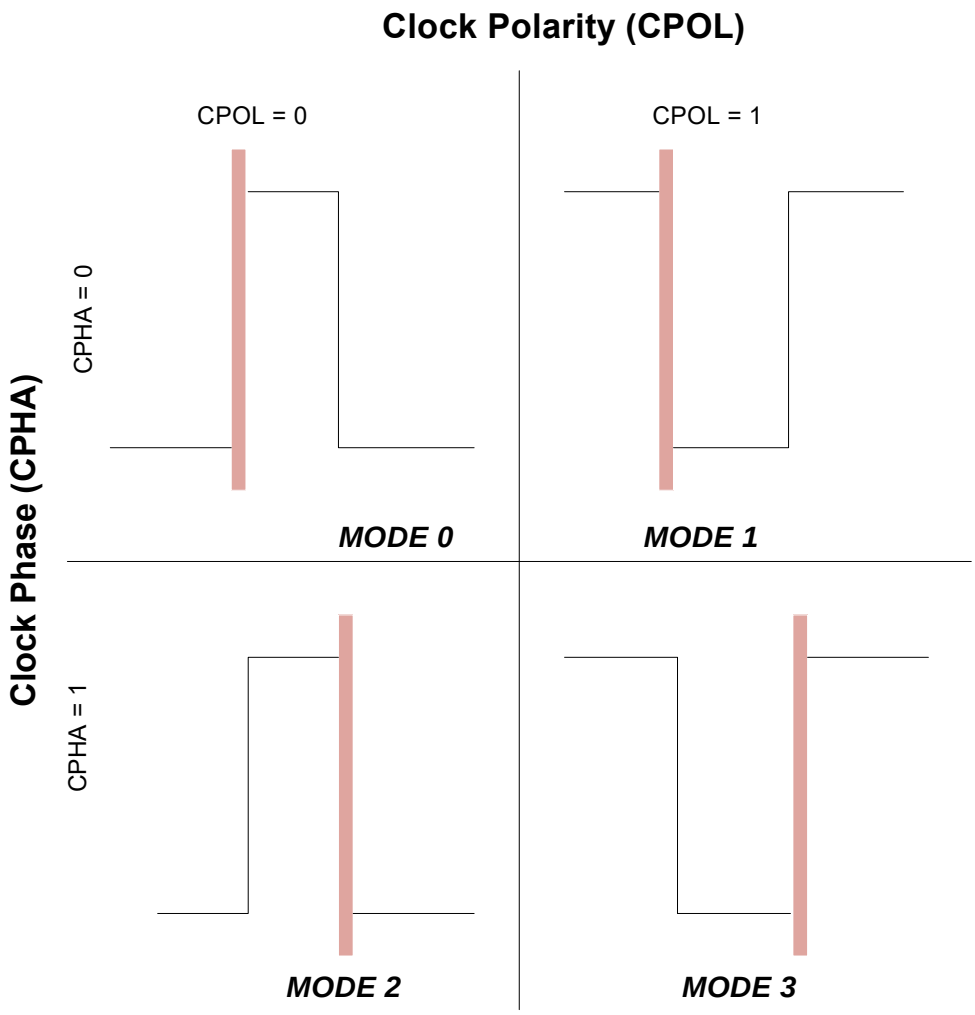
For proper communication with the slave SPI, mode must be configured. The master must use the same clock polarity (CPOL) and phase (CPHA) as the slave. The *mode* parameter allows defining the necessary SPI communication mode.

It is possible to configure four modes using CPOL and CPHA values, as shown in the following table.

Table 2-4. Configure Four Modes using CPOL and CPHA Values

Mode	CPOL	CPHA
0	0	0
1	1	0
2	0	1
3	1	1

The timing diagram for all the modes is shown below.



At CPOL = 0 normal clock mode.

- For CPHA = 0, data is captured on the clock's rising edge (low¨high transition) and data is propagated on a falling edge (high¨low clock transition).
- For CPHA = 1, data is captured on the clock's falling edge and data is propagated on a rising edge.

At CPOL = 1 inverse clock mode.

- For CPHA = 0, data is captured on clock's falling edge and data is propagated on a rising edge.
- For CPHA = 1, data is captured on clock's rising edge and data is propagated on a falling edge.

```
public enum enumSpiBitOrder
   {
         MSB = 0,
         LSB = 1
   }
```

```
public enum enumSpiMode
   {
         Mode_00 = 0,
         Mode_01 = 1,
         Mode_02 = 2,
         Mode_03 = 3
         }
```

Example:

```
>SPI_ConfigureBus 0 0 1000000 0 0 0
<
0 OK
>
```

Perl:

```
my $val = $pp->SPI_ConfigureBus(0, 0, 1000000, "0 0 0");
```

SPI_DataTransfer (IN mode, nvector IN dataIN, nvector OUT dataOUT)

Generates transaction on the PI bus. Parameters are as follows:

mode - State of SS signal during transaction; if 0x02 - SS assert, if 0x08 - SS deassert.

dataIN - Data to be sent to the slave,

dataOUT - Data received from the slave. Size of *dataOUT* equals the size of *dataIN*.

Example:

```
>SPI_DataTransfer 0xA 0x20 0x1 0x0 0x2 0x1 0x3 0x0 0x4 0x0
<ff fe fd fc fb fa f9 f8 f7
0 OK
>
```

Perl:

```
my @arr = $pp->SPI_DataTransfer(0xA, "0x20 0x1 0x0 0x2 0x1 0x3 0x0 0x4
0x0");
```

SPI_GetSupportedFreq (nvector OUT freq)

Returns the set of frequencies that the master can generate. Frequencies are expressed in Hz. One of these frequencies should be used in the SPI_ConfigureBus() function for correct bus initialization.

Example:

```
>SPI_GetSupportedFreq
<4000000 3000000 2400000 2000000 1714286 1500000 1333333 1200000 1090909 1000000
 923077 857143 800000 750000 705882 666667 631579 600000 571429 545455 521739 50
0000 480000 461538 444444 428571 413793 400000 387097 375000 363636 352941 34285
7 333333 324324 315789 307692 300000 292683 285714 279070 272727 266667 260870 2
55319 250000 244898 240000 235294 230769 226415 222222 218182 214286 210526 2068
97 203390 200000 196721 193548 190476 187500 184615 181818 179104 176471 173913
171429 169014 166667 164384 162162 160000 157895 155844 153846 151899 150000 148
148 146341 144578 142857 141176 139535 137931 136364 134831 133333 131868 130435
 129032 127660 126316 125000 123711 122449 121212 120000 118812 117647 116505 11
```

```
5385 114286 113208 112150 111111 110092 109091 108108 107143 106195 105263 10434
8 103448 102564 101695 100840 100000 99174 98361 97561 96774 96000 95238 94488 9
3750 93023 92308 91603 90909 90226 89552 88889 88235 87591 86957 86331 85714 851
06 84507 83916 83333 82759 82192 81633 81081 80537 80000 79470 78947 78431 77922
 77419 76923 76433 75949 75472 75000 74534 74074 73620 73171 72727 72289 71856 7
1429 71006 70588 70175 69767 69364 68966 68571 68182 67797 67416 67039 66667 662
98 65934 65574 65217 64865 64516 64171 63830 63492 63158 62827 62500 62176 61856
 61538 61224 60914 60606 60302 60000 59701 59406 59113 58824 58537 58252 57971 5
7692 57416 57143 56872 56604 56338 56075 55814 55556 55300 55046 54795 54545 542
99 54054 53812 53571 53333 53097 52863 52632 52402 52174 51948 51724 51502 51282
51064 50847 50633 50420 50209 50000 49793 49587 49383 49180 48980 48780 48583 4
8387 48193 48000 47809 47619 47431 47244 47059
0 OK
>
```

Perl:

```
my @arr = $pp->SPI_GetSupportedFreq();
```

SWDIOR (IN addr, OUT data)

Reads content of the register of a PSoC 3, PSoC 4, or PSoC 5 device using the SWD interface.

Example: (PSoC 4)

```
>swdior 0x20000000
<0x20002000
0 OK
>swdiow 0x20000000 0x12345678
<
0 OK
>swdior 0x20000000

<0x12345678
0 OK
>swdior_raw 0x00
<77 14 b1 0b 21
0 OK
>
```

Perl:

```
my $val = $pp->swdior(0x20000000);
$pp->swdiow(0, 123);
$val = $pp->swdior(0x20000000);
```

SWDIOR_RAW (nvector IN input, nvector OUT output)

Performs one read transaction on the SWD-bus. Parameters are as follows:

input - one-byte vector defining APnDP register's area (bit 2, mask 0x04) and 2-bit register address (bits 1:0, mask 0x03).

output - 5-byte vector, where the first byte is the status of transaction, and other 4 bytes are read from the register.

You should consider only three LSB bits of the status byte. They are in the raw format of SWD transaction:

b'001 - ACK

b'010 - WAIT

b'100 - FAULT

Other combinations can be considered as NACK (especially b'111 - which means the device does not respond at all).

Example:

```
>swdior_raw 0x00
<77 14 b1 0b 21
0 OK
>swdiow_raw 0x00
<21
0 OK
>
```

Perl:

```
my $val2 = $pp->swdior_raw(0x00);
$val2 = $pp->swdiow_raw(0x00);
```

SWDIOW (IN addr, IN data)

Writes *data* to the register of the PSoC 3, PSoC 4, or PSoC 5 device using the SWD interface.

Example:

Refer to SWDIOR(IN addr, OUT data)

SWDIOW_RAW (nvector IN input, nvector OUT output)

Performs one write transaction on SWD_bus. Parameters are as follows:

input - 5-byte array with 1-byte header and 4-byte data. The header defines APnDP register's area (bit 2, mask 0x04) and register address (bits 1:0, mask 0x03).

output - 1-byte vector containing the status of the transaction. For status byte description, see SWDIOR_RAW API.

Example:

Refer to SWDIOR_RAW(nvector IN input, nvector OUT output).

SWD_LineReset()

Resets and reinitializes the SWD bus by sending 50 or more clock bits on the SWDCLK line, which keeps SDATA high.

Example:

```
>SWD_LineReset
<
0 OK
```

```
>swdior_raw 0x00
<77 14 a0 1b 31
0 OK
>
```

SWV_ReadData(nvector OUT dataOUT)

Allows the client to retrieve SWV data received from the slave device. The method is expected to be continuously pulled by the client to check whether new data is available.

The *dataOUT* array contains received SWV data generated since the last request. If there is no new SWV data, then API fails and *dataOUT* is empty.

Only MiniProg3 supports this method. Note that the SWV protocol must be set in the programmer before using this method. Use SetProtocol() API to set the protocol and SWV_Setup() to configure frequency and encoding type (UART or Manchester) of data transfer.

Example:

```
>SetProtocol 32
<
0 OK
>SWV_Setup 1 6000000
<
0 OK
>SWV_ReadData
<09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b
05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03
01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13
09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b
05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03
01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13
09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b
05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03
01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13
09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b
05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03
01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13
09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b
05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03
01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13
09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 00
00 00 00 00 80 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08
13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04
0b 05 0607 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c
03 01 02 03 04 0b 05 06 07 08 13 09 0a 0b 0c 03 01 02 03 04 0b 05 06 07 08
13 09 0a 0b 0c 03 01 02 03 04 0b 05
0 OK
>
```

SWV_Setup(enumSWVMode IN mode, IN targetFreq, nvector IN extra)

This method is designed to configure the MiniProg3 for SWV data sampling. It allows the MiniProg3 to handle multiple SWV data modes and multiple bus frequencies.

Only MiniProg3 implements this method. The SWV mode must be set up by SetProtocol() before using this method. The MiniProg3 supports SWV data sampling on both the 5-pin and the 10-pin connector. The SWV input uses the XRES line on the 5-pin header and the TDO line for the 10-pin header.

Parameters are as follows:

mode - specifies whether to use Manchester (0x02) encoding or UART (0x01).

targetFreq – the frequency on which the slave device generates SWV traffic. It must be in the range 188.235 kHz to 24 MHz for UART and 188.235 kHz to 12 MHz for Manchester. This parameter is passed in Hz.

extra – a reserved parameter for future use.

```
enum enumSWVMode
{
TX8 = 0x01,
MANCHESTER = 0x02
} enumSWVMode;
```

Example:

```
>SWV_Setup 1 230400
<
0 OK
>
```

TableRead(IN blockid, OUT xa,nvector OUT block, OUT result)

Does a TableRead supervisory operation (see the TRM). Parameters are as follows:

blockid - indicates which table should be returned.

xa - CPU_A and CPU_X registers value.

block - returned table.

Example:

```
>TableRead 0x00
<0x00001021
00 0b 13 01 00 00 00 00
0x0012fcb4
0 OK
>
```

TestClock(IN numberOfClocks)

The programmer sends *numberOfClocks* through the SCLK line to the acquired PSoC device.

Example:

```
>TestClock 11
<
1 OK
>
```

ToggleReset (IN polarity, IN duration)

Generates a reset signal on the XRES line depending on polarity and duration parameters.

Where:

polarity: XRES_POLARITY_NEGATIVE = 0x00

XRES_POLARITY_POSITIVE = 0x01

duration - length of the reset signal, specified in milliseconds

Example:

```
>ToggleReset 01 05
<
0 OK
>
```

Perl:

```
$pp->ToggleReset(0x01, 0x05);
```

UpdateProgrammer(string IN arguments)

Updates the programmer to the version in the home directory.

Example:

```
>UpdateProgrammer MINIProg1/868C72CF030D .
<
0 OK
>
```

Verify()

Verifies the contents of the device against the current hex file.

Example:

```
>Verify
<
256 OK
>
```

Perl:

```
$pp->Verify();
```

VerifyProtect()

Verifies that the protection of the blocks equals the protection level specified for that block in the hex file. It does not verify that the contents of the blocks match with the hex file.

Example:

```
>VerifyProtect
<
0 OK
>
```

WriteBlock(IN blockid, nvector IN block, OUT result)

Writes *blockid* with contents *block* (*block* must be 64 bytes).

Example:

```
>WriteBlock 0x01 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01
0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06
0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04
<0x00000001
6 OK
>
```

Perl:

```
$pp->ProgramBlock($blockID, $dataIN);
```

WriteBlock1(IN bank, IN blockid,nvector IN block, OUT result)

Writes *blockid* with contents *block* (*block* must be 64 bytes) in *bank.*

Example:

```
>WriteBlock1 0x00 0x01 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x0A 0x01 0x02 0x03 0x04
<0x00000001
6 OK
>
```

WriteIO(IN address, IN size, nvector IN data)

Writes a data vector to the I/O registers of the acquired PSoC device in the current register's bank. The data is written in the consecutive registers starting from the given address.

Example:

```
>WriteIO 0x03 0x03 0x00 0x12 0x0A
<
0 OK
>
```

WriteRAM(IN address, IN size, nvector IN data)

Writes a data vector to the SRAM of the acquired PSoC device in the current SRAM bank. The data is written in the consecutive addresses starting from the given address.

Example:

```
>WriteRam 0x03 0x03 0x00 0x12 0x0A
<
0 OK
>
```

help

Returns the list of all supported APIs.

quit

Exits the application.

version

Returns the version of the software.

3. Examples

This chapter gives example runs for programming a chip and programming a block.

3.1 Example 1: Programming of Flash Rows for PSoC 6

This example shows how to program 10 rows of flash for the PSoC 6 family. Modify this script if you need to write more rows to add pairs of 'PSoC6_ProgramRowFromHex X' and 'PSoC6_VerifyRow-FromHex X' APIs, where 'X' is the row number.

```
OpenPort MiniProg3/1229DD001023 .
SetAcquireMode "Reset"
SetProtocol 8
SetProtocolClock 224
SetProtocolConnector 1
HEX_ReadFile "C:\\HEX1.hex"
HEX_GetRowsCount

DAP_Acquire

PSoC6_EraseAll

PSoC6_ProgramRowFromHex 0
PSoC6_VerifyRowFromHex 0
PSoC6_ProgramRowFromHex 1
PSoC6_VerifyRowFromHex 1
PSoC6_ProgramRowFromHex 2
PSoC6_VerifyRowFromHex 2
PSoC6_ProgramRowFromHex 3
PSoC6_VerifyRowFromHex 3
PSoC6_ProgramRowFromHex 4
PSoC6_VerifyRowFromHex 4
PSoC6_ProgramRowFromHex 5
PSoC6_VerifyRowFromHex 5
PSoC6_ProgramRowFromHex 6
PSoC6_VerifyRowFromHex 6
PSoC6_ProgramRowFromHex 7
PSoC6_VerifyRowFromHex 7
PSoC6_ProgramRowFromHex 8
PSoC6_VerifyRowFromHex 8
PSoC6_ProgramRowFromHex 9
PSoC6_VerifyRowFromHex 9
PSoC6_ProgramRowFromHex 10
PSoC6_VerifyRowFromHex 10
```

3.1.1 Sample Output

```
>OpenPort MiniProg3/1229DD001023 .
<
0 OK
>SetAcquireMode "Reset"
<
0 OK
>SetProtocol 8
<
0 OK
>SetProtocolClock 224
<
0 OK
>SetProtocolConnector 1
<
0 OK
>HEX_ReadFile "C:\\HEX1.hex"
<0x00080e00
0 OK
>HEX_GetRowsCount
<0x0000001c
0 OK
>
>DAP_Acquire
<
0 OK
>
>PSoC6_EraseAll
<
0 OK
>
>PSoC6_ProgramRowFromHex 0
<
0 OK
>PSoC6_VerifyRowFromHex 0
<0x00000001
0 OK
>PSoC6_ProgramRowFromHex 1
<
0 OK
>PSoC6_VerifyRowFromHex 1
<0x00000001
0 OK
>PSoC6_ProgramRowFromHex 2
<
0 OK
>PSoC6_VerifyRowFromHex 2
<0x00000001
0 OK
>PSoC6_ProgramRowFromHex 3
<
0 OK
```

```
>PSoC6_VerifyRowFromHex 3
<0x00000001
0 OK
>PSoC6_ProgramRowFromHex 4
<
0 OK
>PSoC6_VerifyRowFromHex 4
<0x00000001
0 OK
>PSoC6_ProgramRowFromHex 5
<
0 OK
>PSoC6_VerifyRowFromHex 5
<0x00000001
0 OK
>PSoC6_ProgramRowFromHex 6
<
0 OK
>PSoC6_VerifyRowFromHex 6
<0x00000001
0 OK
>PSoC6_ProgramRowFromHex 7
<
0 OK
>PSoC6_VerifyRowFromHex 7
<0x00000001
0 OK
>PSoC6_ProgramRowFromHex 8
<
0 OK
>PSoC6_VerifyRowFromHex 8
<0x00000001
0 OK
>PSoC6_ProgramRowFromHex 9
<
0 OK
>PSoC6_VerifyRowFromHex 9
<0x00000001
0 OK
>PSoC6_ProgramRowFromHex 10
<
0 OK
>PSoC6_VerifyRowFromHex 10
<0x00000001
0 OK
```

3.2 Example 2: Programming a Chip

```
OpenPort MINIProg1/848E4756090B “c:/Program Files/Cypress/Programmer”
HEX_ReadFile "c:\\CY8C24794.hex"
SetAcquireMode “Power”
Acquire
Calibrate
EraseAll
Program
Verify
VerifyProtect
Protect
CheckSum 0
HEX_ReadChecksum
```

3.2.1 Sample Output

```
Prompt> c:/Program Files/Cypress/Programmer/ppcli.exe -t "--runfile pro-
gram-example" --quit
>OpenPort MINIProg1/848E4756090B “c:/Program Files/Cypress/Programmer/
x.y”
<
0 OK
>HEX_ReadFile "c:\\CY8C24794.hex"
<
0 OK
>SetAcquireMode “Power”
<
0 OK
>Acquire
<
0 OK
>Calibrate
<
1 OK
>EraseAll
<0x00000001
1 OK
>Program
<
6 OK
>Verify
<
0 OK
>VerifyProtect
<
0 OK
>CheckSum 0
<0x00007563
1 OK
>HEX_ReadChecksum
<0x7563
0 OK
>
```

```
Prompt>
```

3.2.2 Perl Example

```
use PSoC_Programmer;
my $pp = new PSoC_Programmer;
$pp->start("ppcli.exe");
my @ports = $pp->getports();
$pp->OpenPort("MINIProg1/048E4756080B","C:/Program Files/Cypress/Program-
mer/");
#$result = $pp->OpenPort($ports[0],"C:/Program Files/Cypress/Programmer/
");
#print ("result = ",$result,"\n");
$pp->HEX_ReadFile("C:/test/test.hex");
$pp->SetAcquireMode("Reset");
$pp->SetProtocol(2);
$pp->PowerOn();
$pp->Acquire();
$pp->GetSiliconId();
$pp->Calibrate0();
$pp->EraseAll();
$pp->Program();
$pp->Verify();
$pp->Protect();

my $dev_checksum = $pp->CheckSum(0);
my $hex_checksum = $pp->HEX_ReadChecksum();

if(hex($dev_checksum) != hex($hex_checksum))
{
print "Device checksum $dev_checksum \n";
print "Hex file checksum $hex_checksum \n";
print "programming failed\n";
}
else
{
print "programming succeeded\n"
}

$pp->SetAutoReset(1);
$pp->ReleaseChip();

$pp->quit();
1;
```

3.3 Example 3: Programming a Block

```
OpenPort MINIProg1/848E4756090B “C:/Program Files/Cypress/Programmer/”
SetAcquireMode “Power”
Acquire
Calibrate
EraseBlock 0x40
ReadBlock 0x40
```

```
WriteBlock 0x40 01 02 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ReadBlock 0x40
```

3.3.1 Sample Output

```
Prompt> c:/Program Files/Cypress/Programmer/ppcli.exe -t "--runfile write-
one-block-example" --quit
>OpenPort MINIProg1/848E4756090B “C:/Program Files/Cypress/Programmer/”
<
0 OK
>SetAcquireMode “Power”
<
0 OK
>Acquire
<
0 OK
>Calibrate
<
1 OK
>EraseBlock 0x40
<0x00000000
8 OK
>ReadBlock 0x40
<00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 0
0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00
00 00 00 00 00 00 00 00 00 00
0x00000000
8 OK
>WriteBlock 0x40 01 02 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
<0x00000000
6 OK
>ReadBlock 0x40
<01 02 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 0
0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00
00 00 00 00 00 00 00 00 00 00
0x00000000
8 OK
>
Prompt>
```

3.3.2 Perl Example

```
#!perl
use PSoC_Programmer;
```

```
use strict;
use warnings;

my $status;
my $pp = new PSoC_Programmer;
$pp->start("C:/Program Files/Cypress/Programmer/ppcli.exe");

my @ports = $pp->getports();

$pp->OpenPort($ports[3],"C:/Program Files/Cypress/Programmer/ppcli.exe");

my $pv = $pp->GetProgrammerVersion();
$status = $pp->SetAcquireMode("Power");
$status = $pp->Acquire;
$status = $pp->EraseBlock(0x40);
my @block = $pp->ReadBlock(0x40);
$block[0]= 0x1;
$block[1]= 0x2;
$block[2]= 0x3;
$status = $pp->WriteBlock(0x40,@block);
my @block = $pp->ReadBlock(0x40);
$pp->quit();
1;
```

3.4 Example: Write NVL/Flash Data for PSoC 3 ES3

```
OpenPort MiniProg3/1134CC000006 .
SetAcquireMode “Reset”
SetProtocol 8
SetProtocolClock 224
SetProtocolConnector 1
SetPowerVoltage 5.0
PowerOn
DAP_Acquire
DAP_GetJtagID
PSoC3_GetTemp 0x1 0xFFF 0x4
PSoC3_WriteNvlArray 0x80 0x00 0x00 0x00 0x08
PSoC3_ReadNvlArray 0x80
DAP_Acquire //If ECC bit in NVL is changed then need to reacquire
PSoC3_WriteRow 0x00 0xFF 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00
```

```
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05
0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08
PSoC3_ReadRow 0x00 0xFF 0x00
PSoC3_ProtectArray 0x00 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
PSoC3_ReadProtection 0x00
ClosePort
```

3.4.1 Sample Output

```
>OpenPort MiniProg3/1134CC000006 .
<
0 OK
>SetAcquireMode “Reset”
<
0 OK
>SetProtocol 8
<
0 OK
>SetProtocolClock 224
<
0 OK
>SetProtocolConnector 1
<
0 OK
>SetPowerVoltage 5.0
<
0 OK
>PowerOn
<
0 OK
>DAP_Acquire
<
0 OK
>DAP_GetJtagID
<1e 02 80 69
0 OK
>PSoC3_GetTemp 0x1 0xFFF 0x4
<0x01
0x22
0 OK
>PSoC3_WriteNvlArray 0x80 0x00 0x00 0x00 0x08
<
0 OK
>PSoC3_ReadNvlArray 0x80
<00 00 00 08
0 OK
>DAP_Acquire //If ECC bit in NVL is changed then need to reacquire
```

```
<
0 OK
>PSoC3_WriteRow 0x00 0xFF 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04
0x05 0x06 0x07 0x08 0x09 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08
<
0 OK
>PSoC3_ReadRow 0x00 0xFF 0x00
<01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05
06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00
01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05
06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00
01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05
06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00
01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05
06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00
01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05
06 07 08 09 00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 00
01 02 03 04 05 06 05 06 07 08
0 OK
>PSoC3_ProtectArray 0x00 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
<
0 OK
>PSoC3_ReadProtection 0x00
<ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0 OK
>ClosePort
<
0 OK
>
```

Revision History

Document Title: PSoC Programmer Command-Line Interface (CLI) Guide Document Number: 001-16300			
Revision	**ECN No.**	**Issue Date**	**Description of Change**
**	1449036	Jun 15, 2007	Put changes to the original CLI Guide in a new template and assigned a Spec Number.
*A	2507445	May 22, 2008	Added new command descriptions for PSoC Programmer 3.
*B	2794466	Oct 29, 2009	Added new APIs.
*C	2878959	Apr 08, 2010	New APIs, updated two APIs, Examples updated
*D	2998963	Aug 02, 2010	4 new APIs GetPower2 (...) SPI_ConfigureBus (...) SPI_DataTransfer (...) SPI_GetSupportedFreq (...) 2 APIs updated GetPower1(OUT power, OUT voltage, string OUT strError) SetProtocol(enumInterfaces IN protocol, string OUT strError)
*E	3607597	May 3, 2012	Removed _StartSelfTerminator(), PSoC 4_SiliconID() APIs from the guide. SWV_ReadData(), SWV_Setup() added. Updated SetProtocol(). Most sections rewritten to conform to current quality standards. More examples added to command descriptions. More explanation added to ReadIO(), ReadProgram(), AcquireChip(), AcquireChipWithDelay(). Updated description of SetProtocolConnector to include SWD mode for TrueTouch Bridge.
*F	3704591	Aug 03, 2012	Updated CheckSum1(), ClosePort(), DAP_Acquire(), DAP_AcquireChip(), DAP_WriteIO (), GetProgrammerVersion(), HEX_ReadChipProtection(), I2C_ReadData(), I2C_SendData(), I2C_SetSpeed(), Protect(), PSoC4_CheckSum(), RunFile(), SetAcquireMode(), SetAutoReset(), SWDIOR_RAW(), SWV_ReadData(), SWV_Setup();

Document Title: PSoC Programmer Command-Line Interface (CLI) Guide Document Number: 001-16300			
Revision	**ECN No.**	**Issue Date**	**Description of Change**
*G	3828175	Dec 03, 2012	Added new APIs HEX_WriteEEPROM(), DAP_ReadRaw(), DAP_WriteRaw(), JTAG_SetIR(), JTAG_ShiftDR() Updated definition of PSoC3_EraseRow() API. Updated example for GetProgrammerCapabilities() API. Add new APIs and JTAG_SetChainConfig() to the Table 1-1 in section 1.5 "Command Overview"; Obsoleted API PSoC3_SetJtagChainConfig() Updated paths in OpenPort() API in Perl examples from section 3.1.2 and 3.2.2 Added new example "Write NVL/Flash Data for PSoC 3 ES3" to the section 3 "Examples"
*H	3902076	Feb 12, 2013	Removed obsolete API PSoC3_ReadIO(), PSoC3_WriteIO() from Table 1-1. Added Description of DAP_ReadIO API(), "help" API, and JTAG_ShiftIR() API.
*I	4046552	Jul 01, 2013	Corrected typo in PSoC4_GetSiliconID API description.
*J	4207146	Dec 02, 2013	Replaced SetHexFile with HEX_ReadFile in Chapters 2 and 3.
*K	4273553	Feb 06, 2014	Removed references to obsolete APIs.
*L	4506509	Sep 16, 2014	Updated the description of the following APIs: PSoC3_WriteNvlArray(), PSoC3_ReadNvlArray(), and PSoC3_CheckSum().
*M	4735318	Apr 22, 2015	Updated the commands in the Examples chapter for more clarity.
*N	4908796	Sep 07, 2015	Updated Overview chapter. Removed Perl examples for APIs which are not implemented in Perl module.
*O	5283060	Aug 26, 2016	Updated the PSoC4_GetFlashInfo command. Updated Table 1-1: Added APIs for GetRowsPerArrayInFlash API and FM0+ devices. Updated Chapter 2 with new APIs.
*P	5560941	Dec 20, 2016	Added APIs for PSoC 6 devices.
*Q	5840674	Jul 21, 2017	Migrated to the new template.
*R	5958435	Nov 11, 2017	Updated Commands: PSoC6_ProtectAll VerifyProtect Added Command: PSoC6_EraseRow
*S	6215567	Jun 25, 2018	No technical updates. Completing Sunset Review.
*T	6398278	Nov 30, 2018	Added new APIs: HEX_GetDataInRange, HEX_GetDataSizeInRange, FL_LoadElf, FL_IsApiExists, FL_ExecAPI, FL_SetRamForAlgorithms, FL_PrepareTarget, FL_ExecCmsisApiInit,FL_ExecCmsisApiUnInit, FL_ExecCmsisApiEraseSector, FL_ExecCmsisApiProgramPage, FL_ExecCmsisApiVerify, FL_ReleaseTarget. Fixed typos in few APIs.
*U	6678889	Sep 20, 2019	Added note to PSoC4_GetFlashInfo
*V	6754358	Dec. 17, 2019	Updated PSoC6_WriteProtection(BYTE IN lifeCycle, nvector IN secureRestrict, nvector IN deadRestrict, BOOL IN voltageVerification) chapter on page 63