AN84858

PSoC® 4 Programming Using an External Microcontroller (HSSP)

Author: Tushar Rastogi
Associated Part Family: All PSoC 4 Parts
Related Application Notes: AN73054, AN44168
To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/AN84858.

AN84858 shows you how to implement PSoC® 4 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 4 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 4.

Contents

1 Introduction

PSoC 4 device programming refers specifically to the programming of the nonvolatile memory in PSoC 4 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 4 device. For more information on the PSoC 4 architecture and to learn how to create projects for PSoC 4 using the PSoC Creator™ software, refer to AN79953 - Getting Started with PSoC® 4.

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 for a PSoC 4 device. The project provided with this application note uses a PSoC 5LP device as a host programmer to program the target PSoC 4 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 4 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-M0 CPU in PSoC 4. It 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 Cortex-M0 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 PSoC 4 target 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 4.
- 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. PSoC 4 supports only SWD interfaces.

2 HSSP Firmware Architecture

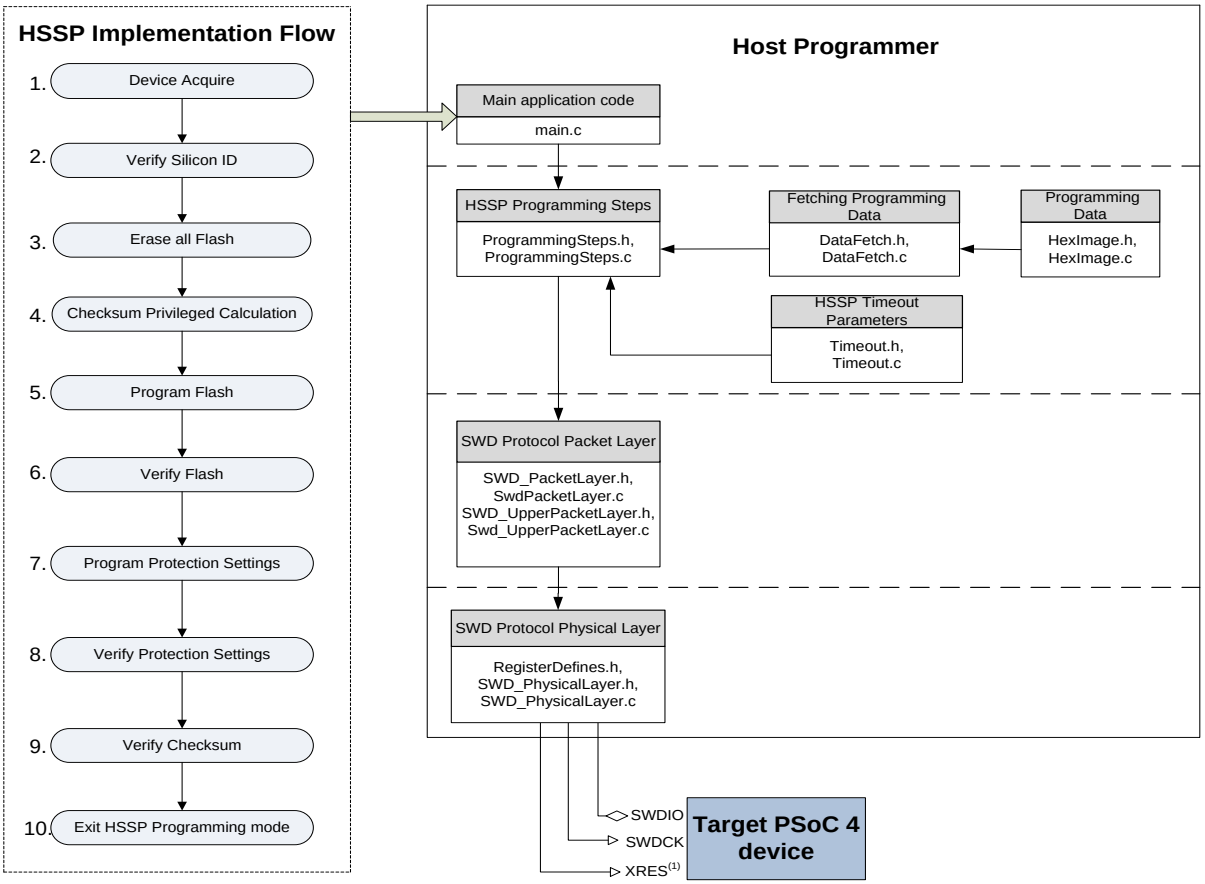
HSSP for PSoC 4 is implemented in multiple layers using modular C code. These layers are as follows:

1. SWD Protocol Physical layer
2. SWD Protocol Packet layer
3. HSSP Programming Steps layer

See Figure 1 for the flow of control among these layers.

Refer to the A_Hssp_Programmer project, which uses PSoC 5LP as the external host programmer, attached with this application note for the implementation of this firmware.

Figure 1. PSoC 4 HSSP Firmware Architecture



(1) For power cycle mode programming, device power rails need to be toggled instead of the reset (XRES) pin

The HSSP Programming Steps layer uses the “Fetching Programming Data” interface to extract the programming data from the source of the data (for example, the hex file data provided by any communication interface—I^2C, SPI, UART, or USB or host flash—as Figure 1 shows. In addition, the “HSSP Programming Step” layer uses the HSSP Timeout Parameters interface to configure timeouts in its programming APIs.

All of the layers used in the firmware architecture, as shown in Figure 1, are discussed in the following sections.

2.1 SWD Protocol Physical Layer

Files that constitute the SWD Protocol Physical layer are described in Table 1.

Table 1. SWD Protocol Physical Layer Files

Source Files	Description
RegisterDefines (*.h* file)	This file defines the port number, pin number, input/output register, and drive mode register of the programming pins.
SWD_PhysicalLayer (*.c* and *.h* files)	These files contain macros and functions to manipulate the programming pins. The pins are defined in the *RegisterDefines.h* file.

The codes in these files are written for PSoC 5LP as the host microcontroller. If these files are ported to any other host microcontroller, then you should modify all the functions and macros appropriately.

Note: Refer to “Pin Names and Requirements” in the programming specifications of the respective device listed in the Related Documentation section for details on the pin configurations on the host side.

2.2 SWD Protocol Packet Layer

Files that constitute the SWD Protocol Packet layer are described in Table 2.

Table 2. SWD Protocol Packet Layer Files

Source Files	Description
SWD_PacketLayer (*.c* and *.h* files)	These files define the packet routines for sending the SWD Read and SWD Write packets per the SWD protocol.
SWD_UpperPacketLayer (*.c* and *.h* files)	These files use the functions defined in SWD_PacketLayer to implement functions that directly read and write to the DAP register and CPU address space.

The functions defined in this layer are called directly by the functions in the *ProgrammingSteps.c* file.

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 Figure 1 shows.

2.3 Fetching Programming Data

Files used for fetching the programming data to the upper layer functions are described in Table 3.

Table 3. Data Fetch Layer File

Source File	Description
DataFetch (*.c* and *.h* files)	These files contain the routines to fetch the programming data from the *HexImage.c* file and then pass that data to the functions in the *ProgrammingSteps.c* file. The programming data includes flash row data, flash protection data, chip protection data, Silicon ID, checksum, and the total number of flash rows.

Modify the definitions of the functions based on the method that you use to get the HSSP programming data.

Note: Refer to the section “Interface for Receiving HSSP Programming Data” for information on how to modify the HSSP source code.

2.4 HSSP Programming Steps

This layer includes the files named *ProgrammingSteps.c* and *ProgrammingSteps.h*. These files contain the top-level functions of the HSSP application. These functions are described in sequence as follows:

1. **Device Acquire**: In this step, the device is acquired by sending a specific sequence through the SWD interface after a device reset. As a result, the host programmer can control the Cortex-M0 CPU and other system resources, such as SRAM and registers. The PSoC 40xx and PSoC 4xx7_BLE family of devices requires the internal main oscillator (IMO) frequency to be set at 48 MHz before flash erase/write operations. This operation is also included in the Device Acquire routine for these devices.

   Note: Some of the devices in the PSoC 4000 family do not have a dedicated reset (XRES) pin, and have to be reset by toggling the device power rails. This is referred to as Power Cycle Mode programming. Refer to the Power Cycle Mode Programming section for details on the changes required for modifying the code to support power cycle mode programming.
2. **Verify Silicon ID**: This step verifies that the acquired device is the same as the one for which the hex file was generated.
3. **Erase All Flash**: This step erases all user rows and corresponding flash protection.
4. **Checksum Privileged Calculation**: After all the user rows are erased, this step calculates the checksum of the privileged rows, which is used to verify the checksum of the user rows in Step 9.
5. **Program Flash**: In this step, flash is programmed using the programming data in the hex file and SROM API calls.
6. **Verify Flash**: This step is used to verify the flash data programmed in the previous step with the data in the hex file. This step is optional but highly recommended.
7. **Program Protection Settings**: In this step, row protection settings and chip protection settings from the hex file are written to the specific flash area.
8. **Verify Protection Settings**: Both protection settings are matched with the settings in the hex file.
9. **Verify Checksum**: This step matches the checksum of the user data in the flash with the checksum in the hex file. It uses the checksum of the privileged rows calculated in step 4.
10. **Exit HSSP Programming Mode:** This step releases the target PSoC 4 device from programming mode.

Each of these steps is described by functions made up of basic SWD instructions. Refer to the programming specifications document of the respective device listed in the Related Documentation section for detailed information.

The functions declared in the *ProgrammingSteps.h* file access the functions, definitions, and global variables from three layers: SWD Protocol Packet layer, DataFetch layer, and Timeout layer. The functions provided in the *ProgrammingSteps.c* and *ProgrammingSteps.h* files cover all the steps required to program the target PSoC 4 device.

2.5 HSSP Timeout Parameters

Files that constitute the Timeout layer are described in Table 4.

Table 4. Timeout Layer Files

Source File	Description
Timeout (.c and .h files)	These files contain the timestamp definitions and the delay routines used in HSSP.

Timestamp definitions are derived from the electrical timing specifications provided in the PSoC 4 Device Programming Specifications. The values of these timestamp parameter definitions in *Timeout.h* are for a PSoC 5LP host programmer running at a clock frequency of 63 MHz.

Timestamp definitions and the delay routine are used in the function definitions in the *ProgrammingSteps.c* file.

To learn how to calculate the timestamp parameters for a specific host programmer, see the section Calculating HSSP Timeout Parameters.

2.6 HSSP Programming Data

Files that contain the programming data to be stored in the host programmer are described in Table 5.

Table 5. Files Containing the Programming Data

Source Files	Description
HexImage (*.c* and *.h* files)	These files contain the data to be programmed into the target device. They also store the target device parameters used in HSSP programming, such as silicon ID, checksum, total number of flash rows, and number of bytes per flash row.

The data in these files is stored in PSoC 5LP flash memory as an array of constants.

These files are generated by the C# application Hex File Parser, which is provided with this application note. This application generates these files by taking the PSoC 4 hex file as the input. The details of this application are provided in Appendix A.

For details on the hex file format, see "Appendix B. Intel Hex File Format" in the programming specifications document of the respective device listed in the Related Documentation section of this document.

For host programmers that lack the memory capacity to store the programming data in the on-chip memory, the *HexImage.c* and *HexImage.h* files are not required. In such cases, the HSSP programming data is typically sent to the host as packets through a communication interface, such as I^2C, UART, or USB.

Note: Refer to 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.

2.7 Main Application Code

The *main.c* file is the main application code that calls the top-level HSSP programming steps in the sequence shown in Figure 1. The function `ProgramDevice()` in *main.c* executes all the steps. Each step must be executed successfully before you can proceed to the next step.

The HSSP operation is aborted if a FAILURE is returned after any step. The error status returned by the `ReadHsspErrorStatus()` function is used to identify the cause of the error. The status of the HSSP operation, along with the error status register, is displayed on a character LCD in the attached HSSP project.

Note: The character LCD and `Pin_Start` routines are specific to the PSoC 5LP host programmer and, therefore, should be modified as required for any other host programmer.

2.8 HSSP Error Status

When any of the top-level steps in the HSSP application returns 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. Use the status byte to infer error details from the bit field definitions. See Figure 2 for the bit fields returned by this function.

Figure 2. HSSP Error Status Register

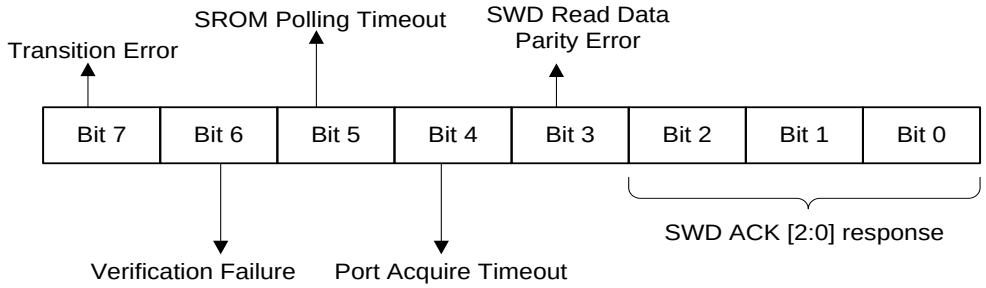


Table 6 describes the bit field definitions of this status register.

Table 6. HSSP Error Status Register

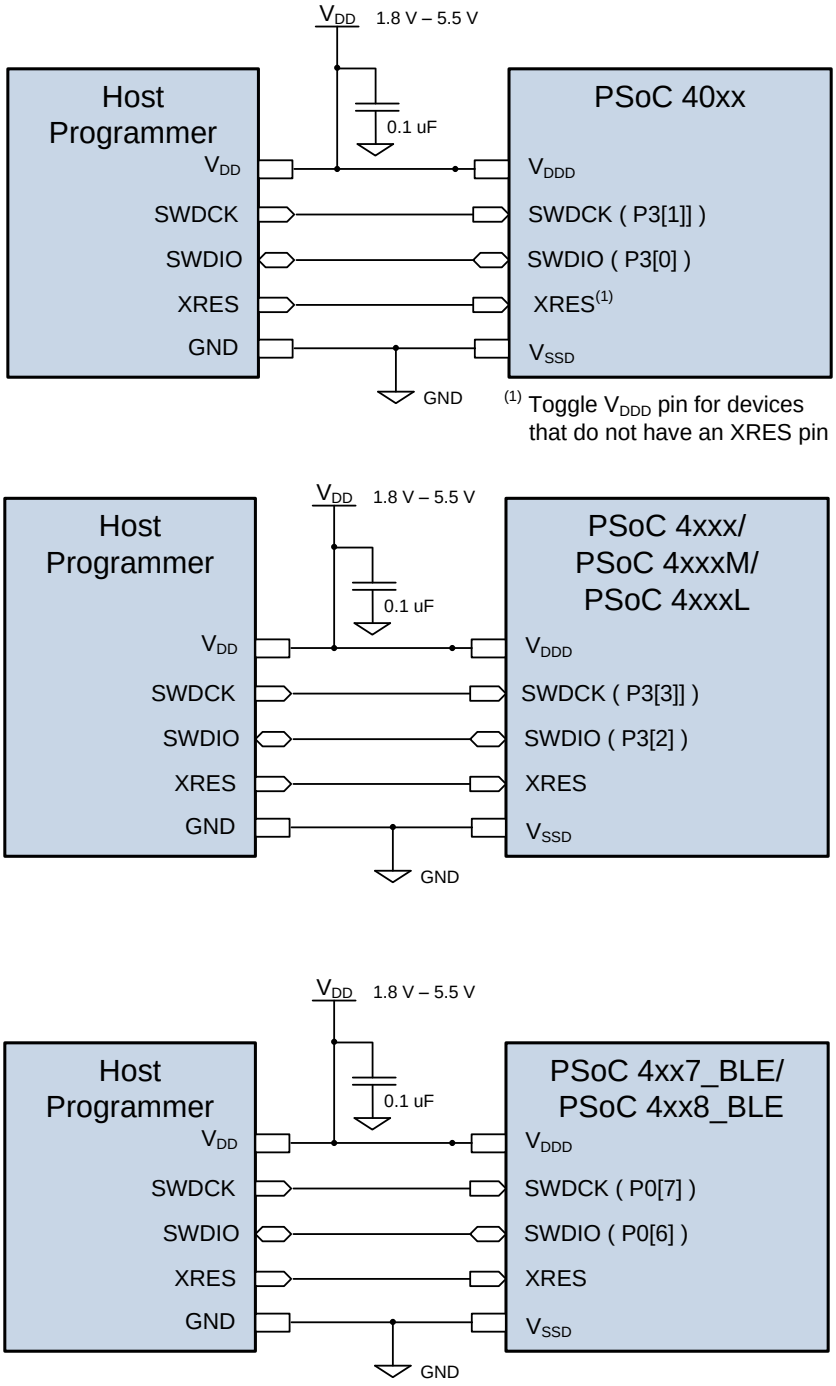
Bit	Field Name	Description
[2:0]	SWD ACK	These three bits store the acknowledge response of previous SWD transactions.
3	SWD Read Data Parity Error	If this bit is set, it indicates a parity error in the data received by the host.
4	Port Acquire Timeout	If this bit is set, it indicates that the device was not acquired within the timeout window.
5	SROM Polling Timeout	If this bit is set, It indicates that SROM operations exceed 1 second.
6	Verification Failure	This bit is set in multiple steps. Depending upon the step where failure occurred, the reason can be inferred.
7.	Transition Error	If this bit is set, it indicates that the chip protection settings read from the chip and the hex file indicates a wrong transition.

To learn more about this register and for a detailed explanation of the bit fields, see Appendix C: Bit Field Definitions of HSSP Error Status Register.

3 Hardware Connections for PSoC 4 HSSP Programming

HSSP programming in PSoC 4 uses the SWD interface pins (SWDIO and SWDCK) and the external reset pin (XRES).The host programmer pin drive mode requirements are explained in the “Physical Layer” section in the programming specifications document listed in the Related Documentation section of this document. Figure 3 shows the basic hardware connection required between the host programmer and the target PSoC 4 device.

Figure 3. Basic Host/Target Connections



4 Porting the HSSP Application to a Host Programmer

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

4.1 Files That Must Be Ported

The PSoC 5LP host programmer-based project includes files specific to PSoC 5LP. While porting the HSSP application code to any other host programmer, you must port only the files listed in Table 7.

Table 7. Files That Must Be Ported

Header Files to Be Ported	Source Files to Be Ported
SWD_PhysicalLayer.h	*SWD_PhysicalLayer.c*
SWD_PacketLayer.h	*SWD_PacketLayer.c*
SWD_UpperPacketLayer.h	*SWD_UpperPacketLayer.c*
Timeout.h	*Timeout.c*
HexImage.h	*HexImage.c*
DataFetch.h	*DataFetch.c*
ProgrammingSteps.h	*ProgrammingSteps.c*
RegisterDefines.h	

4.2 Code Changes Required While Porting

Make the following changes to each of the files while porting the attached HSSP application code to any host programmer other than PSoC 5LP. Note that only some files need to be modified while porting.

1. ***RegisterDefines.h***: Modify the definitions in this header file for the port numbers, pin numbers, mask values, output registers, input registers, and drive mode registers according to the host programmer used.
2. ***SWD_PhysicalLayer.h***: All the bit-banging macros defined in this header file are used in the function definitions in the *SWD_PhysicalLayer.c* file. Modify these macros according to the host programmer used.
3. ***SWD_PhysicalLayer.c***: Modify all the bit-banging functions defined in this file according to the host programmer used.
4. ***Timeout.h***: Modify the three timeout parameter definitions described in Table 8 according to the host programmer used.

Table 8. Timing Parameters

S.No.	Timing Parameter
1	XRES_PULSE_100US
2	DEVICE_ACQUIRE_TIMEOUT
3	SROM_POLLING_TIMEOUT

To learn how to calculate the timeout parameters for a specific host programmer, see Calculating HSSP Timeout Parameters.

5. ***HexImage.c***, ***HexImage.h***: These files contain the data to be programmed into the target device, defined as an array of constants. For the PSoC 5LP host programmer, the data to be programmed is stored in the flash memory of the host PSoC 5LP.

Some host programmers may 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, remove these files.

6. ***DataFetch.c***: The definitions for the functions should be modified based on the method used to get the programming data.

   See Interface for Receiving HSSP Programming Data for information on modifying the HSSP source code according to the method used to get the programming data.

7. ***main.c***: The APIs for the character LCD and Pin_Start pin in the *main.c* file are specific to a PSoC 5LP host programmer. Therefore, you should either remove them or modify them while porting to any other host programmer.

In all of the above specified files, the following comments precede the code that needs to be modified based on the host programmer. This helps you identify the code to be modified.

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

5 Calculating HSSP Timeout Parameters

Modify the values of the timeout parameters defined in the *Timeout.h* file according to the host programmer used.

A separate test project, "C_Hssp_TimeoutCalc," is provided with the application note. This project illustrates the procedure to calculate the timeout parameters for a PSoC 5LP host programmer. Create a similar test project for any other host programmer to calculate those timeout values.

The test project provides test functions in two files: *TimeoutCalc.h*and*TimeoutCalc.c*. These test functions toggle a test pin during code execution. The timeout parameters are measured by measuring the LOW pulse width of the signal on the test pin using an oscilloscope.

To calculate these timeout parameters, see the explanation in the macro definitions in the *TimeoutCalc.h* header file of the project. Now look at the significance of each of these timeout values:

5.1 DEVICE_ACQUIRE_TIMEOUT

This macro is used in the function `DeviceAcquire()` in the *ProgrammingSteps.c* file. To program the device, the device must be acquired within X ms after you do a device reset using the XRES pin, where X is the maximum time window for acquiring the device as defined in Table 9.

Table 9. Timeout for Acquiring Device

Device Family	Timeout (ms) [1]
PSoC40xx	2.0
PSoC41/42xx	1.5
PSoC4xxxM	2.0
PSoC 4xxxL	2.0
PSoC4xx7_BLE / PSoC4xx8_BLE	2.0

[1]The timeout for acquiring the device in the case of Power Cycle mode programming must be longer, ~ 30 ms.

Note: The recommended minimum frequency of the SWDCK clock, which meets the timing requirement to acquire the device is 1.5 MHz. See the "Step 1. Acquire Chip" subsection of the programming specifications document of the respective device listed in the Related Documentation section for more details.

The device-acquiring sequence consists of two steps:

1. Do a line Reset, which is a standard ARM command to reset the debug access port (DAP).
2. Read the DAP.

DEVICE_ACQUIRE_TIMEOUT indicates the maximum number of times a host can send the device-acquiring sequence in a specific time window after device reset as given in Table 9.

To calculate this macro, uncomment the `TestModeTimeout()` function in the *main.c* file of the project, and then program the device.

Measure the LOW pulse width on the test pin for one iteration of the device-acquiring sequence by using an oscilloscope. Then, calculate this macro as follows:

DEVICE_ACQUIRE_TIMEOUT = (X ms/low pulse width), where X is the maximum time window for acquiring the device as defined in Table 9.

Code 1: Calculating DEVICE_ACQUIRE_TIMEOUT Parameter

```
Unsignedshort timestamp = 0;
Unsignedlongchip_DAP_Id = 0;

/* Make the pin LOW before sending SWD clock train */
TESTPIN_OUTPUT_LOW;

for(timestamp = 0; timestamp < 1; timestamp++)
{
    Swd_LineReset();
    Read_DAP(DPACC_DP_IDCODE_READ, &chip_DAP_Id);
}
/* Make the pin HIGH after sending SWD clock train */
TESTPIN_OUTPUT_HIGH;
```

5.2 SROM_POLLING_TIMEOUT

This macro is used in the SROM polling operations during HSSP programming. It is used while polling the result of the nonvolatile memory read and write operations through SROM requests in the target device.

When the host requests a SROM system call through the SWD interface, it waits a maximum of one second while reading the CPUSS_SYSREQ register for status. If a response is not received, the host aborts the HSSP operation.

SROM_POLLING_TIMEOUT indicates the maximum number of times the CPUSS_SYSREQ register can be read in a time window of one second.

To calculate this macro, uncomment the `TestSromPollingTimeout()` function in the *main.c* file of the project, and then program the device.

Measure the LOW pulse width on the test pin after sending 10 iterations of the SROM polling sequence by using an oscilloscope. Then calculate the macro as follows:

SROM_POLLING_TIMEOUT = (1 s/LOW pulse width)*10

Code 2: Calculating the SROM_POLLING_TIMEOUT Parameter

```
Unsignedshort timestamp = 0;
Unsignedlongstatuscode = 0;

/* Make the pin low before sending SWD clock train */
TESTPIN_OUTPUT_LOW;
```

```
for(timestamp = 0; timestamp < 10; timestamp++)
{
        Read_IO (CPUSS_SYSREQ, &statusCode);

        /*performing SROM_SYSREQ_BIT | SROM_PRIVILEGED_BIT */
        statusCode &= (SROM_SYSREQ_BIT | SROM_PRIVILEGED_BIT);
}

/* Make the pin high after sending SWD clock train */
        TESTPIN_OUTPUT_HIGH;
```

This macro signifies the number of times CPUSS_SYSREQ can be read and checked if SROM_SYSREQ_BIT and SROM_PRIVILEGED_BIT are set to 0 in a 1-second interval.

5.3 XRES_PULSE_100US

To reset the target device, the host generates an active LOW signal with a minimum pulse width of 5 µs on the XRES line. In the HSSP code, a pulse width of 100 µs is generated on the XRES pin.

The function `TestDelayHundredUs()` is defined in the *TimeoutCalc.c* file. It uses the XRES_PULSE_100US timeout parameter in a “for” loop to introduce the 100-µs delay, as given in Code 3.

To calculate this macro, uncomment the `TestDelayHundredUs()` function in the *main.c* file of the project, and then program the device. Measure the pulse width of the test pin on an oscilloscope.

Code 3: Delay Routine

```
unsignedshort timestamp;

/* Make the pin low before start of the delay */
TESTPIN_OUTPUT_LOW;

/* For loop to introduce the 100 us delay */
for(timestamp = 0; timestamp < XRES_PULSE_100US; timestamp++)
{
        /* Do Nothing */
}

/* Make the pin high after end of the delay */
TESTPIN_OUTPUT_HIGH;
```

For power cycle mode programming, this delay routine signifies the time for which the device power rails are OFF. For power cycle mode programming, there are no requirements for this pulse width. The recommendation is to have a 1-ms delay before turning ON the power rails.

6 Interface for Receiving HSSP Programming Data

The files *DataFetch.c* and *DataFetch.h* contain 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 the files *HexImage.c* and *HexImage.h*. The data fetch routines access this data from the PSoC 5LP flash memory to perform HSSP.

However, not all host programmers may have the on-chip memory to store the HSSP programming data. When that is the case, the programmer can use a communication interface (such as SPI, USB, or UART) to get the programming data. Also, all the function definitions in the *DataFetch.c* file should be modified appropriately.

The following example is a reference that shows the modifications required for the `Hex_ReadRowData()` function. You can perform similar modifications for other functions as well.

Original Code

Code 4: Original Code to Get Flash Data

```
void HEX_ReadRowData(unsignedshortrowCount, unsignedchar * rowData)
{
/* Maximum value of 'i' can be 256 */
unsignedshort i;

for(i = 0; i<BYTES_PER_FLASH_ROW; i++)
    {
       rowData[i] = FlashData_HexFile[rowCount][i];
    }
}
```

Modified Code

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

Code 5: Modified Code to Get Flash Data

```
void HEX_ReadRowData(unsignedshortrowCount, unsignedchar * rowData)
{
/* Maximum value of 'i' can be 256 */
unsignedshort i;

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

for(i = 0; i<BYTES_PER_FLASH_ROW; i++)
    {
       rowData[i] = /* PLACE THE UART BUFFER ARRAY HERE */
    }
}
```

7 HSSP Timing Validation

The host programmer must meet the timing specifications for PSoC 4 programming to achieve a robust HSSP implementation. Those specifications are given in “Appendix D. Timing Specifications of the SWD Interface” of the programming specifications document of the respective device listed in the Related Documentation section.

The host programmer must meet the timing parameters specified for the SWD interface and programming mode entry. To validate the timing, capture the SWDIO, SWDCK, and XRES signals on an oscilloscope. For power cycle mode programming, the device power rails should be monitored instead of the XRES pin. Using the captured waveforms, you can verify the timing parameters against the corresponding values provided in the programming specification.

8 Power Cycle Mode Programming

Device programming starts with a device reset to acquire the device and enter programming mode. The recommended method of resetting the device from the host side is to toggle the device XRES pin. But some lower pin count devices in the PSoC 4000 family do not have an XRES pin, and the host has to toggle the device V_{dd} pin to reset the device (power cycle mode). All the projects provided with the application note work using the XRES programming mode because the PSoC 5LP host processor on the development kit does not have the hardware connections to toggle the power supply of the target PSoC 4 device. So, when porting the projects to your host processor and adding power cycle mode support, the following changes and considerations need to be made in the project.

1. The DEVICE_ACQUIRE_TIMEOUT value, discussed in the section DEVICE_ACQUIRE_TIMEOUT, has to be modified to reflect the much longer 30-ms timeout window required for power cycle mode programming. So, for example, if the DEVICE_ACQUIRE_TIMEOUT define has been calculated to be 20 for a 2 ms XRES mode acquire timing, then that define should be changed to 300 for a 30 ms power cycle mode acquire timing.
2. The function definitions related to toggling of the XRES pin on the host side should be modified as appropriate to toggle the device power rails. These functions are - `SetXresHigh()`, `SetXresLow()`.
3. If an I/O pin of the host processor is used to power the PSoC 4 device directly to toggle the power from the host side, ensure that the I/O pin can source the current required for device operation as specified in the respective PSoC 4 device datasheet. If the I/O pin is not able to source the required current, the output voltage (V_{OH}) typically decreases, and can potentially go even below the minimum PSoC 4 device operating voltage. This can cause programming failures. Such a voltage droop scenario can be identified by observing the power rail voltages on the oscilloscope.

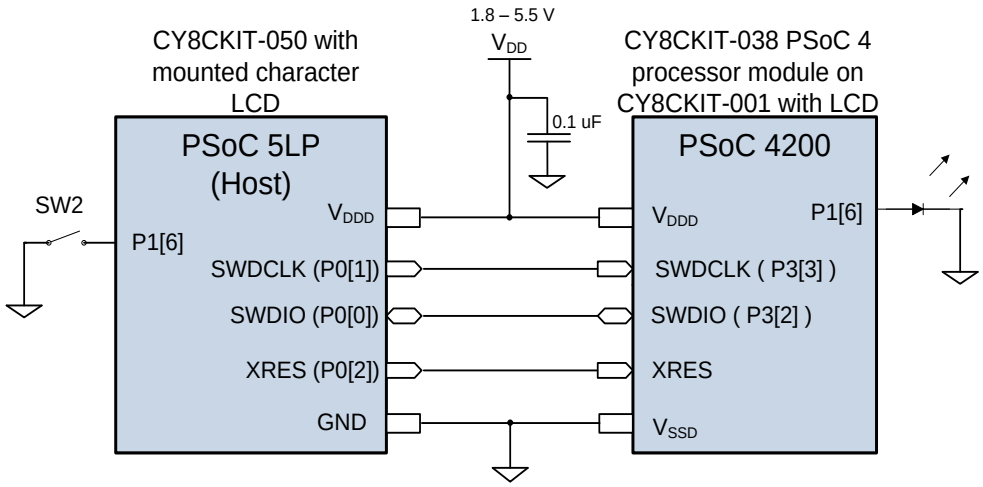
9 Testing the Example Projects

The project file *HexImage.h* defines parameters CY8C40xx_FAMILY, CY8C4xxxM_FAMILY, CY8C4xx7_BL_FAMILY, CY8C4xx8_BL_FAMILY, and CY8C4xxxL_FAMILY. These parameters are automatically set to 1 if the hex file parser application for the respective family is used to generate the *HexImage.c* and *HexImage.h* files.

9.1 For CY8CKIT-038 PSoC 4 Development Kit

To test the HSSP project on a PSoC 4 processor module (CY8CKIT-038) on the CY8CKIT-001 DVK, using PSoC 5LP as the host, use the A_Hssp_Programmer project attached with the application note. Program the project in PSoC 5LP of the CY8CKIT-050 DVK. Use Figure 4 to help you make the connections between the host and target. Pressing SW2 starts the HSSP operation.

Figure 4. Host/Target Connections



The hex file included by default in this project toggles pin P1[6] of PSoC 4 at 1-Hz frequency and displays “PSoC Programmed” on the character LCD mounted on the CY8CKIT-001 DVK after a successful programming operation. To start programming, press SW2 on the PSoC 5LP host. If programming is successful, pin P1[6] begins to toggle and the character LCD displays the message. If programming is unsuccessful, PSoC 5LP displays the cause of the error on the LCD mounted on the PSoC 5LP kit.

Note: If you are using any other host programmer, modify the source code as explained in Porting the HSSP Application to a Host Programmer. Then, test the project by making the basic connections illustrated in Figure 3.

9.2 For Kits with Onboard PSoC 5LP Programmer (KitProg)

To test the HSSP project on the kits listed in Table 10, use the B_Hssp_Pioneer project attached with this application note. Using this kit, you do not need an external host microcontroller; PSoC 5LP is present as an onboard microcontroller.

The onboard PSoC 5LP has bootloader firmware that can load and run new bootloadable applications through USB. Therefore, the HSSP project is built as a bootloadable project, and you can download *the B_Hssp_Pioneer.cyacd* project to PSoC 5LP via USB to be used as an HSSP host programmer.

This project uses a USB-to-UART component to display the programming outputs on a HyperTerminal, which is a standard program used for serial communication.

The project requires the changes listed in Table 10 to work on CY8CKIT-040, CY8CKIT-042, CY8CKIT-044, and CY8CKIT-042-BLE. Note that corresponding changes should be done in the *RegisterDefines.h* file in the project.

Table 10. Pin Assignments for Different Kits

Host Pin	CY8CKIT-040	CY8CKIT-042	CY8CKIT-043	CY8CKIT-044	CY8CKIT-046	CY8CKIT-042-BLE
Pin_SWDCK	P12[3]	P2[1]	P12[3]	P12[3]	P12[3]	P12[3]
Pin_SWDIO	P12[2]	P2[0]	P12[2]	P12[2]	P12[2]	P12[2]
Pin_XRES	P12[4]	P2[4]	P12[4]	P12[4]	P12[4]	P12[4]

To test this project, follow these steps:

1. Prepare the B_Hssp_Pioneer project:
   a) Generate the files containing the programming data (*HexImage.c*, *HexImage.h*) for your target PSoC 4 device using the HexFile Parser.
   b) Replace the existing *HexImage.c* and *HexImage.h* files with the generated ones.
   c) Build the B_Hssp_Pioneer project.
2. Enter PSoC 5LP bootloader in the Pioneer kit:
   a) Remove the power supply by unplugging the USB cable.
   b) Press and hold the reset switch (SW1) of the kit and plug in the USB cable.

The status LED starts to blink, indicating that PSoC 5LP is in the bootloader mode. Figure 5 shows the PSoC 4 Pioneer Kit and the location of the status LED and reset switch.

Figure 5. PSoC 4 Pioneer Kit

3. Bootload the HSSP project to PSoC 5LP.
    a) Open the bootloader host tool. Choose **Tools**>**Bootloader Host** in PSoC Creator.
    b) Click on the Filter button, and then click on the 'Show USB Devices' checkbox.
    c) Enter '0xF13B' in the PID field (see Figure 6).

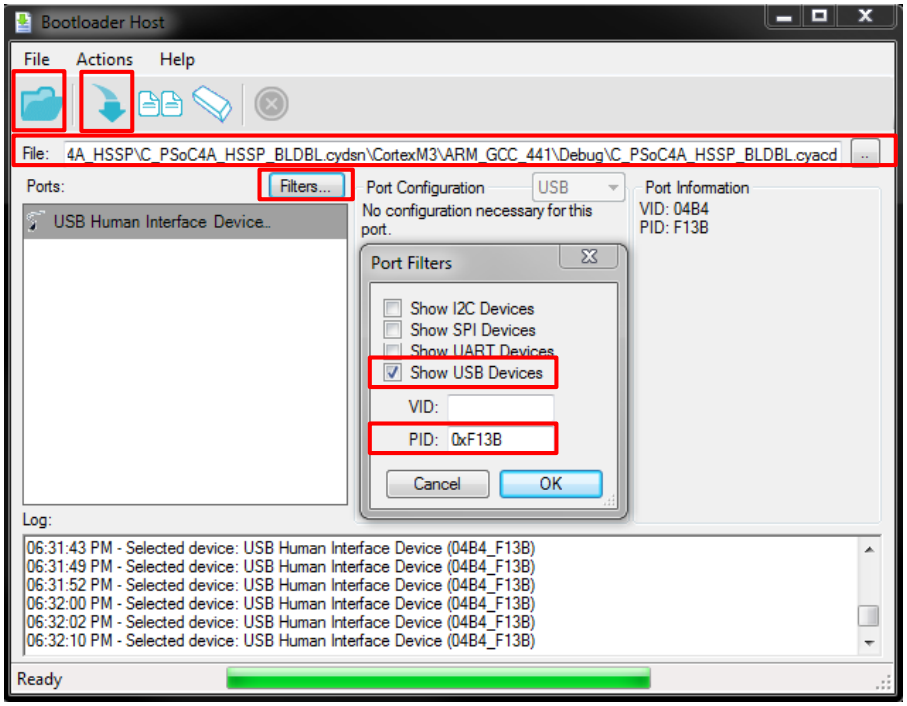
PSoC 5LP bootloader is listed as USB Human Interface Device in the Port list.

a) Click on the **Open** button in the GUI and select the *B_Hssp_Pioneer.cyacd* file from the following path:

`..\ AN84858 \ B_Hssp_Pioneer.cydsn \ CortexM3 \ARM_GCC_441 \Debug \B_Hssp_Pioneer.cyacd`

b) Click on the **Program** button to bootload the file to PSoC 5LP.

Figure 6. Bootloader Host



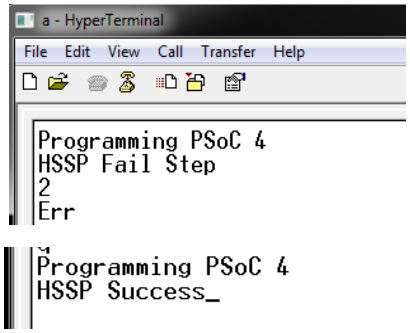
4. Install the USBUART driver for the kit:
    a) Unplug and reconnect the USB cable.
    b) If Windows fails to install the USBUART drivers, open the Device Manager. In the “Other devices” list, “USBUART” should be present.
    c) Double-click it to open its properties.
    d) Click the **Update Driver** button.
    e) Select the option “Browse my computer for driver software.”
    f) The driver is provided in the attached project under the folder USBUART. Navigate to the location where you saved the project attached with this application note, and select the driver in the “USBUART driver” folder.

    Click the **Next** button. Windows generates a warning because the driver files are not signed. Ignore the warning and proceed. Now the USBUART driver is installed on your machine
5. Use HyperTerminal to start HSSP programming:
    a) Open HyperTerminal on your computer. If you do not have it, download any terminal application for serial communication from the Internet.
    b) In the UART configuration window, set the baud rate at 9600, data bits as 8, stop bits as 1, parity as No Parity, and Hardware Control as None.
    c) Press any alphanumeric button on the keyboard to start programming. If the operation is successful, the terminal shows “HSSP Success” (see Figure 7).

    In addition, you can see a red LED on the Pioneer Kit toggling in 1-second intervals. This is the default PSoC 4 project programmed using PSoC 5LP.

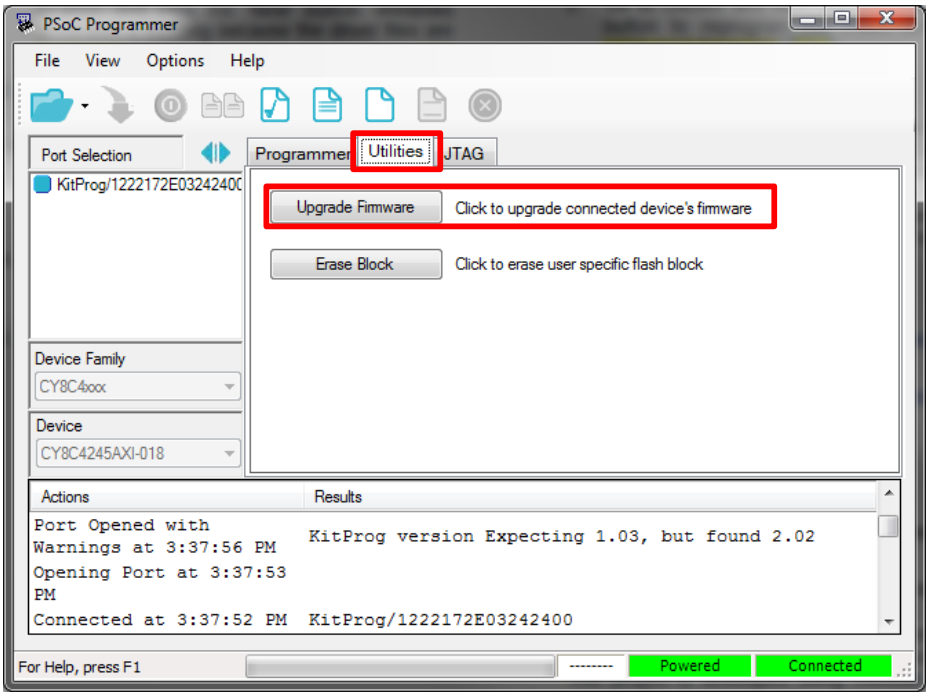
    If the programming operation fails, the terminal displays the step and error code that you can use to debug the project (see Figure 7).

Figure 7. Terminal Display



6. To reprogram PSoC 5LP with the kit firmware, follow these steps:
    a) Open PSoC Programmer (Version 3.22).
    b) Enter the bootloader as described in step 1.
    c) Go to the Utilities tab and click the **Upgrade Firmware** button to reprogram PSoC 5LP with the original kit firmware (Figure 8).

Figure 8. Upgrade Firmware using PSoC Programmer



10 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 4 device, powering of all the PSoC 4 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, listed in the section - PSoC 4 Programming Specifications - for details on the hardware connections and configuration.
- **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 1.5 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.

11 Summary

The HSSP application is useful for developing in-system programming solutions for a PSoC 4 device. It provides a cheap and robust method for programming PSoC 4 devices using an onboard embedded microcontroller as a host programmer. The portable and modular C code provided with this application note greatly reduces the time it takes to develop such HSSP applications.

12 Related Documentation

12.1 Application Notes

AN73054 – PSoC® 3 / PSoC 5LP Programming Using an External Microcontroller (HSSP)

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

12.2 PSoC 4 Programming Specifications

CYBL10x6x, CY8C4127_BL, CY8C4247_B: Programming Specifications

CY8C4XXXM Programming Specifications

CY8C41XX, CY8C42XX Programming Specifications

CY8C4000 Programming Specifications

12.3 PSoC 4 Architecture Technical Reference Manuals

PSoC 4000 Family: PSoC® 4 Architecture Technical Reference Manual (TRM)

PSoC 4100 and 4200 Family: PSoC® 4 Architecture Technical Reference Manual (TRM)

PSoC 4100M/4200M Family: PSoC® 4 Architecture Technical Reference Manual (TRM)

PSoC 41X7_BLE/42X7_BLE Family: PSoC® 4 BLE Architecture Technical Reference Manual (TRM)

12.4 Webpage

General PSoC Programming

13 List of Attached Projects

AN84858.cywrk: This workspace contains three projects to demonstrate the HSSP application.

- A_Hssp_Programmer: This is the PSoC 4 HSSP application project. It has a PSoC 5LP device, which acts as the host programmer to program the target PSoC 4 device. This project is developed using modular C code to program the device according to the steps described in the programming specifications document of the respective device listed in the Related Documentation section.
- B_Hssp_Pioneer: This is the PSoC 4 HSSP application project for testing on the PSoC 4 Pioneer Kit. It uses the onboard PSoC 5LP programmer to program the PSoC 4 device.
- C_Hssp_TimeoutCalc: This project is used to calculate the timestamp parameters used in the PSoC 4 HSSP projects.

In addition, a C# application to extract information from the hex file is attached with the application note:

- Hex File Parser application: This C# application extracts the required information from the hex file and parses it in a *.c/h* file. This file is stored in the flash of the microcontroller and is used to directly access the programming data.

About the Author

Name:	Tushar Rastogi
Title:	Applications Engineer
Background:	Tushar has a bachelor's degree in Electronics and Communication Engineering from MNNIT, Allahabad.

A Appendix A: Hex File Parser Application

The data for programming PSoC 4 is available in the hex file (*.hex*) format, which is explained in the section "Appendix B. Intel Hex File Format" of the programming specifications document of the respective device listed in the Related Documentation section.

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 metadata in hexadecimal format. The metadata includes information on the hex file record type, extended linear address data, and so on. This metadata is used to categorize the programming data into 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 the *.c* and *.h* (*HexImage.c, HexImage.h)* files, which store only the programming data from the hex files. The programming data is stored as an array of constants in the *HexImage.c* and *HexImage.h* files.

The C# application with the source code is provided along with the application note. To use the C# application, you must install .NET Framework version 3.5 or higher on your computer.

Note: Separate Hex File Parser applications are provided for the PSoC 4000, PSoC 4100/4200, PSoC 4100M/4200M, PSoC 4xx7_BLE, PSoC 4xx8_BLE, and PSoC 4xxxL device families.

C# Application Name: Hex File Parser

Development environment: Microsoft Visual Studio 2010 (Version 10.0.30319.1 RTMRel)

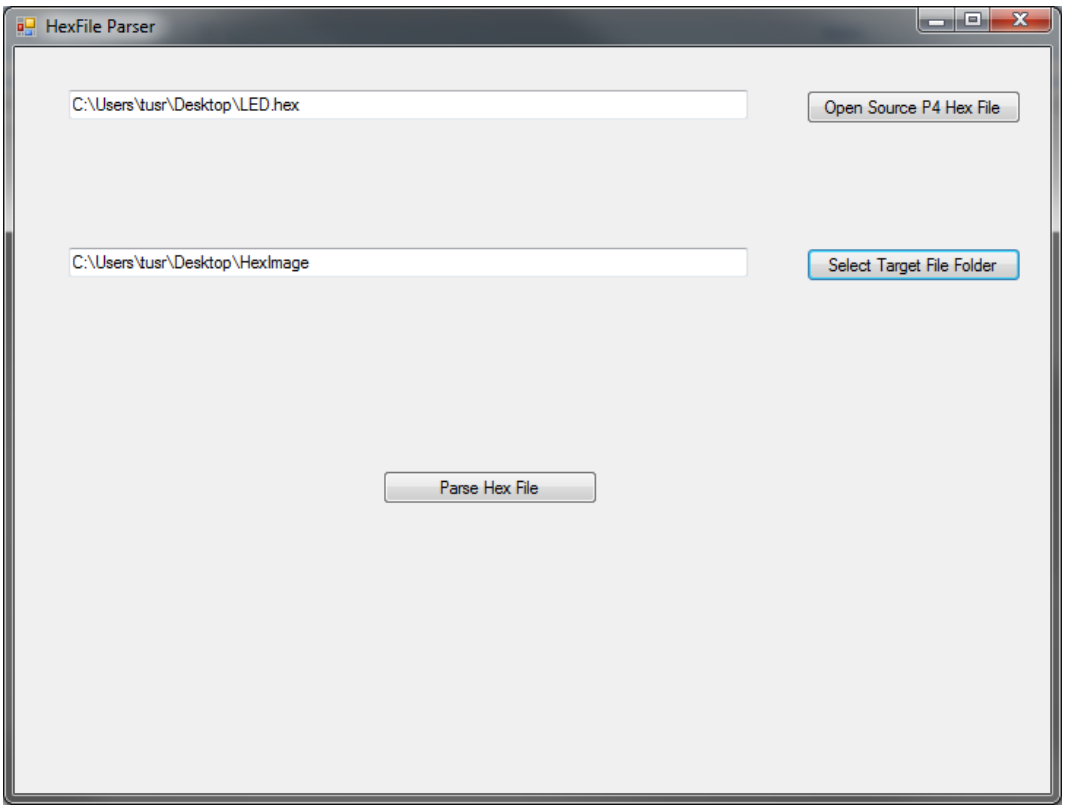
Source code: See the C# source project for details. The project source code can be viewed and edited by downloading and installing the freely available Microsoft Visual Studio 2010 Express edition. Simply right-click the *Form1.cs* file in the **Solution Explorer** window and select the **View Code** option. This action opens the source code of *Form1.cs*.

A.1 Using the Hex File Parser Application

Four sets of parser applications are available in the folder 'C# Application'. Choose the appropriate application depending on the PSoC 4 family you are using.

Open the executable file of the Hex File Parser application in the folder `C# Application\HexFile Parser.exe` of the attached *.zip* file. A GUI screen pops up, as Figure 9 shows.

Figure 9. Hex File Parser Application



1. Select the hex file that needs to be programmed by clicking the **Open Source P4 Hex File** button and navigating to the location of the file.
2. Select the destination folder location in which the parsed .c and .h files (*HexImage.h*, *HexImage.c*) should be created. Click the **Select Target File Folder** button to select the folder location.

   Note Make sure that there are no files with the names *HexImage.h* or *HexImage.c* already in the target folder location. Delete any such files or select a new folder location. A message is displayed to notify you if the same file names are already present in the target folder location.
3. After selecting the hex file and the target folder location, click the **Parse Hex File** button to generate the *.c* and *.h* files. After parsing is complete, a message is displayed.

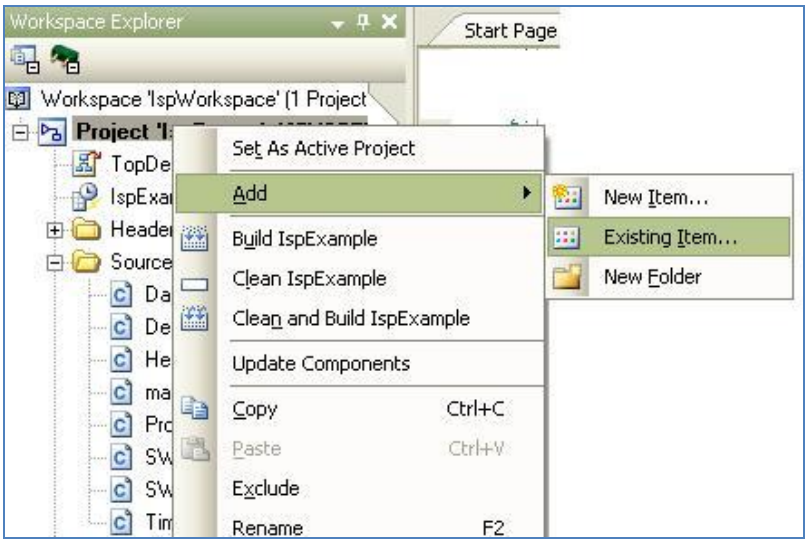
A.2 Adding the Generated Files to PSoC Creator Example Project

To add the generated *HexImage.c and HexImage.h* files to the PSoC Creator example project provided with this application note, follow the steps in this section. They are required if you need to program a hex file into a target device using PSoC 5LP as a host programmer. These steps apply to the project A_Hssp_Programmer and B_Hssp_Pioneer for PSoC 4 HSSP.

1. Select the Target File Folder location in the GUI, as shown in Figure 9. This is the folder in which the *main.c* file of the project is located.

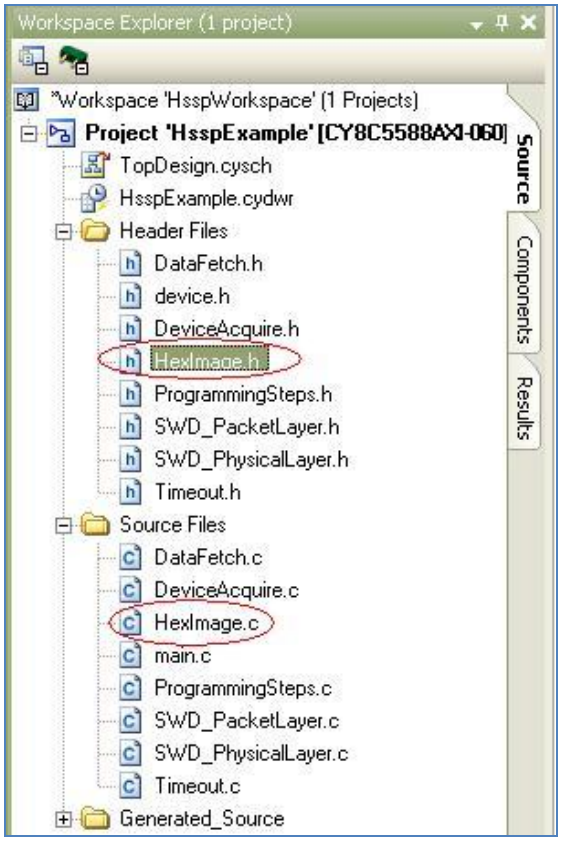
2. After the *HexImage.c* and *HexImage.h* files have been generated in the previous location, add those files to the project workspace in PSoC Creator by clicking **Add Existing Item** in Workspace Explorer, as Figure 10 shows.

Figure 10. Adding Files to PSoC Creator Project



3. After these files are selected, the Workspace Explorer window appears, as Figure 11 shows.

Figure 11. Project Workspace showing *HexImage.c* and *HexImage.h* Files



4. If you need to replace these files in the project with those corresponding to a different hex file, delete the existing files from the Workspace Explorer and project folder by right-clicking and deleting the files. Then follow steps 1 through 0 to add the new files to the project.

A.2.1 Using the *HexImage.c*, *HexImage.h* Files for the PSoC 4xxxL/ PSoC 41x8_BLE/ 42x8_BLE Family Devices:

The HSSP code examples use a PSoC 5LP device as a host programmer to program the target PSoC 4 device. Because the HSSP algorithm in the host PSoC 5LP will consume a portion of its 256 KB Flash memory, the entire hex file data of larger flash memory devices, like the devices belonging to the 256 KB PSoC 4xxxL/PSoC 41x8_BLE/42x8_BLE families, cannot be stored in the flash memory of the host PSoC 5LP. To meet the code size requirements, the programming data corresponding to the last few flash rows is deleted from the *HexImage.c* and *HexImage.h* files. The corresponding logic has also been incorporated in the `HEX_ReadRowData(…)` function in the *DataFetch.c* file. The changes done are listed below. Refer to the 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 256 by editing the array declaration as below:

```
extern unsigned char const flashData_HexFile[(NUMBER_OF_FLASH_ROWS_HEX_FILE - 256)][FLASH_ROW_BYTE_SIZE_HEX_FILE];
```

The choice of 256 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 256 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 `HEX_ReadRowData (…)` in the *DataFetch.c* file, the entire flash row data is loaded with zero in the case of the flash row number being in the last 256 rows of the target PSoC 5LP device.

B Appendix B: Status Codes for SROM Request

All of the programming-related operations are implemented as system call functions. They are executed out of the SROM in the privileged mode of operation.

You have no access to read or modify the SROM code. The DAP or the Cortex-M0 CPU requests the system call by writing the function opcode and function parameters to certain registers and then requesting the SROM to execute the function. Based on the function opcode, the SROM executes the corresponding system call from its memory and updates the function execution status in a register. The DAP or the CPU should read this status register for the pass/fail result of the function execution.

When SROM_SYSREQ_BIT (bit 31) and SROM_PRIVILEGED_BIT (bit 28) of the CPUSS_SYSREQ register are cleared, that indicates the completion of the system call. The CPUSS_SYSARG register is read to check the success or failure status of the system call.

If the 32-bit value read from the register is 0xAXXXXXXX (X denotes don't care values), the system call was successful. If the value read from the register is in the form 0xF00000YY, it indicates failure, and YY indicates the reason for failure. Table 11 shows the list of error codes.

For more details, refer to the "Nonvolatile Memory Programming" chapters of the respective PSoC 4 Architecture Technical Reference Manuals (TRM) listed in the Related Documentation section.

Table 11. Error Status Codes and Reason for Failure

Status Code (32-Bit Value in CPUSS_SYSARG Register)	Description
0xAXXXXXXX	Success. The value "X" denotes the "don't care" value, which contains 0 as returned by the SROM unless the API returns the parameter directly to the CPUSS_SYSARG register.
0x F0000001	Invalid Chip Protection Mode: This API is not available during the current chip protection mode.
0x F0000003	Invalid Page Latch Address: The address within the page latch buffer is either out of bounds or the size provided is too large for the page address.
0x F0000004	Invalid Address: The row ID or byte address provided is outside the available memory.
0x F0000005	Row Protected: The row ID provided is that of a protected row.
0x F0000006	SRAM Address Invalid: The SRAM address is out of bounds.
0x F0000007	Resume Completed: All non-blocking APIs have been completed. The resume API cannot be called until the next non-blocking API.
0x F0000008	Pending Resume: A non-blocking API was initiated, and it must be completed by calling the resume API before any other APIs may be called.
0x F0000009	System Call Still In Progress: A resume or non-blocking API is still in progress. The SPC ISR must fire before attempting the next resume.
0x F000000A	Checksum Zero Failed: The calculated checksum was not zero.
0x F000000B	Invalid Opcode: The opcode is not a valid API opcode.
0x F000000C	Key Opcode Mismatch; The opcode provided does not match key1 and key2.
0x F000000E	Invalid Start Address: The start address is greater than the end address provided.
0xF0000012	Invalid Flash Clock: The CY8C40xx family of devices must set the IMO to 48 MHz and the HF clock source to the IMO clock before write/erase operations.

C Appendix C: Bit Field Definitions of HSSP Error Status Register

The HSSP Error Status Register contains the status of the current HSSP operation. When any of the top-level steps in the HSSP application returns a failure status, the ReadHsspErrorStatus() function is called from the main application code to get the details of the error. This function returns the contents of this register. See Figure 2 for the bit fields returned by this function.

For a successful HSSP operation, all bits except bit 0 of this 8-bit register must be zero. If bit 0 is set, it indicates that the SWD packet received an OK ACK, as Table 12 shows.

C.1 Bits[2:0] – SWD Acknowledge Response (SWD ACK [2:0])

This is the 3-bit acknowledgment response for an SWD packet sent by the target device to the host programmer. The possible ACK codes are listed in Table 12.

Table 12. SWD ACK Response Codes

ACK[2:0]	ACK Response Meaning	Description
001	OK (SUCCESS)	This means that a previous SWD transaction was successful.
010	WAIT	This error code indicates that the target has returned five WAIT ACK responses consecutively.
100	FAULT	This error code indicates that there is a parity error in the 4-byte data packet sent by the host during the previous SWD write packet.
Any other code	Undefined code	Treat this as a 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.

WAIT ACK is received if the host programmer tries to clock SWDCK at a frequency higher than the maximum specified value of SWDCK in the programming specifications.

C.2 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.

C.3 Bit 4 – Port Acquire Timeout

This bit is set if the SWD packets that are part of acquiring the target device (Step 1. The `DeviceAcquire()` function in the *ProgrammingSteps.c* `file`) are not completed successfully. If this bit is set, the HSSP operation must be aborted and retried.

There are two possible causes for 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 PSoC 4 programming mode, see “Step 1: Acquire Chip” in the programming specifications document of the respective device listed in the Related Documentation section.

C.4 Bit 5 – SROM Polling Timeout Error

Flash programming in PSoC 4 is done by using SROM APIs. This bit is set when the `PollSromStatus()` function returns an error status. The CPUSS_SYSREQ register is polled for the status code for 1 second. This bit is set if the time exceeds 1 second or if the status code returned by this function corresponds to FAILURE. This function is called in all the HSSP steps to read the status of SROM system requests.

If this bit is set, the host programmer must call the `ReadSromStatus()` function in *ProgrammingSteps.h* to read and display the value of the status code returned during polling.

For more details on SROM status codes, see Appendix B: Status Codes for SROM Request.

C.5 Bit 6 – Verification Failure

This error bit, which is set in multiple steps, can fail in verification for multiple reasons, as Table 13 explains.

Table 13. Verification Errors – Steps and Reasons

Error Step	Error	Reason for Error
Device Acquire (Step 1)	Device ID Verification Error	IDCODE returned by device does not match the Cortex-M0 DAP ID (0x0BB11477).
Verify Silicon ID (Step 2)	Silicon ID Verification Error	Silicon ID information in the hex file does not match the Silicon ID read from the target device.
Verify Flash (Step 6)	Flash Data Verification Error	Flash data does not match the data in the hex file.
Verify Protection Settings (Step 7)	Flash Protection Data Verification Error	Row protection data or chip protection data read from the silicon does not match the hex file data.
Verify Checksum (Step 9)	Checksum Verification Error	Checksum value of the flash data in the target device does not match the checksum data in the hex file.

It is clear from the previous conditions that bit 6 can be set in many verification error cases. Based on the step in which the bit is set, you can infer the cause of the verification failure. For example, if the bit is set in the “Verify Silicon ID” step, the host programmer application can determine that the error is due to the mismatch of the silicon ID.

C.6 Bit 7 – Transition Error

This bit is set when the chip protection settings read from the chip and the chip protection settings stored in the hex file indicate a wrong transition.

See “Appendix A. Chip-Level Protection” of the programming specifications document of the respective device listed in the Related Documentation section to learn about protection modes and the state diagram for valid and invalid transitions.

D Appendix D: HSSP Functions

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

Table 14. 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 XRES 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 15. Functions in *SWD_PacketLayer.h*

Function	Description
`Swd_WritePacket()`	Sends an SWD write packet. This function operates on the global variables swd_PacketHeader, swd_PacketAck, and swd_PacketData[].
`Swd_ReadPacket()`	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 reading from a specific address.
`SwdLineReset()`	Resets the SWD line by sending 51 SWDCK clock cycles with SWDIO line HIGH. Used to acquire the debug access port (DAP) during step 1 of programming.

Table 16. Functions in *SWD_UpperPacketLayer.h*

Function	Description
`Read_DAP`	Reads a 32-bit data from the specific DAP register and writes it to Swd_PacketData[].This function uses the Swd_ReadPacket() function to read the data.
`Write_DAP`	Writes a 32-bit data to the specific DAP register. This function uses the Swd_WritePacket() function to write the data.
`Read_IO`	Reads a 32-bit data from the specified address of the CPU address space. This function is implemented by using the Read_DAP() and Write_DAP() functions. Returns “true” if all SWD transactions succeeded (ACKed).
`Write_IO`	Writes a 32-bit data into the specified address of the CPU address space. This function is implemented by using the Write_DAP() function. Returns “true” if all SWD transactions succeeded.

Table 17. Function in *Timeout.h*

Function	Description
`DelayHundredUs()`	Introduces a delay of 100 µs to generate an active LOW pulse signal lasting for 100 µs on the XRES pin.

Table 18. Functions in *DataFetch.h*

Function	Description
`HEX_ReadSiliconId()`	Copies the device silicon ID data from the *HexImage.c* file to an indicated destination array.
`HEX_ReadRowData()`	Copies the flash row data from the *HexImage.c* file to an indicated destination array. The flash row number is also passed as a parameter to this function.
`HEX_ReadRowProtectionData()`	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.
`HEX_ReadChipProtectionData`	Copies the chip protection data from the *HexImage.c* file to an indicated destination.
`HEX_ReadChecksumData()`	Copies the checksum data from the *HexImage.c* file to an indicated destination.
`GetFlashRowCount()`	Returns the total number of flash rows in the target device from the *HexImage.c* file.

Table 19. Functions in *ProgrammingSteps.h*

Function	Description
`DeviceAcquire()`	Enters the programming mode and acquires the target device.
`VerifySiliconId()`	Verifies whether the silicon ID of the target device and the hex file match.
`EraseAllFlash()`	Erases the entire flash memory of the target device, including the flash protection data.
`ChecksumPrivileged()`	Calculates the checksum of the privileged data in flash.
`ProgramFlash()`	Programs the flash memory of the target device.
`VerifyFlash()`	Verifies whether the flash data programmed to the target device matches the hex file flash data.
`ProgramProtectionSettings()`	Programs the row protection data and chip protection data to the target device.
`VerifyProtectionSettings()`	Verifies whether the row protection data and chip protection data programmed to the target device matches the data in the hex file.
`VerifyChecksum()`	Verifies whether the checksum data read from the target device matches the hex file checksum data.
`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.
`ReadSromStatus()`	Returns the CPUSS_SYSARG status register value in the case of an SROM polling timeout error condition.

Document History

Document Title: AN84858 - PSoC® 4 Programming Using an External Microcontroller (HSSP)

Document Number: 001-84858

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3995049	TUSR	05/14/2013	New Application Note for HSSP applications on PSoC 4.
*A	4073726	GMRL	07/22/2013	Removed submit document feedback link.
*B	4183545	RNJT	11/05/2013	Updated associated project.
*C	4223201	RNJT	12/17/2013	Added information on CY8C40xx family.
*D	4347319	RNJT	04/15/2014	Updated projects for PSoC Creator 3.0 SP1. Completing Sunset Review.
*E	4675817	RNJT	03/04/2015	Updated for PSoC 4xxxM family.
*F	4767408	ARVI	05/19/2015	Updated for the PSoC 4xx7_BL family. Added Table 9. Timeout for Acquiring Device
*G	4867159	VVSK	08/24/2015	Updated the C# application to work with hex files of all the devices in the supported families Added sections on Power cycle mode programming, Tips and Tricks for debugging HSSP issues Projects updated to add support for PSoC 41xx8_BLE/42xx8_BLE families
*H	4922631	RNJT	09/16/2015	Updated for the PSoC 4xxxL family Updated the example projects to support CY8CKIT-046
*I	5068136	RNJT	12/30/2015	Updated the example projects to PSoC Creator 3.3 SP1.

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