CY8C21x45, CY8C22x45, CY8C24x94, CY8C28xxx, CY8C29x66, CY8CTST120, CY8CTMA120, CY8CTMG120, CY7C64215

PSoC® 1 ISSP Programming Specifications

Document No. 001-15239 Rev. *L

October 5, 2015

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

Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
www.cypress.com

License

Copyrights

Disclaimer

Flash Code Protection

Cypress products meet the specifications contained in their particular Cypress Data Sheets. Cypress believes that its family of PSoC products is one of the most secure families of its kind on the market today, regardless of how they are used. There may be methods that can breach the code protection features. Any of these methods, to our knowledge, would be dishonest and possibly illegal. Neither Cypress nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Cypress is willing to work with the customer who is concerned about the integrity of their code. Code protection is constantly evolving. We at Cypress are committed to continuously improving the code protection features of our products.

Contents

1. Overview

This document is the programming specifications manual for the PSoC® 1 device families: CY8C21x45, CY8C22x45, CY8C24x94, CY8C28xxx, CY8C29x66, CY8CTST120, CY8CTMA120, CY8CTMG120, CY7C64215. This reference manual provides information about the hardware connections and programming vectors required to develop your own PSoC 1 programmers. PSoC 1 can be programmed using the In-System Serial Programming (ISSP) protocol. Refer to the other programming specifications documents for information about how to program the rest of the PSoC 1 devices.

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 (ISSP) protocol, a proprietary protocol used by Cypress.

This programming 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. Refer to the application note AN44168 for a practical implementation with source code of the Host Programming solution. Refer to the General PSoC Programming web page 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, supplies the clocking, and sends commands to the target. The target receives data from the programmer and supplies data upon a read request. The target 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. Refer to Appendix B on page 27 for more information regarding the *Intel.hex* file format.

There are two important points that you should remember while developing a Host Programming application. These are:

1. You should not compare the programming vectors provided in this application note 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 for programming the target device. The programming vectors provided in this application note are the recommended ones for developing your own host-side interface to program a PSoC 1 device.
2. Even though the ISSP protocol uses a bidirectional data line for communication between the host and the target device, it is not related to I^2C protocol in any manner.

1.2 Document History

Document Title: CY8C21x45, CY8C22x45, CY8C24x94, CY8C28xxx, CY8C29x66, CY8CTST120, CY8CTMA120, CY8CTMG120, CY7C64215 PSoC® 1 ISSP Programming Specifications

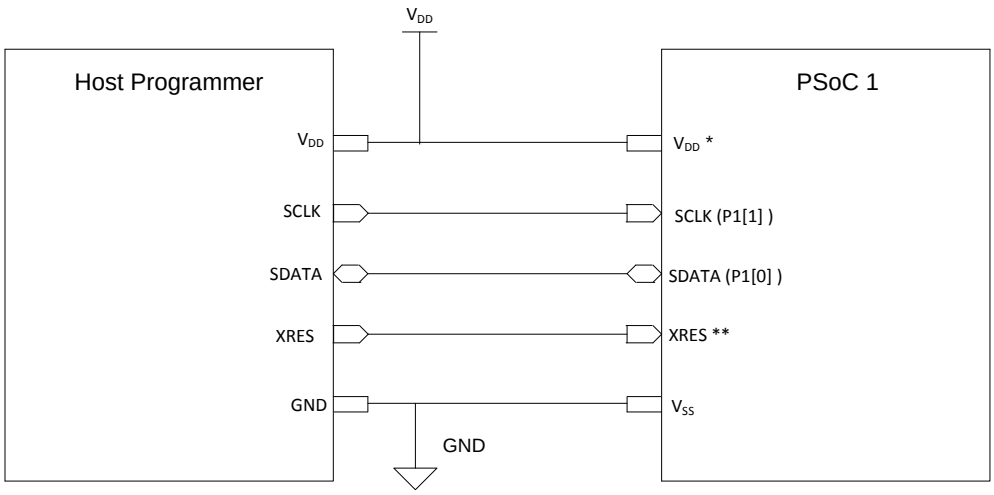
Document Number: 001-15239

Revision	Orig. of Change	Submission Date	Description of Change
**	HMT	09/17/2007	Converted to new template.
*A	MAXK/AESA	07/31/2008	Updated Power Cycle Mode on page 12. Updated Appendix B on page 27. Converted to latest application note template.
*B	FKL/PYRS	02/16/09	Updated CY8C28xxx chip family information. Added CY8C28xxx to Table 5.
*C	MAXK/AESA	10/26/09	Added CY8CTST120, CY8CTMA120, and CY8CTMG120 devices.
*D	ROBC	11/17/09	Added CY8C21345 and CY8C22x45 devices
*E	MAXK	10/01/2010	Updated Figure 3-4 on page 11, Figure 3-7 on page 13, Figure 3-9 on page 14, Figure 3-12 on page 16, Figure 3-13 on page 17, and Figure 3-15 on page 19 Updated content in 3.1.3 Wait and Poll on page 10 section. Updated Text in 3.5 Secure Procedure on page 17 section Updated Table A-1 on page 23.
*F	VVSK	12/06/2010	Updated 1.1 Introduction on page 5. Updated 3.2.1 Reset Mode on page 12 and 3.2.2 Power Cycle Mode on page 12. Added 3.8 Erase Block Procedure on page 19. Updated Table A-1 on page 23. Updated Appendix B on page 27.
*G	VVSK	03/02/2011	Updated Associated Application Notes in page 1 as AN2026a, AN2026c, AN2026d, AN44168, AN59389. Updated Abstract. Updated 1.1 Introduction on page 5. Added Host Programmer - PSoC® 1 Programming Interface chapter on page 7. Added 2.2 Using an External Crystal Oscillator on page 8. Added 2.3 Pin Loading Requirements on page 8. Renamed the section Clocking as 3.1.2 Clocking, Data Format, and Timing Diagrams on page 10 under Programming Flow on page 9 and updated the same section. Deleted the section Command Format under Programming Flow on page 9. Added the section 3.1.3 Wait and Poll on page 10 under Programming Flow on page 9. Updated the sections 3.2.2 Power Cycle Mode on page 12, 3.2.3 Verify Silicon ID Procedure on page 13 under See “Initialize Target Procedure” on page 11.. Updated 3.3 Program Procedure on page 14. Updated Appendix A on page 23.
*H	VVSK	05/19/2011	Changed Title. Changed Associated Part Family. Changed Associated Application Notes. Modified Abstract. Updated 1.1 Introduction on page 5. Updated Host Programmer - PSoC® 1 Programming Interface chapter on page 7. Updated Table 4-3 on page 22. Updated Appendix A on page 23 and Table A-1 on page 23. Updated Appendix B on page 27.
*I	VVSK	08/29/2011	Minor changes throughout document. Changed to TRM format
*J	RJVB	04/19/2012	Updated Table 4-1 and Table 4-2
*K	RJVB	10/08/2014	No technical updates. Completing Sunset Review.
*L	RJVB	10/05/2015	Updated Figure 3-1. Added 3.6 Verify Secure Procedure on page 17. Updated Table A-1 with the VERIFY-SECURE=SETUP vector.

2. Host Programmer - PSoC® 1 Programming Interface

Figure 2-1 shows the connections between the host programmer and the target PSoC® 1 device. If you use a Miniprog1 programmer, refer to the knowledge base article at www.cypress.com/?id=4&rID=50010 for information about the part number of the Miniprog1 programming header.

Figure 2-1. Host Programmer - PSoC 1 Interface



* For Programming in Power Cycle mode, the Host Programmer must be able to toggle power to the PSoC 1 device.

** XRES pin in PSoC 1 is an 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 datasheet for information on XRES pin availability. Use Power Cycle mode if XRES is not available.

2.1 Programming Pin Drive Modes

The electrical pin connections between the programmer and the target shown in Figure 2-1 are listed in Table 2-1. 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 2-1.

Table 2-1. 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 the PSoC 1 drives the SDATA line to indicate it has started up completely or to send data back to the host, SDATA is in a strong drive configuration. When it waits for data or receives data from the 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 pin can be driven both high and low when it is in resistive pull-down mode. Because there is an internal pull-down resistor (5.6 kΩ) on the SDATA line, the presence of external pull-up resistors on the SDATA line might cause the host to miss the high-to-low transition on the target device. This is caused by the resistive voltage divider. You should not use external pull-up resistors on the SDATA line for this reason.

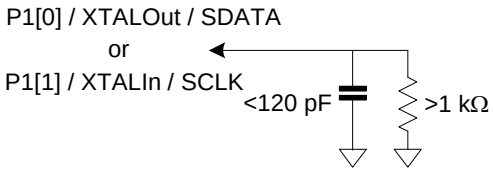
2.2 Using an External Crystal Oscillator

The programming pins on PSoC 1 (SCLK (P1[1], SDATA (P1[0]) are also shared by the external 32-kHz crystal. If your design uses the external 32-kHz crystal, 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.) Too much trace length on these signals could adversely affect the operation of the oscillator. During programming, the 32-kHz crystal loading does not add loading to the programming pins.

2.3 Pin Loading Requirements

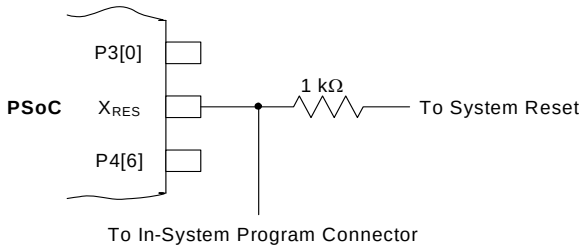
The SDATA and the SCLK pins each have three functions. These pins are configurable as an external 32-kHz crystal, I^2C interface pins, and as general-purpose IO pins.
The equivalent load on these pins should not exceed 120 pF in parallel with a 1-kΩ resistor.

Figure 2-2. Maximum Load Data and SCLK Pins



The XRES signal is a single-function pin. You should connect this signal 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 you can place a resistor in series with the signal source and the XRES pin. The programmer is then connected on the pin side of the register. See Figure 2-3. This allows the programmer to overdrive the XRES pin.

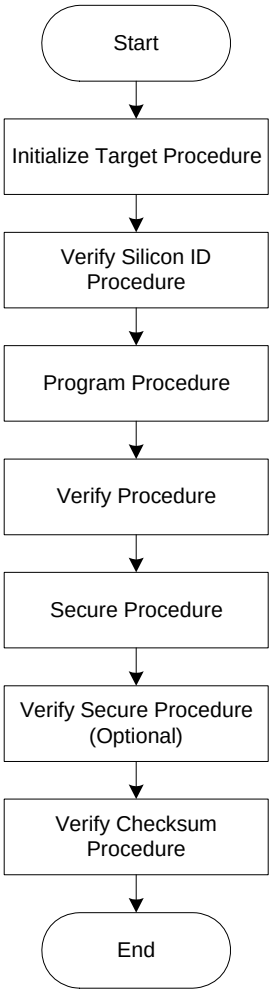
Figure 2-3. XRES Connection



3. Programming Flow

To make target programming successful, use the steps in the programming flow shown in Figure 3-1. Each procedure is explained in the following sections. If you do not complete these steps the flash can be programmed incorrectly.

Figure 3-1. Target Programming Flow



3.1 Programming Concepts

3.1.1 Vectors

Vectors are the binary representation of the commands necessary to perform the various operations involved in the programming flow. Many individual vectors are associated with each procedure in the programming flow. (See Appendix A on page 23). 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.

3.1.2 Clocking, Data Format, and Timing Diagrams

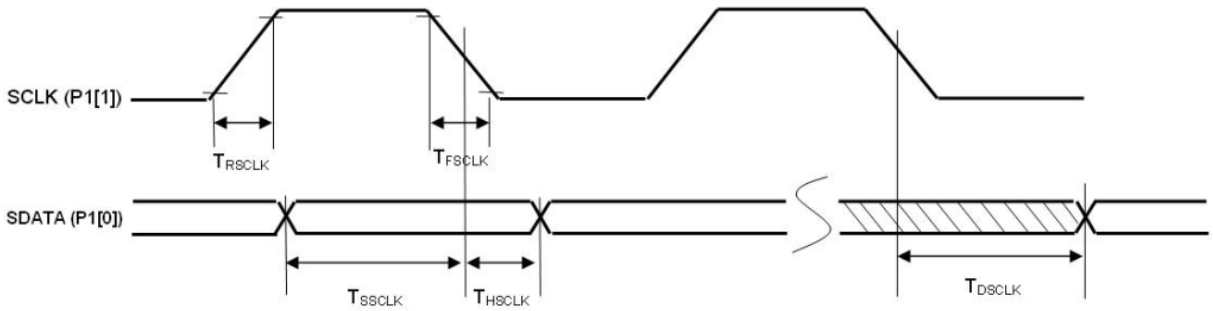
The host programmer always writes and reads SDATA on the rising edge of SCLK, while the target writes and reads on the falling edge. Figure 3-2 on page 10 shows the Timing waveforms of the SDATA, SCLK lines. Refer to Table 4-2 on page 21 for the timing specifications mentioned in Figure 3-2 on page 10.

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 4-2 on page 21). 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 to a HI-Z state. It continues driving the line only after the target sends the byte. The programmer supplies clocks even when it has released (HI-Z) the SDATA line.

During the “Wait and Poll” procedure, the programmer releases (HI-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 (Tpoll) as shown in the “Wait and Poll” timing diagram.

Figure 3-2. SCLK, SDATA Timing Diagrams

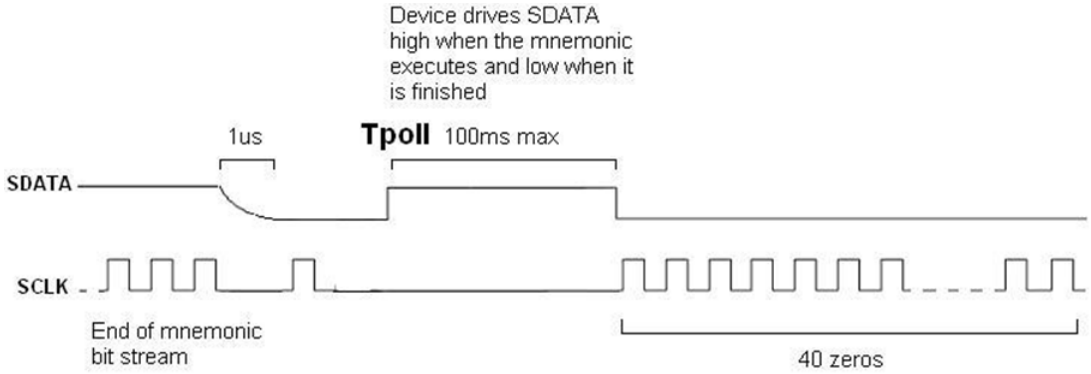


3.1.3 Wait and Poll

After a mnemonic bit stream is sent, the programmer clocks in a “Z” to the device (with enough setup time for the device SDATA pin to drift low to V_{ilp} by the device’s internal pull down resistor, typically 1 µs). One SCLK clock cycle must complete before SDATA transitions from low to high. SCLK is then held low. The target device pulls SDATA high when the mnemonic begins 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 SDATA 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 Tpoll (from Table 4-2 on page 21).

When it observes the transition to low, 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 3-3.

Figure 3-3. “Wait and Poll” Timing Diagram

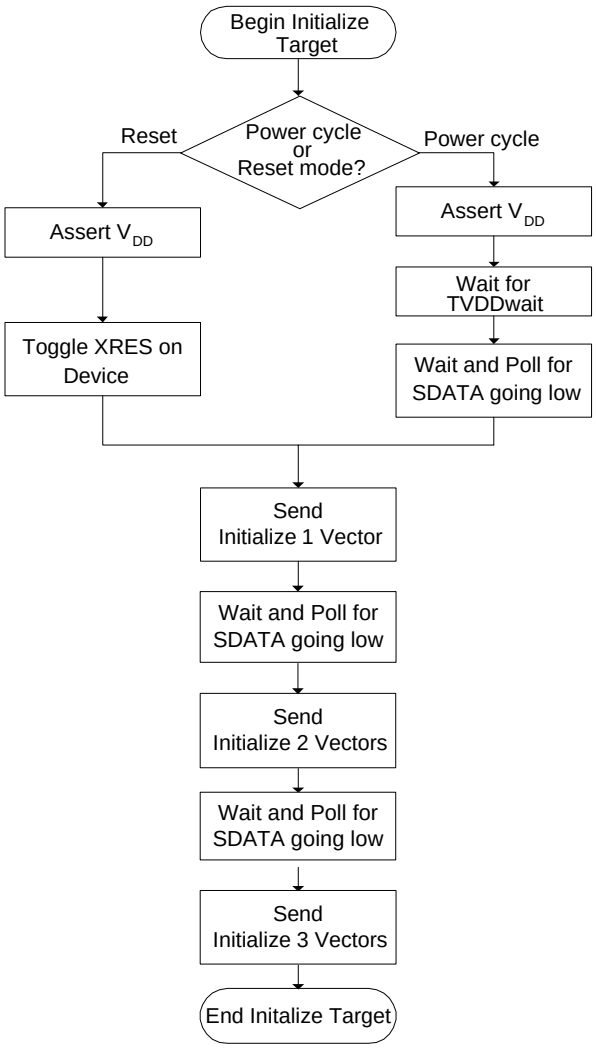


3.2 Initialize Target Procedure

The Initialize Target Procedure places the chip into the programming mode. This is done by using the reset mode or power cycle mode.

Reset mode is the preferred method for initiating communication with the target. However, in the case of CY8C24794, there is no XRES pin, so power cycle mode is the only option. Because power cycle mode involves cycling power to the target, in-circuit field programming may involve PCB layout considerations in the design phase.

Figure 3-4. Initialize Target Procedure

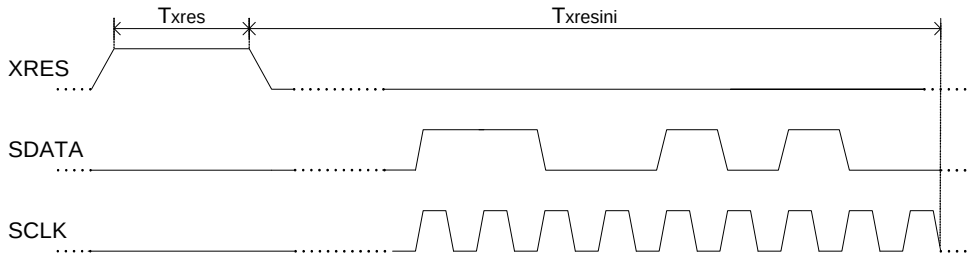


3.2.1 Reset Mode

The timing to enter programming mode with Reset is shown in Figure 3-5. 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} (see Table 4-2 on page 21). After XRES is driven low, there is a window of time specified by $T_{xresini}$, as shown in Table 4-2 on page 21, 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 a high-to-low transition on SDATA, and 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. (Appendix A on page 23).

Figure 3-5. Using Reset to Initialize



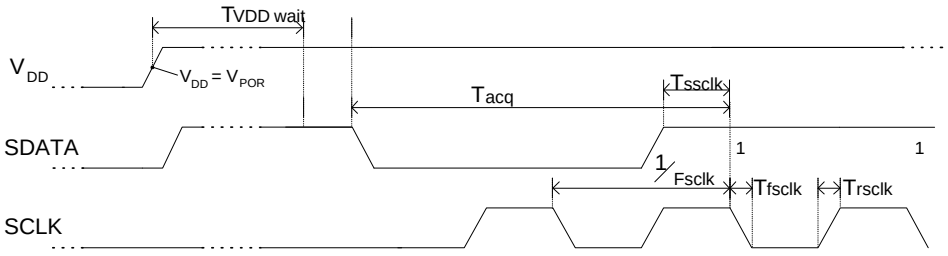
3.2.2 Power Cycle Mode

To initiate communication with the target using power cycle mode, apply V_{DD} to the target as shown in Figure 3-6. The target attempts 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 could be detected. As a result, the programmer must wait for the time specified by $T_{VDDwait}$ (Table 4-2 on page 21) before beginning to wait and poll. The programmer also 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 (see Table 4-2 on page 21). Next, send Initialize 2 vectors and wait for a 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 achieves this if the pin is left floating externally.

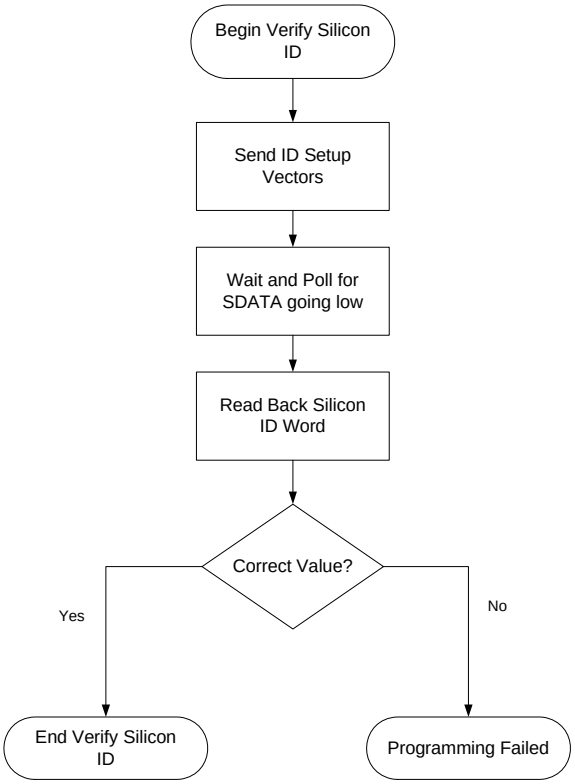
Figure 3-6. Using Power Cycling to Initialize



3.2.3 Verify Silicon ID Procedure

The Verify Silicon ID Procedure (see Figure 3-7 on page 13) returns the package-specific silicon ID value from the target. This is used by the programmer to verify the package type of the target.

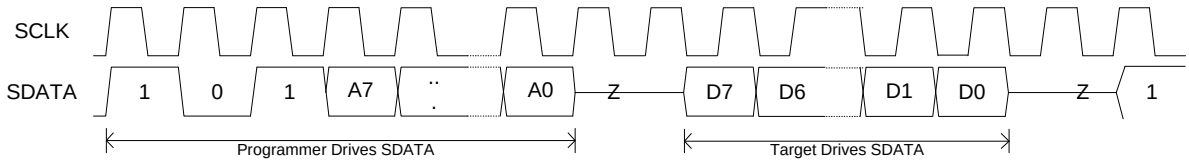
Figure 3-7. Verify Silicon ID Procedure



The first step in the Verify Silicon ID Procedure is for the programmer to send the ID-Setup vector-set. The programmer then drives the SDATA line into a HI-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 target at a specific address is shown in Figure 3-8. The READ-ID-WORD vector given in Appendix A on page 23 is based on this READ BYTE sequence with address replaced with specific value, and the data byte to be read replaced by the expected Silicon ID byte. Two bytes must be read to get a complete Silicon ID word.

The vectors in Appendix A on page 23 under READ-ID-WORD show the device-specific values read from the target. For example, a LLLLLLLL LLLHHHLH denotes a 0x00ID hex read back from CY8C24794. The programmer must compare the value in the READ-ID-WORD vector (Appendix A on page 23) and the value returned by the target. If these values do not match, the programmer must terminate the programming flow.

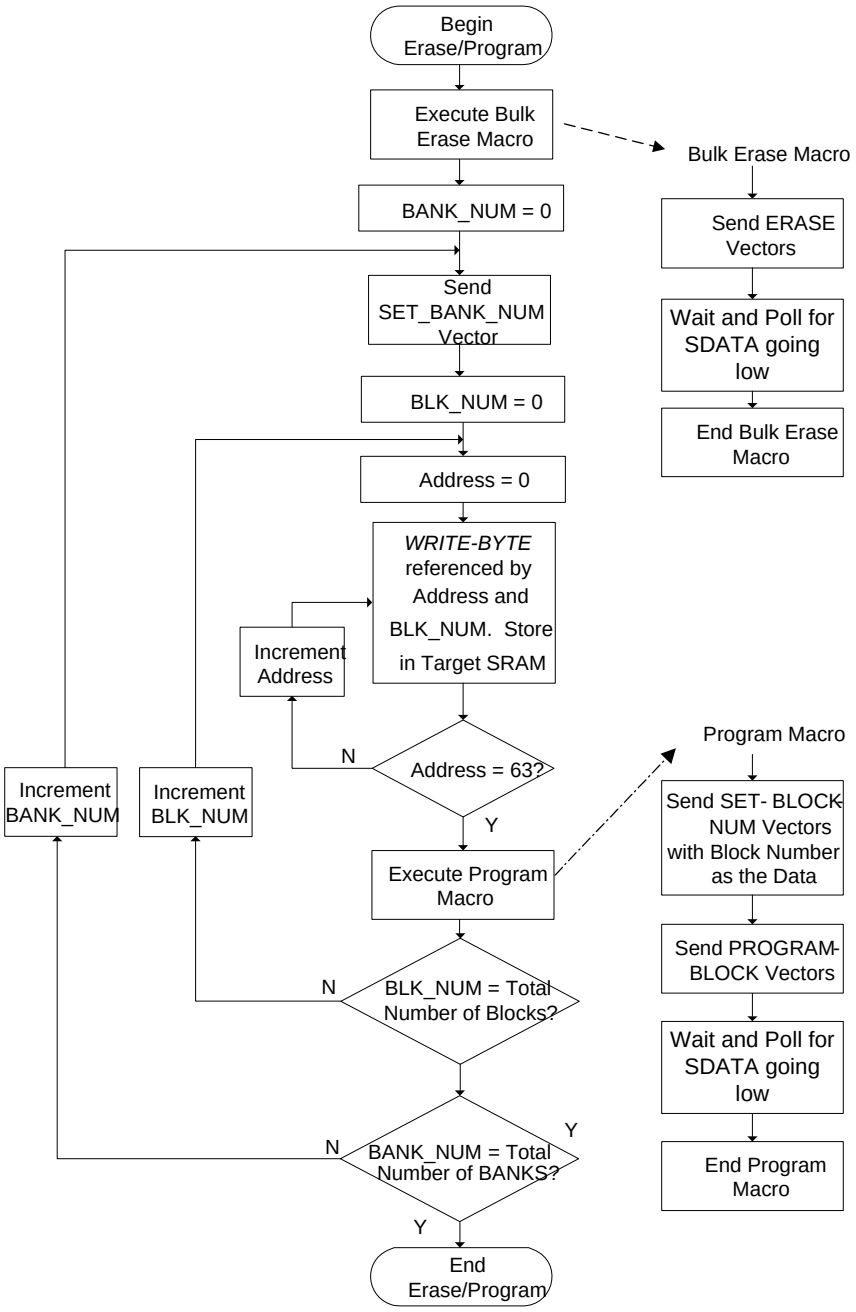
Figure 3-8. READ-BYTE (D7..D0) from Target PSoC 1 (At address A7..A0)



3.3 Program Procedure

The Program Procedure is responsible for the actual programming of the Flash.

Figure 3-9. Program Procedure



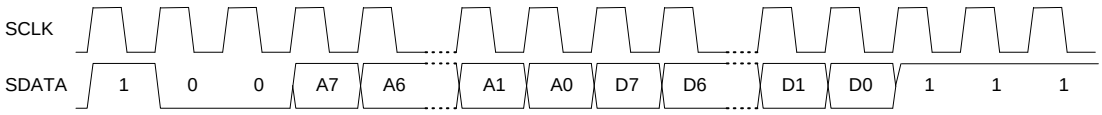
A Bulk Erase operation must be executed to prepare the flash for programming.

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* (see Appendix B on page 27) 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 3-10.

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 SDATA 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 3-10. WRITE-BYTE (D7..D0) to Target PSoC 1 (At address A7..A0)



3.4 Verify Procedure

The Verify Procedure, as shown in Figure 3-12 on page 16, 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 (see Appendix A on page 23) 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. Each Read Block operation reads a 64-byte block from Flash and stores the data in the target's SRAM. The programmer must then use the READ-BYTE vector (see Figure 3-11) 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 denotes an incorrect flash write; the programmer must terminate the programming flow as a failure.

Figure 3-11. READ-BYTE From Target for Verify operation

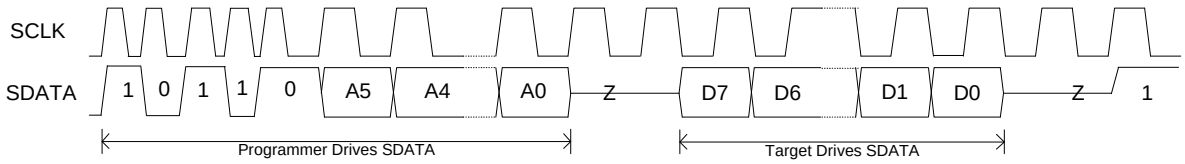
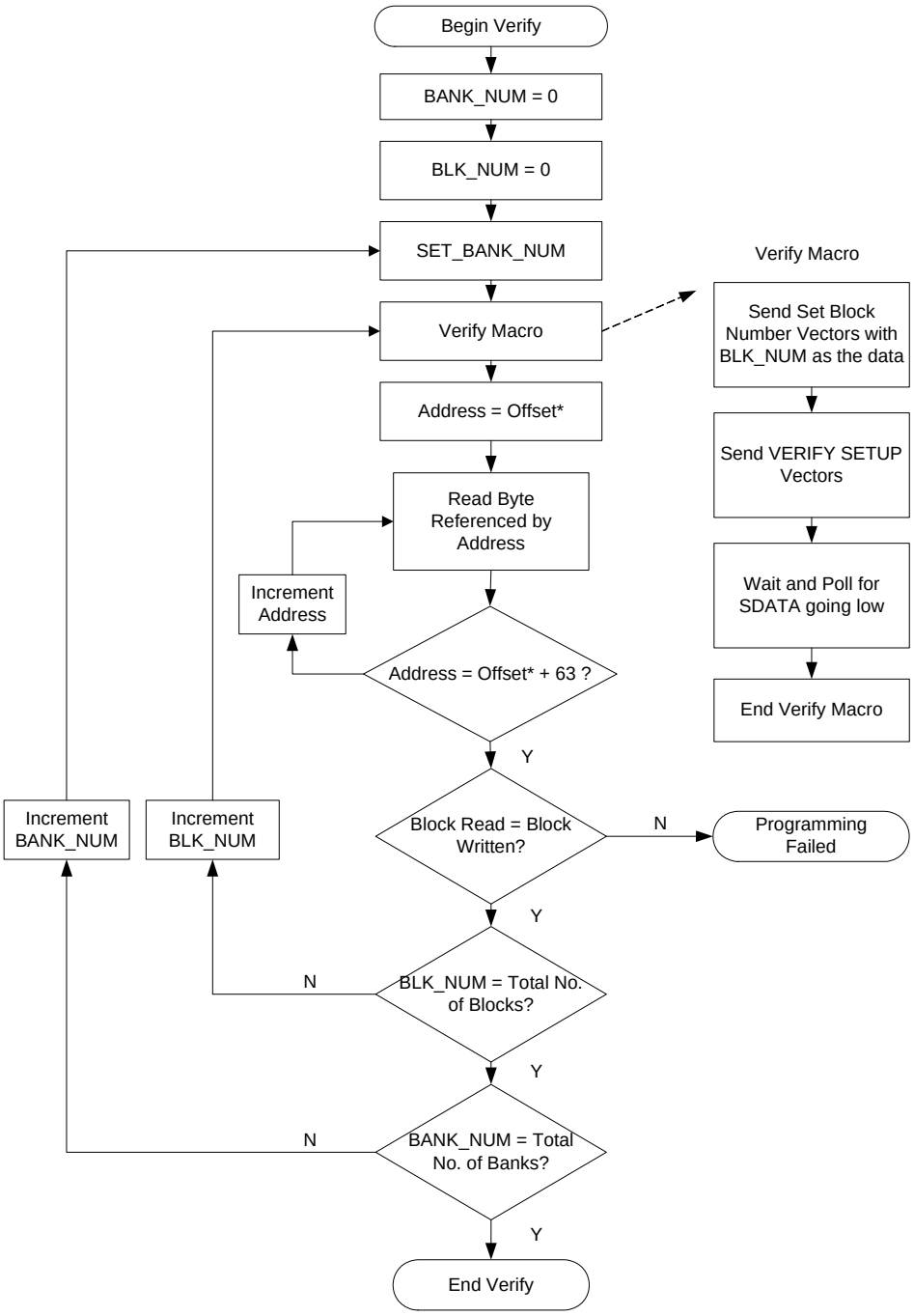


Figure 3-12. Verify Procedure (Offset = 0x80 in figure)

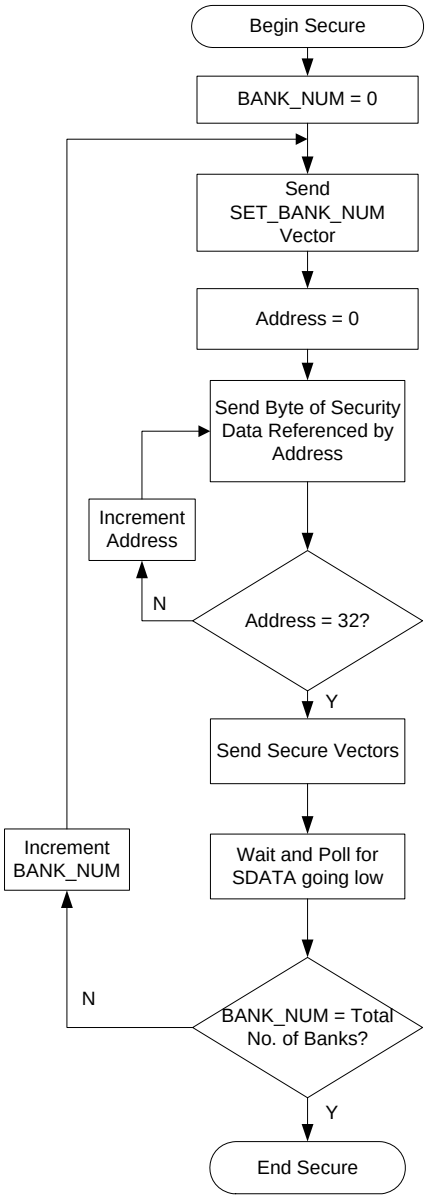


3.5 Secure Procedure

The Secure Procedure (shown in Figure 3-13), writes the user-determined security values to the target for each block.

After the flash is programmed and verified, each bank of the flash must be secured separately. Each 8K bank is secured by 32 bytes of security data. The 32 bytes of security data are 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 32 bytes are written to the target, the appropriate SECURE vector-set is sent to the target and the programmer waits and polls SDATA while the operation executes. The security data is located in the *.hex* file (Appendix B on page 27).

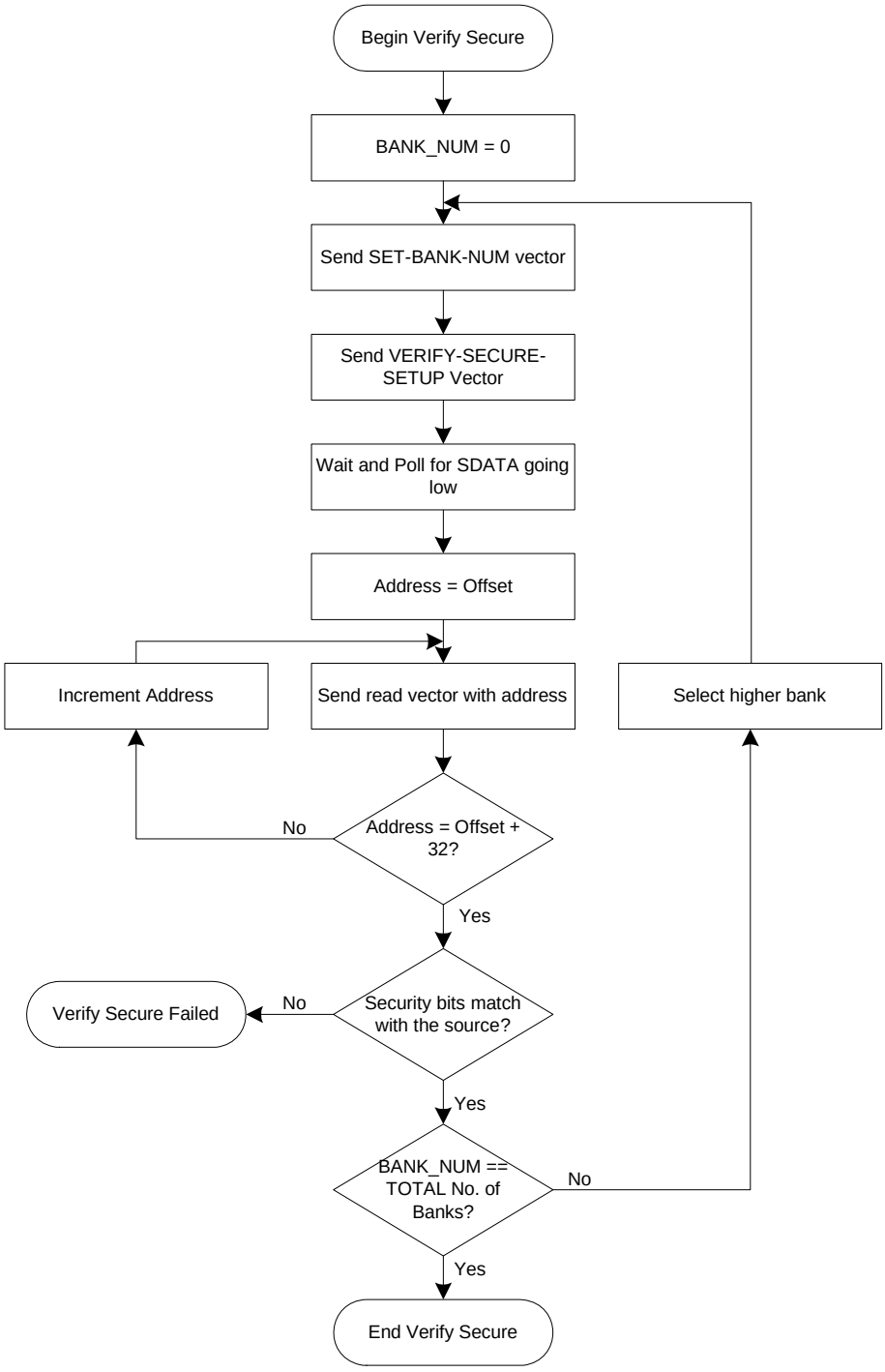
Figure 3-13. Secure Procedure



3.6 Verify Secure Procedure

The Verify Secure Procedure shown in Figure 3-14 verifies the security data (protection bits) stored in the target device. Note that this is an optional step in the programming flow. The procedure involves a loop to read (32 × Number of Banks) bytes of security data and verify with the hex file. The SET-BANK-NUM vector is sent to select the bank and then the VERIFY-SECURE-SETUP vector is sent to collect the security bits from the protection area to the SRAM area of the target device. The programmer waits and polls SDATA while this security bit transfer is in progress. When the transfer is complete, the programmer sends the READ-BYTE vector to read the 32 bytes of security data from the SRAM area of the target device. The programmer verifies the received security data with the hex file. If it matches, the process is repeated for the remaining banks. Otherwise, the programmer will flag an error.

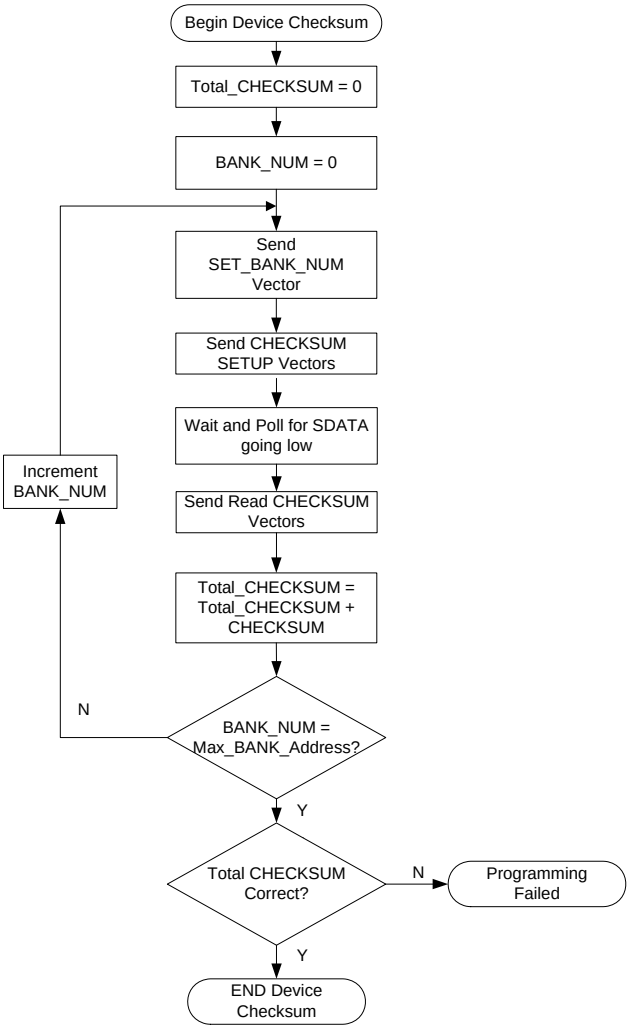
Figure 3-14. Verify Secure Procedure



3.7 Verify Checksum Procedure

The Verify Checksum Procedure (shown in Figure 3-15), causes the target to generate a checksum value for the data in flash.

Figure 3-15. Verify Checksum Procedure



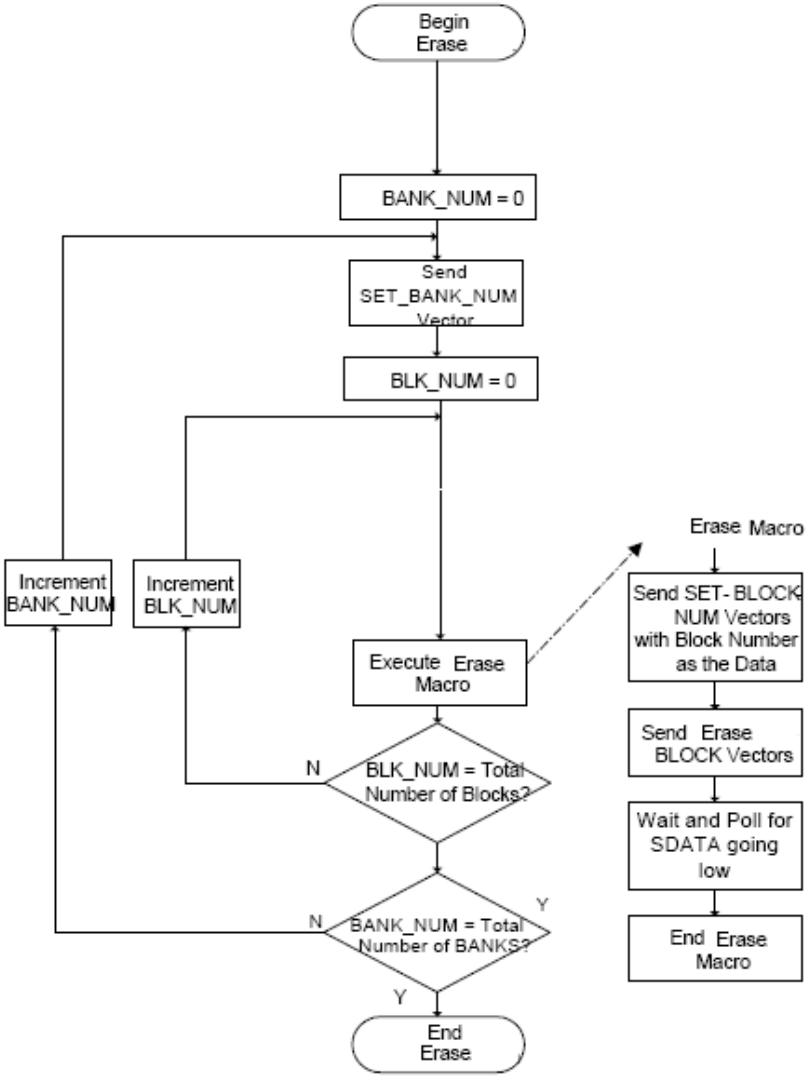
To get the Checksum Value from the target, the programmer sends the appropriate CHECKSUM-SETUP vector to the target. The programmer releases the SDATA line, then waits and polls. After the target signals that the operation is complete, the READ-CHECKSUM vector reads back the two-byte checksum value from the target. This value from the target is compared to the device checksum value from the *.hex* file (Appendix B on page 27). If the values are not equal, a programming error has occurred.

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

3.8 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 typically needed to update a few blocks of flash for partial firmware updates. In this case, the Erase Block and Program Block vectors are sent by the host to the target device. Note that this “Erase Block” is not used or required in the general programming flow, as shown in Figure 3-9 on page 14. This is because the Bulk Erase Macro is used in Figure 3-9 on page 14, which erases all the blocks of flash. Although the Bulk Erase function can be used to erase all of the flash data and the protection settings at any time, the “Erase Block” function can execute only if the Write protection feature for that particular block is turned off.

Figure 3-16. Erase Block Procedure



As shown in Figure 3-16, initialize the Bank number with the starting bank number, and the Block number with the starting block number (zero in Figure 3-16), and iterate for required number of blocks (all blocks are erased one by one in Figure 3-16).

4. Specifications and Definitions

4.1 DC Programming Specifications

Table 4-1. 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 IOL = 0.1 mA)		
V_{ohv} (Output High Voltage During Programming or Verify IOH = 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	

4.2 AC Programming Specifications

Table 4-2. 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 Setup 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 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 pursued. Therefore, a delay of $T_{vddwait}$ is needed after V_{DD} is applied and before WAIT-AND-POLL.
2. This applies to the WAIT-AND-POLL mnemonic. The SDATA 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).

4.3 Device Address and Block Definitions

Table 4-3. Device Address and Block Definitions

Device Part Numbers	CY8C22x45, CY8C24x94, CY8C28xxx, CY8CTST120, CY8CTMA120, CY8CTMG120, CY7C64215	CY8C29x66	CY8C21x45
Byte Addresses within a Block	0–63	0–63	0–63
Max_byte_address	63	63	63
Block Addresses within a Flash Bank	0–127	0–127	0–127
Max_block_address	127	127	127
Flash Bank Addresses	0–1	0–3	0
Max_bank_address	1	3	0

Appendix A. Programming Vectors for CY8C21x45, CY8C22x45, CY8C24x94, CY8C28xxx, CY8C29x66, CY8CTST120, CY8CTMA120, CY8CTMG120, CY7C64215

Table A-1. Programming Vectors

Name	Data
Vector	Bit Stream (Executed From Left Bit to Right)
Initialize-1	1100101000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000 1101111011100000000111110111101100000000111 1001111100000111010111100111110010000001111 1101111010100000000111110111101000000001111 1001111101110000000111110111110010011000111 1101111101001000000111110111100000001001111 1101111100000000000111110111111110001001011
Initialize-2	1101111011100000000111110111101100000000111 1001111100000111010111100111110010000001111 1101111010100000000111110111101000000001111 1001111101110000000111110111110010011000111 1101111101001000000111100111110100000000111 1101111000000000011011111011111000000000111 11011111111000010010111
Initialize-3 3V	1101111011100000000111110111101000000001111 1101111010100000000111110111101100000100011 1101111100001010001111110111110011111100011 1101111101000110000111110111111110001001011 0000000000000000000000011011110111000000000111 1101111010000000011111110111101010000000111 1101111011000001000111110111110000010000111 1101111100111101010111101111101000110000111 1101111011100010000111110111111110001001011 0000000000000000000000011011110111000000000111 1101111010000000011111110111101010000000111 1101111011000001000111110111110000010001111 1101111100111111011111101111101000110000111 1101111111100010010111000000000000000000000 1101111011100000000111110111101000000001111 1101111010100000000111110111101100000100011 1101111100001100000111110111110011101000111 1101111101000110000111110111110010000111 1101111111100010010111000000000000000000000
Initialize-3 5V	1101111011100000000111110111101000000001111 1101111010100000000111110111101100000100011 1101111100001010001111110111110011111100111 1101111101000110000111110111111110001001011 0000000000000000000000011011110111000000000111 1101111010000000011111110111101010000000111 1101111011000001000111110111110000010000111 1101111100111101010111101111101000110000111 1101111011100010000111110111111110001001011 0000000000000000000000011011110111000000000111 1101111010000000011111110111101010000000111 1101111011000001000111110111110000010001111 1101111100111111011111101111101000110000111 1101111111100010010111000000000000000000000 1101111011100000000111110111101000000001111 1101111010100000000111110111101100000100011 1101111100001100000111110111110011101000111 1101111101000110000111110111110010000111 1101111111100010010111000000000000000000000

Table A-1. Programming Vectors *(continued)*

Name	Data
ID-SETUP	11011110111000100001111101110000000000010111 11011110111000000001111101111011000000000111 10011111000001110101111001111100100000011111 11011110101000000001111101111010000000011111 10011111011100000001111101111001001100000111 11011111010010000001111001111010000000000111 11011110000000001101111101111100000000000111 11011111111100010010111
	READ-ID-WORD (CY8C21345) 10111111000ZLLLLLLLLZ1101111111001ZHHLHLLHHZ1
	READ-ID-WORD (CY8C21645-24xxXA) 10111111000ZLLLLHLLLZ1101111111001ZHHLHHLHLZ1
	READ-ID-WORD (CY8C21645-12xxXE) 10111111000ZLLLLHLLLZ1101111111001ZHHLHHLLHZ1
	READ-ID-WORD (CY8C22345) 10111111000ZLLLLLLLLZ1101111111001ZHHLHLLLHZ1
	READ-ID-WORD (CY8C22345H-24xxXA) 10111111000ZLLLLHHLLZ1101111111001ZHHLHLLLHZ1
	READ-ID-WORD (CY8C22545-24xxXI) 10111111000ZLLLLLLLLZ1101111111001ZHHLHLLHLZ1
	READ-ID-WORD (CY8C22645-24xxXA) 10111111000ZLLLLLLLLZ1101111111001ZHHLHHLHLZ1
	READ-ID-WORD (CY8C22645-12xxXE) 10111111000ZLLLLLLLLZ1101111111001ZHHLHHLLHZ1
	READ-ID-WORD (CY8C24794) 10111111000ZLLLLLLLLZ1101111111001ZLLLHHHLHZ1
	READ-ID-WORD (CY8C24894) 10111111000ZLLLLLLLLZ1101111111001ZLLLHHHHHZ1
	READ-ID-WORD (CY8C24994) 10111111000ZLLLLLLLLZ1101111111001ZLHLHHLLHZ1
	READ-ID-WORD (CY8C28000) 10111111000ZLLLLLLLLZ1101111111001ZHHHLLLLLZ1
	READ-ID-WORD (CY8C28445) 10111111000ZLLLLLLLLZ1101111111001ZHHHLLLLHZ1
	READ-ID-WORD (CY8C28545) 10111111000ZLLLLLLLLZ1101111111001ZHHHLLLHLZ1
	READ-ID-WORD (CY8C28645) 10111111000ZLLLLLLLLZ1101111111001ZHHHLLLHHZ1
	READ-ID-WORD (CY8C28243) 10111111000ZLLLLLLLLZ1101111111001ZHHHLLHLLZ1
	READ-ID-WORD (CY8C28643) 10111111000ZLLLLLLLLZ1101111111001ZHHHLHLHLZ1
	READ-ID-WORD (CY8C28452) 10111111000ZLLLLLLLLZ1101111111001ZHHHLLHLHZ1
	READ-ID-WORD (CY8C28413) 10111111000ZLLLLLLLLZ1101111111001ZHHHLLHHLZ1
	READ-ID-WORD (CY8C28513) 10111111000ZLLLLLLLLZ1101111111001ZHHHLHLHHZ1
	READ-ID-WORD (CY8C28433) 10111111000ZLLLLLLLLZ1101111111001ZHHHLLHHHZ1
	READ-ID-WORD (CY8C28533) 10111111000ZLLLLLLLLZ1101111111001ZHHHLHHLLZ1
	READ-ID-WORD (CY8C28403) 10111111000ZLLLLLLLLZ1101111111001ZHHHLHLLLZ1
	READ-ID-WORD (CY8C28623) 10111111000ZLLLLLLLLZ1101111111001ZHHHLHLLHZ1
	READ-ID-WORD (CY8C29466) 10111111000ZLLLLLLLLZ1101111111001ZLLHLHLHLZ1
	READ-ID-WORD (CY8C29566) 10111111000ZLLLLLLLLZ1101111111001ZLLHLHLHHZ1
	READ-ID-WORD (CY8C29666) 10111111000ZLLLLLLLLZ1101111111001ZLLHLHHLLZ1
	READ-ID-WORD (CY8C29866) 10111111000ZLLLLLLLLZ1101111111001ZLLHLHHLHZ1

Table A-1. Programming Vectors *(continued)*

Name	Data
	READ-ID-WORD (CY8CTST120-56xxxx) 10111111000ZLLLLLHHLZ110111111001ZLLLHHHHHZ1
	READ-ID-WORD (CY8CTST120-00xxxx) 10111111000ZLLLLLHHLZ110111111001ZLLLHHLHHZ1
	READ-ID-WORD (CY8CTMA120-56xxxx) 10111111000ZLLLLLHLHZ110111111001ZLLLHHHHHZ1
	READ-ID-WORD (CY8CTMA120-00xxxx) 10111111000ZLLLLLHLHZ110111111001ZLLLHHLHHZ1
	READ-ID-WORD (CY8CTMA120-100xxxx) 10111111000ZLLLLLHLHZ110111111001ZLHLHHLLHZ1
	READ-ID-WORD (CY8CTMG120-56xxxx) 10111111000ZLLLLLHHHZ110111111001ZLLLHHHHHZ1
	READ-ID-WORD (CY8CTMG120-00xxxx) 10111111000ZLLLLLHHHZ110111111001ZLLLHHLHHZ1
	READ-ID-WORD (CY7C64215-28xxxx) 10111111000ZLLLLLLLLZ110111111001ZLLLHHHHLZ1
	READ-ID-WORD (CY7C64215-56xxxx) 10111111000ZLLLLLLLLZ110111111001ZLHLHLLHHZ1
SET-BANK-NUM	1101111011100010000111110111110100000000dd111 11011110111000000111 where dd = bank #
SET-BLOCK-NUM	10011111010ddddddd111 where ddddddd = block #
BULK ERASE	1001111110000010101111100111111110010101101111 1101111011100000000111110111101100000000111 1001111100000111010111100111111001000000111111 1101111010100000000111110111101000000001111 1001111101110000000111110111110010011000011 1101111101001000000111110111100000000000101111 1101111100000000000111110111111110001001011
WRITE-BYTE	10010aaaaaaddddddd111 where ddddddd = data in, aaaaaa = address (6 bits)
PROGRAM-BLOCK	1001111110001010100111100111111100101011011 1101111011100000000111110111101100000000111 1001111100000111010111100111111001000000111111 1101111010100000000111110111101000000001111 1001111101110000000111110111110010011000011 1101111101001000000111110111100000000000101111 1101111100000000000111110111111110001001011
VERIFY-SETUP	1101111011100000000111110111101100000000111 1001111100000111010111100111111001000000111111 1101111010100000000111110111101000000001111 1001111101110000000111110111110010011000011 1101111101001000000111110111100000000000101111 1101111100000000000111110111111110001001011
READ-BYTE	10110aaaaaaZDDDDDDDDZ1 where DDDDDDDD = data out, aaaaaa = address (6 bits)
SECURE	1001111110001010100111100111111100101011011 1101111011100000000111110111101100000000111 1001111100000111010111100111111001000000111111 1101111010100000000111110111101000000001111 1001111101110000000111110111110010011000011 1101111101001000000111110111100000000000100111 1101111100000000000111110111111110001001011
VERIFY-SECURE-SETUP	1101111011100000000111110111101100000000111 1001111100000111010111100111111001000000111111 1001111110100000000111100111111110000000011 1101111010100000000111110111101000000001111 1001111101110000000111110111110010011000011 1101111101001000000111110111100000000010000111 1101111100000000000111110111111110001001011

Table A-1. Programming Vectors *(continued)*

Name	Data
CHECKSUM-SETUP	11011110111000000001111101111011000000000111 10011111000001110101111001111100100000011111 11011110101000000001111101111010000000011111 10011111011000000001111101111001001100000111 11011111010010000001111001111101010000000111 11011110000000001111111101111100000000000111 11011111111100010010111
READ-CHECKSUM	10111111001ZDDDDDDDDDZ110111111000ZDDDDDDDDZ1 where DDDDDDDDDDDDDDDD = Device Checksum data out
ERASE BLOCK	10011111100010101001111001111111001010110111 11011110111000000001111101111011000000000111 10011111000001110101111001111100100000011111 11011110101000000001111101111010000000011111 11011111001001100001111101111010010000000111 11011110000000000111111101111100000000000111 11011111111100010010111

Notes

1 = Logic high = V_{ihp}

0 = Logic low = V_{ilp}

Z = HI-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 ($V_{out} = V_{ohv}$)

L = Low data read from the device ($V_{out} = V_{olv}$)

If the Programmer has delays between executing the different mnemonics, SDATA must be HI-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 in scenarios where ISSP bus of CY8C20x36/46A/66A/96A/CY8CTMG2xx/CY8CTST2xx parts are shared with other PSoC devices, you must take care to avoid CY8C20x36/46A/66A/96A/CY8CTMG2xx/CY8CTST2xx parts seeing key 'AC52' in reset state. Refer to the knowledge base article www.cypress.com/?id=4&rID=45442 for details.

Appendix B. *Intel.hex* File Format for CY8C21x45, CY8C22x45, CY8C24x94, CY8C28xxx, CY8C29x66, CY8CTST120, CY8CTMA120, CY8CTMG120, CY7C64215

Intel.hex file records are a text representation of hexadecimal coded binary data. Only ASCII characters are used, so the format is portable across 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 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 represents 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, and so on) 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 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. 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 (such as CR only).

B.1 Example Flash Program Data Record

```
:4000C000505152535455565758595A5B5C5D5E5F000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000E8(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
505152535455565758595A5B5C5D5E5F000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
```

```
       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.
```

B.2 Example Security Data Records

```
:020000040010ea(CR/LF)
:40000000555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555
55555555555555555555555555555555555555555555580(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
55555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555
55555555555555555555555555555555
          - 64 data bytes – here bytes have 0x55 data
80        - The record checksum, calculated as above.
(CR/LF)   - End of this record.
```

B.2.1 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 uses multiple records (one to access the extended memory space, and others for the 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 blocks' 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.

B.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 0x3a 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.
```

B.3.1 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 uses two records (one to access the extended memory space, and the other for the data). The extended linear address record that precedes the checksum 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.

B.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.
```

B.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 4-3 on page 22.