AN73054

PSoC® 3 and PSoC 5LP Programming Using an External Microcontroller (HSSP)

Author: Vivek Shankar Kannan

Associated Project: Yes

Associated Part Family: All PSoC 3 and PSoC 5LP parts

Software Version: PSoC Creator™ 3.3 CP1 or later

Related Application Notes: AN84858, AN44168, AN73854

AN73054 shows you how to implement PSoC 3 or PSoC 5LP device programming with an external microcontroller by using modular C code. In this process, called Host Sourced Serial Programming (HSSP), the host microcontroller programs PSoC 3 or PSoC 5LP through the serial wire debug (SWD) interface. The C code is written so that it can be ported to any microcontroller with minimal changes, speeding up HSSP application development for PSoC 3 or PSoC 5LP.

Contents

1 Introduction

PSoC 3 or PSoC 5LP device programming refers to the programming of the nonvolatile memory in the device by using an external host programmer. The host can be the programmer supplied by Cypress (CY8CKIT-002 MiniProg3), a third-party programmer, or a custom programmer (for example, an onboard microcontroller). This application note explains how to implement a host programmer to program a PSoC 3 or PSoC 5LP device. For more information on the device architecture and to learn how to create projects for PSoC 3 or PSoC 5LP using the PSoC Creator™ software, refer to the application notes: AN54181 – Getting Started with PSoC® 3, and AN77759 – Getting Started with PSoC® 5LP.

1.1 Types of Programmers

The type of device programmer you choose depends on the stage of product development:

Prototyping: A programmer must be able to perform the following functions:

1. Program the device.
2. Debug the device to troubleshoot the application.

The programmers used during prototyping must also interact with the integrated design environment (IDE)—for example, PSoC Creator™—to accomplish the programming and debugging operations. A few examples are Cypress's MiniProg3 or the PSoC® 5LP Prototyping Kit, which can be used as a low-cost programmer/debugger.

Production: You require a programmer that can program multiple devices. It parses the hex file to extract the necessary information and implements programming through the programming interface, such as SWD.

There are two major categories of programmers:

- In-system programmers can program the target device directly on the end-application PCB. You can connect the external programmer to the device's programming pins to do in-system programming.
- Socket programmers require the target device to be placed on the programmer hardware socket for programming. After programming, solder the target device to the end-application PCB. Most third-party production programmers are of the socket type.

In both in-system and socket programming, the programmer implements an HSSP algorithm and generates signals to program the hex file's data.

This application note provides the C code to implement an HSSP programmer. You can easily port this C code to any host microcontroller with minimal changes. By porting, you reduce the time required to develop HSSP applications. The project provided with this application note uses a PSoC 5LP device as a host programmer to program the target PSoC 3 or PSoC 5LP device.

Before reading this application note, review the programming specifications document of the respective device listed in the Related Documentation section. This document explains the programming interface, programming algorithm, hardware connection, and electrical timing specifications required to program a PSoC 3 or PSoC 5LP device. This application note is a practical implementation of the programming specifications.

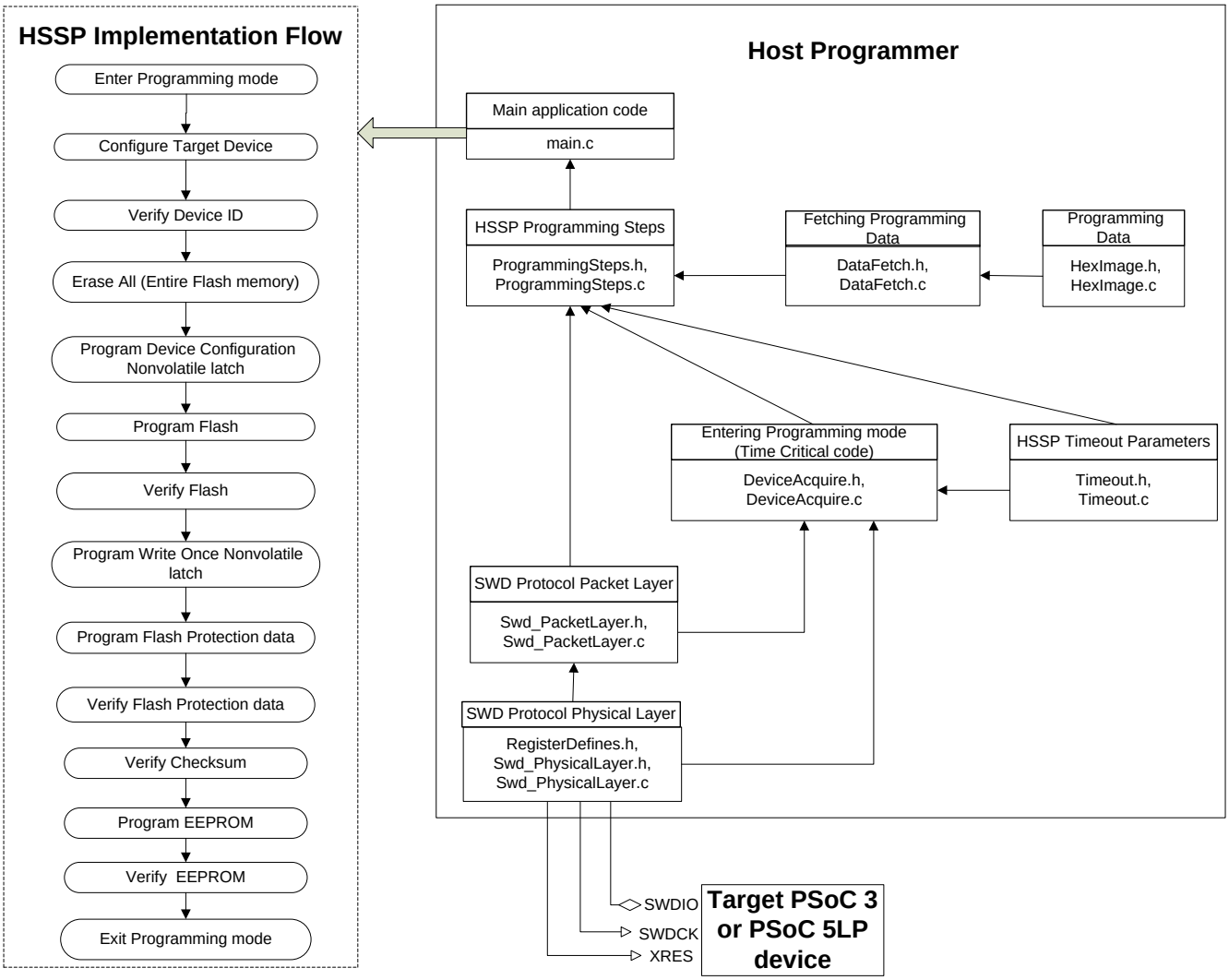
1.2 Terms and Definitions

1. **Serial wire debug (SWD)**: Developed by ARM, the SWD protocol uses only two wires—SWDCLK (clock) and SWDIO (bidirectional data line)—to program and debug.
2. **Debug access port (DAP):** DAP is the program/debug interface between SWD and the Cortex-M3 CPU in PSoC 5LP. PSoC 3, which has an 8051 CPU, has a test controller block that is similar to the DAP in PSoC 5LP. The DAP includes a debug port (DP) and an access port (AP).
   - DP is responsible for the physical connection to the programmer/debugger.
   - AP provides the interface between the DAP module and the CPU, the flash memory, and so on.
3. **HSSP:** HSSP refers to the programming of the target device on the board using a host microcontroller. The target PSoC is programmed through the SWD interface. In this application note, HSSP uses a bit-banging implementation to program the target device. Bit-banging programming refers to the technique in which programming pins are manipulated using a software code that resides in the host programmer.
4. **Differences between bootloading and HSSP:** In embedded systems, bootloaders are also used to update the system firmware. Bootloading and HSSP differ in the following key aspects:
   - Bootloaders are used to update the flash memory of the device over a standard communication protocol. Bootloaders can update only a specific portion of the flash memory, known as the bootloadable area.
   - On the other hand, HSSP supports complete programming of the flash memory in PSoC.
   - Bootloaders can use any standard communication interface (such as, USB, I^2C, SPI, and UART) to update the firmware, while HSSP uses an SWD or JTAG interface to program the flash. This application note project supports only SWD interface.

2 HSSP Firmware Architecture

Figure 1 shows the call tree graph of the HSSP firmware architecture and the high-level sequence of steps in the HSSP application. Those steps displayed under *HSSP Implementation Flow* in Figure 1 are explained in detail in the PSoC 3 or PSoC 5LP Programming Specifications. The function of each layer of the code in the HSSP firmware architecture is explained in the following sections. Appendix B lists the functions used in the different layers of the HSSP firmware.

Figure 1. PSoC 3 and PSoC 5LP HSSP Firmware Architecture



2.1 SWD Protocol Physical Layer

This layer includes the files *RegisterDefines.h*, *Swd_PhysicalLayer.h*, and *Swd_PhysicalLayer.c*. These files have the driver routines to manipulate the SWDIO, SWDCK, and XRES pins used for programming the target device. The code in these files applies when PSoC 5LP is the host programmer. For any other host processor, you must modify the function definitions and the macro definitions in the files appropriately.

RegisterDefines.h: This header file contains the definitions for the port number, pin number, output register, input register, and the drive mode register of the programming pins. The definitions are specific to PSoC 5LP as the host programmer and should be modified for the specific host used. The register definitions are used in the *Swd_PhysicalLayer.c*, *Swd_PhysicalLayer.h* files to define the bit banging routines for the programming pins.

Swd_PhysicalLayer.h, ***Swd_PhysicalLayer.c***: These files contain the macros and the functions to manipulate the programming pins. They use the register definitions in *RegisterDefines.h* to access the programming pins. Note that the bit banging routines are provided both as macros and as functions. That is because the procedure to enter the programming mode of PSoC 3 and PSoC 5LP (defined as "Step 1: Enter Programming Mode" in the programming specification) has strict timing requirements. Therefore, bit banging macros are used to reduce the execution time.

2.2 SWD Protocol Packet Layer

This layer includes the files *Swd_PacketLayer.h*, *Swd_PacketLayer.c*. These files have the packet routines for sending the SWD read and the SWD write packets. These packet routines are called by the functions in *DeviceAcquire.c* and *ProgrammingSteps.c*. To meet the critical timing requirements in entering the programming mode of PSoC 3 or PSoC 5LP, a separate SWD write packet function `Swd_WritePacketFast(…)` is defined that has been optimized for execution time by having few function calls in its definition. This function expects the pre-calculated parity bit of the 4-byte data as a function parameter. This function uses the bit banging macros defined in *Swd_PhysicalLayer.h*, and another function `Swd_SendByteFast(…)` to send a single SWD write packet quickly.

All of these SWD packet functions operate on three global variables - `Swd_ packetHeader`, `Swd_packetAck`, and `Swd_packetData[]`, that are accessed by the functions in the top layer files, as shown in Figure 1.

2.3 HSSP Timeout Parameters

This layer includes the files *Timeout.h*, *Timeout.c*. These files have the time stamp definitions and the delay routines used in HSSP. The time stamp definitions are derived from the electrical timing specifications provided in the PSoC 3 PSoC 5LP Device Programming Specifications. The values of these time stamp parameter definitions in *Timeout.h* are for a PSoC 5LP host programmer running at a clock frequency of 66 MHz. To learn how to calculate these time stamp parameters for a specific host programmer, see the section Calculating HSSP Timeout Parameters. A delay routine `DelayHundredUs(…)`, defined in *Timeout.c* is used to generate an active low pulse of 100 µs on XRES pin to reset the target device. The time stamp definitions and the delay routine are used in the function definitions in files *DeviceAcquire.c* and *ProgrammingSteps.c*.

2.4 Entering HSSP Programming Mode

This layer includes the files *DeviceAcquire.h*, *DeviceAcquire.c*. These files contain the routines to acquire the target device to enter the HSSP programming mode (`AcquireTargetDevice(…)`), and to release the target device to exit the HSSP programming mode (`ReleaseTargetDevice(…)`).

The first step in HSSP is to enter the programming mode of the target device. For this step to be executed successfully, the host programmer must meet the timing requirements defined in "Step 1: Enter Programming Mode" in the programming specification. To meet these timing requirements, the function `AcquireTargetDevice(…)` is written in a performance (execution time) optimized manner. This is done by using very few internal function calls in the function definition, and also by using the bit banging macros defined in the file *Swd_PhysicalLayer.h*. This function also calls the `Swd_WritePacketFast(…)` function in *Swd_PacketLayer.h* to send the SWD acquire packets to the target device.

2.5 HSSP Programming Data

This layer includes the files *HexImage.h*, *HexImage.c*. These files contain the data to be programmed in to the target device, which includes the flash data, flash protection data, EEPROM, and the NVL data. The files also store the target device parameters used in HSSP programming, such as the device silicon ID, Checksum value of the flash data, number of flash rows, number of flash arrays, number of EEPROM rows, and the number of bytes for each flash row. In the project provided with the application note, the programming data is stored in the flash memory of the PSoC 5LP host programmer as an array of constants.

HexImage.h and *HexImage.c* files are generated from the PSoC 3 or PSoC 5LP hex file. See "Appendix A.1. Intel Hex File Format" in the programming specifications document for details on the hex file format. *HexImage.h*, *HexImage.c* files are generated by the application ***HexFileParser_Psoc3_5lp*** that is provided with this application note. This application generates these files by taking the PSoC 3 or PSoC 5LP hex file as the input. The details of this application are provided in Appendix A.

For host programmers that lack the memory capacity to store the programming data in the on-chip memory, the *HexImage.h, HexImage.h* files are not required. In such cases, the HSSP programming data will possibly be sent to the host as packets through a communication interface, such as I^2C, UART, and USB. See the section Interface for Receiving HSSP Programming Data for information on modifying the HSSP source code according to the method used to get the programming data.

2.6 Fetching Programming Data

This layer includes the files *DataFetch.h*, *DataFetch.c*. These files have the functions to fetch the programming data from the *HexImage.c* file and pass that data to the functions in the *ProgrammingSteps.c* file. These include the functions to get the flash row data, flash protection data, EEPROM data, NVL data, JTAG ID, checksum, total number of flash rows, and the number of bytes for each flash row. The definition of the functions needs to be modified based on the method used to get HSSP programming data. See the section Interface for Receiving HSSP Programming Data for information on modifying the HSSP source code.

2.7 HSSP Programming Steps

This layer includes the files *ProgrammingSteps.h*, *ProgrammingSteps.c*. These files contain the top-level functions of the HSSP application as shown in the **HSSP Implementation Flow** in Figure 1 These functions access the functions, definitions, and global variables from three layers namely - *Swd_PacketLayer.h*, *DeviceAcquire.h*, and *DataFetch.h*. The functions provided in these files cover all the steps required to program the target device.

2.8 Main Application Code

The *main.c* file is the main application code that calls the top-level HSSP functions in the sequence shown in Figure 1. Each step must be executed successfully to proceed to the next step. The HSSP operation is aborted if a failure code is returned after any step. For a failure code, the error status returned by the `ReadHsspErrorStatus()` is used to identify the cause of the error. The function `ProgramDevice(…)` in *main.c* does all of the steps mentioned in Figure 1. The status of the HSSP operation along with the error status register is displayed on a character LCD in the HSSP project provided with the application note. The character LCD routines are specific to the PSoC 5LP host programmer and should be modified as required for any other host programmer.

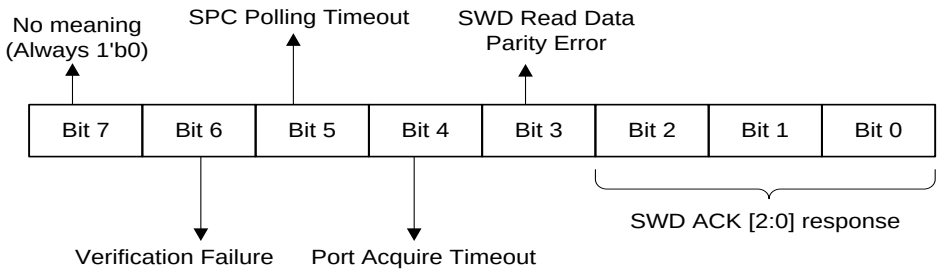
2.8.1 HSSP Error Status

When any of the top-level steps in the HSSP application return a failure status, the `ReadHsspErrorStatus()` function is called from the main application code to get the details of the error. This function returns the status byte, and from the bit field definitions of the status byte, the error details can be inferred. The bit field definitions of the status byte returned by this function are given in the following figure.

2.8.2 Bit Field Definitions of HSSP Error Status Register

For a successful HSSP operation, all bits except ‘Bit 0’ must be zero. ‘Bit 0’ being set indicates that the SWD packet received an OK ACK as shown in Table 1.

Figure 2. HSSP Error Status Register Bit Field Definitions



- **Bits[2:0] - SWD Acknowledge response (Swd ACK [2:0])** - This is the 3-bit acknowledgment response for a SWD packet sent by the target device to the host programmer. The possible ACK codes are listed in Table 1.

Table 1. SWD ACK Response Codes

ACK[2:0]	ACK Response Meaning
001	OK (SUCCESS)
010	WAIT
100	FAULT
Any other code	Undefined code. Treat this as FAULT response

All the responses except the OK ACK response require that the host abort the HSSP operation and restart from the first step. Even for an OK ACK, the rest of the bit fields (bits 3 to 6) in the status register in Figure 2 should not be set. If any of the other bit fields are set even with the OK ACK, the HSSP operation must be aborted and restarted.

The WAIT ACK will be set when a continuous WAIT ACK code is received for five consecutive tries of the SWD packet. WAIT ACK will be received if the host programmer tries to clock SWDCK at a frequency higher than the maximum specified value of SWDCK in the programming specifications.

A FAULT ACK, typically, will be received if there is a parity error in the 4-byte data packet sent by the host during the previous SWD write packet. Any other ACK code received by the host should also be treated similarly to FAULT ACK.

- **Bit 3 - SWD Read Data Parity Error** - The host programmer sets this bit if a parity error occurs in the data received from the target device. The host must abort the HSSP operation and try again.

- **Bit 4 - Port Acquire Timeout -** This bit is set if the SWD packets that are part of acquiring the target device (`AcquireTargetDevice(…)` in *`DeviceAcquire.c`*) are not completed successfully. The HSSP operation must be aborted if this bit is set, and retried. Two reasons can cause this timeout error: Either the hardware connection fails between the host programmer and the target device, or the host programmer fails to meet the timing requirements to enter the target device programming mode. For details on the timing requirements to enter the programming mode, see “Step 1: Enter Programming Mode” in Programming Specifications document.

- **Bit 5 - SPC Polling Timeout Error -** After every nonvolatile memory operation such as flash read and flash write, the host programmer must poll a status register (called SPC Status Register) to check if that operation is complete. If a success response is not received within 1 second (the maximum polling duration), this “SPC Polling Timeout” bit is set, and the HSSP operation must be aborted and restarted. The polling timeout typically occurs if the wrong parameters have been sent to the target device for a nonvolatile memory operation.

  If this bit is set, the host programmer must call the `ReadSpcStatusReg()` function in *`ProgrammingSteps.h`* to read the value of the SPC Status Register. See “Table A-3. Status Codes for an SPC Command” in the Programming Specifications document for details on the SPC status codes.

- **Bit 6 - Verification Failure -** This error bit is set under the following conditions.
  - Device ID Verification Error – If the Device silicon ID information in the hex file does not match the ID read from the target device, this bit is set. This means that the hex file to be programmed is not meant for the target device identified, and the HSSP operation is aborted. This can occur in the “Verify Device ID” step in the HSSP programming sequence.
  - Flash Data Verification Error - In HSSP a verify operation ensures that the data to be programmed matches with the flash data programmed in the target device. If the data mismatch, this bit is set. This can occur in the “Verify Flash” step in the HSSP programming sequence.
  - Flash Protection Data Verification Error – This is similar to the “Flash Data Verification Error” except that the comparison is done on the flash protection data. This can occur in the “Verify Flash Protection” step in the HSSP programming sequence.

- Checksum Verification Error – If the checksum value of the flash data in target device does not match with the checksum data in hex file, this bit is set. This can occur in the “Verify Checksum” step in the HSSP programming sequence.
- EEPROM Verification Error – This is similar to the “Flash Data Verification Error” except that the comparison is done on the EEPROM data. This can occur in the “Verify EEPROM” step in the HSSP programming sequence. Note that EEPROM program and verify operations are carried out only if the EEPROM data is present in the hex file.

It is clear from the above conditions that bit 6 can be set for multiple verification error cases. Based on the step in which the bit is set, the cause of the verification failure can be inferred. For example, if this bit is set in the “Verify Device ID” step, the host programmer application can identify that the error is due to mismatch of the Device ID.

- **Bit 7 -** No meaning for this bit. It is always zero.

3 Porting the HSSP Application to a Host Programmer

The project provided with this application note uses PSoC 5LP as a host programmer for the target device. In the HSSP application, the host programmer could also be any other microcontroller. This section explains the changes required to port this HSSP code to the specific host used to program the target device.

3.1 Files that need to be Ported

The PSoC 5LP host programmer-based project includes files that are specific to PSoC 5LP. While porting the HSSP code to any other host programmer, only the following files must be ported.

- **Header Files (*.h* files) to be ported:** *RegisterDefines.h, Swd_PhysicalLayer.h, Swd_PacketLayer.h, Timeout.h, DeviceAcquire.h, HexImage.h, DataFetch.h, ProgrammingSteps.h*
- **Source Files (*.c* files) to be ported:** *Swd_PhysicalLayer.c, Swd_PacketLayer.c, Timeout.c, DeviceAcquire.c, HexImage.c, DataFetch.c, ProgrammingSteps.c, main.c*

3.2 Code Changes Required while Porting

The following changes should be done to each of the files while porting the HSSP code to a host programmer other than PSoC 5LP. Note that only some files need to be modified while porting as explained. The rest of the files need not be modified.

- ***RegisterDefines.h*:** The definitions in this header file for the port numbers, pin numbers, mask values, output registers, input registers, and the drive mode registers should be modified according to the host programmer used.
- ***Swd_PhysicalLayer.h*:** The entire bit banging macros defined in this header file need to be modified based on the host programmer. Note that these macros are defined in the PSoC 5LP host programmer project to meet the timing requirements for entering the target device programming mode. It is recommended to have these macros for any other host programmer as well as to meet the timing requirements.
- ***Swd_PhysicalLayer.c*:** The entire bit banging functions defined in this file should be modified according to the host programmer used.
- **Timeout.h:** The four timeout parameter definitions namely - `XRES_PULSE_100US`, `TIME_WINDOW_68US`, `DEVICE_ACQUIRE_TIMEOUT`, and `SPC_POLLING_TIMEOUT` – need to be modified according to the host programmer used. See the section Calculating HSSP Timeout Parameters for the explanation on calculating these timeout parameters for a specific host programmer.
- ***DeviceAcquire.c* (if necessary):** The `AcquireTargetDevice()` function in this file has been written in a performance optimized (reduced execution time) manner in order to meet the timing requirements to enter the target device programming mode. If this function cannot meet the timing requirements defined in the programming specification without making changes, then the definition should be modified appropriately. One way to reduce the function execution time is to replace all the function calls inside the function with inline code.

- ***HexImage.c, HexImage.h*:** These files contain the data to be programmed into the target device defined as an array of constants. For PSoC 5LP host programmer, the data to be programmed is stored in the flash memory of host PSoC 5LP. Some host programmers might lack the capacity to store the programming data in their on-chip memory. Instead, they can use a communication interface, such as USB, SPI, or UART, to get the programming data. In such a case, these files need to be removed.
- ***DataFetch.c*:** The definitions for the functions must be modified based on the method used to get the programming data. See the section Interface for Receiving HSSP Programming Data for information on modifying the HSSP source code according to the method used to get programming data.
- ***main.c*:** The character LCD functions in *main.c* to display the HSSP application status on the LCD are for a PSoC 5LP host programmer, and should either be removed or modified while porting to any other host programmer.

In all of the above specified files, the code that needs to be modified based on the host programmer are preceded by the below comments. This will help in easy identification of the code to be modified.

```
/****** USER ATTENTION REQUIRED *****
 ****** HOST PROCESSOR SPECIFIC *****/
```

3.3 Calculating HSSP Timeout Parameters

The values of the timeout parameters defined in *Timeout.h* need to be modified according to the host programmer used. A separate test project, “Hssp_TimeoutCalc”, is provided with the application note to illustrate the procedure to calculate these timeout parameters for a PSoC 5LP host programmer. A similar test project should be created for any other host programmer to calculate these timeout values. The test project provides test functions in two files - *TimeoutCalc.h, TimeoutCalc.c*. These test functions toggle a test pin during code execution. Calculate the timeout parameters by measuring the low pulse width of the signal on the test pin. See the explanation in the macro definitions in the *TimeoutCalc.h* header file of the project for calculating these timeout parameters. The significance of each of these timeout values are as follows.

3.3.1 Timeout parameter name: TIME_WINDOW_68US

This constant definition is used in the function `AcquireTargetDevice()` in *DeviceAcquire.c* file to enter the target device programming mode. It is referred to as T_{BOOT} (68 µs) in the programming specifications document. During this time T_{BOOT}, the host must generate SWDCK at a frequency of $f_{SWDCK\_ACQUIRE}$. This frequency parameter value is provided in the programming specifications, and its minimum value is 1.4 MHz. The host must also drive the SWDIO line low during the period T_{BOOT}. The code snippet to do this in firmware is below. `TIME_WINDOW_68US` parameter determines the duration of clocking (T_{BOOT}) by the host programmer.

Code 1. Clocking for Time T_{BOOT}

```
/* Generate SWDCK clock waveform for 68 µS with SWDIO low. The timeout constant
TIME_WINDOW_68US used below comes
from Timeout.h */
unsigned short time_elapsed = 0;
SWDIO_OUTPUT_LOW;
do
{
        SWDCK_OUTPUT_LOW;
        SWDCK_OUTPUT_HIGH;
        time_elapsed++;
}
while(time_elapsed < TIME_WINDOW_68US);
```

To calculate this timeout value, see the explanation of the definition `TIME_WINDOW_68US` in the file *TimeoutCalc.h* in the “`Hssp_TimeoutCalc`” project.

3.3.2 Timeout parameter name: DEVICE_ACQUIRE_TIMEOUT

This constant definition is used in the function `AcquireTargetDevice()` in *DeviceAcquire.c* file to enter the target device programming mode. This timeout corresponds to the timing specification $T_{TESTMODE}$ in the Programming Specifications document. $T_{TESTMODE}$ defines the maximum time available for the host to enter device programming mode after doing a device reset using the XRES pin. `DEVICE_ACQUIRE_TIMEOUT` definition signifies the maximum number of SWD packets the host can send in this time window.

To calculate this timeout value, see the explanation of the definition `DEVICE_ACQUIRE_TIMEOUT` in the file *TimeoutCalc.h* in "`Hssp_TimeoutCalc`" project. This value should always be greater than or equal to 4 for any host programmer. That is because a minimum of four SWD packets are required to enter the programming mode.

3.3.3 Timeout parameter name: SPC_POLLING_TIMEOUT

This constant definition is used in the system performance controller (SPC) polling operations during the HSSP Programming. This is used while polling the result of nonvolatile memory read and write operations through the SPC in the target device. When the host has issued commands to the SPC through the SWD interface, it waits for a maximum of 1 second for a response from the SPC. If the response is not received, the host would abort the HSSP operation. This definition `SPC_POLLING_TIMEOUT` signifies the number of SWD read packets that can be sent in 1 second interval. When the host detects during polling that the number of SWD packets sent have crossed the limit defined by `SPC_POLLING_TIMEOUT`, it aborts the HSSP operation.

To calculate this timeout value, see the explanation of the definition `SPC_POLLING_TIMEOUT` in the file *TimeoutCalc.h* in "`Hssp_TimeoutCalc`" project.

3.3.4 Timeout parameter name: XRES_PULSE_100US

To reset the target device, the host has to generate an active low signal with a minimum pulse width of 1 µs on XRES line. In the HSSP code, a pulse width of 100 µs is generated on the XRES pin. This is accomplished by using the delay routine `DelayHundredUs()` between the XRES toggling events as shown below.

Code 2. Generating a Reset Pulse

```
XRES_OUTPUT_LOW;
DelayHundredUs();

XRES_OUTPUT_HIGH;
```

The function `DelayHundredUs()` is defined in the file *Timeout.c*. It uses the `XRES_PULSE_100US` timeout parameter in a for loop to introduce the 100 µs delay as given below.

Code 3. Delay Routine

```
void DelayHundredUs(void)
{
    unsigned short timestamp;


    for(timestamp = 0; timestamp <
    XRES_PULSE_100US; timestamp++)
    {

    }
}
```

To calculate this XRES_PULSE_100US timeout value, see the explanation of the definition `XRES_PULSE_100US` in the file *TimeoutCalc.h* in the "`Hssp_TimeoutCalc`" project.

3.4 Interface for Receiving HSSP Programming Data

The files *DataFetch.c*, *DataFetch.h* have the functions to fetch the data to be programmed into the target device. In the example project, the programming data is stored in the on-chip flash memory of the PSoC 5LP host programmer in files *HexImage.c*, *HexImage.h*. The data fetch routines access this data from the PSoC 5LP flash memory to do HSSP. But not all host programmers have the on-chip memory to store the HSSP programming data. Instead, the programmer might use a communication interface, such as an SPI or an UART, to get the programming data. For such a use case, all of the function definitions in *DataFetch.c* file should be modified appropriately. The following example is a reference that shows the modifications required for the `LoadFlashRowData(...)` function. Similar modifications should be done for other functions as well.

Original Code

Code 4. Original Code to Get Flash Data

```
void LoadFlashRowData(unsigned short rowNumber, unsigned short rowByteSize, unsigned
char *rowData)
{
    unsigned short i;

    for(i = 0; i < rowByteSize; i++)
    {
        /* Get the constant array data from
        HexImage.c and store in
        Flash row buffer */
        rowData[i] =
           flashData_hexFile[RowNumber][i];
    }
}
```

Modified Code

If the programming data is received through a communication interface such as an UART, then the modified code should be similar to the following one.

Code 5. Modified Code to Get Flash Data

```
void LoadFlashRowData(unsigned short rowNumber, unsigned short rowByteSize, unsigned
char *rowData)
{
    unsigned short i;

    /* ADD WAITING CODE HERE FOR THE UART
    BUFFER TO GET THE FLASH DATA */

    for(i = 0; i < rowByteSize; i++)
    {

        rowData[i] = /* PLACE THE UART
        BUFFER ARRAY HERE */;
    }
}
```

3.5 HSSP Timing Validation

The host programmer must meet the timing specifications for the PSoC 3 or PSoC 5LP programming for a robust HSSP implementation. Those timing specifications are given in the section “Programming Specifications” in the PSoC 3 Programming Specifications document, PSoC 5LP Programming Specifications document. The host programmer must meet the timing parameters specified for the SWD interface, programming mode entry. This timing validation can be done by capturing the SWDIO, SWDCK, and XRES signals on an oscilloscope. From the captured waveforms, the timing parameters can be verified against the corresponding values provided in the programming specification.

4 Tips and Tricks for Debugging HSSP Issues

Porting the HSSP code from the PSoC 5LP host processor used in the code example to your own processor architecture might be a complex task depending on the other system level constraints on the host processor side. This section helps you in troubleshooting the most commonly encountered issues while developing an HSSP application for your hardware platform.

- **Hardware Setup Validation**: The first step is to ensure that the hardware connections are done properly for the HSSP operation. This includes making the correct pin connections between the host processor and the target PSoC device, powering of all the PSoC voltage domains, and ensuring the host SWD pins drive mode settings are configured appropriately. Refer to the "Physical Layer" section of the respective device programming specification for details on the hardware connections and configuration.
- **Power Cycle Mode Programming**: The HSSP projects provided with this application note use the XRES pin to generate the reset event. The same reset event can also be triggered by toggling the power to the device which is referred to as power cycle mode programming. Power cycle mode programming is more complex to implement compared to XRES mode programming. Review the section "SWD Programming using Power Cycle Mode" in the respective programming specification document for design considerations in power cycle mode.
- **Timing Validation**: When porting the host PSoC 5LP code to your host processor, ensure that the timeout parameters used in the code are modified to reflect the host processor code timing. Refer to the section - Calculating HSSP Timeout Parameters – for information on modifying the timeout parameters while porting the code. The first step of the HSSP "Device Acquire" has strict timing requirements with regards to entering the programming mode. One of the important requirements is to ensure that the frequency of SWDCK clock line is at least 1.4 MHz to meet the acquire window timing. Ensure that there are no interrupt events in the host processor, which can affect the code execution time for completing the "Device Acquire" step on the host processor side. Ensure that the host processor is able to meet all the timing requirements explained in "Step 1 – Acquire Chip" of the respective device programming specification document.
- **HSSP Algorithm Validation**:
  - While porting the HSSP code, if any changes were made to the SWD packet layer files shown in Figure 1, ensure that the SWD packet format is the same as that mentioned in the section "Serial Wire Debug (SWD) Format" in the device programming specification.
  - Cypress qualified programmers like MiniProg3, KitProg can be used to validate and debug the steps like "Erase Flash", "Program Flash" in Figure 1. For example, to check if the host processor erased the entire flash memory, the MiniProg3 programmer and the "Read" option in the PSoC Programmer GUI can be used to verify that the entire flash data is zero. The "Checksum" option in PSoC Programmer GUI can be used to ensure that the checksum of the flash data programmed in to the device matches the checksum of the hex file that is fed as the input file to the GUI. Additionally, the "Patch Image" option in the PSoC Programmer GUI can be used to identify the number of flash rows for which there is a data mismatch between the device flash content and the hex file data.

5 Summary

The HSSP application is extremely useful for developing in-system programming solutions for the PSoC 3 and PSoC 5LP devices. It provides a cheap, robust method for programming the PSoC 3 or PSoC 5LP using an on-board embedded microcontroller as a host programmer. The portable, modular C code provided with this application note greatly reduces the time to develop such HSSP applications.

6 Related Documentation

6.1 Application Notes

AN84858 – PSoC 4 Programming Using an External Microcontroller (HSSP)

AN44168 – PSoC® 1 Device Programming using External Microcontroller (HSSP)

6.2 Programming Specifications

PSoC 3 Device Programming Specification

PSoC 5LP Device Programming Specification

6.3 PSoC Architecture Technical Reference Manuals

PSoC 3 Architecture Technical Reference Manual (TRM)

PSoC 5LP Architecture Technical Reference Manual (TRM)

6.4 Webpage

General PSoC Programming

6.5 List of Attached Projects

AN73054.cywrk:

This workspace contains three projects to demonstrate the HSSP application.

- *A_PSoC3_Hssp_Programmer*: This project is the PSoC 3 HSSP application. It is a PSoC 5LP based project in which a PSoC 5LP device acts as a host programmer to program the target PSoC 3 device.
- *B_PSoC5LP_Hssp_Programmer*: This project is the PSoC 5LP HSSP application. It is a PSoC 5LP based project in which a PSoC 5LP device acts as a host programmer to program the target PSoC 5LP device.
- *C_Hssp_TimeoutCalc*: This project is used to calculate the time stamp parameters used in the PSoC 3 HSSP, PSoC 5LP HSSP projects.

A Appendix A

A.1 Hex File Parser Application

The data for programming the PSoC 3 or PSoC 5LP is available in the hex file (*.hex*) format, which is explained in the section "Appendix A.1 - Intel Hex File Format" in the PSoC 3 Programming Specifications document, PSoC 5LP Programming Specifications document. The hex file is not in a format that can be used by a host to program the target device. This file, which is generated by the PSoC Creator software, contains both the programming data and the meta data in hexadecimal format. The meta data includes information on the hex file record type, extended linear address data, and so on. This meta data is used to categorize the programming data in to the flash code region data, flash configuration region data, flash protection data, and so on. A C# application has been developed in the Visual Studio development environment that parses the hex file, and generates *.c*, *.h* (`HexImage.c, HexImage.h`) files that store only the programming data from the hex files. The programming data is stored as an array of constants in the `HexImage.c, HexImage.h` files.

The C# application with the source code is provided along with the application note. To use the C# application, .NET framework version 4.0 or higher must be installed in your computer.

C# Application Name: HexFileParser_Psoc3_5lp

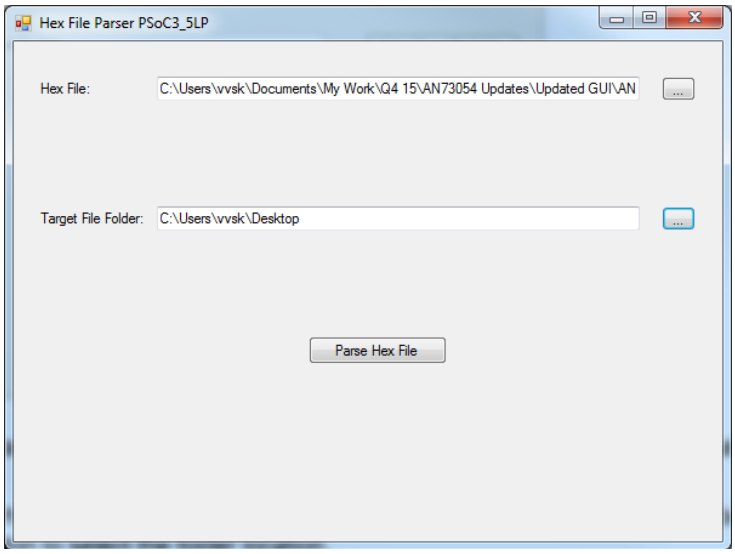
Development Environment: Microsoft Visual C# 2010 Express

Source code: See the C# source project for details. The project source code can be viewed, edited by downloading and installing the freely available Microsoft Visual C# 2010 Express software. . The source code can be viewed and edited by right clicking on the *Form1.cs* file in the *Solution Explorer* window, and then selecting on the *View Code* option. This action will open the source code of *Form1.cs*.

A.1.1 Using the Hex File Parser Application

Open the executable file of Hex File Parser application in the folder "*HexFileParser_Psoc3_5lp \HexFileParser_Psoc3_5lp \bin\Release*". A graphical user interface (GUI) screen, as shown in Figure 3 will pop-up.

Figure 3. Hex File Parser Application



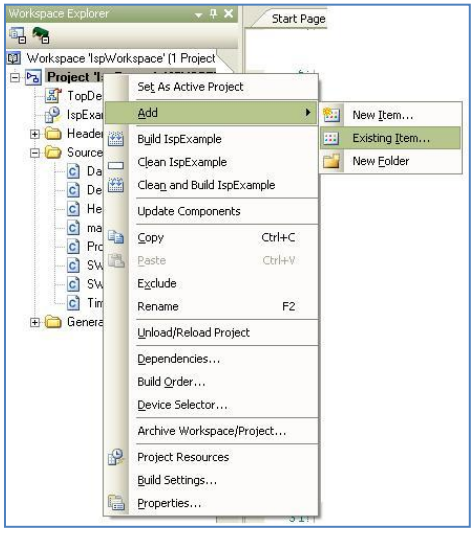
- Select the hex File that needs to be programmed.
- Select the target file folder location in which the parsed *.c*, *.h* files (`HexImage.h, HexImage.c`) should be created.
- After selecting the hex file and the target folder location, click the **Parse Hex File** button to generate the *.c*, *.h* files. A message will be displayed once the parsing operation has been completed.

A.1.2 Adding the Generated *HexImage.c, HexImage.h* files to PSoC Creator Example Project

To add the generated *HexImage.c, HexImage.h* files to the PSoC Creator example project provided with this application note, follow the below steps. These steps are required if you need to program a hex file into target device using PSoC 5LP as a host programmer. The below steps are applicable either for the project *A_PSoC3_Hssp_Programmer* in case of PSoC 3 HSSP or the project *B_PSoC5LP_Hssp_Programmer* in case of PSoC 5LP HSSP.

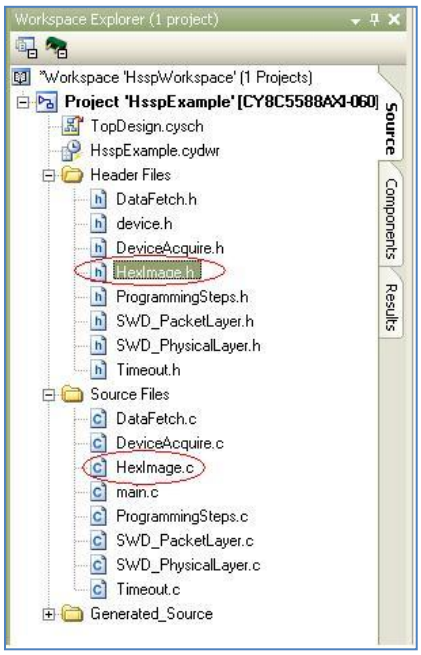
1. **Select the Target File Folder** location in GUI in Figure 3 as the same folder in which *main.c* file of the project is located.
2. After the *HexImage.c, HexImage.h* files have been generated in the above folder location, add those files to the project workspace in PSoC Creator by using **Add Existing Item** option in Workspace explorer as shown in Figure 4.

Figure 4. Adding *HexImage.c, HexImage.c* Files to PSoC Creator Project



3. Once the *HexImage.c, HexImage.h* files are selected, the Workspace Explorer window of the project should appear as shown in the following figure.

Figure 5. Project Workspace Showing *HexImage.c, HexImage.h* Files



3. **Important note on using the *HexImage.c*, *HexImage.h* files for the PSoC 5LP HSSP project:** The PSoC 5LP HSSP project (*B_PSoC5LP_Hssp_Programmer*) uses a PSoC 5LP device as a host programmer to program the target PSoC 5LP device. Since the HSSP algorithm in the host PSoC 5LP will consume a portion of its 256 KB Flash memory, the entire hex file data cannot be stored in the flash memory of host PSoC 5LP. In order to meet the code size requirements, the programming data corresponding to the last few flash rows is deleted from the *HexImage.c*, *HexImage.h* files. The corresponding logic has also been incorporated in the `LoadFlashRowData(…)` function in the file *DataFetch.c*. The changes done are listed below. Refer to the *B_PSoC5LP_Hssp_Programmer* project code for viewing the exact changes.

   a. In the *HexImage.h* file, the number of flash rows in the array declaration is reduced by `NUMBER_OF_FLASH_ROWS_REMOVED` by editing the array declaration as below.

      ```
      extern unsigned char const FlashData_HexFile[(NUMBER_OF_FLASH_ROWS_HEX_FILE - NUMBER_OF_FLASH_ROWS_REMOVED)][FLASH_ROW_BYTE_SIZE_HEX_FILE];
      ```

      The macro define, `NUMBER_OF_FLASH_ROWS_REMOVED`, for specifying the number of rows to be removed is manually added in HexImage.h file, and assigned a value 224 in the project. The choice of 224 rows was arrived so that the code size of the project fits the host PSoC 5LP flash memory capacity.

   b. In the *HexImage.c* file, the last `NUMBER_OF_FLASH_ROWS_REMOVED` flash rows are deleted from the definition of the array `FlashData_HexFile[][]` corresponding to the changes done in the array declaration in the *HexImage.h* file.

   c. In the definition of the function `LoadFlashRowData(…)` in the file `DataFetch.c`, the entire flash row data is loaded with zero in the case of the flash row number being in the last `NUMBER_OF_FLASH_ROWS_REMOVED` rows of the target PSoC 5LP device.

B Appendix B

B.1 HSSP Functions

The following tables list the public functions in each layer of the HSSP firmware architecture. These functions are used to communicate among the layers of the HSSP firmware, as shown in Figure 1.

Table 2. Functions in *Swd_PhysicalLayer.h*

Function	Description
`SetSwdckHigh()`	Sets the host SWDCK pin high
`SetSwdioHigh()`	Sets the host SWDIO pin high
`SetXresHigh()`	Sets the host XRES pin high
`SetSwdckLow()`	Sets the host SWDCK pin low
`SetSwdioLow()`	Sets the host SWDIO pin low
`SetXresLow()`	Sets the host SWDCK pin low
`SetSwdckCmosOutput()`	Configures the host SWDCK pin for CMOS output drive mode
`SetSwdioCmosOutput()`	Configures the host SWDIO pin for CMOS output drive mode
`SetXresCmosOutput()`	Configures the host XRES pin for CMOS output drive mode
`SetSwdckHizInput()`	Configures the host SWDCK pin for high-impedance digital input drive mode
`SetSwdioHizInput()`	Configures the host SWDIO pin for high-impedance digital input drive mode
`SetXresHizInput()`	Configures the host XRES pin for high-impedance digital input drive mode
`ReadSwdio()`	Returns the current state of the SWDIO input pin

Table 3. Functions in *Swd_PacketLayer.h*

Function	Description
`Swd_WritePacket()`	Sends a SWD write packet. This function operates on the global variables Swd_packetHeader, Swd_packetAck, and Swd_packetData[].
`Swd_WritePacketFast()`	Sends a SWD write packet. This function operates on the global variables Swd_packetHeader, Swd_packetAck, and Swd_packetData[]. This function expects the precalculated even parity bit of Swd_packetData[]as a parameter. This function has been optimized for execution time to meet the timing requirements for entering the target device programming mode.
`Swd_RawReadPacket()`	Sends a single SWD read packet. This function operates on the global variables Swd_packetHeader, Swd_packetAck, and Swd_packetData[]. This function is used for doing a continuous read from a specific address.
`Swd_ReadPacket()`	Sends a SWD read packet twice to read from a specific address. This function operates on the global variables Swd_packetHeader, Swd_packetAck, and Swd_packetData[].
`JtagToSwdSequence()` *(Only for PSoC 5LP HSSP)*	Sends the initial packets of the JTAG to SWD switching sequence. This includes all the steps involved in the switching except reading of the Device JTAG ID. The complete switching is implemented in the private function SwitchToSwd() in *ProgrammingSteps.c*. This function is present only for the PSoC 5LP HSSP. PSoC 3 HSSP does not require this switching sequence in its implementation.

Table 4. Functions in *Timeout.h*

Function	Description
`DelayHundredUs()`	Introduces a delay of 100 µs used for generating an active low pulse signal of duration 100 µs on the XRES pin.

Table 5. Functions in *DeviceAcquire.h*

Function	Description
`AcquireTargetDevice()`	Sends the sequence to enter the programming mode of the target device. The function has been optimized for execution time to meet the timing requirements for entering the PSoC 3 or PSoC 5LP programming mode.
`ReleaseTargetDevice()`	Exits the PSoC 3 or PSoC 5LP programming mode by generating a pulse on the XRES pin

Table 6. Functions in *DataFetch.h*

Function	Description
`LoadDeviceSiliconId()`	Copies the device silicon ID data from the *HexImage.c* file to an indicated destination array
`LoadDeviceConfigNvl()`	Copies the device configuration NVL data from the *HexImage.c* file to an indicated destination array.
`LoadWriteOnceNvl()`	Copies the write-once NVL data from the *HexImage.c* file to an indicated destination array
`LoadFlashRowData()`	Copies the flash row data from the *HexImage.c* file to an indicated destination array. The flash row number and the flash row byte size are also passed as parameters to this function.
`LoadFlashProtectionData()`	Copies the flash row protection data from the *HexImage.c* file to an indicated destination array. The byte size of the protection data is also passed as a parameter to this function.
`LoadChecksumData()`	Copies the checksum data from the *HexImage.c* file to an indicated destination array
`LoadEepromRowData()`	Copies the EEPROM row data from the *HexImage.c* file to an indicated destination array. The EEPROM row number is also passed as a parameter to this function.
`GetFlashRowCount()`	Returns the total number of flash rows in the target device from the *HexImage.c* file
`GetFlashRowByteSize()`	Returns the size in bytes of the flash row from the *HexImage.c* file
`GetFlashArrayCount()` *(Only for PSoC 5LP HSSP)*	Returns the number of flash arrays from the *HexImage.c* file. This function is required only for PSoC 5LP HSSP which can have multiple flash arrays. PSoC 3 has only one flash array and this function is not present in PSoC 3 HSSP.
`GetEepromRowCount()`	Returns the total number of EEPROM rows in the target device from the *HexImage.c* file.
`GetEepromSectorCount ()`	Returns the total number of EEPROM sector rows in the target device from the *HexImage.c* file.

Table 7. Functions in *ProgrammingSteps.h*

Function	Description
`EnterProgrammingMode()`	Enters the programming mode of the target device
`ConfigureTargetDevice()`	Configures the target device for programming
`VerifyDeviceId()`	Verifies whether the device silicon IDs of the target device and the hex file match
`EraseFlash()`	Erases the entire flash memory of the target device including the flash protection data
`ProgramDeviceConfigNvl()`	Programs the device configuration NVL of the target device
`ProgramFlash()`	Programs the flash memory of the target device
`VerifyFlash()`	Verifies whether the flash data programmed to the target device and the hex file flash data match
`ProgramWriteOnceNvl()`	Programs the write-once NVL of the target device.
`ProgramFlashProtection()`	Programs the flash protection data to the target device
`VerifyFlashProtection()`	Verifies whether the flash protection data programmed to the target device and the hex file flash protection data match
`VerifyChecksum()`	Verifies whether the checksum data read from the target device and the hex file checksum data match
`ProgramEeprom()`	Programs the EEPROM memory of the target device. If there is no EEPROM data present in the hex file, this API is not called in the main code.
`VerifyEeprom()`	Verifies whether the EEPROM data programmed to the target device and the hex file EEPROM data match. If there is no EEPROM data present in the hex file, this API is not called in the main code.
`ExitProgrammingMode()`	Exits the target device programming mode by generating an active low pulse signal on the XRES pin
`ReadHsspErrorStatus()`	Returns the error status of the HSSP operation
`ReadSpcStatusReg()`	Returns the SPC status register value in case of a SPC polling timeout error condition

C Appendix C

C.1 Differences between PSoC 3 HSSP and PSoC 5LP HSSP

The application note provides two separate projects – *A_PSoC3_Hssp_Programmer* and *B_PSoC5LP_Hssp_Programmer* – for PSoC 3 HSSP and PSoC 5LP HSSP respectively. From a higher level implementation standpoint, both the HSSP projects have very little differences. But there are differences in the internal implementation of the functions in PSoC 3 HSSP and PSoC 5LP HSSP. The differences are explained in this section for the knowledge of the developer migrating between PSoC 3 HSSP and PSoC 5LP HSSP. There is no need to implement these differences as two separate HSSP projects are provided for PSoC 3 HSSP and PSoC 5LP HSSP respectively. The differences are as follows:

- The internal implementation of all the top level HSSP functions (*HSSP Implementation Flow* in Figure 1) is different between PSoC 3 HSSP and PSoC 5LP HSSP. For example, the register addresses in PSoC 5LP have the upper 16-bit offset address as 0x4000, while for PSoC 3 it is 0x0000. These differences stem from the differences in the SWD vectors provided in the PSoC 3 programming specifications and PSoC 5LP programming specifications respectively.
- The SPC polling function definitions (`IsSpcIdle()` and `IsSpcDataReady()`) in the file *ProgrammingSteps.c* differ between PSoC 3 and PSoC 5LP. The SPC status register is the least significant byte in case of PSoC 3 and the third least significant byte in case of PSoC 5LP.
- PSoC 5LP HSSP has a function `GetFlashArrayCount()` in the file *DataFetch.h* to get the number of Flash arrays in the target PSoC 5LP device. This is not present in PSoC 3 HSSP because there is only one Flash array in all PSoC 3 device families.

Document History

Document Title: PSoC® 3 and PSoC 5LP Programming Using an External Microcontroller (HSSP) – AN73054

Document Number: 001-73054

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3466083	VVSK	12/19/2011	New Application Note
*A	3478580	VVSK	12/30/2011	Title change to add PSoC 5 support Content change to add PSoC 5 support
*B	3814249	VVSK	16/11/2012	Updated for PSoC 5LP.
*C	5046300	VVSK	12/16/2015	Updated the projects to PSoC Creator 3.1 CP1. Added support for EEPROM programming and verification Added fix in PSoC 5LP HSSP project to configure XRES event similar to power cycle event Updated template

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com