Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

AN59389

Host Sourced Serial Programming for CY8C20xx6A,CY8C20xx6AS, CY8C20xx6L and CY8C20xx7/S

Author: Chris Hammer
Associated Project: Yes
Associated Part Family: CY8C20xx6A, CY8C20xx6AS CY8C20xx6L and CY8C20xx7
Software Version: PSoC Designer™ 5.4 CP1
Related Documents: ISSP Programming Specifications

Host Sourced Serial Programming (HSSP) is a method of in-circuit serial programming (using ISSP protocol) of the CY8C20xx6A, CY8C20xx6AS, CY8C20xx6L and CY8C20xx7 devices from an onboard host processor. AN59389 explains how to use and port the HSSP code example, provided along with this application note, to the desired host processor for programming the CY8C20xx6A, CY8C20xx6AS, CY8C20xx6L and CY8C20xx7 devices.

1 Introduction

Cypress's PSoC microcontrollers are easy-to-use, flexible, and have 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 even completely reprogram the PSoC for a different function.

Cypress created the HSSP Code Example to give system designers a starting point to create their own serial programming software. Designers have to make minimal modifications to the code to make it compatible with their specific host programmer. The Code Example covers only the CY8C20xx6A, CY8C20xx6AS, CY8C20xx6L, and CY8C20xx7 devices and provides a high level of abstraction. For more information on serial programming, refer to ISSP Programming Specifications.

This application note describes the implementation on a high level. Protocol details and meaning of the vectors are proprietary and intentionally omitted.

2 Overview

The HSSP Code Example has four major parts: main function, sub functions for various programming steps, low-level I/O functions, and definition files. The system designer's direct involvement with the code is to set certain properties through `#defines` to provide code to fill a 128-byte buffer with programming data and to provide low-level drivers for the host I/O.

PSoC devices are programmed in two different modes: Reset and Power Cycle. Reset mode, which is the preferred programming mode, is used only when the system is powered externally. In this case, the XRES pin on the target PSoC is toggled at the end of the process to bring it out of programming mode and resume normal operation. In the Power Cycle mode, the host microcontroller switches the PSoC's power on and off.

In each programming mode, the host needs 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) in the Power Cycle mode. The software influences these pins.

The SDATA pin on the host processor must be bidirectional. The host must be able to change the properties of this pin so that it drives a signal to the PSoC, is released to High-Z state, and is read.

3 Property Selection

The designer must set two properties: Label and Description. To do this, comment or uncomment certain `#defines` in the `ISSP_DIRECTIVES.H` file. These `#defines` 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: Programming mode

Label: `PROGRAMMING MODE`

Description: Comment out this `#define` if you use the power cycle mode. Uncommenting the `#define` causes the target to be programmed in reset mode.

Property: Target PSoC Device

Label: `TARGET PSOC`

Description: Select the target CY8C20xx6, or CY8C20xx7 PSoC in this section. Only one device is enabled at any given time and every other device is commented out.

3.1 Low-Level Driver Modifications

The designer gives 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_DRIVER_ROUTINES.C`.

- **Port Bit Masks:** There are four port bit masks that must be adjusted for the specific host processor being used. Note that though there are four bits to 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 1 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 1 µs and there is a 3-µs overhead. Therefore, the function generates a delay of n+3 µs, where n is the parameter passed to the function. To adjust the delay time for your host processor, modify the `#define`s 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. The descriptions are also available in the Appendix.

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

3.2 Loading Data into RAM Buffer

The HSSP code takes data from a 128-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 a 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 `fLoadSecurityData()`. These functions are found in `ISSP_DRIVER_ROUTINES.C` and are marked with "Processor Specific" and "User Attention Required." In their original state, these functions call two secondary functions that load the buffer with pseudo test data for debugging purposes. In the final version, delete or comment out these calls.

3.3 Modifying Flash Block Sequence or Quantity

In some cases you have to 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:

- If the programming loop is modified, the same changes must be applied to the verify loop to avoid verification failure.
- The code accumulates the checksum as it goes. It examines the checksum against the entire flash up to that point. If you program only a section of flash, set the variable iChecksumData accordingly.

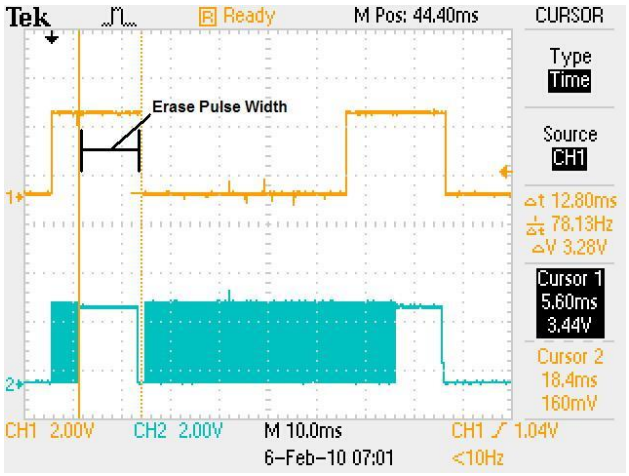
4 Verifying with Built In Test Points

One of the most critical factors in successful host sourced programming is getting the erase and write pulse widths right. 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` `#ifdef` in `main.c`. There are a few functions associated with the debugging mode that are similar to pin manipulation functions mentioned earlier in this application note. The system designer must provide host specific code to drive a pin high, low, or to toggle it.

Proper debugging requires monitoring TP and SDATA lines, and 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 single sequence mode from the rising edge of the TP channel.

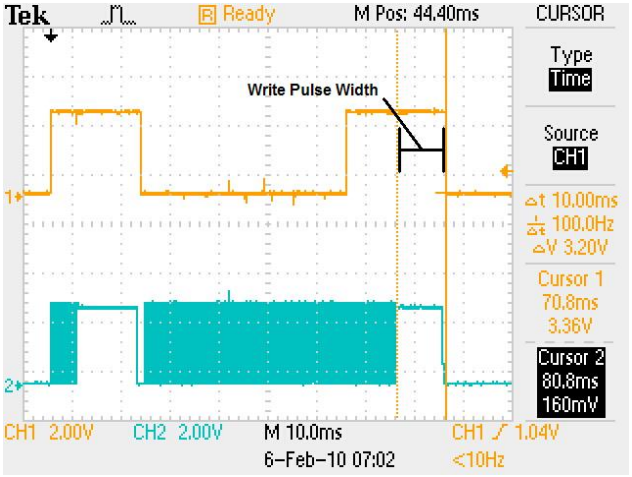
The erase pulse width is measured from the end of the data burst to the TP falling edge, as shown in Figure 1. Note that the TP rising edge does not line up with the end of the data burst. But the TP rising edge is expected to line up due to the delay caused by the overhead between the instant the TP pin is driven high and the host starts sending the data out.

Figure 1. 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 2 shows the programming pulse width 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 2. Measuring the Write Pulse Width



Refer to the device datasheets of CY8C20xx6A and CY8C20xx7 for ideal erase and write pulse widths. The measured values must be within -3% to +15% 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 $Flash_{DR}$ in the DC Programming Specifications section of the device datasheets.
[2] Specified with symbols of $Flash_{ENPB}$ and $Flash_{ENT}$ in the DC Programming Specifications section of the device datasheets.

5 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; however, 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 1.8 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.
- There is an upper limit on SCLK's frequency. The frequency is specified with the FSCLK symbol in the AC Programming Specifications section of the CY8C20xx6A and CY8C20xx7 device datasheets.

6 Summary

The HSSP program has a built-in error reporting section that is useful for debugging. Read the `bErrorNumber` variable to find out about potential problems. The `ISSP_ERRORS.H` file contains a list of all caught errors.

The last step in successful HSSP programming is to reset the PSoC device to bring it out of programming mode. To do this, call the `ReStartTarget()` function.

This application note provides some HSSP codes that give designers the flexibility to create their own serial programming software. This document also explains how to set the right erase and write pulse widths to ensure successful programming.

Appendix A. Port Bit Manipulation Functions

Function Name	Description
`SetSCLKStrong()`	Sets the SCLK pin to an output (Strong drive mode)
`SetSCLKHiZ()`	Releases the SCLK pin to HI-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: AN59389 - Host Sourced Serial Programming for CY8C20xx6A,CY8C20xx6AS, CY8C20xx6L and CY8C20xx7/S

Document Number: 001-59389

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2878828	XCH	02/15/2010	New Application Note.
*A	3211470	PPKS	03/31/2011	Updated Introduction: Updated description.
*B	3543236	KPOL	03/06/2012	Added CY8C20766A and CY8C20746A parts related information in all instances across the document. Removed CY8CTMG2xx and CY8CTST2xx family of devices related information in all instances across the document. Updated attached Associated Project: Changed security setting from 0x1B to 0x00 (Unprotected Mode) in API fLoadSecurityData(). Updated to new template.
*C	3628131	ZINE	05/30/2012	Added CY8C20xx6L and CY8C20xx6AS parts related information in all instances across the document.
*D	3702931	ZINE	08/03/2012	Added CY8C20xx7 part related information in all instances across the document.
*E	3759065	ZINE	09/28/2012	Added CY8C20x45 and CY8C20x55 parts related information in all instances across the document.
*F	4646445	KPOL	02/14/2015	Updated Software Version as “PSoC Designer™ 5.4 CP1” in page 1. Updated Document Title to read as “Host Sourced Serial Programming for CY8C20xx6A, CY8C20xx6AS, CY8C20xx6L and CY8C20xx7/S - AN59389”. Removed CY8C20045 and CY8C20055 parts related information in all instances across the document. Updated attached Associated Project: Code example is rebuild in PSoC Designer 5.4 CP1. Updated to new template.
*G	4729112	VAIR	04/17/2015	No content change, triggered by sunset review.
*H	5832279	AESATMP8	07/27/2017	Updated logo and Copyright.
*I	6167430	VAIR	05/07/2018	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

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 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2010-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.