Host Sourced Serial Programming

AN44168

Author: Navid Kamran

Associated Project: Yes
Associated Part Family: CY8C2xxxx (excluding 20xxx, 25xxx and 26xxx)
GET FREE SAMPLES HERE
Software Version: PSoC Designer™ 4.4
Associated Application Notes: AN2014, AN2026, AN2026a, AN2026b

Application Note Abstract

Host Sourced Serial Programming (HSSP) is a method of programming Cypress' Programmable System On-Chip (PSoC®) devices in-system. HSSP is particularly useful for functions such as firmware field upgrades and calibration. This application note explains how to use the Cypress provided code and modify it for a host processor to program PSoCs.

Introduction

Cypress' PSoC microcontrollers are known for their ease of use, flexibility, and 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.

The HSSP source code was created by Cypress to give system designers a starting point to create their own serial programming software. The designer has to 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. For more detailed information on serial programming, refer to application notes AN2014, AN2026, AN2026a, and AN2026b.

Overview

The HSSP source code has four major parts: main function, sub functions for various programming steps, low level IO functions, and definition files. The system designer's direct involvement with the code is to set certain properties via `#define`s to provide code to fill a 64-byte buffer with programming data and to provide low level drivers for the host IO.

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. However, some PSoC devices do not have an XRES pin. These devices are programmed in power cycle mode only. In power cycle mode, the host microcontroller switches the PSoC's power on and off.

In each programming mode, the host requires three IO pins. These are serial data (SDATA), serial clock (SCLK), and external reset (XRES) in reset mode, and SDATA, SCLK, and PSoC power (PWR) in power cycle mode. These pins are manipulated from the software.

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. For more information on programming modes, refer to application note AN2014[1].

Property Selection

The designer must set the three properties given below. To do this, comment or uncomment certain `#define`s in the `ISSP_DIRECTIVES.H` file. These `#define`s are clearly marked with "User Attention Required" and are easy to find. You may also do a page search for individual labels. An explanation for each property, along with its label is provided below.

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

Property: Programming mode
Label: `PROGRAMMING MODE`
Description: Comment this `#define` out 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 given time and every other device is commented out. If the device is not on the list, contact your local Cypress FAE or call the Cypress Applications Hotline at the number at the end of this document.

Note
1. Application note AN2014: Design for In-System Serial Programming (ISSP).

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_DRIVER_ROUTINES.C`.

- **Port Bit Masks:** There are four port bit masks that must be adjusted for the specific host processor you are using. Note that though there are four bits to set, only three are used in programming depending on your 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. So 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()
```

Loading Data into 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 `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.

Modifying Flash Block Sequence or Quantity

In some cases you may 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:

- Flash bank number is set only once per block write. This is applicable only to CY8C29x66 and CY8C24x94 family of products as other families have only one bank. For more information about banks see the Supervisory ROM (SROM) chapter of 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. It examines the checksum against the entire Flash up to that point. If you program only a section of Flash, you should set the variable `iChecksumData` accordingly.

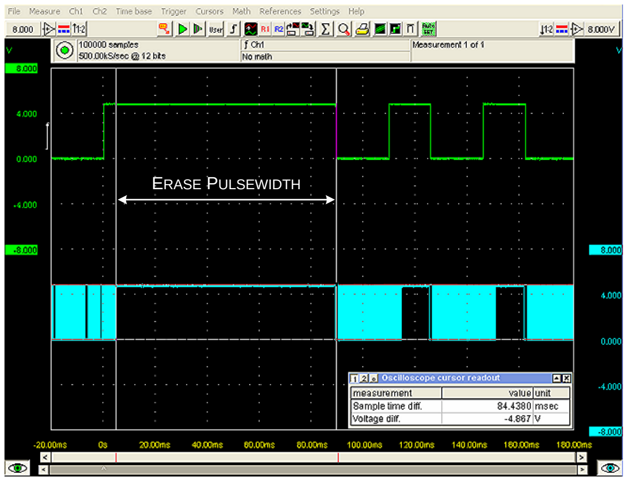
Verifying with Built In Test Points

One of the most critical factors in successful host sourced programming is getting the erase and write pulse widths right. To help the user 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 `ISSP_DRIVER_SOUTINES.H`. There are a 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 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 recommended way to do this is to use a 2-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. This is expected due to the delay caused by the overhead between the instant the TP pin is driven high and the start of sending the data out by the host.

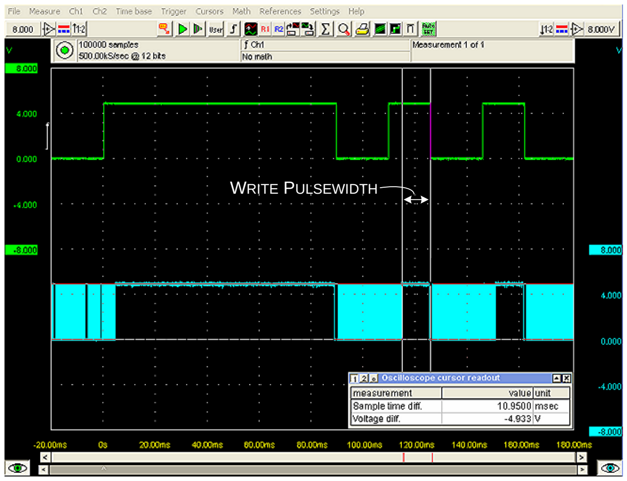
Figure 1. Measuring the Erase Pulse Width



measurement	value	unit
Sample time diff.	84.4380	msec
Voltage diff.	-4.867	V

The programming pulse width is also measured from the end of the data burst to the TP falling edge. Figure 2 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 2. Measuring the Write Pulse Width



measurement	value	unit
Sample time diff.	10.9500	msec
Voltage diff.	-4.933	V

Table 1 summarizes ideal erase and write pulse widths for various PSoC devices. 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[2], and fewer Flash erase and write cycles than expected[3].

Table 1.Erase and Write Pulse Widths for Various PSoCs

Part Number	**Erase Pulse Width (ms)**			**Write Pulse Width (ms)**		
	Min	***Ideal***	***Max***	***Min***	***Ideal***	***Max***
CY8C21x23	19.4	20	23	77.6	80	92
CY8C21x34	19.4	20	23	38.8	40	46
CY8C24x23A	19.4	20	23	77.6	80	92
CY8C24x94	38.8	40	46	38.8	40	46
CY8C27x43	77.6	80	92	9.7	10	11.5
CY8C29x66	77.6	80	92	38.8	40	46

Constraints

The comments at the beginning of main.c include useful and important information that system designers should consider. The HSSP code has some constraints that are explained in those comments; however, below 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 3.0V.
- The programming procedure is completed in one voltage range *only*. If the device is initialized at 5.0V, the entire procedure must be completed in 5.0V range and with 5.0V vectors.
- CY8C20x34 and obsolete PSoCs are currently not supported.
- There is an upper limit on SCLK's frequency. It is specified with a symbol of F_{SCLK} under the AC Programming Specifications section of the device data sheet.

Final Steps

The HSSP program has a built in error reporting section that is very handy when 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.

Notes
2. Specified with a symbol of $Flash_{DR}$ under the DC Programming Specifications section of the device data sheet.
3. Specified with symbols of $Flash_{ENPB}$ and $Flash_{ENT}$ under the DC Programming Specifications section of the device data sheet.

Appendix: 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)

About the Author

Name:	Navid Kamran
Title:	Applications Engineer
Background:	Navid has a BSEE from University of Washington, Seattle. He is an amateur musician and records and produces music in his spare time.
Contact:	navid.kamran@cypress.com
Note:	The HSSP code and application note are based on extensive work done by Mr. Mark Latham of Liquid Logic.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/