AN44168

PSoC® 1 Device Programming using External Microcontroller (HSSP)

Associated Project: Yes

Associated Part Family: CY8C21x23, CY8C21x34, CY21x45, CY8C22x45, CY8C23x33, CY8C24x23, CY8C24x94, CY8C27x43, CY8C28xxx, CY8C29x66, CY8CTST1xx, CY8CTMG1xx, CY8CTMA120

Software Version: PSoC Designer™ 5.4 SP1

Related Documents: For a complete list, click here.

AN44168 shows you how to implement PSoC® 1 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 1 through the In-System Serial Programming (ISSP) 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 1. The code is built based on the programming procedure explained in the PSoC 1 ISSP Programming Specifications.

Contents

1 Introduction

Cypress's Programmable System-on-Chip (PSoC) is easy-to-use and flexible, with a cost-effective mix of reprogrammable analog and digital resources. These features provide many opportunities for creative designs, one of which is programming the PSoC serially by an on-board host processor. This method is used to install or update firmware in field, or reprogram the PSoC for a different function.

The HSSP source code was created by Cypress to give system designers a starting point to create their own serial programming software. The designers must make minimal modifications to the code to make it compatible with their specific host programmer. The source code covers a wide range of PSoC devices and provides a high level of abstraction. The devices covered by this application note are listed in 'Associated Part Family'. For more information on ISSP, see the PSoC 1 ISSP Programming Specifications at www.cypress.com/?rID=2907 and www.cypress.com/?rID=2908. See the application note AN59389 for HSSP application for the CY8C20xx6, CY8CTMG2xx, and CY8CTST2xx devices. AN59389 implements the programming procedure explained in the ISSP Programming Specifications - CY8C20045, CY8C20055, CY8C20065, CY8C20xx6A, CY8C20xx7.

Differences between Bootloader and HSSP:

In embedded applications, bootloaders update the system firmware over a standard communication interface. In addition to supporting bootloaders, PSoC 1 allows in-system programming using an HSSP application. The key differences between HSSP and bootloading are explained in this section. Choose the method that best meets your requirements.

- Partial firmware update versus complete programming of the device: HSSP supports only complete programming of the flash memory in PSoC 1 and not a partial firmware upgrade. However, a bootloader can be used to update only a specific portion (bootloadable area) of the flash memory. A portion of the flash memory (bootloader area) is reserved for doing the bootloading operation and this portion cannot be updated.
- Communication interface between the external host and PSoC 1: The bootloaders use one of the many standard communication interfaces, including USB, I^2C, UART, and SPI, to upgrade the PSoC 1 firmware. HSSP in PSoC 1 always uses the ISSP protocol to do the programming.

2 HSSP Firmware Architecture

HSSP for PSoC 1 is implemented in multiple layers using modular C code. These layers are as follows:

1. ISSP protocol physical layer
2. ISSP protocol packet layer
3. HSSP programming steps layer

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

2.1 ISSP Protocol Physical Layer

File that constitutes the ISSP protocol physical layer is described below:

Source Files	Description
ISSP_PHYSICALLAYER (.C)	This file contains macros and functions to manipulate the programming pins - SDA and SCL, and control the target device's reset and power pin. Bit banging is done to generate signals at these pins. The functions in this file are called by higher level functions from the packet layer.

The firmware, in this file, is written for PSoC 1 CY8C29466 as the host microcontroller. You should modify all the functions and macros appropriately for a different host microcontroller.

2.2 ISSP Protocol Packet Layer

File that constitutes the ISSP protocol packet layer is described below:

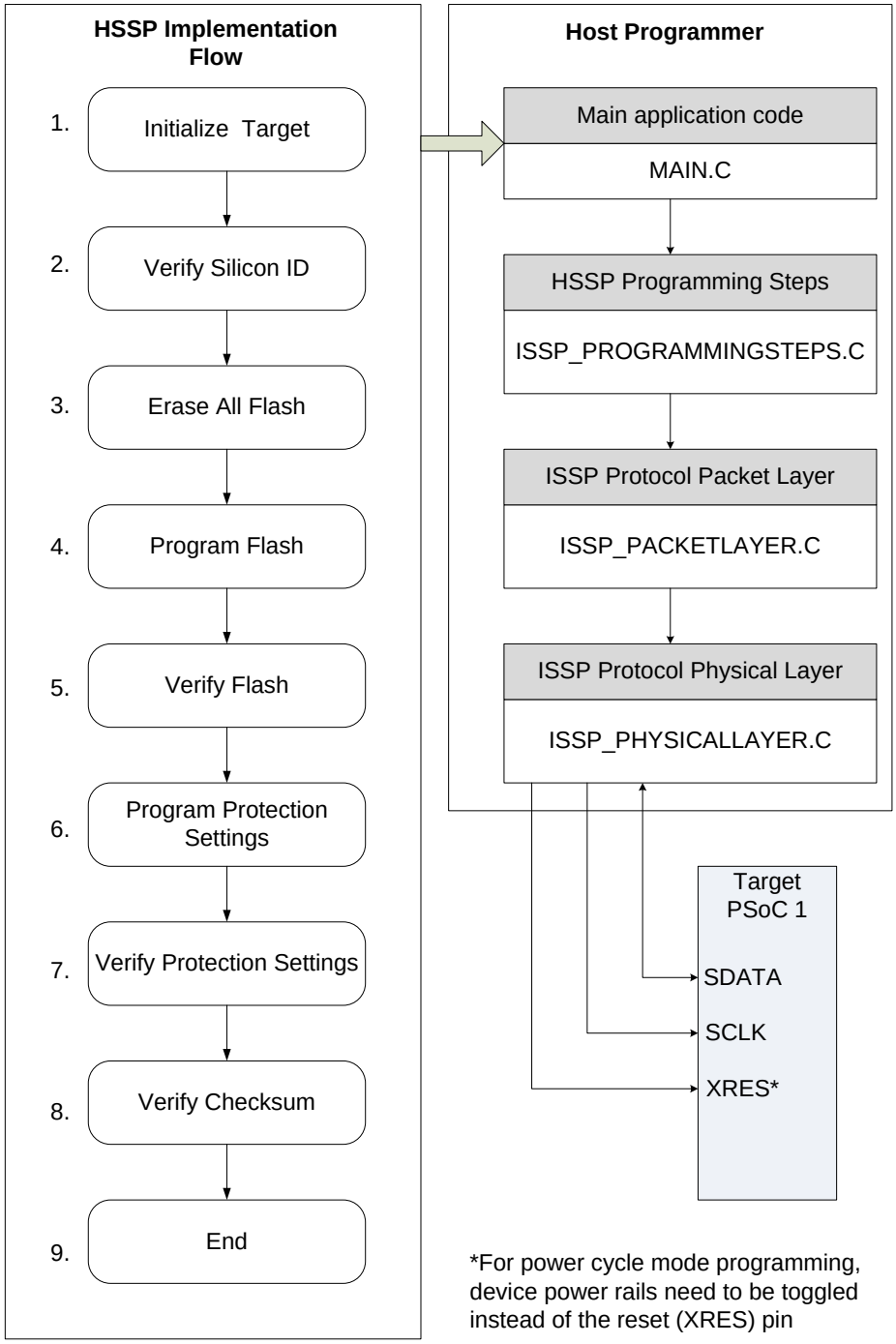
Source Files	Description
ISSP_PACKETLAYER (.C)	This file use the functions defined in ISSP_PhysicalLayer to send out the ISSP vectors. ISSP vector is nothing but a sequence of bits representing a set of instructions. These instructions are sent by host processor for the target device to execute. There are various vectors defined for different stages of programming. For details on the ISSP vectors, refer PSoC 1 ISSP Programming Specifications. The functions in this file are responsible for reading data and writing vector packets to the target device. The functions defined in this layer are called directly by the functions in the ISSP_PROGRAMMINGSTEPS.C file.

2.3 HSSP Programming Steps

File that constitutes the HSSP Programming Steps is described below:

Source Files	Description
ISSP_PROGRAMMINGSTEPS (.C)	This file contains the top level functions of the HSSP application. The details of each step are in the PSoC 1 ISSP Programming Specifications.

Figure 1. PSoC 1 HSSP Firmware Architecture



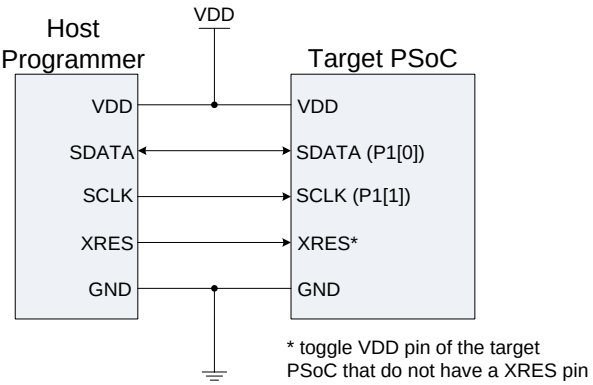
3 Hardware Connections for PSoC 1 HSSP Programming

Figure 2 shows the basic hardware connection required between the host programmer and the target PSoC 1 device. PSoC devices are programmed in two different modes: reset and power cycle. Reset mode, which is the preferred programming mode, is used when the PSoC device has an XRES pin. Devices without an XRES pin must be programmed in Power Cycle mode. In this case, HSSP microcontroller switches device power ON and OFF. In each programming mode, the host requires three I/O pins. These are serial data (SDATA), serial clock (SCLK), and external reset (XRES) in the reset mode, and SDATA, SCLK, and PSoC power (PWR) control in the power cycle mode. These pins are manipulated from the firmware.

The SDATA pin on the host processor must be bi-directional. The host must be able to change the properties of this pin; strong drive while writing and high-Z state while reading.

For more information on programming modes, Programming interface connections, and Programming specifications, see the PSoC 1 ISSP Programming Specifications at www.cypress.com/?rID=2907 and www.cypress.com/?rID=2908.

Figure 2. Basic Host/Target Connections



4 Porting the HSSP Application to a Host Programmer

The project provided with this application note, uses PSoC 1 CY8C29466 as the host programmer. 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

Table 1 shows the files that must be ported to the host device of the system.

Table 1. Files to be Ported

Header Files to be ported	Source Files to be Ported
ISSP_DEFS.H	ISSP_PHYSICALLAYER.C
ISSP_DELAYS.H	ISSP_PACKETLAYER.C
ISSP_DIRECTIVES.H	ISSP_PROGRAMMINGSTEPS.C
ISSP_ERRORS.H	
ISSP_REVISION.H	
ISSP_VECTORS.H	

4.2 Code Changes Required While porting

Table 2 shows updates required in the source files while porting the attached HSSP application code to any host programmer other than PSoC 1.

Table 2. Code Changes

File	Description	Changes required
ISSP_DEFS.H	This file contains number of banks, blocks per bank and security bytes per bank information for all PSoC 1 devices.	No change required
ISSP_DELAYS.H	This file contains timer setting for wait and poll event, XRES assertion and power cycling the target device.	Adjust the delay counts based on the host processor execution speed.
ISSP_DIRECTIVES.H	This file provides option to the user to select the target PSoC 1 device, enable/disable the debugging code (UART, test point), select target voltage and programming mode – reset or power cycle.	See section Property Selection.
ISSP_ERRORS.H	This file contains #define statements for the different errors during programming process.	No change required
ISSP_EXTERN.H	This file contains function and variable prototypes.	No change required
ISSP_REVISION.H	This file is for revision control.	No change required
ISSP_VECTORS.H	This file contains vectors for all ISSP instructions.	No change required
ISSP_PHYSICALLAYER.C	This file contains low level drivers for communicating with the target device.	See section Low-Level Driver Modifications.
ISSP_PACKETLAYER.C	This file contains functions to create vector packets for various stages of programming.	Code updates required for getting the program data. See section Loading Data into the RAM Buffer.
ISSP_PROGRAMMINGSTEPS.C	This file contains code which calls individual functions for each stage of the programming process.	No change required

4.3 Property Selection

The designer must set the three properties Property, Label, and Description. To do this, comment or uncomment certain `#define` statements in the `ISSP_DIRECTIVES.H` file. These `#define` statements are clearly marked with 'User Attention Required' and are easy to find. You can also do a page search for individual labels. An explanation for each property and its label follows.

Property: UART Debug
Label: `USE_TX`
Description: Comment out this `#define` statement to disable the code which sends out messages over UART; uncomment to enable execution of UART functions.

Property: Test Point
Label: `USE_TP`
Description: Comment out this `#define` statement to disable execution of code which bit bangs the test pin. This pin is used for debugging purpose. It is not required in the end design.

Property: Target supply voltage
Label: `TARGET SUPPLY VOLTAGE`
Description: Comment out this `#define` statement if the target runs at 3.3 V; uncomment it if the target voltage is 5 V.

Property: Programming mode
Label: `PROGRAMMING MODE`
Description: Comment out this `#define` statement if power cycle mode is used. Uncommenting the `#define` causes the target to be programmed in reset mode.

Property: Target PSoC Device
Label: `TARGET PSOC`
Description: Select the target PSoC in this section. Only one device is enabled at any time and every other device is commented out. If the device is not on the list, contact Cypress Technical Support.

4.4 Low-Level Driver Modifications

The designer must provide host-specific code to manipulate the pins involved in programming the target PSoC. These APIs are marked 'Processor Specific' and 'User Attention Required' and are found in `ISSP_PHYSICALLAYER.C`.

- **Port Bit Masks:** There are four port bit masks that must be adjusted for the specific host processor being used. Note that though four bits must be set, only three are used in programming, depending on the choice of programming method—SDATA, SCLK, and XRES in reset mode; SDATA, SCLK, and PWR in power cycle mode.
- **Delay(n) Function:** This function is adjusted so that each iteration of the while loop takes at least 2 µs. Generally, there is no upper limit for the loop time. However, the longer this loop takes, the longer it takes to program the target. For example, if the host microcontroller is also a PSoC, each iteration takes about 2 µs with a 7-µs overhead. Therefore, the function generates a delay of 2n+7 µs, where n is the parameter passed to the function. To adjust the delay time for your host processor, modify the `#define` statements in `ISSP_DELAYS.H`.
- **Port Bit Manipulation Functions:** These functions manipulate host pins to generate signals needed to program the PSoC. They deal with driving pins high and low and releasing pins to high-Z state. A list of these functions follows. Most of the functions are self- explanatory, but they are all documented within the code.

```
fSDATACheck()
SCLKHigh()
SCLKLow()
SetSCLKStrong()
SetSDATAHigh()
SetSDATALow()
SetSDATAHiZ()
SetSDATAStrong()
SetXRESStrong()
AssertXRES()
DeassertXRES()
SetSCLKHiZ()
SetTargetVDDStrong()
ApplyTargetVDD()
RemoveTargetVDD()
```

- **UART Functions:** These functions are used for debugging purpose. Messages are sent out over UART to notify progress of the programming process. User should modify below functions based on the host processor. TX8 user module is placed in the design for the UART-transmitter.

```
InitTx()
SendDebugMessage()
PrintReceivedSiliconID()
DisplaySecurityValues()
```

- **Test Point Functions:** These are also used for debugging purpose. Modify the following functions to work with the host processor:

```
InitTP()
SetTPHigh()
SetTPLow()
ToggleTP()
```

4.5 Loading Data into the RAM Buffer

The HSSP code takes data from a 64-byte buffer to program PSoC flash blocks sequentially. This process starts at the lowest block address. After the first block is programmed, the same buffer is used to program further flash blocks.

The designer must provide code to fill this buffer depending on the data source (USB, RS-232, SD Card, and so on). There are two functions to be written for the specific host processor used—`LoadProgramData()` and `LoadSecurityData()`. These functions are found in `ISSP_PACKETLAYER.C` and are marked with 'Processor Specific' and 'User Attention Required'. Currently, these functions load incrementing values into the buffer.

5 Modifying Flash Block Sequence or Quantity

In some cases, you must program a specific area in flash. An example is an area set aside for characterization, calibration, or firmware field upgrades. These features are usually implemented using the EEPROM User Module. However, in some cases programming them directly into the PSoC saves code space if that is a limitation.

You can change the start address of the target block and the order in which the blocks are programmed. This does not cause any problems as each programming sequence includes the block address. However, remember the following points:

- Flash bank number is set only once for each block write. This is applicable only to the CY8C29x66, CY8C28xxx, CY8C24x94, CY8CTST120, CY8CTMG120 and CY8CTMA120 families of products as other families have only one bank. For more information about banks, see the PSoC Technical Reference Manual (TRM). The `FLS_PR1` register determines which flash block the programming calls affect.
- If the programming loop is modified the same changes must be applied to the verify loop; otherwise, verification fails.
- The code accumulates the checksum as it goes. The code examines the checksum against the entire flash up to that point. If you program only a section of flash, set the variable `iChecksumData` accordingly.

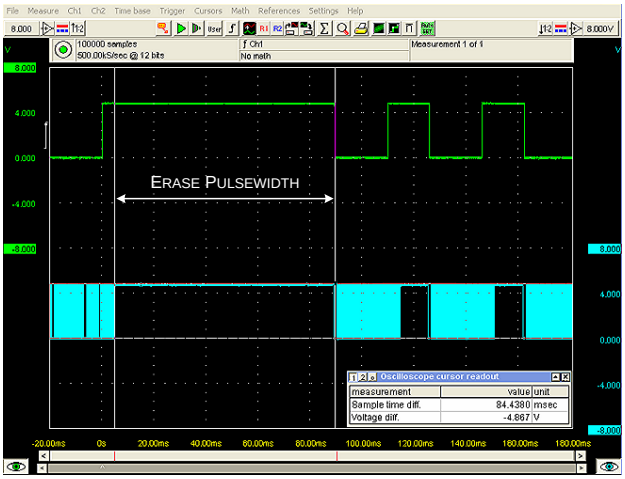
6 Verifying Flash Write Time with Built-In Test Points

One of the most critical factors in successful host-sourced programming is using the correct erase and write pulse widths. To help you with the process, a few strategically placed test point (TP) calls are implemented in the program. To enable this debugging mode, uncomment the `USE_TP #define` in `ISSP_DIRECTIVES.H`. There are few functions associated with the debugging mode, which are similar to pin manipulation functions mentioned earlier in this application note. The system designer must provide a host-specific code to drive a pin high, low, or to toggle it.

Proper debugging requires monitoring TP and SDATA lines. Both erase and programming pulses must be measured. The best way to do this is to use a two-channel oscilloscope and have it trigger in the single-sequence mode from the rising edge of the TP channel.

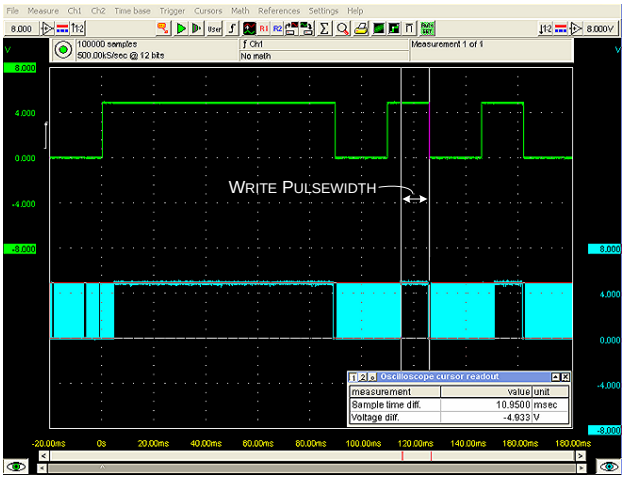
The erase pulse width is measured from the end of the data burst to the TP falling edge, as shown in Figure 3. Note that the TP rising edge does not line up with the end of the data burst. This is expected due to the delay caused by the overhead between the instant the TP pin is driven high and the host starts to send the data out.

Figure 3. Measuring the Erase Pulse Width



The programming pulse width is also measured from the end of the data burst to the TP falling edge. Figure 4 shows this measurement. As with the erase pulse width, the rising edge of the TP signal does not line up with the end of the data burst.

Figure 4. Measuring the Write Pulse Width



See the ’AC Programming Specifications’ table in the respective device datasheet to find the ideal erase and write pulse widths for various PSoC devices. The measured values must be within –3 percent to +15 percent of the ideal values. Failure to meet this requirement results in improper programming, which has undesirable side effects such as shorter than specified flash data retention[1], and fewer flash erase and write cycles than expected[2].

[1] Specified with a symbol of $Flash_{DR}$ under the DC Programming Specifications section of the device data sheet.

[2] Specified with symbols of $Flash_{ENPB}$ and $Flash_{ENT}$ under the DC Programming Specifications section of the device data sheet.

7 UART Debugging Messages

The present code with UART enabled, sends out debug messages. Figure 5 shows an example of status messages sent out during programming process.

Figure 5. Status Messages

8 Constraints

The comments at the beginning of *main.c* include useful and important information that the system designers should consider. The HSSP code has some constraints that are explained in those comments. The following is a brief summary.

- Serial programming occurs only within the temperature range of 5 °C and 50 °C.
- The HSSP program does not support voltages below 3.0 V.
- The programming procedure is completed in one voltage range only. If the device is initialized at 5.0 V, the entire procedure must be completed in 5.0 V range and with 5.0 V vectors.

- CY8C20x34 and obsolete PSoC versions are currently not supported. See the header section in *main.c* of the attached project for a list of devices not supported by this application note.
- The upper limit of SCLK's frequency is specified with a symbol of F_{SCLK} under the "AC Programming Specifications" section of the device datasheet.
- Cypress does not recommend sharing ISSP bus lines of the CY8C20x36/46A/66A/96A/ CY8CTMG2xx/ CY8CTST2xx parts with other PSoC devices. However in scenarios where the ISSP bus of the CY8C20x36/46A/66A/96A/ CY8CTMG2xx/ CY8CTST2xx parts are shared with other PSoC devices, avoid CY8C20x36/46A/66A/96A/ CY8CTMG2xx/ CY8CTST2xx parts seeing key 'AC52' in reset state. For more information, see the knowledge base article, CY8C20X36A/46A/66A/96A: issue with sharing ISSP bus.

9 Summary

This application note provides a ready-to-use example HSSP code that gives designers the flexibility to create their own serial programming software. It provides a good starting point for engineers implementing PSoC 1 Host Programming solutions, and reduces the time to develop the solution by giving a portable C source code.

10 Related Documents

- PSoC® 1 ISSP Programming Specifications - CY8C21x23, CY8C21x34, CY8C23x33, CY8C24x23A, CY8C27x43, CY8CTMG110, CY8CTST110
- PSoC® 1 ISSP Programming Specifications - CY8C21x45, CY8C22x45, CY8C24x94, CY8C28xxx, CY8C29x66, CY8CTST120, CY8CTMA120, CY8CTMG120, CY7C64215
- ISSP Programming Specifications
- AN59389 - Host Sourced Serial Programming for CY8C20xx6A

A Appendix A: Port Bit Manipulation Functions

Table 3. Port Bit Manipulation Functions

Function Name	Description
`SetSCLKStrong()`	Sets the SCLK pin to an output (Strong drive mode).
`SetSCLKHiZ()`	Releases the SCLK pin to high Z.
`SetSDATAHigh()`	Sets the SDATA pin high.
`SetSDATALow()`	Sets the SDATA pin low.
`SetSDATAStrong()`	Sets the SDATA pin to an output (Strong drive mode).
`SetSDATAHiZ()`	Releases the SDATA pin to high Z (to be driven by the target).
`AssertXRES()`	Sets the XRES pin high.
`DeassertXRES()`	Sets the XRES pin low.
`SetXRESStrong()`	Sets the XRES pin to an output (Strong drive mode).
`ApplyTargetVDD()`	Provide power to the target PSoC.
`RemoveTargetVDD()`	Remove power from the target PSoC.
`SetTargetVDDStrong()`	Sets the PWR pin to an output (Strong drive mode).

Document History

Document Title: AN44168 - PSoC® 1 Device Programming using External Microcontroller (HSSP)

Document Number: 001-44168

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2198867	JVY	03/12/2008	New Application Note.
*A	2710245	FKL	05/22/2009	Added support for CY8C28xxx devices.
*B	2880651	XCH	02/18/2010	Added support for CY8CTST110, CY8CTST120, CY8CTMG110, CY8CTMG120, CY8CTMA120. Modified timing equation to match project update which changed CPU from 24 MHz to 12 MHz.
*C	3119376	VVSK	12/23/2010	Added support for CY8C21x45, CY8C22x45 and CY8C23x33 devices. Removed the table specifying programming pulse widths, and instead added a note to refer device datasheet.
*D	3173698	VVSK	02/15/2011	Changed text in abstract to be clear on AN intent. Reorganized sections according to their relevance.
*E	3386899	VVSK	09/28/2011	Title changed to make it clear for new users. Updated to new template.
*F	3618021	KERI	05/22/2012	Added a paragraph on “Differences between Bootloader and HSSP” in the introduction section. Changed the software version from PD5.1 to "PSoC Designer™ 5.2 SP1". Updated to new template.
*G	4035071	KERI	06/20/2013	Changed the software version to PD5.3. Added info on “Wait and Poll” method in “Other Implementation Features” section. Updated reference documentation links.
*H	4914547	RJVB	09/11/2015	Updated Software Version as “PSoC Designer™ 5.4 SP1” in page 1. Updated Abstract. Removed HSSP Overview. Removed HSSP Implementation. Added HSSP Firmware Architecture. Added Hardware Connections for PSoC 1 HSSP Programming. Added Porting the HSSP Application to a Host Programmer. Updated Verifying Flash Write Time with Built-In Test Points: Renamed “Verifying with Built-In Test Points” with “Verifying Flash Write Time with Built-In Test Points” in heading and updated description. Added UART Debugging Messages. Updated Summary. Updated to new template. Updated attached Associated Project: Added protection bits verification stage, removed clocks at SCLK line during TPOLL event, added UART for debugging, and updated with PSoC Designer 5.4 SP1. Completing Sunset Review.
*I	5706010	AESATMP9	04/21/2017	Updated logo and copyright.

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

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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