CY8C21x12, CY8C21x23, CY8C21x34, CY8C23x33, CY8C24x23A, CY8C27x43, CY8CTMG110, CY8CTST110

PSoC® 1 ISSP Programming Specifications

Document #: 001-13617 Rev. *K

December 1, 2016

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
http://www.cypress.com

Copyrights

Contents

1. Overview

1.1 Introduction

In-circuit programming is convenient for prototyping, manufacturing, and in-system field updates. PSoC® 1 devices can be programmed in-system using the in-system serial programming protocol (ISSP), a proprietary protocol used by Cypress.

This reference manual provides programming timing and vectors so that developers and programmer vendors can create their own in-system programming solutions for a PSoC 1 device. See application note AN44168 for practical implementation of the host programming solution. See General PSoC Programming for a list of programming solutions available for PSoC 1.

There are two participants in the programming procedure: the programmer and the target device. The programmer communicates serially with the target. The programmer supplies the clocking and sends commands to the target. The target receives data from the programmer and supplies data upon a read request. It drives the data line only upon request from the programmer. The programmer programs the target with the program image contained in the <PROJECT NAME>*.hex* file, which is generated by PSoC Designer™. See "Intel.Hex File Format" on page 22 for more information.

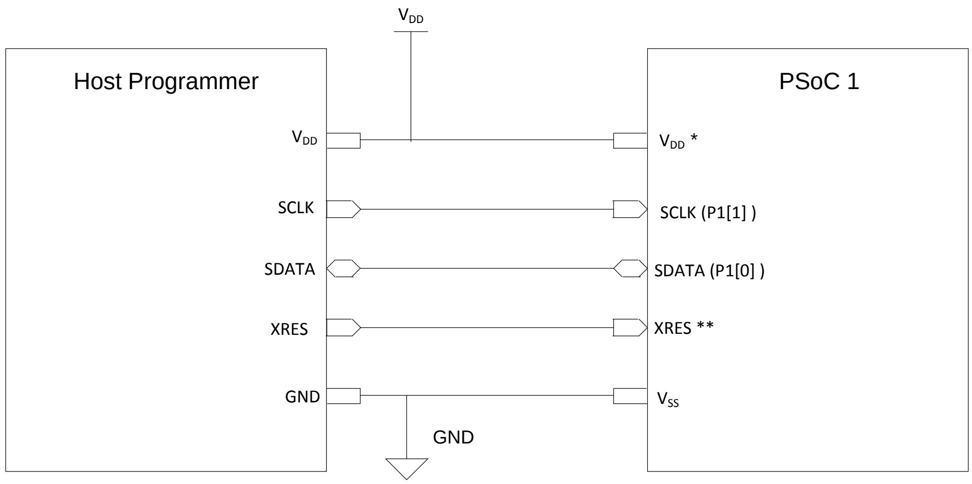
When developing a host programming application, remember the following.

- The programming vectors provided in this document should not be compared with those generated by the MiniProg1, Miniprog3, or ICE-Cube. This is because MiniProg1, MiniProg3, and ICE-Cube follow a slightly different version of the protocol to program the target device. Cypress recommends using the programming vectors provided here to develop your own host side interface and program the PSoC 1 device.
- Even though the ISSP protocol uses a bidirectional data line for communication between host and target device, it is not related to the I^{2}C protocol.

1.2 Host Programmer - PSoC 1 Programming Interface

Figure 1-1 shows the connections between the host programmer and the target PSoC 1 device. If MiniProg1 programmer is used, see the knowledge base article at http://www.cypress.com/?id=4&rID=50010 for information on part number of the MiniProg1 programming header.

Figure 1-1. Host Programmer - PSoC 1 Interface



* To program in Power Cycle mode, the host programmer must be capable of toggling power to the PSoC 1 device.

** XRES pin in PSoC 1 is active high input. It has an internal pull-down resistor to keep it at logic low when left floating. XRES pin is not available in all device packages. Check the device data sheet for information on XRES pin availability. Use Power Cycle mode if XRES is not available.

1.2.1 Programming Pin Drive Modes

The electrical pin connections between the programmer and the target device shown in Figure 1-1 are listed in Table 1-1. This includes two signal pins, a reset pin, a power pin, and a ground pin. Leave the other pins floating. The pin naming conventions and drive strength requirements are also listed in Table 1-1.

Table 1-1. Pin Names and Drive Strengths

Pin Name	Function	Programmer HW Pin Requirements	PSoC 1 Drive mode behavior
P1[0]	SDATA – Serial Data In/Out	Drive TTL Levels, Read TTL, High Z	Strong drive (while sending data to host), Resistive pull down mode (reading data from host, waiting for data from host)
P1[1]	SCLK – Serial Clock	Drive TTL Levels	High Z Digital input
XRES	Reset	Drive TTL Levels. Active High	Active high Reset input with internal resistive pull down
V_{SS}	Power Supply Ground Connection	Low Resistance Ground Connection	Ground connection
V_{DD}	Positive Power Supply Voltage	0 V, 1.8 V, 3.3 V, 5 V. 20 mA Current Capability	Supply voltage

The PSoC 1 SDATA pin drive modes vary during the programming operation. When PSoC 1 drives the SDATA line to indicate that it has started up completely or to send data back to the host, it is in a strong drive configuration. When PSoC 1 waits for data or receives data from host, SDATA is in a resistive pull-down configuration. It is important to design the host external pin drive mode circuitry such that a strong high to resistive low transition can be detected, and also so that the SDATA pin can be driven both high and low when it is in resistive pull down mode. Due to internal pull-down resistor (5.6 kΩ) on SDATA line, the presence of external pull-up resistors on the SDATA line can cause the host to miss the high to low transition on the target device due to resistive voltage divider. Therefore, It is not recommended to use external pull-up resistors on the SDATA line.

1.2.2 Using External Crystal Oscillator

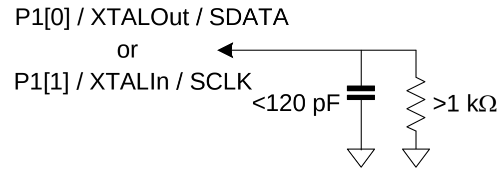
The programming pins on PSoC 1 (SCLK (P1[1]), SDATA (P1[0])) are also shared by the external 32 kHz crystal. If the external 32 kHz crystal is used, the programming connections to ports P1[0] and P1[1] must be kept as short as possible. The total capacitance on each side of the crystal should be close to 25 pF, including the capacitance of the package leads. See the device data sheet for pin capacitance. Excessive trace length on these signals can adversely affect the operation of the oscillator. During programming, the 32 kHz crystal loading does not add loading to the programming pins.

1.3 Pin Loading Requirements

The SDATA and SCLK pins each have three functions. These pins are configurable as an external 32 kHz crystal, I^2C interface pins, and as general purpose I/O pins.

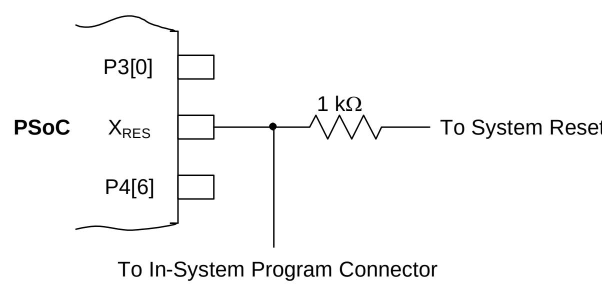
The equivalent load on these pins should not exceed 120 pF in parallel with a 1-kΩ resistor.

Figure 1-2. Maximum Load Data and SCLK Pins



The XRES signal is a single function pin. This signal should be connected directly to the programmer connector. Some designs may drive the XRES signal from another source, such as a system reset, to force reset at a known time. In this case, a resistor may be placed in series with the signal source and the XRES pin. The programmer is then connected on the pin side of the register. See Figure 1-3. This allows the programmer to overdrive the XRES pin.

Figure 1-3. XRES Connection



Document Revision History

Document Title: CY8C21x12, CY8C21x23, CY8C21x34, CY8C23x33, CY8C24x23A, CY8C27x43, CY8CTMG110, CY8CTST110 PSoC® 1 ISSP Programming Specifications

Document Number: 001-13617

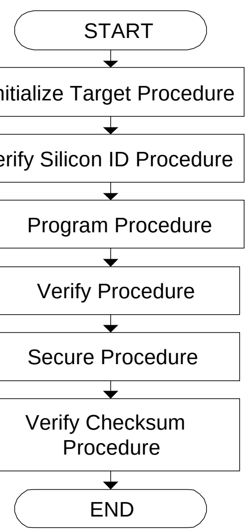
Revision	Issue Date	Origin of Change	Description of Change
**	11/26/2007	FSU	Initial version
*A	07/31/2008	MAXK	Updated Power on Mode on page 7. Updated IntelHex File Format for CY8C21/22/24/24A/27xxx on page 13. Converted to latest application note template. Modified author details.
*B	02/22/2010	XCH	Add CY8CTMG110 and CY8CTST110.
*C	10/01/2010	MAXK	Updated Figure 5, Figure 8, Figure 10, Figure 13, Figure 14, and Figure 15. Updated "Wait and Poll" Timing Diagram section. Updated Table 5.
*D	12/06/2010	VVSK	Updated Introduction. Updated Reset Mode and Power on Mode. Added Erase Block Procedure. Updated Table 5 in Appendix A. Updated Appendix B.
*E	02/04/2011	VVSK	Updated Associated Application Notes in page 1 as AN2026b, AN2026c, AN2026d, AN44168, AN59389. Updated Abstract. Updated Introduction. Added Host Programmer - PSoC 1 Programming Interface, Programming Pin Drive Modes, and Pin Loading Requirements. Updated Clocking, Data format and timing diagrams, added Wait and Poll sub-sections under Programming Flow. Updated Power on Mode, Verify Silicon ID Procedure sub-sections under Initialize Target Procedure. Updated Program Procedure. Updated Verify Procedure.
*F	05/19/2011	VVSK	Changed Title. Changed Associated Part Family. Changed Associated Application Notes. Modified Abstract. Updated Introduction. Updated Host Programmer - PSoC 1 Programming Interface. Updated Table 4. Updated Appendix A and Table 5. Updated Appendix B
*G	09/08/2011	VVSK	Converted to TRM category
*H	04/19/2012	RJVB	Updated Table 2-1 and Table 2-2
*I	12/04/2013	RJVB	Sunset review
*J	04/14/2014	JICG	Added CY8C21x12 part. Added READ-ID-WORD (CY8C21312) and READ-ID-WORD (CY8C21512) in Table A-1.
*K	12/01/2016	RJVB	Updated logo and the copyrights section.

2. Programming Flow

2.1 Target Programming

Successful target programming depends on adherence to the programming flow shown in Figure 2-1. Each procedure is explained in detail in the following sections. Failure to complete these steps can result in incorrectly programmed flash.

Figure 2-1. Target Programming Flow



2.1.1 Vectors

Vectors are the binary representation of the commands necessary to perform various operations involved in the programming flow. Each procedure in the programming flow has many individual vectors associated with it; see "Programming Vectors" on page 18. Each vector is 22 bits long and any number of zeros can be sent between sequential vectors. The target ignores the zero padding and any subsequent '0' on the SDATA line. This continues until the target receives a '1', which is the first bit in the next vector in the vector-set.

2.1.2 Clocking, Data Format, and Timing Diagrams

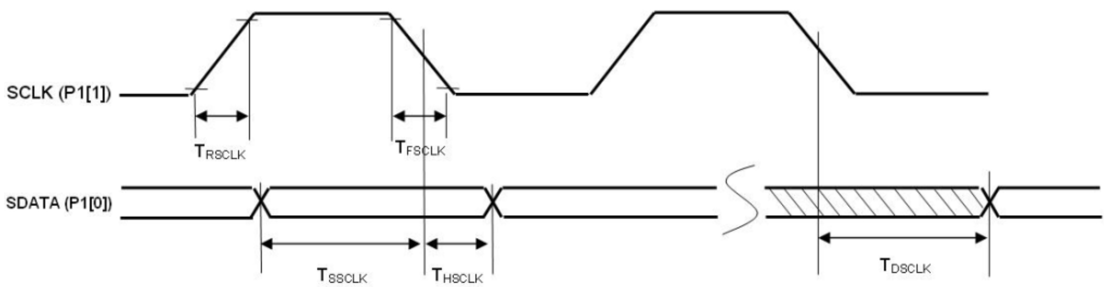
The host programmer always writes and reads on the rising edge of SCLK, while the target writes and reads on the falling edge. See Figure 2-2 that shows the timing waveforms of the SCLK lines. See Table 2-2 on page 16 for the timing specifications mentioned in Figure 2-2.

During the programming flow, the programmer supplies a clock on SCLK to transfer data. This data transfer mode is used while the programmer communicates with the target, either by sending or receiving data. During this time, the programmer can drive the SCLK signal at any frequency that enables reliable data transfer with a maximum transmit frequency of 8 MHz (see F_{SCLK} in Table 2-2 on page 16). The frequency of SCLK does not need to be accurate or consistent, as long as it is less than the 8 MHz limit.

After the programmer requests a read from the target, it releases the SDATA line to a High Z state and resumes driving the SDATA line only after the byte is sent by the target. The programmer supplies clocks even when it has released (High Z) the SDATA line.

During the "Wait and Poll" procedure, the programmer releases (High Z) the SDATA line and must wait for a High to Low transition on SDATA. Clocks are not allowed during the Wait and Poll phase (T_{poll}) as shown in Figure 2-3.

Figure 2-2. SCLK Timing Diagram

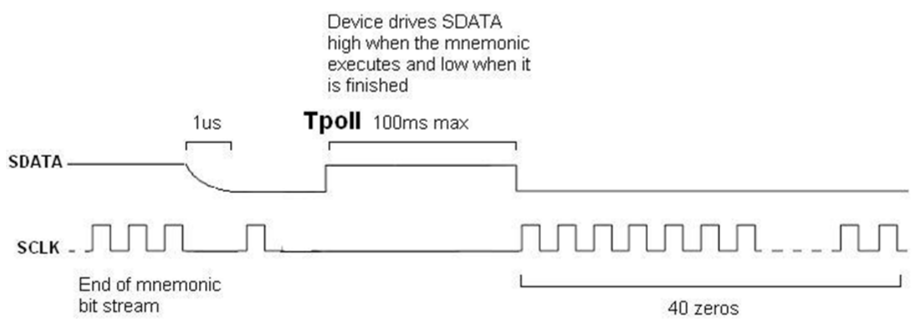


2.1.3 Wait and Poll

After a mnemonic bit stream is sent, the programmer clocks in a “Z” to the device (with enough set up time for the device SDATA pin to drift low to Vilp by the device’s internal pull-down resistor –- typically 1 µs). One SCLK clock cycle is needed before SDATA transitions from low to high. The SCLK is then held low. The target device pulls SDATA high when the mnemonic starts executing. The device outputs logic high on the SDATA pin while the mnemonic is executing and then switches to a logic low when the mnemonic finishes. The programmer must wait and poll the SDATA pin for the high to low transition. The maximum high time is 100 ms; this is the maximum time the programmer should wait for the operation to complete. WAIT-AND-POLL uses AC timing specification T_{poll} (from Table 2-2 on page 16).

When the transition to low is observed on the SDATA line, the programmer must apply a bit stream of 40 zero bits to the SDATA pin of the device and then continue to the next mnemonic. This is shown in Figure 2-3.

Figure 2-3. Wait and Poll Timing Diagram

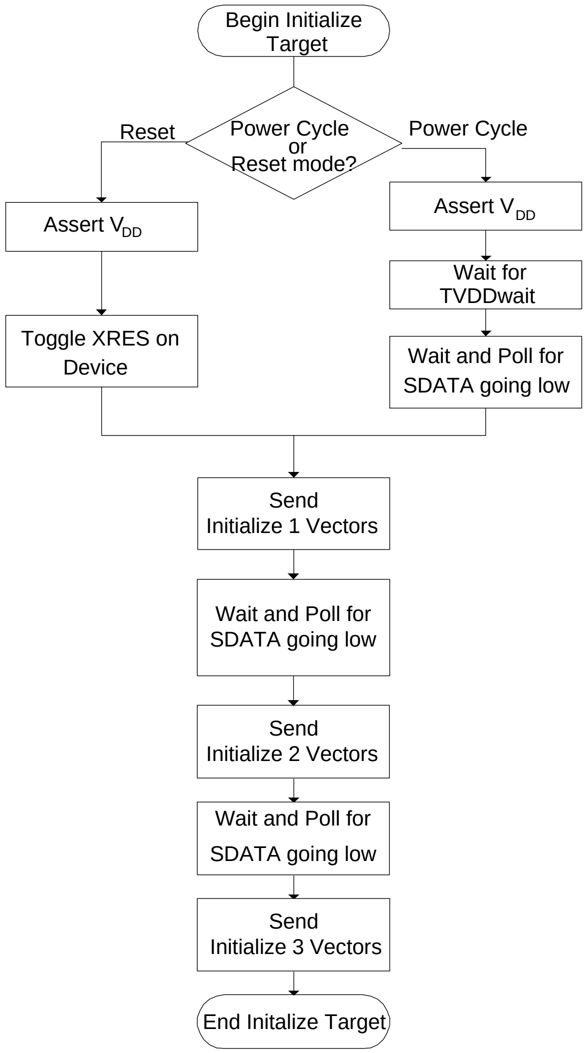


2.2 Initialize Target Procedure

The Initialize Target procedure places the chip into the programming mode. This is done by using Reset mode or Power Cycle mode.

Reset mode is preferred to initiate communication with the target. However, in the case of the 8-pin DIP package, there is no XRES pin and Power Cycle mode is the only option. Power Cycle mode involves cycling power to the target; therefore, in-circuit field programming may involve PCB layout considerations in the design phase.

Figure 2-4. Initialize Target Procedure



2.2.1 Reset Mode

The timing to enter programming mode with XRES is shown in Figure 2-5. To initialize the part using the XRES line, first wait until V_{DD} is stable, and then assert the XRES line for the time specified by T_{xres} (Table 2-2 on page 16). After XRES is driven low, there is a window of time specified by $T_{xresini}$ (Table 2-2 on page 16) in which the first nine bits of the Initialize 1 vector-set must be transmitted.

When the target executes the operation, it drives the SDATA line High. The programmer must wait and poll the SDATA line for a HIGH to LOW transition, which is the signal from the target that the Initialize 1 operation has completed.

Next, send Initialize 2 vectors, wait for HIGH to LOW transition on SDATA line, then send Initialize 3 vectors.

The programmer must sense the system supply and decide which Initialize 3 vectors to supply. If $V_{DD} \leq 3.6$ V, use one set; if $V_{DD} > 3.6$ V, use the other. See "Programming Vectors" on page 18.

2.2.2 Power Cycle Mode

To initiate communication with the target using power cycle mode, apply V_{DD} to the target, as shown in Figure 2-6 on page 11. This causes the target to attempt to drive the SDATA line High. The programmer then waits and polls for a HIGH to LOW transition on the SDATA line, which is the signal from the target that V_{DD} has stabilized. Note that until V_{DD} stabilizes, the SDATA signal is noisy and a false edge can be detected. As a result, the programmer must wait for the time specified by $T_{VDDwait}$ (Table 2-2 on page 16) before beginning to wait and poll. In addition, the programmer must not drive the SCLK signal until the $T_{VDDwait}$ time period has passed.

After the SDATA transition is detected, the programmer must transmit the Initialize 1 vectors in T_{acq} seconds (Table 2-2 on page 16).

Next, send Initialize 2 vectors and wait for HIGH to LOW transition on SDATA. Send the appropriate Initialize 3 vectors for the V_{DD} level applied to the PSoC when it is programmed.

During the power cycle phase of the Initialize Target Procedure, V_{DD} must be the only pin asserted. XRES must be low. The PSoC's internal pull down resistor accomplishes this if the pin is left floating externally.

Figure 2-5. Using Reset to Initialize

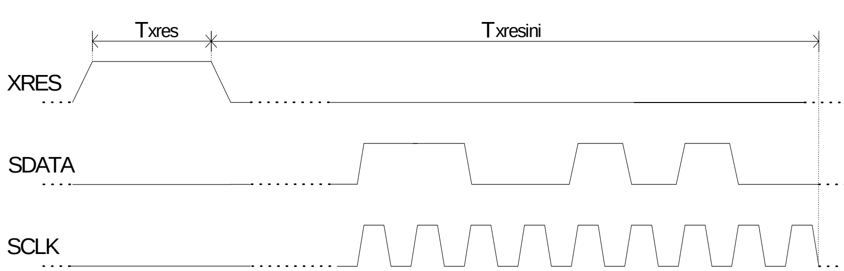
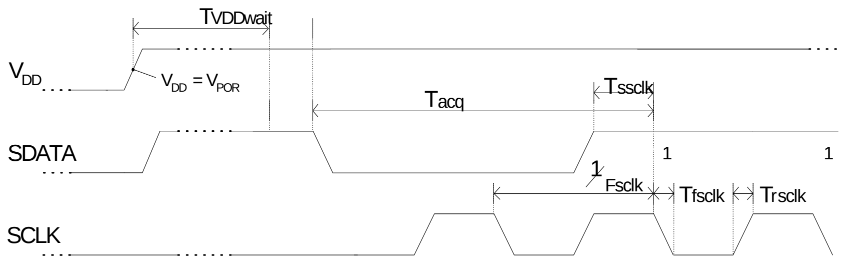


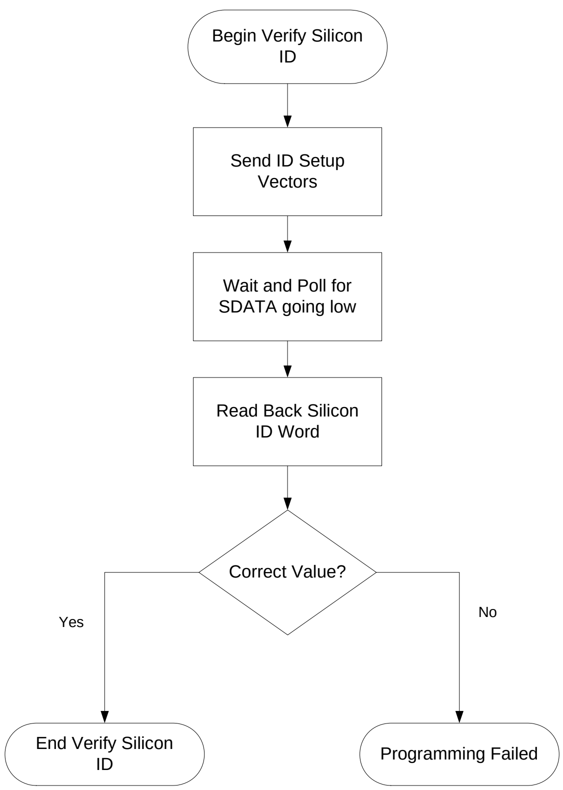
Figure 2-6. Using Power Cycling to Initialize



2.3 Verify Silicon ID Procedure

The Verify Silicon ID procedure (Figure 2-7) returns the package-specific silicon ID value from the target. This is used by the programmer to verify the package type of the target.

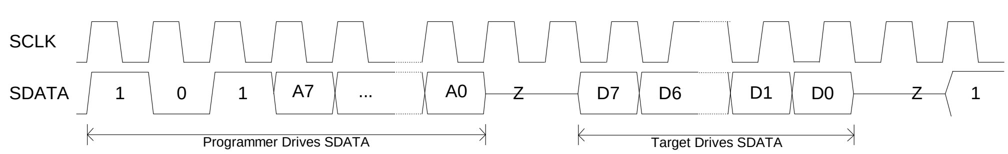
Figure 2-7. Verify Silicon ID Procedure



The first step in this procedure is for the programmer to send the ID-Setup vector-set. The programmer then drives the SDATA line into a High Z state. It waits and polls the SDATA line for a HIGH to LOW transition, which signifies that the target has executed the operation. The silicon ID value can then be read back by using the READ-ID-WORD vector-set. The sequence for a READ BYTE operation from the target at a specific address is shown in Figure 2-8. The READ-ID-WORD vector given in Programming Vectors on page 18 is based on this READ BYTE sequence with address replaced with specific SDATA value, and the data byte to be read replaced by the expected silicon ID byte. Two bytes must be read to obtain a complete silicon ID word.

The vectors in Programming Vectors on page 18 under READ-ID-WORD show the package-specific values read from the target, that is, a LLLLHLLH denotes a 0x09 hex read back from an 8-pin target (CY8C27143). The programmer must compare the value in the READ-ID-WORD vector (Programming Vectors on page 18) and the value returned by the target. If these values do not match, the programmer must terminate the programming flow.

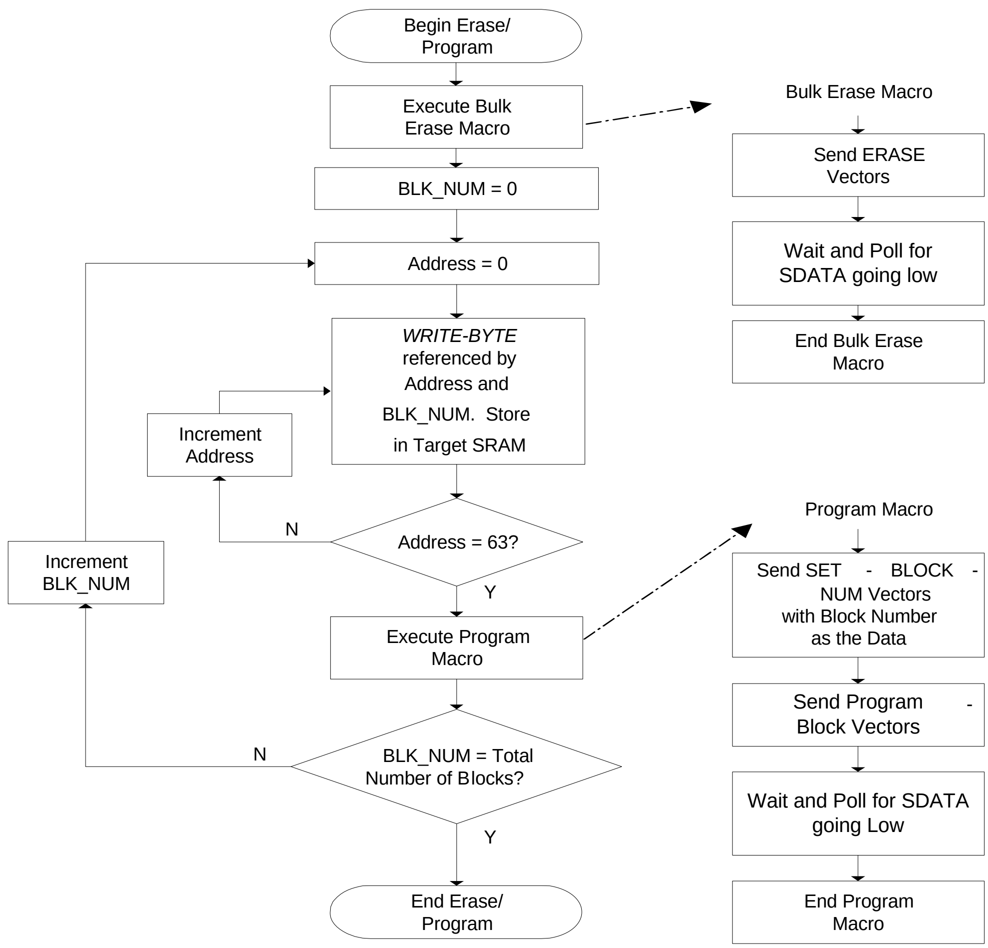
Figure 2-8. READ-BYTE (D7..D0) from Target PSoC 1 (At address A7..A0)



2.4 Program Procedure

The Program procedure is responsible for actual programming of the flash.

Figure 2-9. Program Procedure



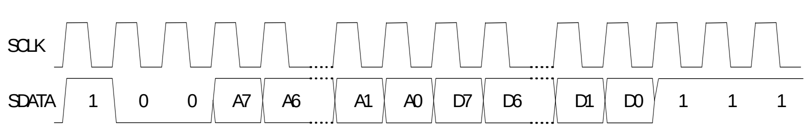
A Bulk Erase operation must be executed to prepare the flash for programming.

The ERASE vector-set is sent. As before, the programmer must wait and poll the SDATA line for a HIGH to LOW transition before continuing with the program procedure.

To place the actual program image into the flash, the program portion of the *.hex* (Programming Vectors on page 18) is read by the programmer in 64-byte blocks. This is written into the SRAM of the target one byte at a time using the WRITE-BYTE vector whose format is shown in Figure 2-10.

After the programmer completely writes the block into the target's SRAM, the block number to be written is set using the SET-BLOCK-NUM vector. Then the PROGRAM-BLOCK is sent. The PROGRAM-BLOCK vector executes a write block operation. Following the previous commands, the programmer must wait and poll the line before continuing. This loop is executed for each 64-byte block of the program image until the entire program is loaded into the flash. Note that data can only be written to flash in 64-byte blocks.

Figure 2-10. WRITE-BYTE (D7..D0) to Target PSoC 1 (At address A7..A0)



2.5 Verify Procedure

The Verify procedure (Figure 2-12) is responsible for verification of the programmed flash.

Flash must be verified to ensure program integrity. This procedure uses a loop to read back the same number of blocks programmed into the flash. To verify a block of flash, the SET-BLOCK-NUM vector (Programming Vectors on page 18) is first sent with the 'dddddddd' in the vector replaced with the block number to be read from flash.

The programmer sends the VERIFY-SETUP vector-set and then waits and polls the SDATA line. Each Read Block operation reads a 64-byte block from flash and stores the data in the target's SRAM starting at address 0x80. The programmer must then use the READ-BYTE vector with an offset of 0x80 (Figure 2-11) to individually read each byte in the block. After the programmer reads the block, the programming software must compare it with the block written to the flash. Data mismatch must terminate the programming flow as a failure.

Figure 2-11. READ-BYTE From Target for Verify Operation

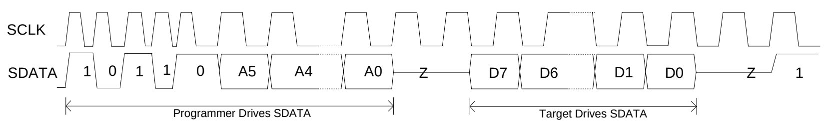
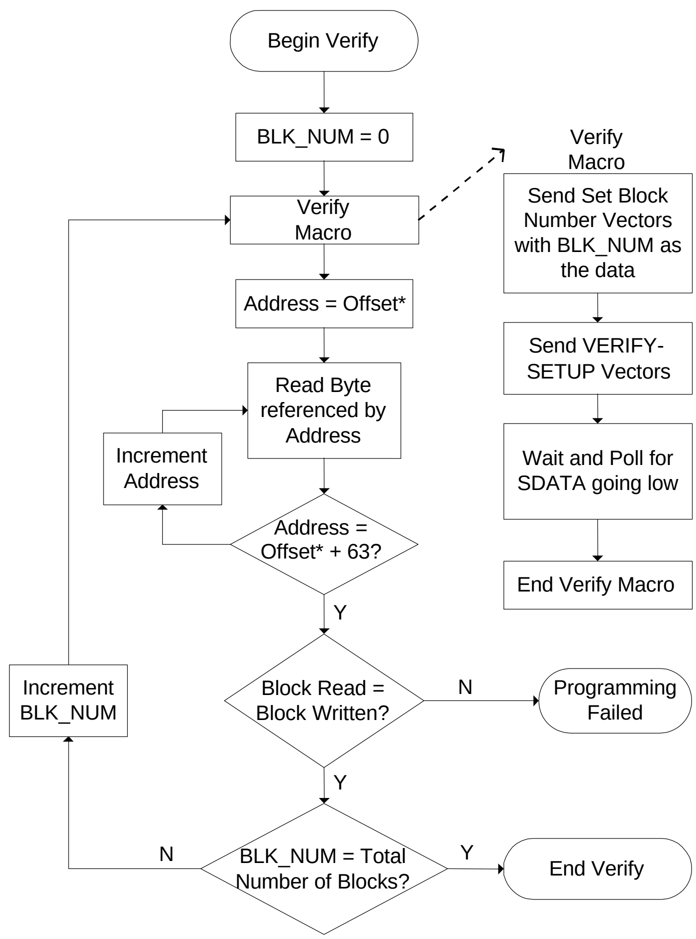


Figure 2-12. Verify Procedure (Offset = 0x80 in figure)

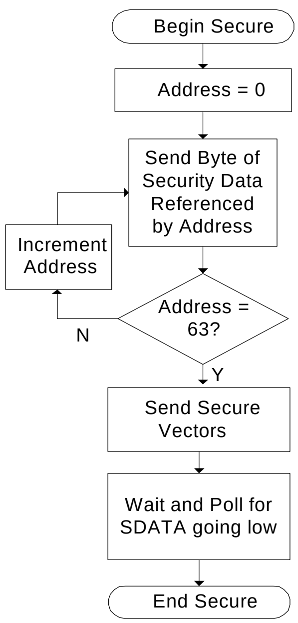


2.6 Secure Procedure

The Secure procedure (Figure 2-13), writes the user-determined security values to the target for each block.

After the flash is programmed and verified, each byte of the 64-byte security block is written to the target SRAM using the WRITE-BYTE vector. This block defines the access modes for each 64-byte block of the program image. After the 64-byte block is written to the target, the appropriate SECURE vector-set is sent to the target and the programmer waits and polls for the operation to execute. There are different SECURE vectors for different part numbers. The correct vector must be specified for the PSoC being programmed. The security data is located in the *.hex* file (see Intel.Hex File Format on page 22).

Figure 2-13. Secure Procedure



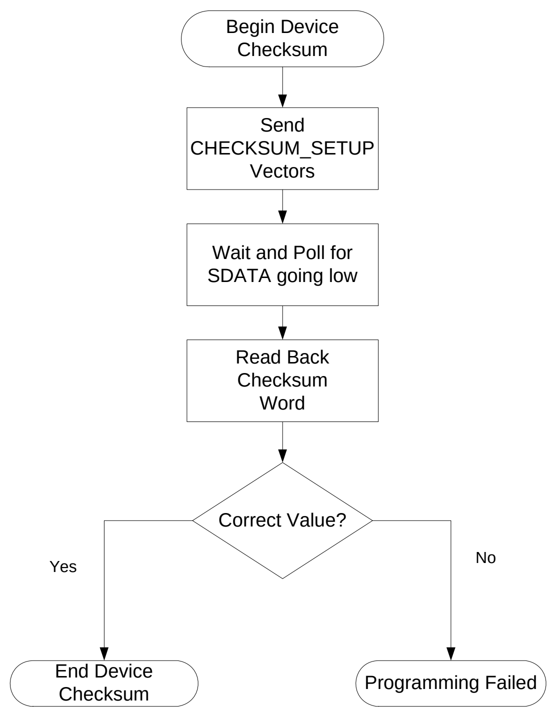
2.7 Verify Checksum Procedure

The Verify Checksum procedure (Figure 2-14), causes the target to generate a Checksum Value for the data in flash.

To get the Checksum Value from the target, the programmer sends the appropriate CHECKSUM-SETUP vector-set to the target. There are different CHECKSUM-SETUP vectors for different part numbers. The correct vector must be specified for the PSoC being programmed. The programmer releases the SDATA line then waits and polls. After the target signals that the operation is complete, the READ-CHECKSUM vector-set is used to read back the two-byte Checksum Value from the target. This value from the target is compared to the Device Checksum Value from the *.hex* file (see Intel.Hex File Format on page 22). If the values are not equal, a programming error has occurred.

To calculate a correct checksum, the entire flash must be programmed.

Figure 2-14. Verify Checksum Procedure



2.8 Erase Block Procedure

The Erase Block procedure is required only when it is necessary to erase a particular number of blocks of flash. This is needed to update a few blocks of flash for partial firmware updates. The Erase Block and Program Block vectors are sent by the host to the target device. Note that the Erase Block is not used or required in the general programming flow as shown in Figure 2-9 on page 12. This is because the Bulk Erase Macro is used, which erases all the blocks of flash. While the Bulk Erase function can be used to erase the entire flash data along with protection settings any time, the Erase Block function can be executed only if the Write protection feature for that particular block is turned off.

Figure 2-15. Erase Block Procedure

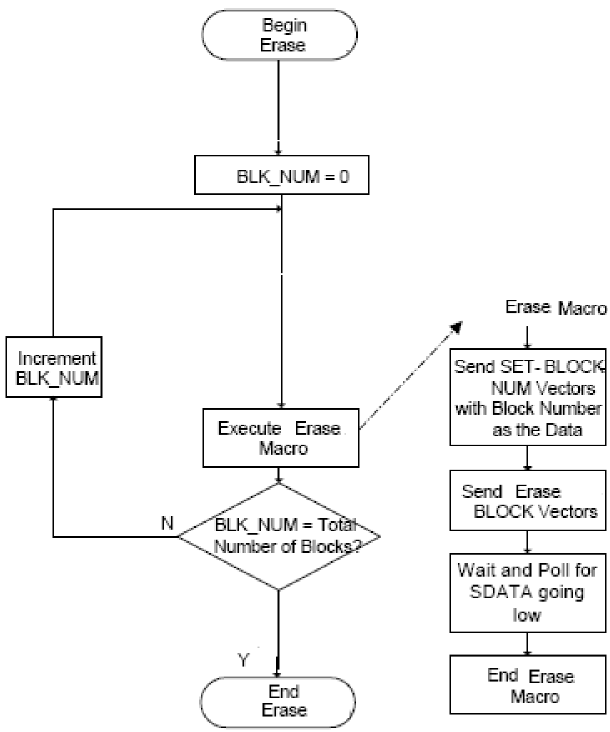


Figure 2-15 initializes with the starting block number (zero in the figure) and iterates for required number of blocks (all blocks are erased one by one).

2.9 Specifications and Definitions

2.9.1 DC Programming Specifications

Table 2-1. DC Programming Specifications

DC Programming Specifications	Minimum	Maximum
I_{DDp} (Supply current during programming or verify)	See the DC Programming Specifications section in the respective device datasheet	
V_{ilp} (Input low voltage during programming or verify)		
V_{ihp} (Input high voltage during programming or verify)		
I_{ilp} (Input current when applying V_{ilp} to P1[0] or P1[1] during programming or verify)		
I_{ihp} (Input current when applying V_{ihp} to P1[0] or P1[1] during programming or verify)		
V_{olv} (Output low voltage during programming or verify I_{OL} = 0.1 mA)		
V_{ohv} (Output high voltage during programming or verify I_{OH} = 5 mA)		
V_{ddp} (V_{DD} for programming and erase)		
V_{dd} (V_{DD} for verify)		
V_{ipor} (Power on reset trip)	See the DC POR and LVD Specifications section in the respective device datasheet	

2.9.2 AC Programming Specifications

Table 2-2. AC Programming Specifications

AC Programming Specifications	Minimum	Maximum
T_{rsclk} (Rise Time of SCLK)	See the AC Programming Specifications section in the respective device datasheet	
T_{fsclk} (Fall Time of SCLK)		
T_{ssclk} (Data set-up time to falling edge of SCLK)		
T_{hsclk} (Data hold time From falling edge of SCLK)		
F_{sclk} (Frequency of SCLK)		
T_{dsclk} (Data-out delay from falling edge of SCLK)		
$T_{vddwait}$ (V_{DD} stable to WAIT-AND-POLL hold off[1])	0.1 ms	1 ms
T_{poll} (SDATA High pulse time[2])	10 µs	100 ms
T_{acq} (Delay from WAIT-AND-POLL to initialize-1[3])	–	3 ms
T_{xres} (Duration of external reset)	See the AC Chip Level Specifications section in the respective device datasheet	
$T_{xresini}$ (Programming mode acquisition window)	–	125 µs

Notes

1. Until V_{DD} stabilizes, SDATA is noisy and the falling edge must not be searched for. Therefore, a delay of $T_{vddwait}$ is needed after V_{DD} is applied and before WAIT-AND-POLL.
2. This applies to WAIT-AND-POLL mnemonic. The SDARA remains high for T_{poll} time.
3. The Initialize-1 bit stream data must not be delayed more than T_{acq} from the end of the WAIT-AND-POLL (measured from SDATA's falling edge).

2.9.3 Device Address and Block Definitions

Table 2-3. Device Address and Block Definitions

Device	Address Numbers (Bytes Within a Block)	Block Numbers (Program Data)	max_data_block
CY8C21123	0–63	0–63	63
CY8C21223	0–63	0–63	63
CY8C21323	0–63	0–63	63
CY8C21234	0–63	0–127	127
CY8C21334	0–63	0–127	127
CY8C21434	0–63	0–127	127
CY8C21534	0–63	0–127	127
CY8C21634	0–63	0–127	127
CY8C23033	0-63	0-127	127
CY8C23433	0-63	0-127	127
CY8C23533	0-63	0-127	127
CY8C24123A	0–63	0–63	63
CY8C24223A	0–63	0–63	63
CY8C24423A	0–63	0–63	63
CY8C27143	0–63	0–255	255
CY8C27243	0–63	0–255	255
CY8C27443	0–63	0–255	255
CY8C27543	0–63	0–255	255
CY8C27643	0–63	0–255	255
CY8CTMG110	0–63	0–127	127
CY8CTST110	0–63	0–127	127

A. Appendix

A.1 Programming Vectors

Table A-1. Programming Vectors for CY8C21x12, CY8C21x23, CY8C21x34, CY8C23x33, CY8C24x23A, CY8C27x43, CY8CTMG110, CY8CTST110

Name	Data
Vector	Bit Stream (Executed From Left Bit to Right)
Initialize-1	11001010000000000000000000000000000000000000 00000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000 11011110111000000001111101111011000000000111 10011111000001110101111001111100100000011111 11011110101000000001111101111010000000011111 10011111011100000001111101111100100110000111 11011111010010000001111101111100000001001111 11011111000000000001111101111111100010010111
Initialize-2	11011110111000000001111101111011000000000111 10011111000001110101111001111100100000011111 11011110101000000001111101111010000000011111 10011111011100000001111101111100100110000111 11011111010010000001111001111101000000001111 11011111000000001101111101111110000000000111 1101111111100010010111

Table A-1. Programming Vectors for CY8C21x12, CY8C21x23, CY8C21x34, CY8C23x33, CY8C24x23A, CY8C27x43, CY8CTMG110, CY8CTST110 *(continued)*

Name	Data
Initialize-3 3V	11011110111000000000111110111101000000000011111 11011110101000000000111110111101100000001000111 110111110000101000111111011111001111111000111 110111110100011000011111011111111100010010111 0000000000000000000000110111101110000000000111 110111101000000000111111011110101000000000111 110111101100000100011111011111000011000000111 110111110011110101011111011111010001100000111 110111101110001000011111011111111100010010111 0000000000000000000000110111101110000000000111 110111101000000000111111011110101000000000111 110111101100000100011111011111000010100011111 110111110011111100111111011111010001100000111 11011111111100010010111000000000000000000000000 11011110111000000000111110111101000000000011111 11011110101000000000111110111101100000001000111 110111110000110000011111011111001111101000111 110111110100011000011111011110111000100000111 11011111111100010010111000000000000000000000000
Initialize-3 5V	11011110111000000000111110111101000000000011111 11011110101000000000111110111101100000001000111 110111110000101000111111011111001111111100111 110111110100011000011111011111111100010010111 0000000000000000000000110111101110000000000111 110111101000000000111111011110101000000000111 110111101100000100011111011111000011000000111 110111110011110101011111011111010001100000111 110111101110001000011111011111111100010010111 0000000000000000000000110111101110000000000111 110111101000000000111111011110101000000000111 110111101100000100011111011111000010100011111 110111110011111110111111011111010001100000111 11011111111100010010111000000000000000000000000 11011110111000000000111110111101000000000011111 11011110101000000000111110111101100000001000111 110111110000110000011111011111001111101000111 110111110100011000011111011110111000100000111 11011111111100010010111000000000000000000000000
ID-SETUP	11011110111000100001111101110000000000000010111 11011110111000000000111110111101100000000000111 10011111000001110101111001111100100000011111 11011110101000000000111110111101000000000011111 10011111011100000000111110111110010011000001111 110111110100100000001111001111101000000000111 110111100000000011011111011111100000000000111 110111111111000100010111
	READ-ID-WORD (CY8C27143) 10111111000ZLLLLLLLLZ110111111001ZLLLLHLLHZ1
	READ-ID-WORD (CY8C27243) 10111111000ZLLLLLLLLZ110111111001ZLLLLHLHLZ1
	READ-ID-WORD (CY8C27443) 10111111000ZLLLLLLLLZ110111111001ZLLLLHLHHZ1
	READ-ID-WORD (CY8C27543) 10111111000ZLLLLLLLLZ110111111001ZLLLLHHLLZ1
	READ-ID-WORD (CY8C27643) 10111111000ZLLLLLLLLZ110111111001ZLLLLHHLHZ1
	READ-ID-WORD (CY8C24123A) 10111111000ZLLLLLLLLZ110111111001ZLLHHLLHLZ1
	READ-ID-WORD (CY8C24223A) 10111111000ZLLLLLLLLZ110111111001ZLLHHLLHHZ1
	READ-ID-WORD (CY8C24423A) 10111111000ZLLLLLLLLZ110111111001ZLLHHLHLLZ1
	READ-ID-WORD (CY8C23533) 10111111000ZLLLLHLLLZ110111111001ZHLHHLLLHZ1

Table A-1. Programming Vectors for CY8C21x12, CY8C21x23, CY8C21x34, CY8C23x33, CY8C24x23A, CY8C27x43, CY8CTMG110, CY8CTST110 *(continued)*

Name	Data
	READ-ID-WORD (CY8C23433) 10111111000ZLLLLHLLLZ110111111001ZHLHHLLLLZ1
	READ-ID-WORD (CY8C23033) 10111111000ZLLLLHLLLZ110111111001ZHLHHLLHLZ1
	READ-ID-WORD (CY8C21123) 10111111000ZLLLLLLLLZ110111111001ZLLLHLHHHZ1
	READ-ID-WORD (CY8C21223) 10111111000ZLLLLLLLLZ110111111001ZLLLHHLLLZ1
	READ-ID-WORD (CY8C21323) 10111111000ZLLLLLLLLZ110111111001ZLLLHHLLHZ1
	READ-ID-WORD (CY8C21234) 10111111000ZLLLLLLLLZ110111111001ZLLHHLHHLZ1
	READ-ID-WORD (CY8C21312) 10111111000ZLLLLHLLLZ110111111001ZLLHHLHHHZ1
	READ-ID-WORD (CY8C21334 and CY8C21334W) 10111111000ZLLLLLLLLZ110111111001ZLLHHLHHHZ1
	READ-ID-WORD (CY8C21434) 10111111000ZLLLLLLLLZ110111111001ZLLHHHLLLZ1
	READ-ID-WORD (CY8C21512) 10111111000ZLLLLHLLLZ110111111001ZLHLLLLLLZ1
	READ-ID-WORD (CY8C21534 and CY8C21534W) 10111111000ZLLLLLLLLZ110111111001ZLHLLLLLLZ1
	READ-ID-WORD (CY8C21634) 10111111000ZLLLLLLLLZ110111111001ZLHLLHLLHZ1
	READ-ID-WORD (CY8CTMG110-32LTXI) 10111111000ZLLLLLHHHZ110111111001ZLLHHHLLLZ1
	READ-ID-WORD (CY8CTMG110-00PVXI) 10111111000ZLLLLLHHHZ110111111001ZLLHHHLLHZ1
	READ-ID-WORD (CY8CTST110-32LTXI) 10111111000ZLLLLLHHLZ110111111001ZLLHHHLLLZ1
	READ-ID-WORD (CY8CTST110-00PVXI) 10111111000ZLLLLLHHLZ110111111001ZLLHHHLLHZ1
SET-BLOCK-NUM	10011111010dddddddd111 where dddddddd=block #
BULK ERASE	10011111100000101011111001111111001010110111 11011110111000000001111101111011000000000111 10011111000001110101111001111100100000011111 11011110101000000001111101111010000000011111 10011111011100000001111101111100100110000111 11011111010010000001111101111000000000101111 11011111000000000001111101111111100010010111
WRITE-BYTE	10010aaaaaadddddddd111 where dddddddd= data in, aaaaaa=address (6 bits)
PROGRAM-BLOCK (CY8C21x12, CY8C21x23, CY8C21x34, CY8C23x33, CY8C24x23A, CY8CTMG110, CY8CTST110)	10011111100010101001111001111111001010110111 11011110111000000001111101111011000000000111 10011111000001110101111001111100100000011111 11011110101000000001111101111010000000011111 10011111011100000001111101111100100110000111 11011111010010000001111101111000000000010111 11011111000000000001111101111111100010010111
PROGRAM-BLOCK (CY8C27x43)	10011111100000101011111001111111001010110111 11011110111000000001111101111011000000000111 10011111000001110101111001111100100000011111 11011110101000000001111101111010000000011111 10011111011100000001111101111100100110000111 11011111010010000001111101111000000000010111 11011111000000000001111101111111100010010111

Table A-1. Programming Vectors for CY8C21x12, CY8C21x23, CY8C21x34, CY8C23x33, CY8C24x23A, CY8C27x43, CY8CTMG110, CY8CTST110 *(continued)*

Name	Data
VERIFY-SETUP	1101111011100000000111110111101100000000000111 1001111100000111010111100111110010000000011111 1101111010100000000111110111101000000000011111 1001111101110000000111110111110010011000000111 1101111101001000000111110111100000000000001111 1101111100000000000111110111111110001001010111
READ-BYTE	10110aaaaaaZDDDDDDDDZ1 where DDDDDDDD= data out, aaaaaa=address (6 bits)
SECURE	1001111110001010100111100111111110010101100111 1101111011100000000111110111101100000000000111 1001111100000111010111100111110010000000011111 1101111010100000000111110111101000000000011111 1001111101110000000111110111110010011000000111 1101111101001000000111110111100000000000100111 1101111100000000000111110111111110001001010111
CHECKSUM-SETUP (CY8C21x12, CY8C21x23, CY8C21x34, CY8C23x33, CY8C27x43, CY8CTMG110, CY8CTST110)	1101111011100000000111110111101100000000000111 1001111100000111010111100111110010000000011111 1101111010100000000111110111101000000000011111 1001111101110000000111110111110010011000000111 1101111101001000000111100111110100000000000111 1101111000000000011111111011111000000000000111 1101111111100010010111
CHECKSUM-SETUP (CY8C24x23A)	1101111011100000000111110111101100000000000111 1001111100000111010111100111110010000000011111 1101111010100000000111110111101000000000011111 1001111101110000000111110111110010011000000111 1101111101001000000111100111110100100000000111 1101111000000000011111111011111000000000000111 1101111111100010010111
READ-CHECKSUM	10111111001ZDDDDDDDDZ110111111000ZDDDDDDDDZ1 where DDDDDDDDDDDDDDDD= Device Checksum data out
ERASE BLOCK	1001111110001010100111100111111110010101100111 1101111011100000000111110111101100000000000111 1001111100000111010111100111110010000000011111 1101111010100000000111110111101000000000011111 1101111100100110000111110111110100100000000111 1101111000000000011111111011111000000000000111 1101111111100010010111

Notes

1 = Logic high = Vihp

0 = Logic low = Vilp

Z = High Z (floating)

D = Data read from device (Most Significant Bit [MSb] of binary data comes out first)

d = Data applied to the device (MSb of the binary data goes in first)

a = Address applied to the device (MSb of the binary data goes in first)

H = High data read from the device (Vout = Vohv)

L = Low data read from the device (Vout = Volv)

If the programmer has delays between executing the different mnemonics, SDATA must be High Z (floating) during these delays.

Note Cypress does not recommend sharing ISSP bus lines of CY8C20x36/46A/66A/96A/CY8CTMG2xx/CY8CTST2xx parts with other PSoC devices. However, when the ISSP bus of CY8C20x36/46A/66A/96A/CY8CTMG2xx/CY8CTST2xx parts are shared with other PSoC devices, take care to avoid CY8C20x36/46A/66A/96A/CY8CTMG2xx/CY8CTST2xx parts seeing key 'AC52' in reset state. See the knowledge base article http://www.cypress.com/?id=4&rID=45442 for details.

A.2 Intel.Hex File Format

The *Intel.hex* file format for CY8C21x12, CY8C21x23, CY8C21x34, CY8C23x33, CY8C24x23A, CY8C27x43, CY8CTMG110, CY8CTST110 is discussed in this section.

Intel.hex file records are a text representation of hexadecimal coded binary data. Only ASCII characters are used and the format is portable across almost all computer platforms.

PSoC Designer generates this file and stores it under the <PROJECT_DIR>/OUTPUT directory.

Each line in an *Intel.hex* file is called a 'record'.

Each line (record) of *Intel.hex* file consists of six parts:

Start code (Colon character)	Byte count (1 byte)	Address (2 bytes)	Record type (1 byte)	Data (N bytes)	Checksum (byte)

The flash program data and end data are made up of a single record. The security data and checksum data are made up of multiple records. These data each have an extended linear address record and one or more data records. Records always begin with a colon (:), followed by the number of data bytes in each record. For the devices, flash program data records always use 64 bytes of data so the hexadecimal value in the file is always 0x40 for that type.

For flash programming data records, the next pair of numbers represent the 16-bit starting address of the data in the record. This is the absolute location in the flash memory. This number must be a multiple of 64 (0x00, 0x40, 0x80, 0xC0,…) for flash program data records because each record contains 64 bytes.

The starting address is followed by a byte representing the record type. If this is 0x00, the next bytes are the actual program data to be stored in flash. A 0x01 indicates that this is the end of the file. A 0x04 indicates an “Extended Linear Address Record” and is used for security data and device checksum data storage (see the following examples).

The security and checksum data use multiple records because they have longer addresses than the other data. The first record, the Extended Linear Address Record, gives the upper bytes of the address of the data in memory. The other records give the lower bytes of the address along with the data.

Following the record type are the Hexadecimal representations of the data to be stored. The last byte is a checksum, which is the least significant byte of the two’s complement of the sum of the values of all fields except the colon field – this is called the record checksum. This is called the record checksum. Note that this value is derived from the binary values of the bytes rather than the ASCII representation.

Typically, a standard CR/LF pair (carriage return/linefeed, 0x0D 0x0A) terminates the record. Other end-of-line conventions are also acceptable (like CR only).

A.2.1 Example Flash Program Data Record

```
:4000C000505152535455565758595A5B5C5D5E5F00000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000E8(CR/LF)
Broken down, it is as follows:
:          - Colon, indicates that this is IntelHex
40         - Number of data bytes to follow = 0x40(40 hex)
00C0       - Starting address in the FLASH for record.
00         - This is the record type -- 0x00 = Data
505152535455565758595A5B5C5D5E5F00000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000
           These are 64 bytes of data in hex as noted above. The
           first byte (0x50) will be stored at 0x00C0, with the
           remaining bytes following in sequence.
E8         - This is record checksum. If you add all of successive bytes
           (note that the address is treated as two individual
           bytes), and truncate it to the lowest eight bits, the
           result is 0x18. The two's complement of 0x18 is 0xE8.
           (This may be derived by subtracting 0x18 from 0x100, or
           by inverting the bits and adding one to the result.)
(CR/LF)    - End of this record.
```

A.2.2 Example Security Data Records

```
:020000040010ea(CR/LF)
:400000005555555555555555555555555555555555555555555555555555555555555555555555555555555555
5555555555555555555555555555555555555555555580(CR/LF)

:         - Colon, indicates IntelHex
02        - Number of data bytes – 2 bytes of data
0000      - Address - zero
04        - This is the record type -- 0x04 indicates
                 Extended Linear Address record
0010      - 2 hex data bytes used – here byte 1 has 0x00,
                 byte 2 has 0x10 data.
                 This indicates that the
                 security data is offset in memory space
                 (0x0010 is used for security data).
ea        - The record checksum, calculated as above.
(CR/LF)   - End of this record.

:         - Colon, indicates that this is IntelHex
40        - Number of data bytes – 64 bytes
0000      - Address – zero
00        - Record type - 0x00 indicates data record
5555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555
5555555555555555555555555555555555555
          - 64 data bytes – here bytes have 0x55 data
80        - The record checksum, calculated as above.
(CR/LF)   - End of this record.
```

Additional Notes on Security Records

The security data must be in the file after all flash program data records are specified.

As seen in the previous example, security data use multiple records (one to access the extended memory space, and the others for data). There is one security data record for every 256 blocks of flash. For devices with under 256 blocks of flash, the record is still 64 bytes long. The most significant bytes are used, and the remainder are ignored. The extended linear address record that precedes the security data record always specify the same data, and as a result, always have the same checksum. This record can be copied from a known good hex file.

The data of the security data record indicates the flash security settings specified in PSoC Designer, in flashsecurity.txt. Each letter in flashsecurity.txt indicates the security settings for one block of flash space. Each letter is encoded into two bits of a hex digit in the security data record. Four block settings are concatenated into two digits of data, in reverse order. The encoding may be further examined by changing flashsecurity.txt and generating hex files.

A.2.3 Example Device Checksum Data Records

```
:020000040020da(CR/LF)
:02000000253a9f(CR/LF)

:         - Colon, indicates that this is IntelHex
02        - Number of data bytes – 2 bytes of data
0000      - Address - zero
04        - This is the record type -- 0x04 indicates
                 Extended Linear Address record
0020          - 2 hex data bytes used – here byte 1 has 0x00,
                     byte 2 has 0x20 data.
                     This indicates that indicates that the checksum
                     data is offset in memory space (0x0020 is use
                     for checksum data).
da        - The record checksum, calculated as above.
(CR/LF)       - End of this record.

:         - Colon, indicates that this is IntelHex
02        - Number of data bytes – 2 bytes of data
```

```
0000       - Address - zero
00         - Record type -- 0x00 indicates data record
253a       - 2 hex data bytes used – here byte 1 has 0x25, byte 2 has 0x39 data. The data is
             a 2 byte checksum of all of the data stored in flash.
9f         - The record checksum, calculated as above.
             (CR/LF)- End of this record.
```

Additional Notes on Device Checksum Data Records

The Device Checksum data must be in the file after all security data records are specified.

As seen in the previous example, Device Checksum data use two records (one to access the extended memory space, and the other for data). The extended linear address record that precedes the checksum data record always specifies the same data, and as a result, always has the same checksum. This record can be copied from a known good hex file.

A.2.4 End Record (End of File)

```
:00000001FF(CR/LF)

:          - Colon, indicates that this is IntelHex
00         - Number of data bytes - zero
0000       - Address - zero
01         - Record type -- 0x01 indicates end record,
           - no data bytes used
FF         - The record checksum, calculated as above.
(CR/LF)    - End of this record.
```

A.2.5 Device Address and Block Definitions

The least significant 6 bits in the IntelHex address define the byte address (0 to 63) within a block. The most significant bits in the IntelHex address define the block number. See Table 2-3 on page 17.