CY8C52xxx, CY8C53xxx, CY8C54xxx, CY8C55xxx

PSoC® 5 Device Programming Specifications

Document #: 001-64359 Rev. *F

December 5, 2012

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

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

License

Copyrights

Disclaimer

Flash Code Protection

Cypress products meet the specifications contained in their particular Cypress Datasheets. 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. Introduction

PSoC® 5 device programming refers to the programming of nonvolatile memory in PSoC 5 using an external host programmer. In the context of external host programmers, nonvolatile memory includes flash memory, EEPROM, and write once nonvolatile latch. PSoC 5 supports programming through the Serial Wire Debug (SWD) interface. The data to be programmed is stored in a hex file. This programming specifications document explains the hardware connections, programming protocol, programming vectors, and the timing information for developing programming solutions for a PSoC 5 device.

1.1 Host Programmer

The host programmer can be the MiniProg3 Programmer supplied by Cypress, a third-party programmer, or a hardware device such as a microcontroller or an FPGA. MiniProg3 programmer is used in the prototype stage of application development for both programming and debugging PSoC 5 devices on board. Third-party programmers are used for production programming of PSoC 5 in large numbers. They are used when the design is finalized and the application needs to go in for mass production. Apart from this, custom developed host programmers such as FPGA or external microcontroller can be used to perform in-system programming of PSoC 5 device either for complete programming or partial firmware upgrade.

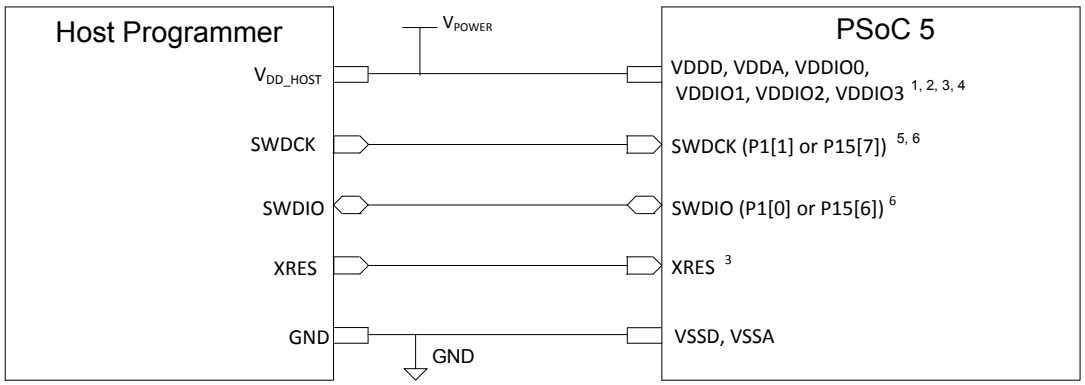
The host programmer programs the PSoC 5 device with the program image contained in the <Project_Name>.hex file, which is generated by PSoC Creator software. See the General PSoC Programming web page for complete information on PSoC programming-related documents, software, and a list of supported third-party programmers.

1.2 Hardware Connections

Figure 1-1 shows the hardware connections between the host programmer and the target PSoC 5 device for programming through the SWD interface. This figure only shows the connections between host programmer and PSoC 5 device. For a complete schematic of the PSoC 5 device for programming, including the PSoC 5 regulator output pins (Vccd, Vcca), see “Example Schematic” on page 59. See the PSoC 5 device datasheet for information on device operating conditions, specifications, and pinouts for different PSoC 5 packages. Table 1-1 lists the host programmer hardware requirements for PSoC 5 pins involved in programming.

SWDCK and SWDIO are the SWD interface programming pins. On the PSoC 5 device, there are two pairs of pins that support SWD: P1[0] SWDIO and P1[1] SWDCK, or P15[6] USB D+ (SWDIO) and P15[7] USB D– (SWDCK) pins. No device configuration setting is required to choose between the two pairs of SWD pins. The internal device logic chooses between the two pair of pins automatically by detecting activity (clock transition on SWDCK lines) on those pins after the device comes out of reset. To reset the PSoC 5 device for programming, either the XRES pin or power cycle mode must be used. Power cycle mode programming involves toggling power to the Vddd, Vdda, and Vddio pins of PSoC 5 to reset the device. All SWD interface programmers support programming using XRES pin, but only some of them support programming using power cycle mode. If power cycle mode programming is needed, ensure that the programmer being used supports power cycle mode.

Figure 1-1. Programming Interface Connections between Host Programmer and PSoC 5



Notes for Figure 1-1:

1. The voltage levels of the host programmer and supply voltage for PSoC 5 I/O pins used in programming should be the same. Port 1 SWD pins and XRES pin in PSoC 5 are powered by the Vddio1 pin. USB SWD pins are powered by the Vddd pin.

   For programming using the Port 1 SWD pins (P1[0], P1[1]) and XRES pin, the host voltage level (VDD_HOST) should be the same as Vddio1 pin of PSoC 5. The remaining PSoC 5 power supply pins (Vddd, Vdda, Vddio0, Vddio2, Vddio3) need not be at the same voltage level as the host programmer.

   For programming using the USB SWD pins (P15[6], P15[7]) and XRES pin, the host voltage level (VDD_HOST) should be the same as the Vddd and Vddio1 pins of PSoC 5. The remaining PSoC 5 power supply pins (Vdda, Vddio0, Vddio2, Vddio3) need not be at the same voltage level as the host programmer.
2. Vdda must be greater than or equal to all other power supplies (such as Vddd and Vddios) in PSoC 5.
3. For power cycle mode programming, XRES pin is not required. The Vddd, Vdda, Vddio0, Vddio1, Vddio2, and Vddio3 pins of PSoC 5 should be tied together to the same power supply; power to these pins should be toggled to reset the device. Ensure that the programmer being used supports power cycle mode. MiniProg3 (rev 7 and newer revisions) supports power cycle mode programming.
4. If the programming sequence includes programming of write once (WO) nonvolatile latch, then the Vddd supply voltage should be less than or equal to 3.3 V and programming temperature should be between 10 °C and 40 °C (Vddd = 3.3 V and TJ = 25 °C ±15 °C). Note that the constraints are only on the Vddd power supply pin and not on Vdda or Vddio power supply pins. The WO NVL programming is an optional step and is required only for applications that require the Device Security feature to be enabled. This step can be ignored for normal programming.
5. When USB SWD pins (P15[6], P15[7]) are used for programming, the unused P1[1] SWDCK pin must be externally connected to ground using external pull-down resistor (around 100-K resistor). This is required for P15[7] SWDCK signal to be seen by the PSoC 5 internal logic.
6. USB SWD pins (P15[6], P15[7]) are not present in devices without USB functionality.

Table 1-1. Host Programmer Requirements for PSoC 5 Programming

Pin	Host Programmer Requirement	PSoC 5 Function	Comment
SWDCK (SWD Clock)	Strong drive (CMOS drive) digital output	High impedance digital input	When USB SWD pins are used for programming, the P1[1] SWDCK pin must be externally connected to ground using external pull-down resistor (around 100-k resistor). This is required for P15[7] SWDCK signal to be seen by PSoC 5's internal logic.
SWDIO (SWD Data)	Write operation: Strong drive (CMOS drive) digital output Read operation: High impedance digital input	Write operation: Strong drive (CMOS drive) digital output Read operation: High impedance digital input	PSoC 5 changes between two drive modes for read and write operations on SWDIO line using Turnaround (TrN) phase of SWD protocol. Host must also change the drive mode of the SWDIO line during this TrN phase. When the host writes to SWDIO, PSoC 5 reads from SWDIO and vice-versa.
XRES	Strong drive (CMOS drive) digital output	Digital Input with internal 5.6 kΩ resistive pull-up	The XRES pin in PSoC 5 is active low input and there is an internal 5.6 kΩ pull-up resistor to Vddio1.
Vdda, Vddd, Vddio	Positive voltage	Digital, analog, I/O power supply	For power cycle mode, the Vddd, Vdda, and Vddio pins of PSoC 5 should be tied together to the same power supply; power to these pins should be toggled to reset the device. See the PSoC 5 device datasheet for specifications on power pins (Vddd, Vdda, Vddios), Ground pins (Vssd, Vssa)
Vssd, Vssa	Low resistance ground connection	Ground for all analog peripherals (Vssa), all digital logic and I/O pins (Vssd)	

Document Revision History

Document Title: CY8C52xxx, CY8C53xxx, CY8C54xxx, CY8C55xxx PSoC® 5 Device Programming Specifications

Document Number: 001-64359

Revision	Issue Date	Origin of Change	Description of Change
**	10/09/2010	VVSK	Initial version
*A	10/21/2010	SRIH	Updated the document properties.
*B	03/23/2011	VVSK	Updated for NPS. Edited and formatted as per the template.
*C	08/29/2011	VVSK	Content updates throughout the document. Converted to the TRM template.
*D	10/11/2011	VVSK	Added information on power cycle mode support.
*E	12/09/2011	VVSK	Modified definition of JTAG to SWD switching sequence. Modified Step 3 Verify Device ID.
*F	12/05/2012	VVSK	Updated Figure 3-1, Figure 3-3, Figure 3-6. Added sections to include Step 11 Program EEPROM and Step 12 Verify EEPROM

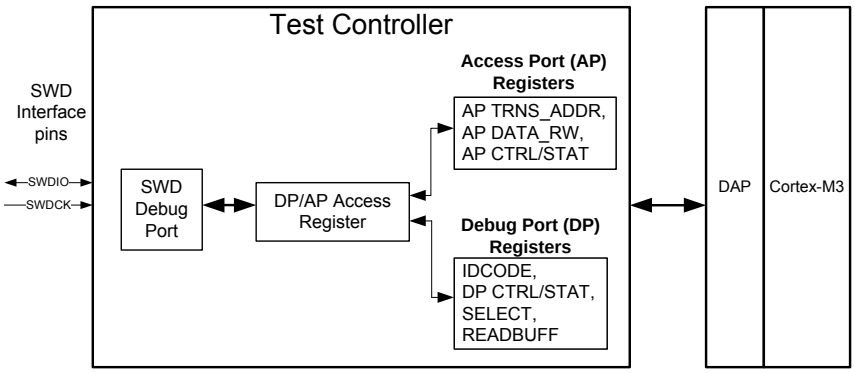
2. PSoC 5 Programming Interface

This section explains the programming interface in PSoC 5 and the registers used for programming PSoC 5. An overview of the SWD interface is also provided. See “Nonvolatile Memory Organization in PSoC 5” on page 56 for details.

2.1 Test Controller Block

The host programmer communicates with the PSoC 5 through the device’s internal Test Controller (TC). The TC is the interface that provides access to the PSoC 5 ARM Debug Access Port (DAP) module, which in turn provides access to the device memory and registers. Using the TC and DAP, the host programmer writes to SRAM, sets internal registers, and programs the device’s flash memory, EEPROM, and NV latches. For more information on DAP functionality, see the Test Controller chapter in PSoC 5 Architecture TRM and the ARM Debug Interface Architecture Specification.

Figure 2-1. PSoC 5 Programming Interface



2.2 Programming Interface Registers

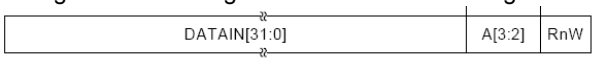
2.2.1 Debug Port/Access Port (DP/AP) Access Register

PSoC 5 architecture has a DP/AP Access register that is 35 bits wide. This register, which is a part of the test controller interface, is used to transfer data between the SWD interface and the Debug Port and Access Port registers in the DAP. The SWD interface enables direct reads and writes of the DP/AP Access register.

2.2.1.1 Writing to DP/AP Access Register

Figure 2-2 shows the structure when writing to the DP/AP Access register from the SWD interface.

Figure 2-2. Writing to the DP/AP Access Register

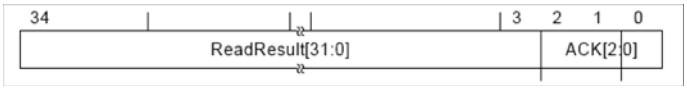


- Bits 34 to 3: (32 bits of data). If the register is less than 32-bits wide, zero-padding must be done for the remaining bits that are sent to PSoC 5
- Bits 2 to 1: 2-bit address for selecting DP or AP registers. These address bits are listed in Table 2-2
- Bit 0: RnW – 1 = read (from PSoC 5 to host programmer); 0 = write (to device from debug host)

2.2.1.2 Reading from DP/AP Access Register

Figure 2-3 shows the structure of the 35-bit data register when reading the DP/AP Access register from the SWD interface.

Figure 2-3. Reading from the DP/AP Access Register



- Bits 34 to 3: (32 bits of data): If the register is less than 32-bit wide (N-bit), it is still required to read the entire 32 bits to complete the transaction. Only the least N-bit data should be considered of the 32-bits read from device.
- Bits 2 to 0: (ACK response code): Depending on the interface, the ACK response is as indicated in Table 2-1. This ACK response is for the previous SWD transfer; if there is an error, it indicates that the previous transfer must be done again.

Table 2-1. ACK Response for SWD Transfers

ACK[2:0]	SWD
OK	001
WAIT	010
FAULT	100

2.2.2 Debug Port (DP)/Access Port (AP) Registers

The DP and AP registers listed in Table 2-2 are part of the ARM Cortex-M3 Debug Access Port (DAP). All the DP/AP registers are 32-bit registers. In the PSoC 5 Cortex-M3, the DAP consists of the SWD Debug Port (SW-DP) and the AHB Access Port (AHB-AP). Note that Table 2-2 does not list all the DP/AP registers; it lists only those DP/AP registers that are required to program PSoC 5. For more information on these ports and their registers, see the ARM Debug Interface Architecture Specification (for SW-DP) and ARM Cortex-M3 Technical Reference Manual (for AHB-AP), available at http://www.arm.com.

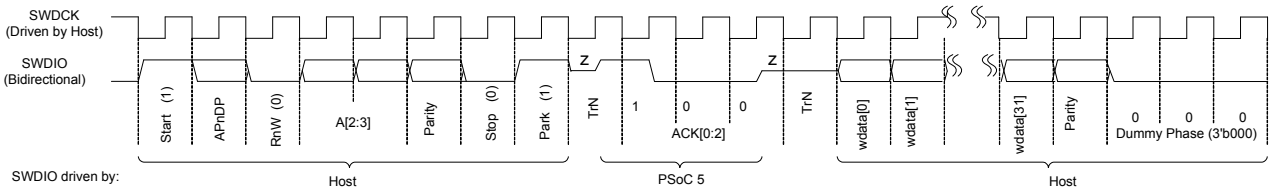
Table 2-2. Debug Port and Access Port Registers (PSoC 5)

Register Name	Register Type	Address (A[3:2])	Function
IDCODE	DP	00	32-bit Device IDCODE register.
DP CTRL/STAT	DP	01	Debug port control/status register. CTRLSEL bit in the SELECT register should be ‘0’ to access this register.
SELECT	DP	10	Access port select – The MS byte of the SELECT register selects which Access Port (AP) is used on AP accesses. Bits [7:4] select which register in the AHB-AP is accessed.
READBUFF	DP	11	Port Acquire key is written to this 32-bit register to acquire port through SWD interface.
AP Control Status (AP CTRL/STAT)	AP	00 (SELECT[7:4] = 0)	AHB-AP control/status register
AP Transfer Address	AP	01 (SELECT[7:4] = 0)	AHB-AP transfer address register. This register holds the 32-bit address that is used for device register access
AP Data Read/Write	AP	11 (SELECT[7:4] = 0)	AHB-AP data read/write register. This 32-bit register holds the data to be read from/written to the address specified by the AP Transfer Address register

2.3 Serial Wire Debug (SWD) Interface

PSoC 5 supports programming through the serial wire debug (SWD) interface. There are two signals in SWD interface: data signal (SWDIO) and a clock for data signal (SWDCK). The host programmer always drives the clock line, whereas either the programmer or the PSoC 5 device drives the data line. The timing diagram for the SWD protocol is given in Programming Specifications chapter on page 29. Host programmer and PSoC 5 device communicate in packet format through the SWD interface. Write packet refers to the SWD packet transaction in which the host writes data to PSoC 5. Read packet refers to the SWD packet transaction in which the host reads data from PSoC 5. The format of the write packet and read packet are illustrated in Figure 2-4 and Figure 2-5, respectively

Figure 2-4. SWD “Write Packet” Timing Diagram



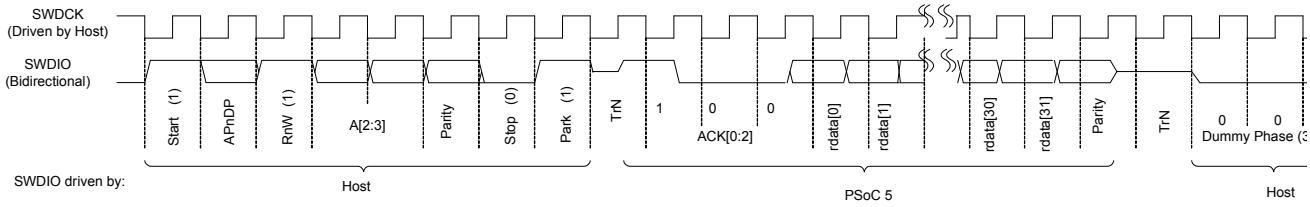
a.) Host Write Operation: Host sends data on the SWDIO line on falling edge of SWDCK and PSoC 5 reads that data on the next SWDCK rising edge (for example, 8-bit header data, Write data(wdata[31:0]), Dummy phase (3'b000))

b.) Host Read Operation: PSoC 5 sends data on the SWDIO line on the rising edge of SWDCK and the host should read that data on the next SWDCK falling edge (Ex: ACK data (ACK[2:0])

c.) The host should not drive the SWDIO line during TrN phase. During first TrN phase (½ cycle duration) of SWD packet, PSoC 5 starts driving the ACK data on the SWDIO line on the rising edge of SWDCK. The host should read the data on the subsequent falling edge of SWDCK. The second TrN phase is 1.5 SWDCK clock cycles. Both PSoC 5 and the Host will not drive the line during the entire second TrN phase (indicated as ‘z’). Host should start sending the Write data (wdata) on the next falling edge of SWDCK after second TrN phase.

d.) “DUMMY” phase is three SWD clock cycles with SWDIO line low. This DUMMY phase is not part of SWD protocol. The three extra clocks with SWDIO low are required for the Test Controller in PSoC 5 to complete the Read/Write operation when the SWDCK clock is not free-running. For a reliable implementation, include three IDLE clock cycles with SWDIO low for each packet. According to the SWD protocol, the host can generate any number of SWD clock cycles between two packets with SWDIO low.

Figure 2-5. SWD “Read Packet” Timing Diagram



a.) Host Write Operation: Host sends data on the SWDIO line on falling edge of SWDCK and PSoC 5 reads that data on the next SWDCK rising edge (for example, 8-bit header data, dummy phase (3'b000))

b.) Host Read Operation: PSoC 5 sends data on the SWDIO line on rising edge of SWDCK and the host should read that data on the next SWDCK falling edge (for example, ACK data (ACK[2:0], Read data (rdata[31:0]))

c.) The host should not drive the SWDIO line during TrN phase. During first TrN phase (½ cycle duration) of SWD packet, PSoC 5 starts driving the ACK data on the SWDIO line on the rising edge of SWDCK. The host should read the data on the subsequent falling edge of SWDCK. The second TrN phase is 1.5 SWDCK clock cycles. Both PSoC 5 and the host will not drive the line during the entire second TrN phase (indicated as ‘z’). Host should start sending the Dummy phase (3'b000) on the next falling edge of SWDCK after second TrN phase.

d.) “DUMMY” phase is three SWD clock cycles with SWDIO line low. This phase is not part of the SWD protocol. The three extra clocks with SWDIO low are required for the Test Controller in PSoC 5 to complete the Read/Write operation when the SWDCK clock is not free-running. For a reliable implementation, include three IDLE clock cycles with SWDIO low for each packet. According to the SWD protocol, the host can generate any number of SWD clock cycles between two packets with SWDIO low.

A complete data transfer requires 46 clocks (not including the optional three dummy clock cycles in Figure 2-4 and Figure 2-5). Each data transfer consists of three phases:

- **Packet request** – External host programmer issues a request to the PSoC 5 device.
- **Acknowledge response** – PSoC 5 sends an acknowledgement to the host.
- **Data** – Data is valid only when a packet request is followed by a valid (OK) acknowledge response.

The data transfer is either:

- PSoC 5 to host, following a read request – RDATA
- Host to PSoC 5, following a write request – WDATA

In Figure 2-4 and Figure 2-5, the following sequence occurs:

1. The start bit initiates a transfer; it is always logic '1'.
2. The APnDP bit determines whether the transfer is an AP access, '1', or a DP access, '0'.
3. The next bit is RnW, which is '1' for a read from the PSoC 5 device or '0' for a write to the PSoC 5 device.
4. The ADDR bits (A[3:2]) are register select bits for the access port or debug port. See Table 2-2 for address bit definitions.
5. The parity bit has the parity of APnDP, RnW, and ADDR. This is an even parity bit. If the number of logical 1s in these bits is odd, then parity must be '1', otherwise it is '0'.

   If the parity bit is not correct, the header is ignored by the target device; there is no ACK response. For the host implementation, the programming operation should be stopped and tried again by doing a device reset.
6. The stop bit is always logic '0'.
7. The park bit is always logic '1' and should be driven high by the host.
8. The ACK bits are the device-to-host response.

   Possible values are shown in Table 2-1. Note that the ACK in the current SWD transfer reflects the status of the previous transfer. OK ACK means the previous packet is successful. WAIT response indicates that the previous packet transaction is not yet complete. For a Fault operation, the programming operation should be aborted immediately.

   a. For a WAIT response, if it is a read transaction, the host should ignore the data read in the data phase. PSoC 5 does not drive the line and the host must not check the parity bit.

   b. For a WAIT response, if it is a write transaction, the data phase is ignored by the PSoC 5 device. But the host must still send the data to be written from an implementation standpoint. The parity data corresponding to the data should also be sent by the host.

   c. For a WAIT response, it means that the PSoC 5 device is processing the previous transaction. The host can try for a maximum of four continuous WAIT responses to see if an OK response is received, failing which, it can abort the programming operation and try again.

   d. For a FAULT response, the programming operation should be aborted and retried by doing a device reset.
9. The data phase includes a parity bit (even parity, similar to the packet request phase).

   a. For a read data packet, if the host detects a parity error, then it must abort the programming operation and restart.

   b. For a write data packet, if the PSoC 5 detects a parity error in the data packet sent by the host, it generates a FAULT ACK response in the next packet.
10. Turnaround (TrN) phase: According to the SWD protocol, the TrN phase is used both by the host and the PSoC 5 device to change the Drive modes on their respective SWDIO line. There are two TrN phases in each SWD packet. During the first TrN phase after packet request, PSoC 5 drives the ACK data on the SWDIO line on the rising edge of SWDCK in TrN phase. This ensures that the host can read the ACK data on the next falling edge. Thus, the first TrN cycle is only for half cycle duration, as shown in Figure 2-4 and Figure 2-5. The location of the second TrN phase is different for read and write packets. The second TrN phase of the SWD packet is one-and-a-half cycle long. Neither the host nor PSoC 5 should drive SWDIO line during both the TrN phases as indicated by 'z' in Figure 2-4 and Figure 2-5.
11. The address, ACK, and read and write data are always transmitted least significant bit (lsb) first.
12. At the end of each SWD packet in Figure 2-4 and Figure 2-5, there is a "DUMMY" phase, which is three SWD clock cycles with SWDIO line held low. This DUMMY phase is not part of the SWD protocol. The three extra clocks with SWDIO low are required for the Test Controller in PSoC 5 to complete the Read/Write operation when the SWDCK clock is not free-running. For a reliable implementation, include three IDLE clock cycles with SWDIO low for each packet. According to the SWD protocol, the host can generate any number of SWD clock cycles between two packets with SWDIO low.

Note The SWD interface can be reset anytime during programming by clocking 51 or more cycles with SWDIO high. To return to the idle state, SWDIO must be clocked low for three or more cycles. The host programmer can begin a new SWD packet transaction from the idle state.

2.3.1 Register Access Using SWD Interface

To access the registers using the SWD interface, in the 8-bit transfer request packet, set the APnDP bit and select the corresponding ADDR bits, as shown in Table 2-2. Table 2-3 shows the 8-bit transfer request packet to access the DP and AP registers for read or write operation. The 8-bit transfer request data in Table 2-3 is transmitted least significant bit first. The 'Start' bit is the least significant bit (LSb) and the 'Park' bit is the most significant bit (MSb) in Table 2-3. Use Table 2-3 and vectors given in SWD Vectors for Programming chapter on page 31 to implement PSoC 5 programming.

Table 2-3. SWD Transfer Request Data Packet for Test Controller DPACC and APACC Register Access

Pseudo Code	Register Name	Type of Operation	SWD Transfer Request Data (LSB sent first)	
			Binary	Hex
DPACC IDCODE Read	IDCODE	Read	8'b10100101	8'hA5
DPACC DP CTRLSTAT Write	DP CTRL/STAT	Write	8'b 10101001	8'hA9
DPACC DP SELECT Write	SELECT	Write	8'b10110001	8'hB1
DPACC READBUFF Write	READBUFF	Write	8'b10011001	8'h99
APACC AP CTRLSTAT Write	AP CTRL/STAT	Write	8'b10100011	8'hA3
APACC ADDR Write	AP Transfer Address	Write	8'b10001011	8'h8B
APACC DATA Read	AP Data Read/Write	Read	8'b10011111	8'h9F
APACC DATA Write	AP Data Read/Write	Write	8'b10111011	8'hBB

The 'AP Transfer Address' register holds the PSoC 5 memory address that needs to be accessed. To read or write PSoC 5's internal registers or SRAM, first write the address to the 'AP Transfer Address' register (Pseudo Code – APACC ADDR Write). For a write operation, write data to the 'AP Data Read/Write' register (Pseudo Code – APACC DATA Write). If it is a read operation, read the 'AP Data Read/Write' register twice (Pseudo Code – APACC DATA Read); the test controller (TC) reads out data through the data line.

For example, to write 32'hB6 to the target device internal register at address 32'h40004720, the following SWD transfers are necessary:

APACC ADDR WRITE [0x40004720]

APACC DATA WRITE [0x000000B6]

The binary data for the two SWD packets, with the bit pattern being least significant bit to most significant bit (from left to right), are as follows.

11010001 (ACK) 00000100111000100000000000000010(0)

11011101 (ACK) 01101101000000000000000000000000(1)

'(ACK)' indicates waiting for ACK from target device. This '(ACK)' is for the previous SWD transfer as explained earlier. The last bit in data phase (enclosed in brackets above) is the parity bit for the 32-bit data.

SWD register read is similar to SWD write operation, except that the read operation should be done twice to get the correct data. First, host should write the address to the APACC ADDR register address. Then, it should read the DATA_RW register twice. The first read initiates the command to the DAP interface and the second read returns the requested value.

For example, to read from address 32'h40004720, the following transfers need to be done:

APACC ADDR Write [0x40004720]

Dummy_data = APACC DATA Read //dummy SWD read

Data = APACC DATA Read //returns actual data

Note The previous two examples do not consider the three dummy clocks cycles required at the end of each SWD packet. They should be appended, as shown in Figure 2-4 and Figure 2-5, if the SWDCK clock is not free running.

To simplify the process, the programmer can have a SWD command interpreter that implements Table 2-3 and outputs data in binary format. An example is as follows. The SWD_packet function recognizes the SWD transfer that is given and puts the corresponding binary data into the outgoing data buffer for transmission.

SWD_packet (APACC_ADDR, 32'h40004720)

SWD_packet (APACC_DATA_WRITE, 32'hB6)

Data= SWD_packet (APACC_DATA_READ)

2.3.2 Switching to SWD Interface

PSoC 5 supports programming only through the SWD interface. It does not support programming through the Joint Test Action Group (JTAG) interface. But it is necessary to send a JTAG-to-SWD switching sequence on SWDIO, SWDCK lines to program through the SWD interface. This switching is required in “Step 2: Configure Target Device” on page 21. The PSoC 5 architecture warrants the switching sequence to SWD interface due to the following reason.

After acquiring the PSoC 5 device, access to the Debug and Access Port (DAP) must be done by setting the ‘d_tst_acc’ bit in the Test Controller register ‘TC.TST_CR4’. The Test Controller automatically switches from SWD to JTAG the first time ‘d_tst_acc’ bit is set, because the DAP always resets to JTAG the first time. JTAG to SWD switching must be done once during programming as a result of this condition. But this switching from SWD to JTAG does not happen for subsequent setting of ‘d_tst_acc’ bit while programming. This is because the DAP remembers its last configuration from that point (until the device is reset by XRES).

2.3.2.1 JTAG to SWD Switching

To switch DAP from JTAG to SWD operation, the sequence is as follows:

1. Send 51 or more **SWDCK** cycles with **SWDIO** HIGH. This ensures that the current interface is in its reset state. The JTAG interface only detects the 16-bit JTAG-to-SWD sequence starting from the Test-Logic-Reset state.
2. Send the 16-bit JTAG-to-SWD select sequence on **SWDIO**. The 16-bit JTAG-to-SWD select sequence is 0b0111_1001_1110_0111, most-significant bit (MSB) first. This can be represented as either:
   a. 0x79E7 transmitted most-significant bit (MSb) first
   b. 0xE79E transmitted least-significant bit (LSb) first.

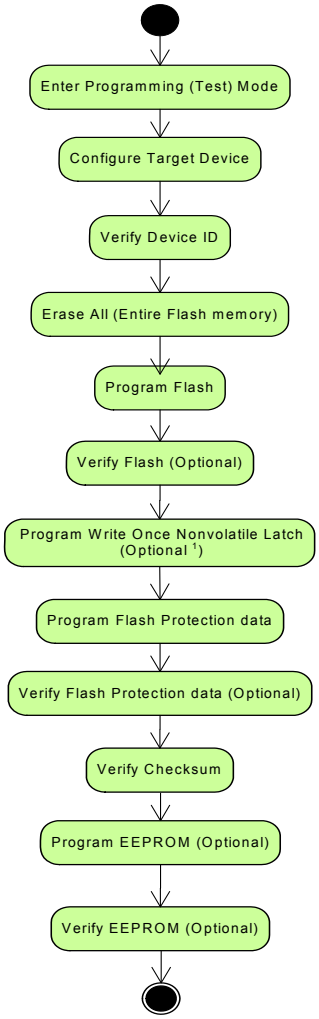
Figure 2-6. First Three Steps of JTAG to SWD Switching

3. Send 51 or more **SWDCK** cycles with **SWDIO** HIGH. This ensures that if DAP is already in SWD operation before sending the select sequence, the SWD interface enters line reset state.
4. Send three or more **SWDCK** cycles with **SWDIO** low. This ensures that the SWD line is in the idle state before starting a new SWD packet transaction.
5. Send the **DPACC IDCODE READ** SWD read packet as given in Table 2-3. There is no need to process the Device ID returned by the PSoC 5 device for this read packet. Ignore the Device ID returned by PSoC 5 in this step.

3. PSoC 5 Programming Flow

Figure 3-1 shows the sequence of steps involved in programming a PSoC 5 device. Each step is discussed in detail in later sections. All steps in Figure 3-1 must be completed successfully for a successful programming operation. The programming operation should be stopped if there is a failure in any of the steps. The SWD packets for each step are provided in SWD Vectors for Programming chapter on page 31.

Figure 3-1. PSoC 5 Programming Flow



Note Programming write once nonvolatile latch (WO NVL) is an optional step that is required only if the Device Security feature is enabled; it is not required for normal device programming. If this step is included in the programming flow, conditions imposed on the PSoC 5 Vddd operating voltage (Vddd ≤ 3.3 V) and programming temperature (TJ = 25 °C ± 15 °C). See Figure 1-1 and "Step 7: Program WO NVL (Optional)" on page 25 for more details. These conditions are not applicable if the WO NVL programming step is not included in the programming sequence.

3.1 Step1: Enter Programming Mode

The first step in PSoC 5 device programming is to enter the Programming mode, also called the Test mode. The host programmer must complete this step successfully for the remaining programming steps to be successful.

The procedure to enter the programming mode depends on the method used to reset the PSoC 5 device. The two methods to reset PSoC 5 are as follows:

- Using the device reset (XRES) pin: In this method, the host programmer drives the XRES pin of PSoC 5 low to do a device reset.
- Power cycle mode: In this method, the host programmer toggles power to PSoC 5's power supply pins (Vddd, Vdda, and Vddios) to do a device reset.

3.1.1 Enter Programming mode through the SWD Interface:

Figure 3-2 shows the sequence of steps to enter programming mode (or test mode) of the PSoC 5 using SWD interface; Figure 3-3 shows the corresponding timing diagram. See Table 4-2 on page 30 for specifications of timing parameters mentioned in Figure 3-2 and Figure 3-3. Figure 3-3 shows both XRES method and power cycle mode of programming. Each of these methods are explained in separate sections.

Figure 3-2. Entering Programming (Test) Mode through SWD Interface

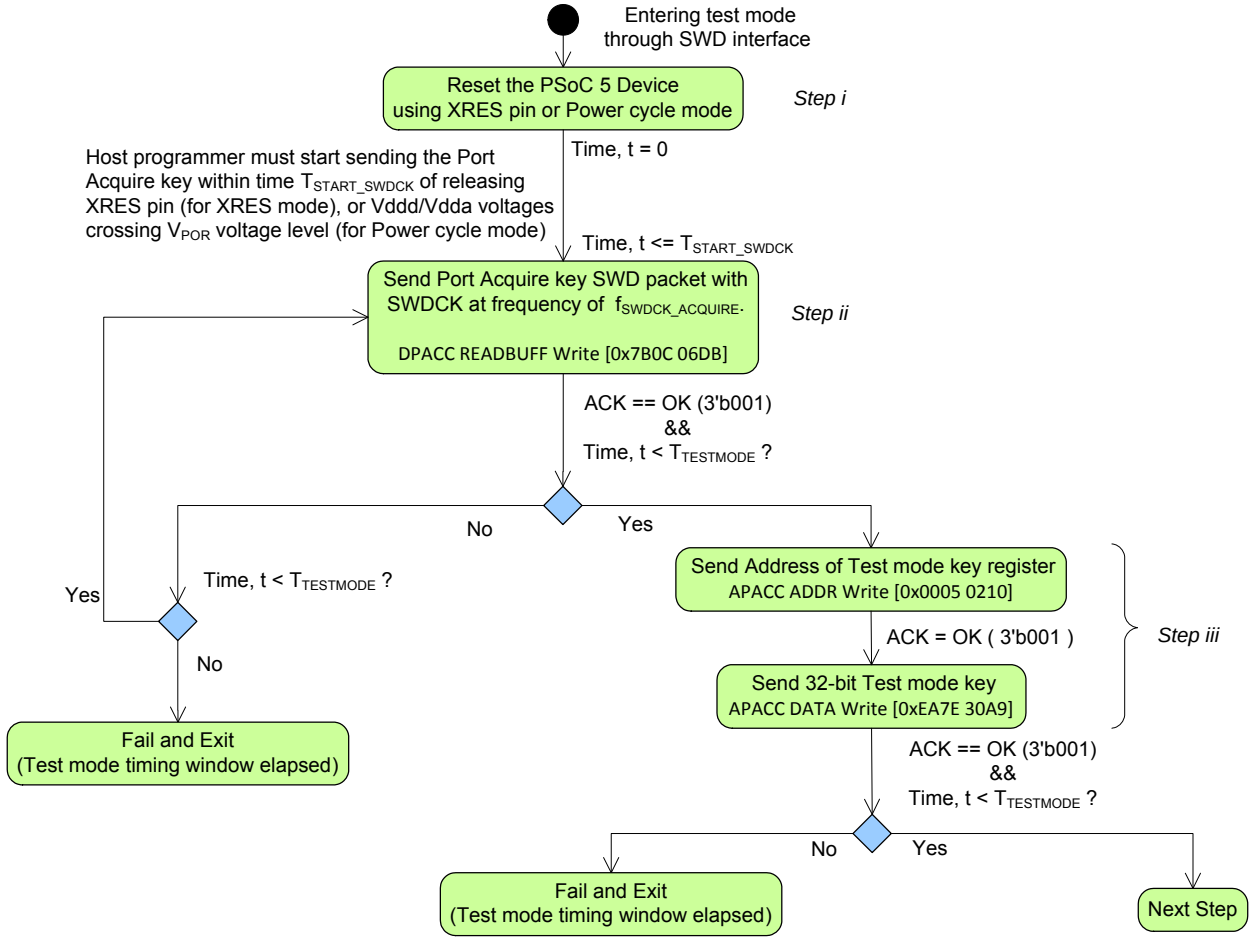
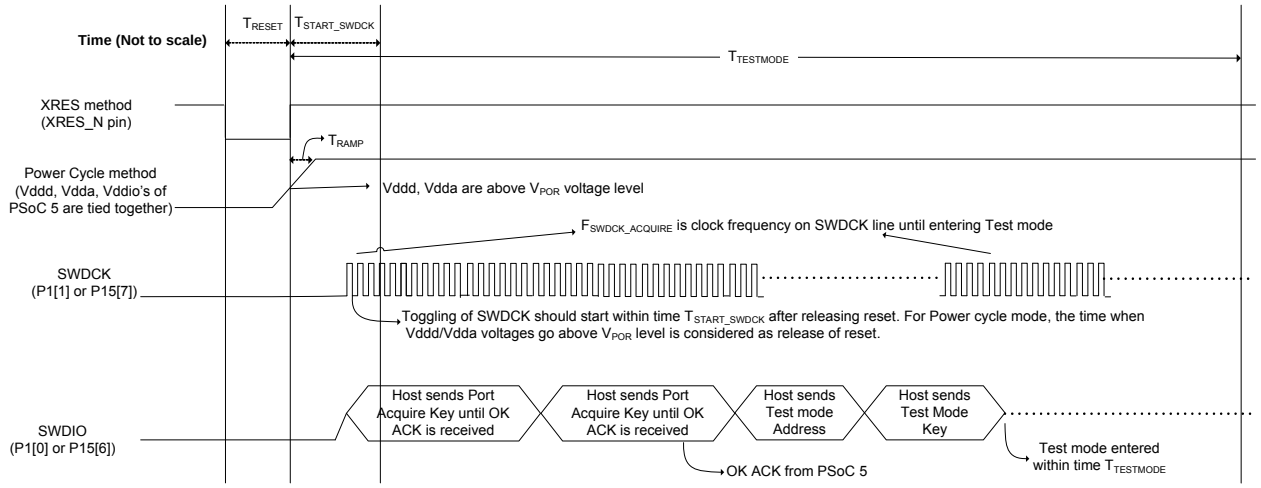


Figure 3-3. Timing Diagram to Enter Test Mode through SWD Interface



3.1.1.1 SWD Programming using XRES Pin

The sequence in Figure 3-3 using SWD interface and XRES pin is as follows.

1. Host programmer drives the XRES pin of PSoC 5 low to cause a device reset. The reset signal is active low, and the reset pulse width is specified by the T_{RESET} timing parameter.
2. Within time T_{START_SWDCK} of releasing XRES signal, the host must start sending the Port Acquire key on SWDIO and SWDCK lines. The host must send this Port Acquire key continuously until an OK ACK is received from PSoC 5. The pseudo code is given here.

```
do
{
/* Write Port Acquire key, Use SWD ADDR =
       2'b11*/
    DPACC READBUFF Write [0x7B0C 06DB]
  //Check port acquire retry time and
       whether OK ACK is received
} while (ACK != "OK" AND time_elapsed <
       TTESTMODE)

// Exit on timeout
if (ACK != "OK" OR time_elapsed >
TTESTMODE) then FAIL_EXIT
```

   If the debug port is disabled, PSoC 5 ignores the first Port Acquire SWD packet sent after releasing reset. It does not return an OK ACK for the first packet. PSoC 5 sends an OK ACK only during the second try of Port Acquire SWD packet. Therefore, the port acquire sequence must be sent continuously on the SWD interface until an OK ACK is received from PSoC 5. The timeout window for this loop is $T_{TESTMODE}$, the programming (test) mode entry window duration.

   Significance of SWDCK frequency $f_{SWDCK\_ACQUIRE}$: In Figure 3-2 and Figure 3-3, the SWDCK frequency during test mode entry is $f_{SWDCK\_ACQUIRE}$. The host programmer must meet this frequency specification to successfully enter PSoC 5 programming mode. After device reset is released, the internal test controller logic in PSoC 5 looks for clock transitions on the SWDCK line. If the test controller logic notices eight SWDCK clock cycles within a time window of $T_{ACQUIRE}$, it extends the time to enter programming mode to $T_{TESTMODE}$. This time window can be anywhere within duration T_{BOOT} (68 µs) after device reset. T_{BOOT} is the time for PSoC 5 boot to complete after device reset is released. By ensuring that SWDCK line is always clocked at a frequency of $f_{SWDCK\_ACQUIRE}$, the host programmer can meet PSoC 5 test mode entry timing requirements. Note that for bit banging host programmers, which cannot generate a constant clock frequency of $f_{SWDCK\_ACQUIRE}$ on the SWDCK line for entire SWDCK packet duration, an alternate acquire method is explained in a later section.
3. After the host programmer receives an OK ACK for Port Acquire sequence, it must write the test mode key to the Test Mode Key register to enter PSoC 5 programming mode. This key must be written within time $T_{TESTMODE}$, as shown in Figure 3-2 and Figure 3-3. By ensuring that SWDCK is clocked at frequency of $f_{SWDCK\_ACQUIRE}$ during this step, the host programmer can enter PSoC 5 programming mode within time $T_{TESTMODE}$. The pseudo code for this step is given here.

```
APACC ADDR Write [0x0005 0210] // Address of
the Test mode key register
APACC DATA Write [0xEA7E 30A9] // Write 32-
bit test mode key

/* Exit on timeout or reception of FAULT
response means the device did not enter
Programming mode within time TTESTMODE. Retry
again by doing reset and restarting.*/
```

```
if (ACK != "OK" OR time_elapsed > T_TESTMODE usec) then FAIL_EXIT
```

3.1.1.2 SWD Programming using Power Cycle Mode:

Power cycle mode programming is identical to XRES method from a programming algorithm standpoint, as shown in Figure 3-2 and Figure 3-3. The only difference is that, instead of driving XRES pin, the host programmer toggles power to the PSoC 5 power supply pins (Vddd, Vdda, Vddio0, Vddio1, Vddio2, and Vddio3) to cause a device reset.

Power cycle method is complex to implement compared to XRES method because it requires special hardware design considerations for power toggling. Power cycle mode programming also requires that Vdda, Vddd, and Vddio power supply pins in PSoC 5 are tied to the same power supply and toggled at the same time, as shown in Figure 3-2. It is recommended to implement the XRES method of programming because it is easier to implement. Power cycle mode programming is required in the following case.

- If it is required to program PSoC 5 using the SWD interface's USB pins (P15[6], P15[7]), then the host programmer can toggle power to USB interface's VBUS pin to cause a device reset and program using the USB SWD pins. In this case, VBUS power pin in the USB interface powers the Vddd, Vdda, and Vddio power supply pins in PSoC 5.

Ramp Rate Requirements for Power Cycle Mode Programming

The maximum power supply ramp rate is specified in the PSoC 5 device datasheet as parameter Svdd. There is no minimum ramp rate requirement specified for power cycle mode. A slower ramp rate requires special hardware considerations as follows:

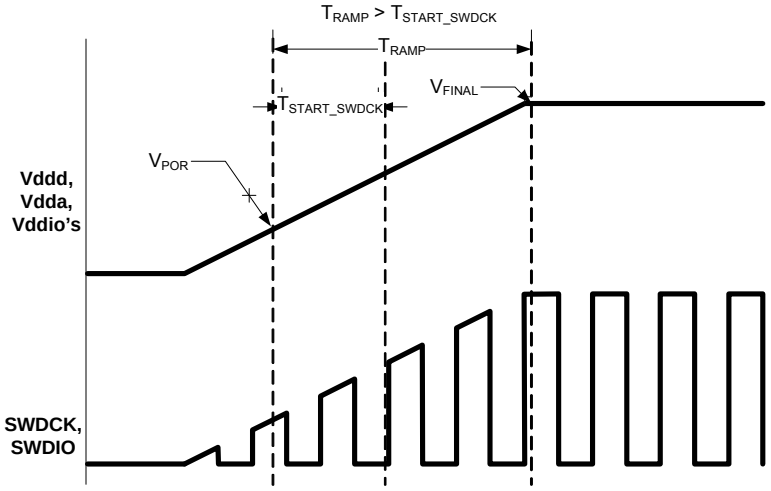
- When power supply ramp duration (T_{RAMP}) from VPOR to final value is less than T_{START_SWDCK}.

  Figure 3-3 shows that the host programmer must start sending the Port Acquire sequence within time duration T_{START_SWDCK} of Vddd and Vdda voltage levels crossing VPOR voltage level specification. If the time (T_{RAMP}) for power supplies to ramp from VPOR to final supply voltage is less than T_{START_SWDCK}, then the host programmer can start sending the Port Acquire sequence after Vddd, Vdda, and Vddio pins have reached final voltage value.

- When power supply ramp duration from VPOR to final value (T_{RAMP}) is more than T_{START_SWDCK}

  In this case, the host programmer cannot wait for power supplies to ramp to the final voltage value before sending the Port Acquire sequence. Otherwise, the host programmer cannot meet the timing requirements to enter PSoC 5 programming mode. The host programmer should implement the power cycle mode shown in Figure 3-4. It should start sending the Port Acquire sequence even as the power supplies (Vddd, Vdda, Vddio) ramp up. Adjust the voltage levels of SWDCK and SWDIO lines to match the instant value of power supply pins. This method is implemented in Cypress's MiniProg3 programmer in which the ramp rate duration (T_{RAMP}) is greater than T_{START_SWDCK}. This implementation ensures that the PSoC 5's test controller is able to detect data (logic levels) on the SWDIO and SWDCK lines even when power supply is ramping.

Figure 3-4. Power Cycle Mode Implementation for $T_{RAMP} > T_{START\_SWDCK}$



3.1.1.3 SWD Programming using Bit Banging Host Programmers:

Some host programmers implement the SWD interface as a bit banging implementation. Examples of such host programmers are microcontrollers in which the SWDIO and SWDCK signals are generated by writing to specific port registers of the microcontroller.

It is not possible for some of the bit banging programmers to generate the SWDCK clock signal at a constant frequency of $f_{SWDCK\_ACQUIRE}$ for entire SWD packet, as shown in Figure 3-2 and Figure 3-3. A modified method of entering PSoC 5 programming mode is given for these programmers. This method is applicable only for programmers that use the XRES pin. It is not applicable for power cycle mode programming due to the constraints it imposes on power supply ramp rates.

Figure 3-5 shows the modified steps to enter test mode of PSoC 5; Figure 3-6 shows the corresponding timing diagram. See Table 4-2 on page 30 for specifications of timing parameters. The primary need for SWDCK clocking at frequency of $f_{SWDCK\_ACQUIRE}$ is to meet the condition of "8 SWDCK clock cycles in time window $T_{ACQUIRE}$". On detection of these eight clocks, the time to enter test mode is extended to $T_{TESTMODE}$. The time window $T_{ACQUIRE}$ can occur anywhere during time T_{BOOT}. To simplify the implementation for bit banging programmers, the method in Figure 3-5 requires the programmer to toggle SWDCK alone at frequency of $f_{SWDCK\_ACQUIRE}$ with SWDIO held low. This ensures that the host programmer meets the initial test mode timing requirements. An example C code that implements Figure 3-5 is given here.

```
/* Set LOOP_COUNT value based on number of
       loop cycles needed to execute the
       “Initial Port Acquire window” loop
       below for time TBOOT */
#define LOOP_COUNT 240

uint16 j = 0; /* Variable to keep track of
       no. of times to generate SWDCK clock
       */

XRES_LOW; /* Generate active reset on XRES
       line for at least for time TRESET */
XRES_HIGH; /* Release XRES */

SWDIO_LOW; /* Hold the SWDIO line low during
       TBOOT */

/*---------------Initial Port Acquire
       window, TBOOT--------------------*/
do
{
       /* Ensure that SWDCK frequency
       is greater than fSWDCK_ACQUIRE */

       SWD_CLOCK_LOW;
       SWD_CLOCK_HIGH;
       j++;
       }while(j < LOOP_COUNT);
/*---------------End of Initial Port Acquire
       window---------------*/

/*Send Port Acquire key, Test mode address,
Test mode key SWD packets at frequency of
fSWDCK_BITBANG to complete all steps within
time TTESTMODE*/
```

After time T_{BOOT}, the programmer must send the port acquire, test mode key SWD packets. These SWD packets should be sent within time $T_{TESTMODE}$.

Figure 3-5. Enter Test Mode through SWD Interface (for bit banging programmers)

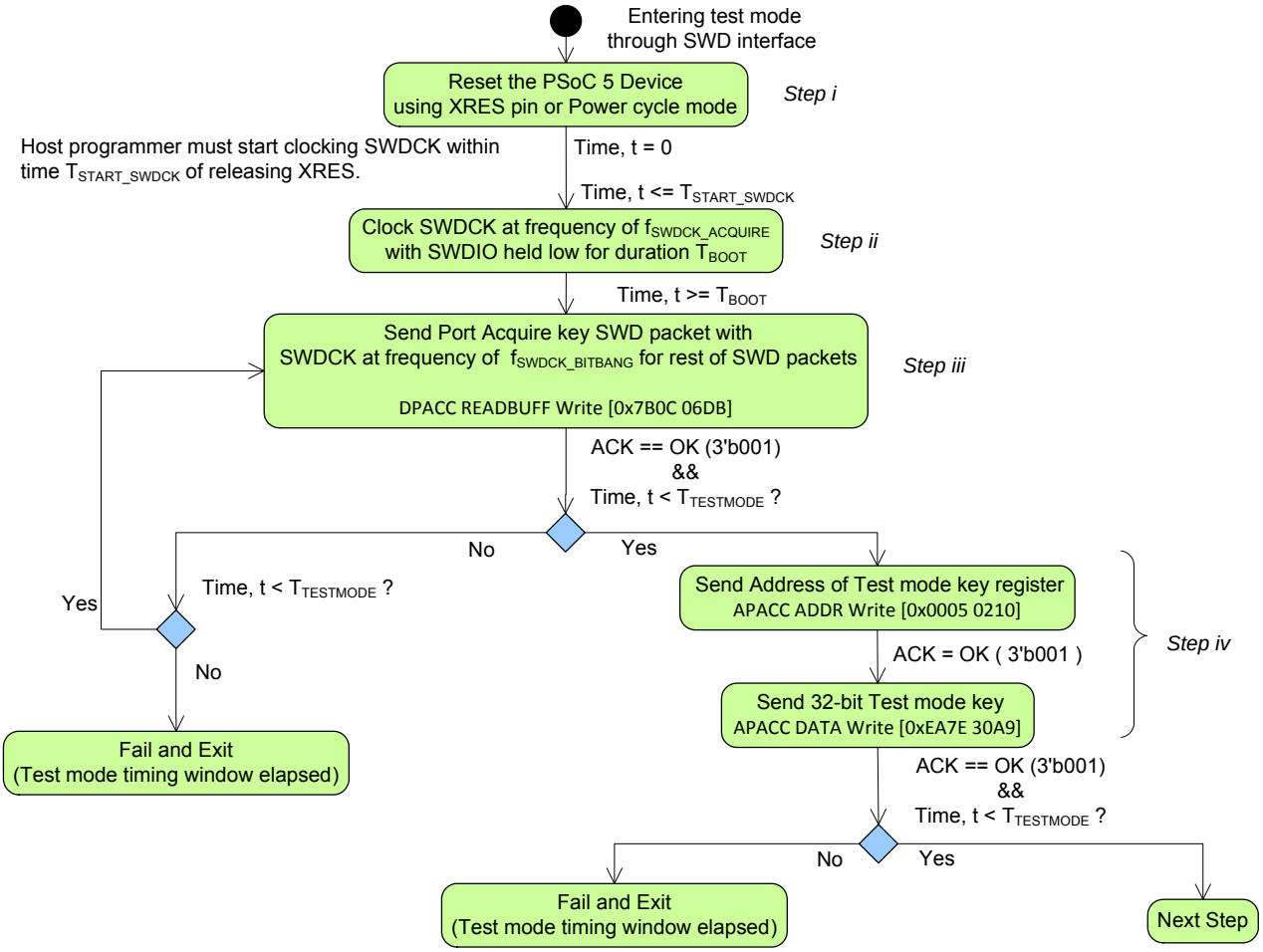
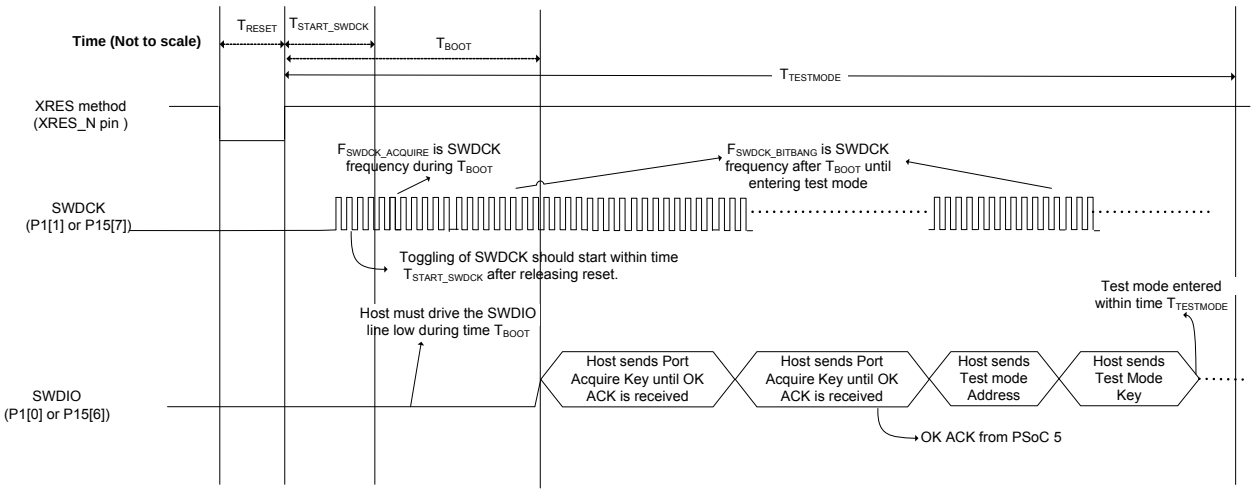


Figure 3-6. Timing Diagram to Enter Test Mode through SWD Interface (for bit banging programmers)



3.1.1.4 Determine $f_{SWDCK\_BITBANG}$:

In Figure 3-5, the programmer must send the SWD packets after time T_{BOOT} at a frequency of $f_{SWDCK\_BITBANG}$. This frequency requirement is to meet the $T_{TESTMODE}$ timing requirement. The value of $f_{SWDCK\_BITBANG}$ depends on bit banging programmer implementation. An example calculation for $f_{SWDCK\_BITBANG}$ that assumes no overhead in sending SWD packets is given here.

In PSoC 5, a maximum of two Port Acquire SWD packet tries are required to get OK ACK. The test mode address and test mode key require another two SWD packets. A maximum of four SWD packets must be sent by the programmer within time ($T_{TESTMODE}$ - T_{BOOT}). Minimum value of $T_{TESTMODE}$ from Table 4-2 on page 30 is 395 µs, and T_{BOOT} is 68 µs; the difference factor is 327 µs. Each SWD packet requires 49 SWDCK clock cycles (including the three dummy clock cycles at end of each SWD packet), and hence 196 SWDCK clock cycles are required for four SWD packets.

$T_{SWDCK\_BITBANG}$(no overhead) $\leq$ (327 µs/196) $\cong$ 1.6 µs

$f_{SWDCK\_BITBANG}$(no overhead) $\geq$ (1/1.6 µs) $\cong$ 0.7 MHz

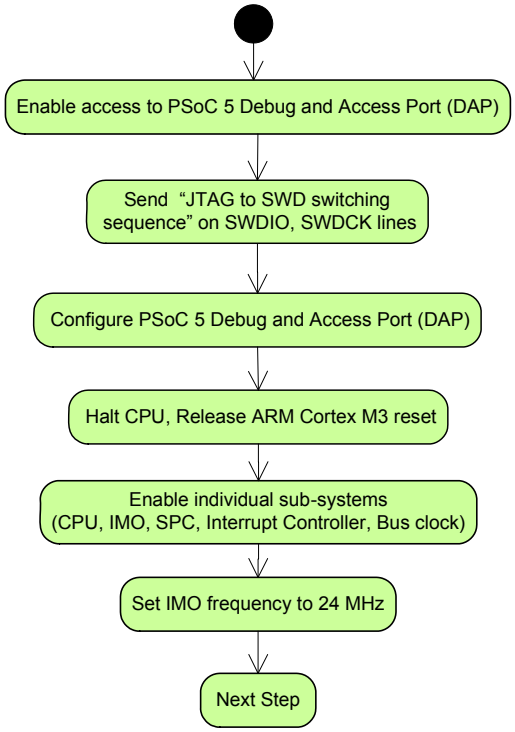
This example calculation assumes no overhead in sending the SWD packets on the host programmer side. The minimum frequency requirement increases with other additional overhead; this is specific to host programmer architecture.

The frequency parameter $f_{SWDCK\_BITBANG}$ refers to the average frequency of the SWDCK clock generated by host programmer. Bit banging programmers cannot generate constant frequency on SWDCK line during entire SWDCK packet. But the average SWDCK frequency must be greater than the minimum value of $f_{SWDCK\_BITBANG}$ so that the programming mode is entered within time $T_{TESTMODE}$.

3.2 Step 2: Configure Target Device

Figure 3-7 shows the sequence to configure the target PSoC 5 device before programming the device.

Figure 3-7. Configuring Target PSoC 5 Device



After entering Programming mode, the host programmer must do certain register writes to configure the target device. These are required to enable the PSoC 5 Debug and Access Port (DAP), configure the PSoC 5 DAP, halt the CPU, activate debug mode, enable different sub-systems (IMO, bus clock, CPU), and configure clocks (IMO).

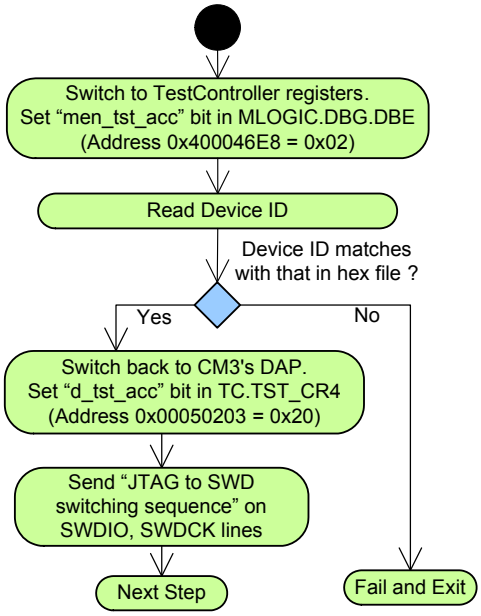
The access to the DAP in PSoC 5 is enabled by setting the 'd_tst_acc' bit in Test Controller register 'TC.TST_CR4'. The Test Controller also automatically switches from SWD to JTAG the first time 'd_tst_acc' bit is set, because the DAP always resets to JTAG the first time. Now, because the DAP is reset to JTAG interface, but the programming is required to be done through the SWD interface, the switching from JTAG to SWD mode must be done. See "JTAG to SWD Switching" on page 14 for detailed information on switching sequence.

3.3 Step 3: Verify Device ID

To ensure that the target device corresponds to the device for which the hex file is meant, the device ID of the target device must be compared against the Device ID information in the hex file. This ensures that hex file is completely compatible with Device under Test (DUT). If there is a mismatch in the device IDs, the programming operation should be stopped. See "Intel Hex File Format" on page 53 for information on the location of device ID in the hex file.

Device ID can be read using the SWD interface with a packet request containing ADDR = 00, APnDP = 0, and RnW = 1. The DAP access should also be disabled, and the device ID should be read in Test Controller mode. The DAP access should be enabled again after reading the device ID. The "JTAG to SWD Switching" on page 14 should be sent after the DAP access is enabled before proceeding to the next step. This is shown in Figure 3-8.

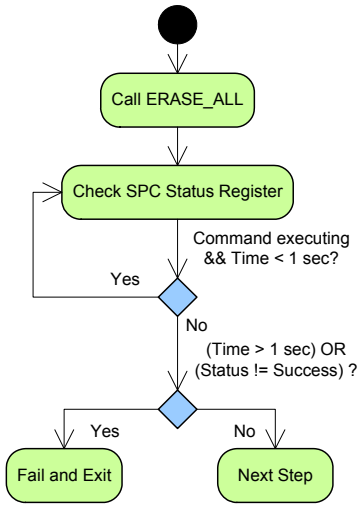
Figure 3-8. Verify Device ID of Target Device



3.4 Step 4: Erase Flash

Figure 3-9 demonstrates the Erase Flash process, which erases all flash data and configuration bytes, and all flash protection rows.

Figure 3-9. Erase Flash Sequence



All the nonvolatile memory (flash, EEPROM, NVL) erase and program operations are done through a simple command and status register interface. The Test Controller (TC) accesses programming operations by writing to the command data register (SPC_CPU_DATA) at address 32'h40004720. After providing a valid command, the host should wait until the command is executed. When a command is completed, the status is available in the status register (SPC_SR). The status register can be polled to see if the command is executed successfully.

These details are explained in "Nonvolatile Memory Programming" on page 56. For more information on nonvolatile memory programming, refer to the PSoC 5 Architecture TRM.

A single command requires several SWD writes to the command data register. The ERASE_ALL command has three parameters, and they should be written to the command data register. After calling ERASE_ALL, the target device starts erasing the entire flash. The ERASE_ALL command should not take longer than 1 second, otherwise an overtime error occurs.

3.5 Step 5: Program Flash

Flash memory in PSoC 5 is programmed in rows. Each row has 256 code bytes and 32 configuration bytes. The row latch to program the flash row is of size 288 bytes. The flash data to be programmed comes from hex file. See "Intel Hex File Format" on page 53 for information on the location of flash programming data in the hex file.

During the programming process, the row latch needs to be loaded with all the 288 bytes. In this scenario, the 256 code bytes should be fetched from main flash data region of hex file at address 0x0000 0000. The 32 configuration bytes should be fetched from the configuration data region of hex file at address 0x8000 0000. The programmer software should concatenate these 32 bytes with the 256 bytes to form the 288 byte row data that needs to be loaded in to the row latch. This step needs to be done to program all flash rows.

There are three parameters to consider in the flash programming process.

- Number of flash arrays (K) of flash memory: The value of 'K' depends on flash memory size. The flash memory in PSoC 5 is organized as flash arrays, where each flash array can have maximum size of 64 KB. Each flash array in turn is organized as rows, where the size of each row is 256 code bytes and 32 configuration bytes. The maximum flash size in the PSoC 5 family is 256 KB, and hence the maximum number of flash arrays possible in PSoC 5 is 4. Note that flash memory size given in device datasheet refers only to the code region of flash and not configuration region. A 256 KB flash memory implies that code region memory size is 256 KB.
  - K=1 for flash memory ≤ 64 KB,
  - K=2 for 64 KB < flash memory ≤ 128 KB,
  - K=3 for 128 KB < flash memory ≤ 192 KB,
  - K=4 for 192 KB < flash memory ≤ 256 KB.
- Number of rows (N) of flash memory: The value of 'N' depends on the flash memory size of the target device. For example, a 256 KB flash memory device has 1024 rows [(256K/256) = 1024 rows]. As mentioned previously, these rows are organized across multiple flash arrays depending on the flash memory size. A 256 KB flash memory has the 1024 rows organized as four flash arrays of 256 rows each. Also, note that the flash size parameter does not consider the size of configuration bytes. For example, a 64 KB flash size means that the code region capacity is 64 KB. It does not include the configuration bytes because this region cannot be used for code space, only for configuration data.
- Number of bytes per row (L) of flash memory: Each row of flash has 256 code bytes and 32 bytes of configuration data.

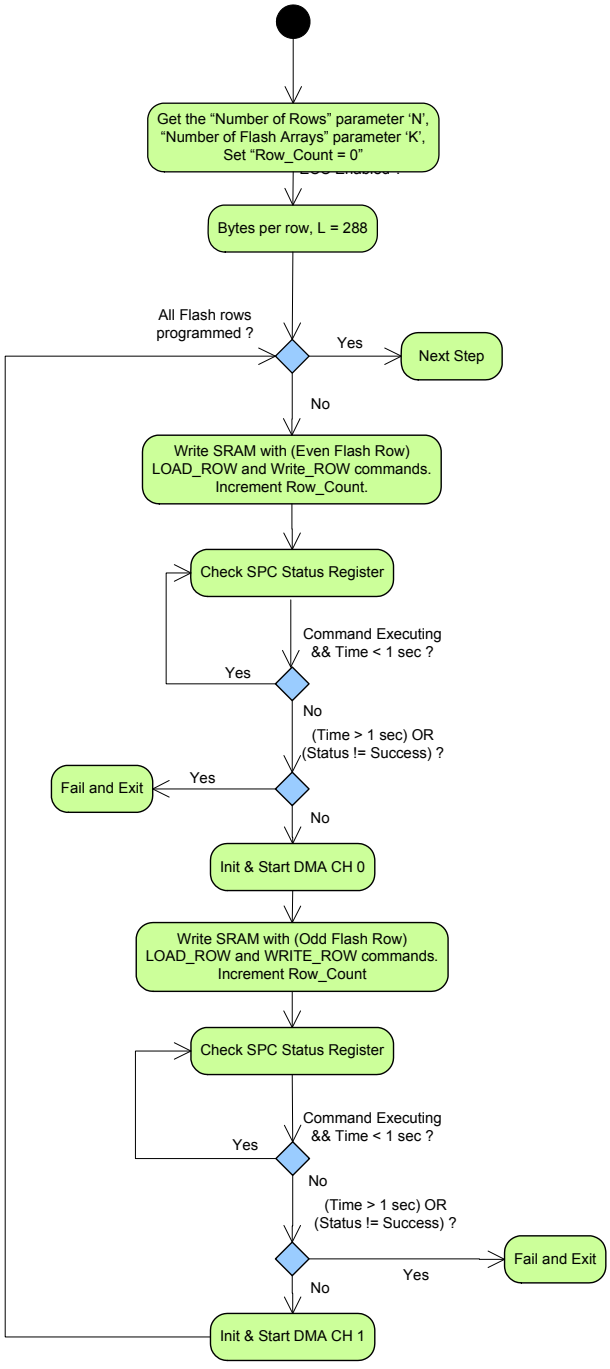
  L=288 bytes

Figure 3-10 demonstrates the flash row programming process. LOAD_ROW and WRITE_ROW commands are required to program flash. The LOAD_ROW command loads one row of flash data into the row latch and the WRITE_ROW command programs the latched data into the specified row of target flash. This process needs to be repeated for every row of flash array, and for all flash arrays. See "Step 5: Program Flash" on page 34 for more details on the command implementation.

It takes time to load and then program each flash row. Direct Memory Access (DMA) can speed up this process, because the DMA runs in parallel with the flash operations. It can call commands through two DMA channels, such that one channel can load row data and then call WRITE_ROW, and the other channel can start loading data for the next row while the previous command is still programming.

WRITE_ROW command is used for programming instead of PROGRAM_ROW as the latter API does not function in PSoC 5. The constant value of the die's temperature is passed to WRITE_ROW API (+25 °C in this case).

Figure 3-10. Program Flash



3.6 Step 6: Verify Flash (Optional)

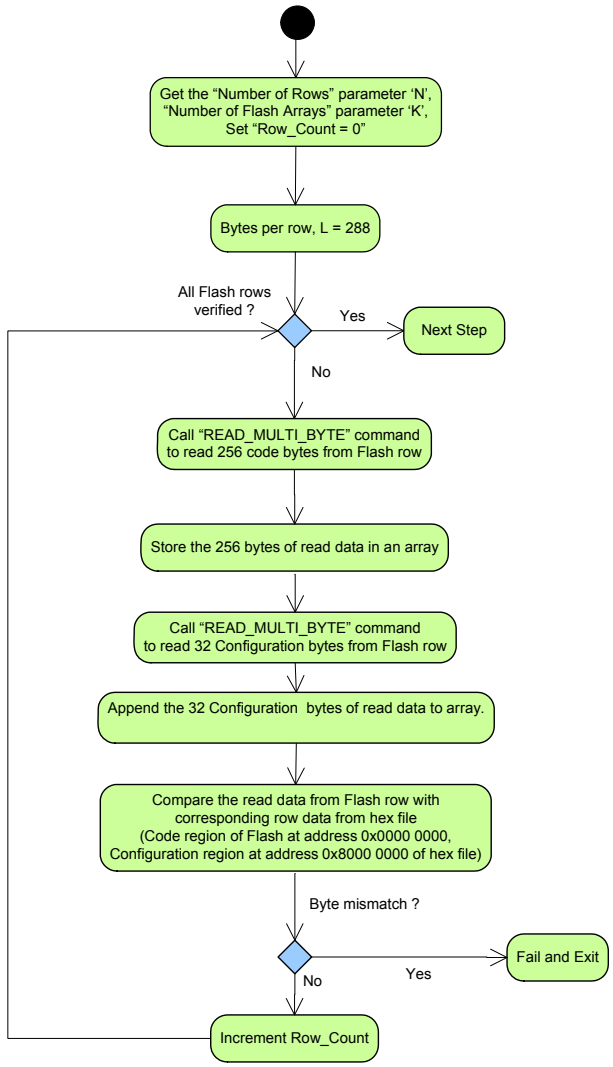
Figure 3-11 demonstrates the flash read process. This optional step allows reading back and verifying data programmed in the Program Flash step. This step should be done before the Checksum Validation step.

The READ_MULTI_BYTE command is used to read out all bytes in flash rows. Each read command can read out a maximum 256 code bytes. Apart from this, the 32 bytes of configuration data need to be read out. To read this data, call the READ_MULTI_BYTE command again, addressed to point to that configuration data. The number of returned data should be set to 32. This cycle needs to be repeated for all flash rows in all flash arrays.

After reading the data for one flash row, it should be verified with the corresponding flash row data in hex file. If there is mismatch in even one of the bytes, the programming process should be stopped and restarted.

Note that in the hex file, the code region in flash row (256 bytes) starts at address 0x0000 0000 of hex file. The 32 configuration bytes for flash row are present starting at address 0x8000 0000 of hex file. 256 bytes from the code region (0x0000 0000 of hex file) and 32 bytes from the configuration region (0x8000 0000 of hex file) must be concatenated to form a flash row.

Figure 3-11. Verify Flash Sequence

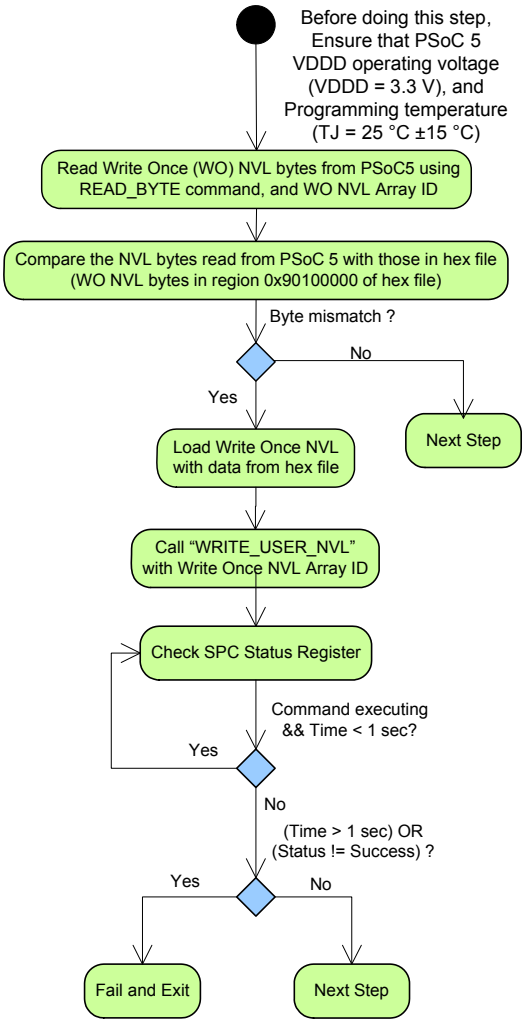


3.7 Step 7: Program WO NVL (Optional)

Programming of Write Once Nonvolatile Latch (WO NVL) is an optional step that is required only if the Device Security feature is required. If this step is included in the programming flow, there are conditions imposed on the PSoC 5 Vddd operating voltage (Vddd ≤ 3.3 V) and programming temperature (TJ = 25 °C ±15 °C). See Figure 1-1 for more details on this condition. These conditions are not applicable if the WO NVL programming step is not included in the programming sequence. Note that programming of WO NVL with the correct 32-bit key will make the device One Time Programmable (OTP). Include this step after understanding its implications and only if it is required for the end application. It is recommended to have this step as an optional selection in your programmer software’s graphical user interface (GUI) such as a checkbox; have it unchecked by default. See “Nonvolatile Memory Organization in PSoC 5” on page 56 for details on the Device Security feature that is supported by WO NVL.

Figure 3-12 shows the Program Write Once Nonvolatile Latch setup flow. This step writes the 4-byte Write Once (WO) NVL. The data to be written to the NVL is located in address 32’h90100000 of the hex file. The LOAD_BYTE and WRITE_USER_NVL commands are used in this step. The LOAD_BYTE command loads the data one byte at a time to a 4-byte latch. The WRITE_USER_NVL command writes the four bytes of data in the latch to NVL. Therefore, the LOAD_BYTE command needs to be called four times, followed by one WRITE_USER_NVL command. The SPC status register needs to be polled to check when the command finishes the write operation. The WRITE_USER_NVL command should not take longer than 1 second, otherwise an overtime error occurs.

Figure 3-12. Program Write Once NVL

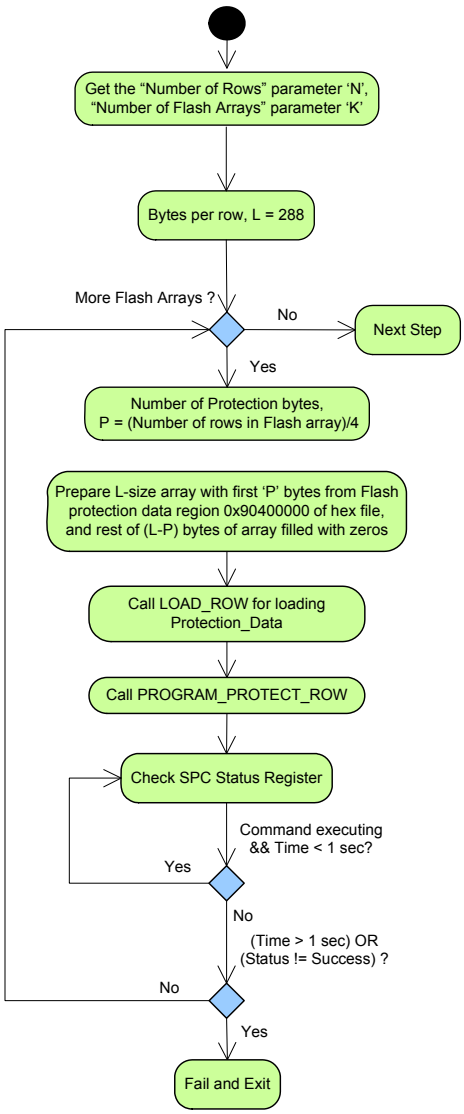


The NVLs in PSoC 5 have much lesser endurance compared to flash and EEPROM memory. Due to this, the write-once NVL is written only if new data needs to be programmed into the latch. This ensures that the latches are programmed only when there is change in the 4-byte security key in hex file, which in turn maximizes the endurance time.

3.8 Step 8: Program Flash Protection

Figure 3-13 shows the sequence to program the protection rows in flash.

Figure 3-13. Program Flash Protection Sequence

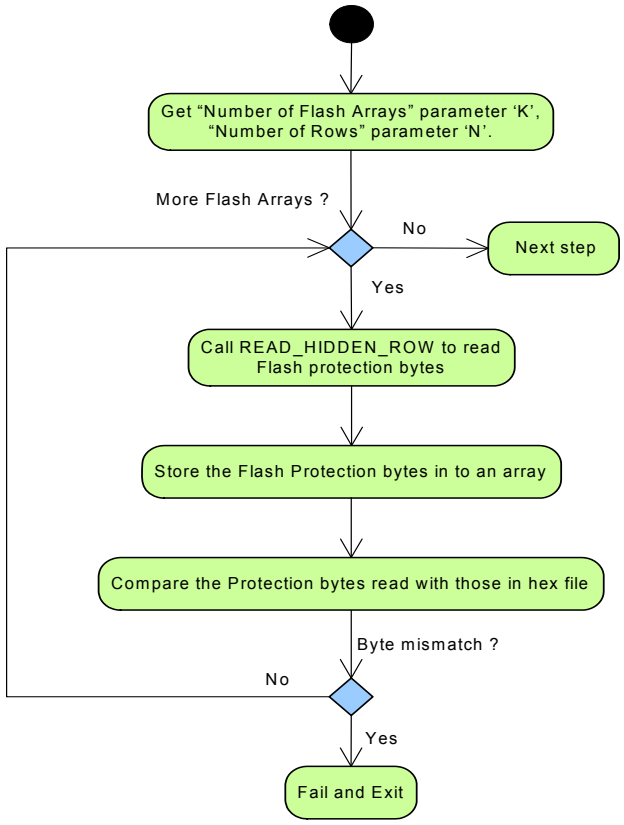


The protection rows start in address 32'h90400000 in the hex file, as shown in "Intel Hex File Format" on page 53. Each protection byte stores protection settings of four flash rows. Each flash array in PSoC 5 can have a maximum of 256 flash rows and hence a maximum of 64 flash protection bytes. The remaining bytes ((L–P) bytes) needed for the LOAD_ROW command are initialized with zeros, as shown in Figure 3-13. This programming of flash protection data should be done for one flash array at a time and should be repeated for all flash arrays.

3.9 Step 9: Verify Flash Protection (Optional)

Figure 3-14 explains the flash protection data verification procedure. This step is optional, and it allows reading back and verifying the data programmed in the Program Flash Protection step. It is recommended that third party programmers include this step to validate data programmed.

Figure 3-14. Verify Flash Protection Sequence

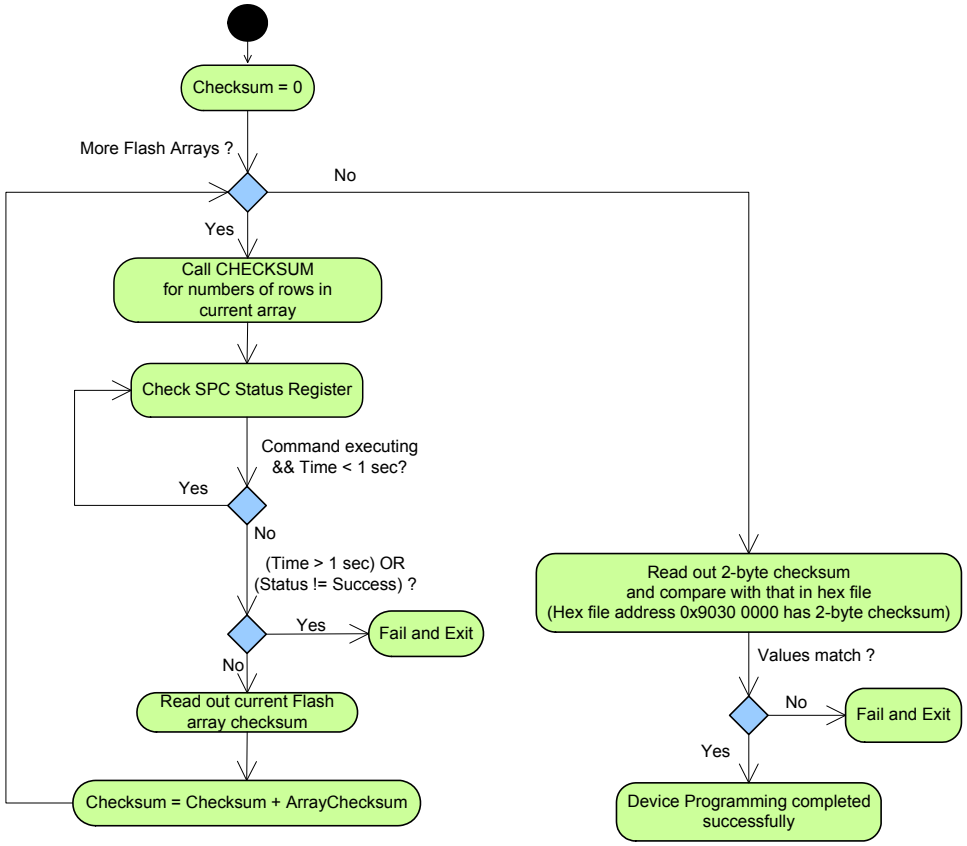


The READ_HIDDEN_ROW command is used to read out all bytes in the Flash Protection row. This command always returns 256 bytes irrespective of the number of valid flash protection bytes. Each protection byte stores protection settings of four flash rows. Each flash array in PSoC 5 can have a maximum of 256 flash rows, and hence maximum of 64 flash protection bytes. The remaining bytes returned by the READ_HIDDEN_ROW command should be ignored during the verification step. The step should be repeated for all the flash arrays.

3.10 Step 10: Checksum Validation

Figure 3-15 demonstrates the checksum validation step. This step validates that the programming operation is successful by doing a checksum on the flash memory data. The computed checksum is only for the code region and the configuration region of flash memory. Flash protection data is not included in the checksum computation. The programmer software needs to locally compute the checksum for all flash rows in all flash arrays, so that it can be compared to the value read out from the target device. The CHECKSUM command is used to compute and return the checksum value, which can be read out through the data register at 32’h40004720. The checksum is a 4-byte value, so four SWD read transfers are required. Only the lower two bytes of this 4-byte value returned from the target device should be taken for comparison as the hex file stores only 2-byte checksum. If the lower 2-byte checksum values mismatch, terminate the programming process. In the hex file, the 2-byte checksum of all flash rows is stored at address 0x9030 0000 of hex file (MSB byte first). This is explained in “Intel Hex File Format” on page 53.

Figure 3-15. Checksum Validation Sequence Block Diagram



3.11 Step 11: Program EEPROM (Optional)

EEPROM nonvolatile memory in PSoC 5 is used to store constant data such as calibration data and lookup table. Some applications might require the EEPROM memory in PSoC 5 to be initialized as part of the device programming sequence. The programmer software can provide a configuration option to the end user to select whether or not to include the EEPROM initialization as part of programming sequence. The "Program EEPROM" and "Verify EEPROM" steps can be included if that option is selected.

The number of rows in the EEPROM memory of the PSoC 5 device can be calculated based on the EEPROM memory size in bytes given in the device datasheet. The EEPROM is written row wise with the programming data coming from the EEPROM region of the hex file, as explained in A.1 Intel Hex File Format.

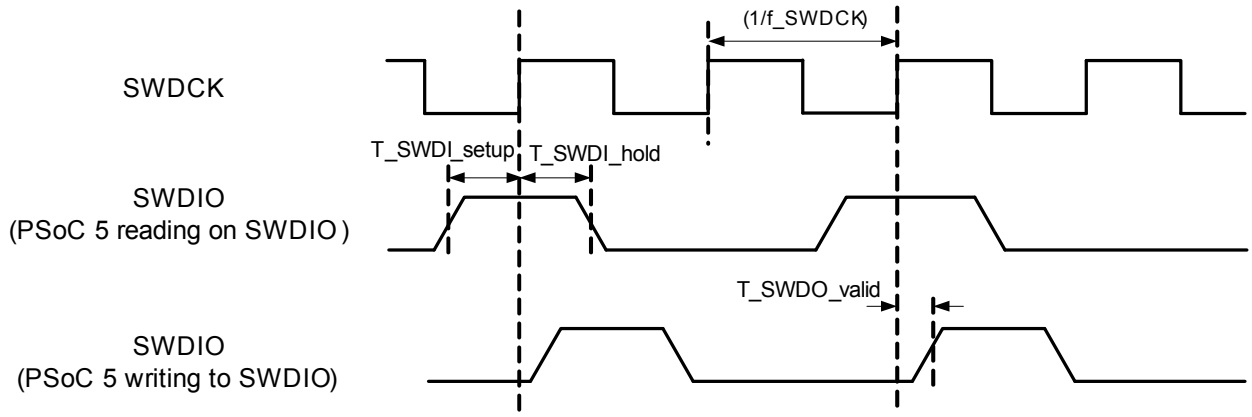
3.12 Step 12: Verify EEPROM (Optional)

This step verifies the integrity of the EEPROM program operation by ensuring the EEPROM data read from the device matches the data in the hex file. This step should be included only if the "Program EEPROM" step is also included. The EEPROM data is read from the device by directly accessing the EEPROM memory address through the Debug and Access Port (DAP) interface. The step is successful if all the EEPROM bytes read from the device matches with the corresponding hex file data.

4. Programming Specifications

4.1 SWD Interface Timing and Specifications

Figure 4-1. SWD Interface Timing



The external host programmer should do all read or write operations on SWDIO line on falling edge of SWDCK, and PSoC 5 will do corresponding write, or read operations on SWDIO on rising edge of SWDCK.

Table 4-1. SWD Interface AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
f_SWDCK	SWDCLK frequency	3.3V ≤ Vddd ≤ 5 V	–	–	8[a]	MHz
		2.7 V ≤ Vddd < 3.3 V	–	–	7	MHz
		2.7 V ≤ Vddd < 3.3 V, SWD over USBIO pins	–	–	5.5	MHz
T_SWDI_setup	SWDIO input setup before SWDCK high	T = 1/f_SWDCK max	T/4	–	–	
T_SWDI_hold	SWDIO input hold after SWDCK high	T = 1/f_SWDCK max	T/4	–	–	
T_SWDO_valid	SWDCK high to SWDIO output	T = 1/f_SWDCK max	–	–	2T/5	

a. The maximum frequency of 8 MHz is less than device datasheet specification as the CPU clock frequency is configured for a fixed frequency of 24 MHz in the programming algorithm, and the f_SWDCK must be no more than 1/3 CPU clock frequency.

4.2 Programming Mode Entry Specifications

Table 4-2. PSoC 5 Programming Mode Entry Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
T_{RESET}	Reset signal (XRES) pulse width (active low)		1	–	–	µs
T_{START_SWDCK}	Maximum time from release of device reset to start of SWDCK signal clocking by host programmer		–	4	–	µs
$T_{ACQUIRE}$	Initial Port Acquire window		6.1	8	9	µs
T_{BOOT}	Time for device boot process to complete after releasing reset		–	68	–	µs
$T_{TESTMODE}$	Time window to enter Programming mode (Test mode)		395	420	430	µs
$f_{SWDCK\_ACQUIRE}$	SWDCK clock frequency during Port Acquire, Test mode entry	f_SWDCK max is from Table 4-1	1.4	–	f_SWDCKmax	MHz
$f_{SWDCK\_BITBANG}$	Average SWDCK clock frequency during Port Acquire, Test mode entry for bit banging SWD interface programmers	f_SWDCK max is from Table 4-1. The minimum frequency is assuming no overhead or delay between SWD packets.	0.7	–	f_SWDCKmax	MHz
V_{POR}	Vddd, Vdda rising trip voltage		–	1	–	V

See the PSoC 5 device datasheet for other specifications such as minimum device operating voltage and nonvolatile memory specifications.

5. SWD Vectors for Programming

5.1 Step 1: Enter Programming Mode

This is the first step in programming procedure; the timing requirements are specified in Table 4.2 on page 30. Depending on the programming interface used, the appropriate method to enter PSoC 3's programming mode should be used from the following methods. A separate method is provided for bit banging programmers that need to program PSoC 3 through SWD interface. Detailed information on all these methods are provided in "Step1: Enter Programming Mode" on page 16.

5.1.1 Method A

```
/*--- Entering Programming mode through SWD Interface using XRES or Power cycle mode---*/
/* --------For Programmers with Hardware SWDCK generation capability------------*/
/* Based on Test mode entry flowchart given in Figure 3-2, Table 4-2 on page 30 */
```

Step i.) Do device reset using XRES pin or Power cycle mode

```
time_elapsed = 0
```

Step ii) Start sending Port Acquire key within time T_{START_SWDCK} of releasing XRES pin high (for XRES mode) or Vddd, Vdda voltages crossing V_{POR} voltage level (for Power Cycle mode). SWDCK frequency during this step should be $f_{SWDCK\_ACQUIRE}$.

```
do
{
/* Write Port Acquire key, Use SWD ADDR = 2'b11*/
    DPACC READBUFF Write [0x7B0C 06DB]

} while (ACK != "OK" AND time_elapsed < T_TESTMODE)//Check port acquire retry time

if (ACK != "OK" OR time_elapsed > T_TESTMODE) then FAIL_EXIT // Exit on timeout
```

Step iii) Send SWD packets for entering test mode. SWDCK frequency during this step should be $f_{SWDCK\_ACQUIRE}$. This step should be completed within time $T_{TESTMODE}$, as given below.

```
APACC ADDR Write [0x0005 0210] // Address of the Test mode key register
APACC DATA Write [0xEA7E 30A9] // Write 32-bit test mode key

/* Exit on timeout or reception of FAULT response which means the device did not enter
Programming mode within time T_TESTMODE. Retry again by doing reset and restarting.*/
if (ACK != "OK" OR time_elapsed > T_TESTMODE) then FAIL_EXIT

else NEXT_STEP /* Entered PSoC 3 Programming mode */
```

5.1.2 Method B

```
/* ---------------Entering Programming mode through SWD Interface using XRES pin------------*/
        /* --------For Bit Banging Host Programmers ------------*/
/* Based on Test mode entry flowchart given in Figure 3-5, Table 4-2 on page 30*/
```

Step i.) Do device reset using XRES pin

```
time_elapsed = 0
```

Step ii.) Clock SWDCK at frequency of $f_{SWDCK\_ACQUIRE}$ for time T_{BOOT}. SWDIO pin of PSoC 3 should be driven low by the Host during time T_{BOOT}. Host should start clocking SWDCK within time T_{START_SWDCK} of releasing XRES pin high.

```
time_elapsed = T_BOOT
```

Step iii) Start sending Port Acquire key in a loop after time T_{BOOT}. Average SWDCK frequency during this step should be $f_{SWDCK\_BITBANG}$.

```
do
{
/* Write Port Acquire key, Use SWD ADDR = 2'b11*/
    DPACC READBUFF Write [0x7B0C 06DB]

} while (ACK != "OK" AND time_elapsed < T_TESTMODE)//Check port acquire retry time

if (ACK != "OK" OR time_elapsed > T_TESTMODE) then FAIL_EXIT // Exit on timeout
```

Step iv) Send SWD packets for entering test mode. Average SWDCK frequency during this step should be $f_{SWDCK\_BITBANG}$. This step should be completed within time $T_{TESTMODE}$ as given below.

```
APACC ADDR Write [0x0005 0210] // Address of the Test mode key register
APACC DATA Write [0xEA7E 30A9] // Write 32-bit test mode key

/* Exit on timeout or reception of FAULT response which means the device did not enter
Programming mode within time T_TESTMODE. Retry again by doing reset and restarting.*/
if (ACK!= "OK" OR time-lapse > T_TESTMODE) then FAIL_EXIT

else NEXT_STEP /* Entered PSoC 3 Programming mode */
```

5.2 Step 2: Configure Target Device

```
APACC ADDR Write [0x0005 0203] //Address of the Test mode register TC.TST_CR4
APACC DATA Write [0x0000 0020] //Switch to "Cortex-M3" IO space (from "Test Mode"), DAP
                               //switches to JTAG automatically. So do SWD switching as below
```

Switching to SWD mode - For programming through SWD interface, the host must switch to SWD mode by sending a switching sequence on SWD lines. See "JTAG to SWD Switching" on page 14 section for details on this. After completing the SWD switching, execute below steps.

```
DPACC DP CTRLSTAT Write [0x50000000] //Configure DP Control & Status Register

DPACC DP SELECT Write [0x00000000] // Clear DP Select Register
```

```
APACC AP CTRLSTAT Write [0x22000002] // Set 32-bit transfer mode of DAP

APACC ADDR Write [0xE000 EDF0]
APACC DATA Write [0xA05F 0003]       //Halt CPU and Activate Debug

APACC ADDR Write [0x4008 000C]
APACC DATA Write [0x0000 0002]       // Release Cortex-M3 CPU Reset

APACC ADDR Write [0x4000 43A0]
APACC DATA Write [0x0000 00BF]       // Enable individual sub-system of chip

APACC ADDR Write [0x4000 4200]
APACC DATA Write [0x0000 0002]       // IMO set to 24 MHz
```

5.3 Step 3: Verify Device ID

```
APACC ADDR Write [0x4000 46E8]
APACC DATA Write [0x0000 0002] //Switch to Test controller mode from DAP mode.


/* Compare the 4-byte Device ID in Hex file (exp_idcode) at address 0x90500002 of hex file
with the Target Device ID. Abort programming operation if Device ID's mismatch
4-byte Device ID in hex file is in Big-endian format. See "Intel Hex File Format" on page 53
for details */

if (DPACC IDCODE Read != exp_idcode) then FAIL_EXIT // Exit on Device ID mismatch

APACC ADDR Write [0x0005 0203] //Address of the Test mode register TC.TST_CR4
APACC DATA Write [0x0000 0020] //Switch to "Cortex-M3" IO space (DAP mode)
```

Switching to SWD Mode. To program through the SWD interface, the host must switch to the SWD mode by sending a switching sequence on SWD lines. See "JTAG to SWD Switching" on page 14 for details. After completing the SWD switching, execute the following steps.

5.4 Step 4: Erase All (Entire Flash Memory)

```
APACC ADDR Write [0x4000 4720]   // SPC data register address
APACC DATA Write [0x0000 00B6]   // First initiation key

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 00DC]  // Second key:00DC(0xD3 + 0x09); 0x09 is Erase All opcode

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 0009]  // ERASE_ALL opcode

/*Read SPC status register to check the status of SPC command. If "Command Success" status is
not received within 1 second, then exit the programming operation */
APACC ADDR Write [0x4000 4722]// SPC status register address
dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

time_elapsed = 0
int32 StatusReg //To store SPC_SR status register value
```

```
do
{
     StatusReg = APACC DATA Read // Save status register value to a local variable
     StatusReg = (StatusReg >> 16) & 0xFF   // Extract status code which is in 3rd byte
} while ((StatusReg != [0x0000 0002]) AND time_elapsed < 1 sec);

if (time_elapsed > 1 sec) then FAIL_EXIT
```

5.5 Step 5: Program Flash

The data for this section is located in address 0x0000 0000 and 0x8000 0000 of the hex file. This step requires three parameters: K- Number of Flash arrays, N- Total number of Flash rows, and L- Number of bytes in row (L=288). K and N are derived from the total flash memory size of the device, and the L value is fixed to 288. See the respective device datasheet for flash memory size of each device.

```
//Calculating total number of rows 'N'
int32 N = (Total_Flash_Code_size)/256; //Each row has 256 code bytes, "Total_Flash_size" is
                                       //in bytes

//Calculating total number of Flash arrays 'K'
if (N % 256 == 0)
{
   byte K= (byte) N/256; //If rows are exact multiple of 256,quotient of (N/256) gives 'K'
}
else
{
   byte K= (byte)((N/256) + 1); //If rows are not exact multiple of 256,increment quotient of
                                // (N/256) by one for 'K's
}
int16 RowsPerArray; //Variable that hold the number of data rows in current Flash array

/* Setting AP Control/Status register configuration register for four-byte access to SRAM.
LSB 3bits: 4 - "4 byte", 2 - "2 byte",  0 - "1 byte" mode - already set during chip initial-
ization */
APACC AP CTRLSTAT WRITE [0x22000002]

// Program all Flash Arrays
for (byte ArrayCount = 0; ArrayCount < K; ArrayCount++)
{
  // Find number of rows in current array
   if (ArrayCount == (K-1))
   {
      RowsPerArray = N – (ArrayCount*256); //Last array may have less than 256 rows
   }
   else
   {
      RowsPerArray = 256; //Except last flash array, rest of them have 256 rows
   }

   int16 RowCount = 0;

   //Program Rows
   while (Row_Count < RowsPerArray)
   {
     //-------------Programming EVEN ROW ----------------------

     //"B6" - SPC_KEY1, "D5" - SPC_KEY2, "02" - LOAD_ROW opcode,
```

```
// ArrayCount - Flash ArrayID
APACC ADDR Write [0x2000 0000]// SRAM address- 32'h20000000
APACC DATA Write [(0x0002 D5B6) | (ArrayCount << 24)] // 4 byte data

int16 Byte_Count = 0

/*Send Row data to SRAM from HEX file. Each row needs 288 bytes (256 Code bytes + 32
  Configuration bytes) for programming. The 256 code bytes for row are present
  starting at address 0x00000000 of hex file. The 32 Configuration bytes are present
  starting at address 0x80000000 of hex file. Thus a single row data is formed by
concatenating these 256 code bytes and 32 configuration bytes to form a 288-byte row
  data. See "Intel Hex File Format" on page 53 for more details. */
while (Byte_Count < L)    // L = 288
{
   APACC ADDR Write [(0x20000000) + Byte_Count + 0x4]
   APACC DATA Write [d3d2d1d0] // Write 4 bytes at a time, 4-bytes are from hex file
   Byte_Count = Byte_Count + 4
}

//"00","00","00" - 3 NOPs for short delay, "B6" - SPC_KEY1
APACC ADDR Write [(0x20000000)+ (L - 1) + 0x05]
APACC DATA Write [0xB600 0000]

 //"D8" -SPC_KEY1+SPC_WR_ROW, "05" -SPC_WR_ROW, " ArrayCount" -Flash Array ID,
//"00" - High Byte of RowCount,"01" – temperature Sign, "19" – temp Magnitude (+25C)
APACC ADDR Write [0x20000000 + (L - 1) + 0x09]
APACC DATA Write [(0x000005D8) | (ArrayCount << 16)]

APACC ADDR Write [0x20000000 + (L - 1) + 0xD]
APACC DATA Write [(RowCount & 0xFF) | (0x1901 << 8)]//Low byte of row number
                                                    //and Die's temperature (+25 C)

//DMA operations

APACC ADDR Write [0x4000 7018]// PHUB_CH0_STATUS Register
APACC DATA Write [0x0000 0000]// Disable chain event, use TDMEM1_ORIG_TD0

APACC ADDR Write [0x4000 7010]// PHUB_CH0_BASIC_CFG register
APACC DATA Write [0x0000 0021] // Enable DMA CH 0

APACC ADDR Write [0x4000 7600]// PHUB_CFGMEM0_CFG0 register
APACC DATA Write [0x0000 0080]// DMA request is required for each burst

APACC ADDR Write [0x4000 7604]// PHUB_CFGMEM0_CFG1 register
APACC DATA Write [0x4000 2000] // Sets upper 16-bit address of destination/source

APACC ADDR Write [0x4000 7800]//PHUB_TDMEM0_ORIG_TD0 register
APACC DATA Write [(0x01FF 0000) + L + 15]  // Set TD transfer counts

APACC ADDR Write [0x4000 7804] // PHUB_TDMEM0_ORIG_TD1 register
APACC DATA Write [0x4720 0000] // Set lower 16-bit address of the destination/source

//Wait until SPC has done previous request
APACC ADDR Write [0x4000 4722]// SPC status register address
dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

time_elapsed = 0
int32 StatusReg //To store SPC_SR status register value
```

```
do
{
    StatusReg = APACC DATA Read
    StatusReg = (StatusReg >> 16) & 0xFF  // Extract status code which is in 3rd byte
} while ((StatusReg != [0x0000 0002]) AND time_elapsed < 1 sec)

if (time_elapsed > 1 sec) then FAIL_EXIT

APACC ADDR Write [0x4000 7014]// PHUB_CH0_ACTION register
APACC DATA Write [0x0000 0001]// This creates a direct DMA request for channel '0'

// DMA will transfer data from SRAM, and call LOAD_ROW and then WRITE_ROW
//When the DMA is transferring data using Channel '0', configure Channel '1'
//to speed up programming time

//-------------Programming ODD ROW ----------------------

Row_Count = Row_Count + 1 // Increment row count and repeat process for the next row

//"B6"-SPC_KEY1, "D5"-SPC_KEY2, "02"-LOAD_ROW opcode, "ArrayCount"-ArrayID
APACC ADDR Write [0x2000 0200]// SRAM address 32'h200
APACC DATA Write [0x0002 D5B6 | (ArrayCount << 24)]// 4-byte data as commented above

/*Send Row data to SRAM from HEX file. Each row needs 288 bytes (256 Code bytes + 32
Configuration bytes) for programming. The 256 code bytes for row are present
starting at address 0x00000000 of hex file. The 32 Configuration bytes are present
starting at address 0x80000000 of hex file. Thus a single row data is formed by
concatenating these 256 code bytes and 32 configuration bytes to form a 288-byte row
data. See "Intel Hex File Format" on page 53 for more details. */

Byte_Count = 0
while (Byte_Count < L)//L = 288
{
     APACC ADDR Write [0x20000000 + Byte_Count + 0x204]
     APACC DATA Write [d3d2d1d0] // Write 4 bytes at a time, 4-bytes are from
                                 //hex file
     Byte_Count = Byte_Count + 4
}

//"00","00","00" - 3 NOPs for short delay, "B6" - SPC_KEY1
APACC ADDR Write [0x20000000 + (L - 1) + 0x205]
APACC DATA Write [0xB600 0000]

//"D8" - SPC_KEY1+SPC_WR_ROW, "05" - SPC_WR_ROW, " ArrayCount" – Array ID
//"00" - High Byte of RowCount,"01" – temp Sign,"19" – temp Magnitude (+25C)
APACC ADDR Write [0x20000000 + (L - 1) + 0x209]
APACC DATA Write [0x000005D8 | (ArrayCount << 16)]// 0xD8 = 0xD3 + 0x05(
                                                   //"WRITE_ROW" opcode)

APACC ADDR Write [0x20000000 + (L - 1) + 0x20D]
APACC DATA Write [(RowCount & 0xFF) | (0x1901 << 8)] //Low byte of row #
                                              //and Die's Temperature

//DMA operations
APACC ADDR Write [0x4000 7028]// PHUB_CH1_STATUS Register
APACC DATA Write [0x0000 0100] // Disable chain event, use TDMEM1_ORIG_TD1
```

```
     APACC ADDR Write [0x4000 7020]// PHUB_CH1_BASIC_CFG register
     APACC DATA Write [0x0000 0021]// Enable DMA CH 0

     APACC ADDR Write [0x4000 7608]// PHUB_CFGMEM1_CFG0 register
     APACC DATA Write [0x0000 0080]// DMA request is required for each burst

     APACC ADDR Write [0x4000 760C]// PHUB_CFGMEM1_CFG1 register
     APACC DATA Write [0x4000 2000]// Sets upper 16-bit address of
                                   //destination/source

     APACC ADDR Write [0x4000 7808]
     APACC DATA Write [(0x01FF 0000) + L + 15]

     APACC ADDR Write [0x4000 780C] // PHUB_TDMEM1_ORIG_TD1 register
     APACC DATA Write [0x4720 0200] // Set lower 16-bit address of the
                                   //destination/source

     //Wait until SPC has done previous request
     APACC ADDR Write [0x4000 4722]// SPC status register address
     dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

     time_elapsed = 0
     int32 StatusReg //To store SPC_SR status register value
     do  // Poll status register
     {
          StatusReg = APACC DATA Read
          StatusReg = (StatusReg >> 16) & 0xFF // Extract status code which is in 3rd byte
     } while ((StatusReg != [0x0000 0002]) AND time_elapsed < 1 sec)

     if (time_elapsed > 1 sec) then FAIL_EXIT

     APACC ADDR Write [0x4000 7024]  // PHUB_CH1_ACTION register
     APACC DATA Write [0x0000 0001] //Creates a direct DMA request to Channel '1'.
                                   // DMA will transfer data from SRAM, and call
                                   //LOAD_ROW and then WRITE_ROW

     Row_Count = Row_Count + 1

   }    //Repeat for all rows of one Flash array

} //Repeat for all Flash arrays

//Make sure that last SPC request is completed
APACC ADDR Write [0x4000 4722]// SPC status register address
dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

time_elapsed = 0
int32 StatusReg //To store SPC_SR status register value
do// Poll status register
{
   StatusReg = APACC DATA Read
   StatusReg = (StatusReg >> 16) & 0xFF  // Extract status code which is in 3rd byte
} while ((StatusReg != [0x0000 0002]) AND time_elapsed < 1 sec)

if (time_elapsed > 1 sec) then FAIL_EXIT
```

5.6 Step 6: Verify Flash (Optional)

This step requires three parameters: K- Number of flash arrays, N- Total number of flash rows, L- Number of bytes in row (L = 288). K and N are derived from the total flash memory size of the device, and the L value is fixed to 288. See the respective device datasheet for flash memory size of each device.

```
//Calculating total number of rows 'N'
int32 N = (Total_Flash_Code_size)/256; //Each row has 256 bytes, "Total_Flash_Code_size" is
                                       //in bytes
//Calculating total number of Flash arrays 'K'
if (N % 256 == 0)
{
   byte K= (byte) N/256; //If rows are exact multiple of 256,quotient of (N/256) gives 'K'
}
else
{
   byte K= (byte)((N/256) + 1); //If rows are not exact multiple of 256,increment quotient of
                                // (N/256) by one for 'K'
}

int16 RowsPerArray; //Variable that hold the number of data rows in current Flash array
int16 byte_index = 0 //Variable to keep track of number of bytes read in a Flash row
byte Data_Array[L] //Array of size 'L' bytes to store one row of data read from device
int32 address

//Read Flash data bytes for all Arrays
for (byte ArrayCount = 0; ArrayCount < K; ArrayCount++)
{
   // Find number of rows in current array
   if (ArrayCount == (K-1))
   {
      RowsPerArray = N – (ArrayCount*256); //Last array may have less than 256 rows
   }
   else
   {
      RowsPerArray = 256; //Except last flash array, rest of them have 256 rows
   }

   int16 RowCount = 0;

   // Iterate through all rows of flash
   while (RowCount < RowsPerArray)
   {
      int32 address = RowCount * 256   //Starting address of Flash row

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [0x0000 00B6]//First initiation key

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [0x0000 00D7]//0xD7= (0xD3 + READ_MULTI_BYTE opcode)

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [0x0000 0004] // READ_MULTI_BYTE opcode

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [ArrayCount]// Array ID

      APACC ADDR Write [0x4000 4720]
```

```
APACC DATA Write [(address >> 16) & 0xFF]//MSB byte2 of 3-byte address

APACC ADDR Write [0x4000 4720]
APACC DATA Write [(address >> 8) & 0xFF]//Byte1 of 3-byte address

APACC ADDR Write [0x4000 4720]
APACC DATA Write [(address >> 0) & 0xFF]//LSB Byte0 of 3-byte address

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 00FF]// Number of bytes to be read minus one

//Wait until Data is ready
ADDR Write [0x4000 4722]// SPC status register address
dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

time_elapsed = 0
int32 StatusReg //To store SPC_SR status register value

do
{
   StatusReg = APACC DATA Read
   StatusReg = (StatusReg >> 16) & 0xFF  //Extract status code which is in 3rd byte
}    while ((StatusReg != [0x0000 0001]) AND time_elapsed < 1 sec)

if (time_elapsed > 1 sec) then FAIL_EXIT

APACC ADDR Write [0x4000 4720]
dummyByte = APACC DATA Read // Dummy SWD read

// Read 256 bytes of row data in to Data_Array
int16 ByteRead = 0, byte_index = 0
while (ByteRead <= 0x0000 00FF)
{
   Data_Array[byte_index] = APACC DATA Read // Save Flash data
   ByteRead = ByteRead + 1
   byte_index = byte_index + 1
}

// Configuration data is addressed as below. MSB bit is '1' to
//specify that addressed memory is ECC (config) memory
address = (RowCount * 32) | 0x00800000;

// Call READ_MULTI_BYTE to read configuration data in ECC memory space

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 00B6]  //First initiation key

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 00D7] //0xD7= (0xD3 + READ_MULTI_BYTE opcode)

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 0004] // READ_MULTI_BYTE opcode

APACC ADDR Write [0x4000 4720]
APACC DATA Write [ArrayCount] // Array ID

APACC ADDR Write [0x4000 4720]
APACC DATA Write [address >> 16) & 0xFF] //MSB Byte 2 of 3-byte address;
```

```
    APACC ADDR Write [0x4000 4720]
    APACC DATA Write [address >> 8) & 0xFF] //Byte 1 of 3-byte address

    APACC ADDR Write [0x4000 4720]
    APACC DATA Write [address >> 0) & 0xFF] //LSB Byte 0 of 3-byte address

    APACC ADDR Write [0x4000 4720]
    APACC DATA Write [0x0000 001F]   //Each row has 32 ECC bytes to be read

    //Wait until Data is ready
    ADDR Write [0x4000 4722]// SPC status register address
    dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

    time_elapsed = 0
    int32 StatusReg //To store SPC_SR status register value

    do
    {
        StatusReg = APACC DATA Read
        StatusReg = (StatusReg >> 16) & 0xFF //Extract statuscode which is in 3rdbyte
    } while ((StatusReg != [0x0000 0001]) AND time_elapsed < 1 sec)

    if (time_elapsed > 1 sec) then FAIL_EXIT

    APACC ADDR Write [0x4000 4720]
    dummyByte = APACC DATA Read // Dummy SWD read

    ByteRead = 0
    while (ByteRead <= 0x000 0001F)
    {
         Data_Array[byte_index] = APACC DATA Read// Save configuration data
         ByteRead = ByteRead + 1
         byte_index = byte_index + 1
    }

    /* Now, the array Data_Array contains a row of Flash data.
    Compare it with data in hex file to check if the correct data has been programmed
    in to Flash row. If there is data mismatch, Abort the Programming operation and
    retry again. Repeat for all Flash rows in all Flash arrays. */

    RowCount = RowCount + 1; // Next Flash row

  } //Repeat for all rows of Flash array

}//Repeat for all Flash arrays
```

5.7 Step 7: Program Write Once Nonvolatile Latch (Optional)

Warning: Programming of Write Once Nonvolatile Latch (WO NVL) is an optional step that is required only if the Device Security feature is required. If this step is included in the programming flow, there are conditions imposed on the PSoC 5 Vddd operating voltage (Vddd ≤ 3.3 V) and programming temperature (T_J = 25 °C ±15 °C). See Figure 1-1 for more details on this condition. These conditions are not applicable if the WO NVL programming step is not included in the programming sequence. Note that programming of WO NVL with the correct 32-bit key makes the device One Time Programmable (OTP). Include this step after understanding the implications of this step and only if it is required for the end application.

```
/* The NV Latches have a lesser endurance, and hence should be written only when the data has
changed. First read the Write Once NVL bytes from target device, and dump in to an array
(Data_Array). Compare the bytes read from the silicon to the NVL bytes in hex file at address
0x90100000. Perform write operation only if there is atleast one byte mismatch */

byte ByteRead = 0  //Variable to track number of bytes that are read
byte Data_Array[4] //4-byte array to store the NVL data read from device

while (ByteRead < 0x0000 0004)
{
  APACC ADDR Write [0x4000 4720]
  APACC DATA Write [0x0000 00B6] // First initiation key

  APACC ADDR Write [0x4000 4720]
  APACC DATA Write [0x0000 00D6] //Second key:00D6(0xD3+0x03);0x03 is ReadByte opcode

  APACC ADDR Write [0x4000 4720]
  APACC DATA Write [0x0000 0003] //0x03 is Read Byte opcode

  APACC ADDR Write [0x4000 4720]
  APACC DATA Write [0x0000 00F8] //Write Once NVL array ID

  APACC ADDR Write [0x4000 4720]
  APACC DATA Write [ByteRead]    //Byte number of Write Once NVL to be read

  // Poll status register bit till data is ready
  ADDR Write [0x4000 4722]// SPC status register address
  dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

  time_elapsed = 0
  int32 StatusReg //To store SPC_SR status register value

  do
  {
     StatusReg = APACC DATA Read
     StatusReg = (StatusReg >> 16) & 0xFF  // Extract status code which is in 3rd byte
  } while ((StatusReg != [0x0000 0001]) AND time_elapsed < 1 sec);

  if (time_elapsed > 1 sec) then FAIL_EXIT //Check if command execution time < 1 sec

  APACC ADDR Write [0x4000 4720]
  dummyByte = APACC DATA Read   //Dummy SWD read, first byte read is garbage
  Data_Array[ByteRead] = APACC DATA Read //Store the data read from device in to
                                         //array
  ByteRead = ByteRead + 1

  //Check if SPC Idle bit is high. Must be in idle state once data byte is read.
  ADDR Write [0x4000 4722]// SPC status register address
```

```
  dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

  time_elapsed = 0
  int32 StatusReg //To store SPC_SR status register value
  do
  {
     StatusReg = APACC DATA Read   //Save status register value to a local variable
     StatusReg = (StatusReg >> 16) & 0xFF  // Extract status code which is in 3rd byte
  } while ((StatusReg != [0x0000 0002]) AND time_elapsed < 1 sec)

  if (time_elapsed > 1 sec) then FAIL_EXIT

}

//Compare the NVL bytes read from target device with those in hex file at address
//0x90100000

ByteRead = 0
byte WriteFlag=0  /* This flag determines whether the NV latch will be programmed or not.
          Flag is set when new data needs to be written; otherwise reset */

while (ByteRead < 0x00000004)
{
   // Replace XX in below line with data at address (0x90100000 + ByteRead) of .hex file
   if(Data_Array[ByteRead] !=  XX)
   {
       WriteFlag=1 //Set the flag if NV latch needs to be programmed
   }
   ByteRead = ByteRead + 1
}

//Check if the WriteFlag is set before programming Write Once NVL
if (WriteFlag == 1)
{
   byte AddrCount = 0
   while (AddrCount < 4)
   {
      APACC ADDR Write [0x4000 4720]// Write to command data register
      APACC DATA Write [0x0000 00B6]// First initiation key

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [0x0000 00D3] // Second initiation key: 0xD3 + 0x00

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [0x0000 0000]// LOAD_BYTE opcode

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [0x0000 00F8]// Array ID of “Write Once NVL”

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [AddrCount]// Byte index in “Write Once NVL”

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [0x0000 00XX]        // Replace XX with data located in
                                          // (0x90100000 + AddrCount) of .hex file

      // Poll status register
      ADDR Write [0x4000 4722]// SPC status register address
```

```
        dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status
        time_elapsed = 0
        int32 StatusReg //To store SPC_SR status register value
        do
        {
            StatusReg = APACC DATA Read
            StatusReg = (StatusReg >> 16) & 0xFF  // Extract status code which is in 3rd byte
        } while ((StatusReg != [0x0000 0002]) AND time_elapsed < 1 sec)

        if (time_elapsed > 1 sec) then FAIL_EXIT // Check if command execution time < 1
                                          //second
        AddrCount = AddrCount + 1 //Increment to load the next NVL byte
       }

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [0x0000 00B6] // SPC_KEY1

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [0x0000 00D9]// SPC_KEY2 + _WRITE_USER_NVL opcode

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [0x0000 0006]// SPC_WRITE_USER_NVL opcode

      APACC ADDR Write [0x4000 4720]
      APACC DATA Write [0x0000 00F8]//Array ID of "Write Once NVL"

      // Poll status register
      ADDR Write [0x4000 4722]// SPC status register address
      dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

      time_elapsed = 0
      int32 StatusReg //To store SPC_SR status register value
      do
      {
         StatusReg = APACC DATA Read
         StatusReg = (StatusReg >> 16) & 0xFF  // Extract status code which is in 3rd byte
      } while ((StatusReg != [0x0000 0002]) AND time_elapsed < 1 sec)

      if (time_elapsed > 1 sec) then FAIL_EXIT//Check if command execution time < 1
                                          //second
 }
```

5.8 Step 8: Program Flash Protection Data

Flash protection data is located in address 32'h9040 0000 in the hex file. This step requires three parameters: K- Number of flash arrays, N- Total number of flash rows, L- Number of bytes in row (L=288). K and N are derived from the total flash memory size of the device, and the L value is fixed to 288. See the respective device datasheet for flash memory size of each device.

```
//Calculating total number of rows 'N'
int32 N = (Total_Flash_Code_size)/256;  //Each row has 256 bytes, "Total_Flash_Code_size" is
                                        //in bytes
//Calculating total number of Flash arrays 'K'
if (N % 256 == 0)
{
   byte K= (byte) N/256; //If rows are exact multiple of 256,quotient of (N/256) gives 'K'
}
else
```

```
{
   byte K= (byte)((N/256) + 1); //If rows are not exact multiple of 256,increment quotient of
                                // (N/256) by one for 'K'
}

int16 RowsPerArray; //Variable that hold the number of data rows in current Flash array
byte protectionPerArray; //Variable that hold the number of security bytes in current
                         //Flash array
int16 Offset =0; //Offset address of current security byte from address 0x9040 0000 of
                 //hex file
//Program protection bytes for all Arrays
for (int ArrayCount = 0; ArrayCount < K; ArrayCount++)
{
   // Find number of rows in current array
   if (ArrayCount == (K-1))
   {
      RowsPerArray = N – (ArrayCount*256); //Last array may have less than 256 rows
   }
   else
   {
      RowsPerArray = 256; //Except last flash array, rest of them have 256 rows
   }

   protectionPerArray = (RowsPerArray/4) //Each Flash protection byte stores
                                         //protection data of 4 Flash rows
   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [0x0000 00B6] // First initiation key
   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [0x0000 00D5] // Second initiation key: 0xD3 + 0x02

   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [0x0000 0002] // LOAD_ROW opcode

   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [ArrayCount]//Flash Array ID

   int16 ByteCount = 0
   while (ByteCount < L)
   {
     APACC ADDR Write [0x4000 4720]

     if (ByteCount < protectionPerArray)
     {
        APACC DATA Write [XX]//Data at address (32'h90400000 + Offset) of
                             //hex file
        Offset = Offset+1; //Increment the offset address in hex file
     }
     else
     {
        APACC DATA Write [0x0000 0000]//Fill bytes greater than protection size with
                                      //zero
     }

     ByteCount = ByteCount + 1
   }

   // After loading the protection data, program it in to the Flash hidden rows
   //using PROGRAM_PROTECT_ROW command
```

```
   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [0x0000 00B6] // First initiation key

   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [0x0000 00DE] // Second initiation key: 0xD3 + 0x0B

   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [0x0000 000B]// PROGRAM_PROTECT_ROW opcode

   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [ArrayCount] //Flash array ID

   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [0x0000 0000] //Row select value is always zero for protection
                                  //data

   // Poll status register
   ADDR Write [0x4000 4722]// SPC status register address
   dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

   time_elapsed = 0
   int32 StatusReg //To store SPC_SR status register value
   do
   {
     StatusReg = APACC DATA Read
     StatusReg = (StatusReg >> 16) & 0xFF  // Extract status code which is in 3rd byte
   } while ((StatusReg != [0x0000 0002]) AND time_elapsed < 1 sec)

   if (time_elapsed > 1 sec) then FAIL_EXIT

 } //Repeat for all Flash arrays
```

5.9 Step 9: Verify Flash Protection Data (Optional)

Flash protection data is located in address 32'h9040 0000 in the hex file. This step requires two parameters: K- Number of flash arrays, N- Total number of flash rows. K and N are derived from the total flash memory size of the device. See the respective device datasheet for flash memory size of each device.

```
//Calculating total number of rows 'N'
int32 N = (Total_Flash_Code_size)/256;  //Each row has 256 bytes, "Total_Flash_Code_size" is
                                        //in bytes
//Calculating total number of Flash arrays 'K'
if (N % 256 == 0)
{
   byte K= (byte) N/256; //If rows are exact multiple of 256,quotient of (N/256) gives 'K'
}
else
{
   byte K= (byte)((N/256) + 1); //If rows are not exact multiple of 256,increment quotient of
                                // (N/256) by one for 'K'
}

int16 RowsPerArray; //Variable that hold the number of data rows in current Flash array
byte protectionPerArray; //Variable that hold the number of security bytes in current
                         //Flash array

int16 byte_index = 0 //Variable to keep track of number of bytes read
```

```
/* Array to store the protection bytes read from PSoC5 Flash array */
byte Data_Array[256];

for (int ArrayCount = 0; ArrayCount < K; ArrayCount++)
{
   // Find number of rows in current array
   if (ArrayCount == (K-1))
   {
      RowsPerArray = N – (ArrayCount*256); //Last array may have less than 256 rows
   }
   else
   {
      RowsPerArray = 256; //Except last flash array, rest of them have 256 rows
   }

   protectionPerArray = (RowsPerArray/4) //Each Flash protection byte stores
                                         //protection data of 4 Flash rows

   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [0x0000 00B6]//First initiation key

   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [0x0000 00DD]//0xDD= (0xD3 + READ_HIDDEN_ROW opcode)

   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [0x0000 000A]// READ_HIDDEN_ROW opcode

   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [ArrayCount]// Flash Array ID

   APACC ADDR Write [0x4000 4720]
   APACC DATA Write [0x0000 0000]// RowID of Protection bytes row

   //Wait until Data is ready
   ADDR Write [0x4000 4722]// SPC status register address
   dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

   time_elapsed = 0
   int32 StatusReg //To store SPC_SR status register value
   do

   {
      StatusReg = APACC DATA Read
      StatusReg = (StatusReg >> 16) & 0xFF  //Extract status code which is in 3rd byte
   } while ((StatusReg != [0x0000 0001]) AND time_elapsed < 1 sec)
   if (time_elapsed > 1 sec) then FAIL_EXIT

   APACC ADDR Write [0x4000 4720]
   dummyByte = APACC DATA Read // Dummy SWD read

   /* Read 256 bytes of row data in to Data_Array. Even though the maximum number of
   protection bytes is only 64 for a Flash array, it is still required to read all the
   256 bytes in Flash protection row to ensure that the SPC returns back to the idle state.
   */
   byte_index = 0
   while (byte_index < 256)
   {
       Data_Array[byte_index] = APACC DATA Read// Save data in to the array
```

```
        byte_index = byte_index + 1
    }

    /* Now, the array Data_Array contains a row of Flash protection data (256 bytes) read from
    the device. Compare the first "protectionPerArray" bytes in the array with the protection
    data in the hex file. In the hex file, the Flash protection bytes are present starting
    from the address 32'h90400000 of the hex file. */

    byte_index = 0
    while (byte_index < protectionPerArray)
    {
      /* hexData[i] is from address (32'h90400000 + (64* ArrayCount) + byte_index) of hex
      file*/
      if (Data_Array[byte_index] != hexData[i])
      {
         FAIL_EXIT /* Byte mismatch. Verify operation for Protection bytes failed. Abort
                 Operation, Exit */
      }

      byte_index = byte_index + 1
     }

  /* Verify operation for Protection bytes passed. Go to next step */

  }//Repeat for all Flash arrays
```

5.10 Step 10: Verify Checksum

The data for this section is located in address 0x90300000 of the hex file. Only the lower two bytes of checksum are stored in the hex file. The MSB byte is stored at address 0x90300000, and the LSB byte is stored at address 0x90300001. This step requires three parameters: K- Number of flash arrays, N- Total number of flash rows, L- Number of bytes in row (L=288). K and N are derived from the total flash memory size of the device, and the L value is fixed to 288. See the respective device datasheet for flash memory size of each device.

```
//Calculating total number of rows 'N'
int32 N = (Total_Flash_Code_size)/256;  //Each row has 256 bytes, "Total_Flash_Code_size" is
                                         //in bytes
//Calculating total number of Flash arrays 'K'
if (N % 256 == 0)
{
   byte K= (byte) N/256; //If rows are exact multiple of 256,quotient of (N/256) gives 'K'
}
else
{
   byte K= (byte)((N/256) + 1); //If rows are not exact multiple of 256,increment quotient of
                                // (N/256) by one for 'K'
}

int16 RowsPerArray; //Variable that hold the number of data rows in current Flash array
int32 chipCheckSum = 0; //32-bit variable used to store the running checksum
//Calculate Checksum for all Arrays
for (byte ArrayCount = 0; ArrayCount < K; ArrayCount++)
{
   // Find number of rows in current array
  if (ArrayCount == (K-1))
   {
      RowsPerArray = N – (ArrayCount*256); //Last array may have less than 256 rows
```

```
}
else
{
   RowsPerArray = 256; //Except last flash array, rest of them have 256 rows
}

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 00B6] //First initiation key

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 00DF] //0xDF = 0xD3 + 0x0C

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 000C] // GET_CHECKSUM opcode

APACC ADDR Write [0x4000 4720]
APACC DATA Write [ArrayCount] //Flash array ID is the current Flash array

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 0000] //Starting row number (lower byte)

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 0000] //Starting row number (higher byte)

APACC ADDR Write [0x4000 4720]
APACC DATA Write [0x0000 0000] //Number of rows minus one (higher byte which is always 0)

APACC ADDR Write [0x4000 4720]
APACC DATA Write [(RowsPerArray - 1)&0xFF] //Number of rows minus one (lower byte)

// Poll status register
ADDR Write [0x4000 4722]// SPC status register address
dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

time_elapsed = 0
int32 StatusReg //To store SPC_SR status register value
do
{
  StatusReg = APACC DATA Read
  StatusReg = (StatusReg >> 16) & 0xFF  // Extract status code which is in 3rd byte
}    while ((StatusReg != [0x0000 0001]) AND time_elapsed < 1 sec)

if (time_elapsed > 1 sec) then FAIL_EXIT
APACC ADDR Write [0x4000 4720]
dummyByte = APACC DATA Read// Dummy SWD read
b3 = APACC DATA Read // Checksum byte 4(MSB byte)
b2 = APACC DATA Read // Checksum byte 3
b1 = APACC DATA Read // Checksum byte 2
b0 = APACC DATA Read // Checksum byte 1(LSB byte)

// Add current array 4-byte checksum to running checksum
chipCheckSum = chipCheckSum + (b3 << 24) + (b2 << 16) + (b1 << 8) + (b0 << 0);

// Poll status register till SPC is IDLE
ADDR Write [0x4000 4722]// SPC status register address
dummy = APACC DATA Read   //Dummy SWD Read, Next Read gives correct status

time_elapsed = 0
```

```
   int32 StatusReg //To store SPC_SR status register value
   do
   {
     StatusReg = APACC DATA Read
     StatusReg = (StatusReg >> 16) & 0xFF  // Extract status code which is in 3rd byte
    } while ((StatusReg != [0x0000 0002]) AND time_elapsed < 1 sec)

    if (time_elapsed > 1 sec) then FAIL_EXIT

  } //Repeat for all Flash arrays

  chipCheckSum = chipCheckSum & (0xFFFF); //Extract only the lower 2-byte checksum

  /* Compare with 2-byte checksum value in hex file (big endian format) at address
  0x90300000. Only the lower two bytes of checksum are stored in the hex file. The MSB byte
  is stored at address 0x90300000, and the LSB byte is stored at address 0x90300001. */
  if (chipCheckSum != file_checksum) then FAIL_EXIT
```

5.11 Step 11: Program EEPROM (Optional)

The data for this section is located in address 0x90200000 of the hex file.

```
/* Get the number of rows in EEPROM based on the EEPROM memory size information in the device
datasheet. Each row has 16 bytes */

byte NumofRows

/* EEPROM_SIZE_IN_BYTES is given in the device datasheet */
NumofRows = EEPROM_SIZE_IN_BYTES / 16

/* Program EEPROM row one by one */

byte Row_Count = 0/* Variable to keep track of current row number */

byte Byte_Count = 0/* Variable to keep track of byte number in a row */

while(RowCount < NumOfRows)
{
       APACC ADDR Write [0x4000 4720]
       APACC DATA Write [0x0000 00B6]/* First SPC Key */

       APACC ADDR Write [0x4000 4720]
       APACC DATA Write [0x0000 00D5]/* Second SPC Key = 0xD3 + 0x02 */

       APACC ADDR Write [0x4000 4720]
       APACC DATA Write [0x0000 0002]  /* Load Row Opcode */

       APACC ADDR Write [0x4000 4720]
       APACC DATA Write [0x0000 0040]  /* EEPROM Array ID */

       /* Load the 16 bytes of EEPROM row one by one by reading from the hex file */

       for(ByteCount = 0; ByteCount < 16; ByteCount++)
       {
              /* EEPROMByteData is located in the hexfile at address          (0x90200000 +
              (RowCount * 16) + ByteCount)  */
              APACC ADDR Write [0x4000 4720]
              APACC DATA Write [EEPROMByteData]

       }

       /* Read SPC status register to check the status of SPC command. If “Command Success”
       statusis not received within 1 second, then exit the programming operation */

       APACC ADDR Write [0x4000 4722]/* SPC status register address */
```

```
        int32 dummy = APACC DATA Read/* Dummy SWD Read */
        int32 StatusReg/* To store SPC_SR status register value */
        time_elapsed = 0
        do
        {
               StatusReg = APACC DATA Read /* Save status register value */
               StatusReg = (StatusReg >> 16) & 0xFF /* status code is in 3rd byte */
        } while ((StatusReg != [0x0000 0002]) AND time_elapsed < 1 sec)
        if (time_elapsed > 1 sec) then FAIL_EXIT
        APACC ADDR Write [0x4000 4720]
        APACC DATA Write [0x0000 00B6]/* First SPC Key */
        APACC ADDR Write [0x4000 4720]
        APACC DATA Write [0x0000 00D8]/* Second SPC Key = 0xD3 + 0x05 */
        APACC ADDR Write [0x4000 4720]
        APACC DATA Write [0x0000 0005]/* Write Row Opcode */
        APACC ADDR Write [0x4000 4720]
        APACC DATA Write [0x0000 0040]/* EEPROM Array ID */
        APACC ADDR Write [0x4000 4720]
        /* MSB byte of the 2-byte row number. Always zero for EEPROM since maximum number of
        rows can only be 128 */
        APACC DATA Write [0x0000 0000]
        APACC ADDR Write [0x4000 4720]
        APACC DATA Write [RowCount]/* LSB byte of the 2-byte row number */
        APACC ADDR Write [0x4000 4720]
        APACC DATA Write [0x0000 0001]   /* Temperature Sign byte */
        APACC ADDR Write [0x4000 4720]
        APACC DATA Write [0x0000 0019]/* Temperature Magnitude byte */
        /* Read SPC status register to check the status of SPC command. If “Command Success”
        status is not received within 1 second, then exit the programming operation */
        APACC ADDR Write [0x4000 4722]/* SPC status register address */
        int32 dummy = APACC DATA Read/* Dummy SWD Read */
        int32 StatusReg/* To store SPC_SR status register value */
        time_elapsed = 0
        do
        {
               StatusReg = APACC DATA Read /* Save status register value */
               StatusReg = (StatusReg >> 16) & 0xFF /* status code is in 3rd byte */
        } while ((StatusReg != [0x0000 0002]) AND time_elapsed < 1 sec)
        if (time_elapsed > 1 sec) then FAIL_EXIT
        RowCount = RowCount + 1 /* Next EEPROM row to be programmed */
}
```

5.12 Step 12: Verify EEPROM (Optional)

```
/* Get the number of rows in EEPROM based on the EEPROM memory size information in the device
datasheet. Each row has 16 bytes */

byte NumofRows

/* EEPROM_SIZE_IN_BYTES is given in the device datasheet */
NumofRows = EEPROM_SIZE_IN_BYTES / 16

int read_address/* Location of EEPROM address to be read */

int read_data/* 4-byte data read from EEPROM */

byte ByteRead = 0  /* Variable to track number of bytes that have been read */

byte Data_Array[16] /* Array to store the EEPROM row data read from the device */

/* Verify the data programmed in to EEPROM, one row at a time */
while(RowCount < NumOfRows)
{
       ByteRead = 0

       /* Read the EEPROM row data from the device in 4-byte chunks and store in the array */
       while(ByteRead < 16)
       {
              /* Address of EEPROM in PSoC 5. 0x40008000 is EEPROM base address */
              read_address = 0x40008000 + (RowCount * 16) + ByteRead

              APACC ADDR Write [read_address]

              dummyByte = APACC DATA Read/* Dummy SWD read */

              read_data = APACC DATA Read/* Actual 4-byte EEPROM data */

              /* Store the 4-byte data in the array */
              Data_Array[ByteRead] = (byte) (read_data)
              Data_Array[ByteRead + 1] = (byte) (read_data >> 8)
              Data_Array[ByteRead + 2] = (byte) (read_data >> 16)
              Data_Array[ByteRead + 3] = (byte) (read_data >> 24)

              ByteRead = ByteRead + 4 /* Read the next 4-bytes */

       }
       /* Verify the row data read from the device against the hex file data */
       for(ByteRead = 0; ByteRead < 16; ByteRead++)
       {
       /* Replace XX below with byte data from the hex file at address
              (0x90200000 + (RowCount * 16) + ByteRead). Verify operation is a failure if
              there is a byte mismatch  */
       if(Data_Array[ByteRead] != XX) then FAIL_EXIT

       }
       RowCount = RowCount + 1 /* Next row */

}
```

A. Appendix

A.1 Intel Hex File Format

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

Each line (record) of the Intel hex file consists of six parts, as shown in Figure A-1.

Figure A-1. Hex File Record Structure

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

- **Start code:** one character - an ASCII colon ':'
- **Byte count:** two hex digits (1 byte) - specifies the number of bytes in the data field
- **Address:** four hex digits (2 bytes) - a 16-bit address at the beginning of the memory position for the data
- **Record type:** two hex digits (00 to 05) - defines the type of data field. The record types used in the hex file generated by PSoC Creator are:
  - 00 - Data record, which contains data and 16-bit address
  - 01 - End of file record, which is a file termination record and has no data. This must be the last line of the file; only one is allowed for every file
  - 04 - Extended linear address record, which allows full 32-bit addressing. The address field is 0000, the byte count is 02. The two data bytes represent the upper 16 bits of the 32 bit address, when combined with the lower 16-bit address of the 00 type record
- **Data:** a sequence of ‘n’ bytes of the data, represented by 2n hex digits
- **Checksum:** two hex digits (1 byte), which is the least significant byte of the two's complement of the sum of the values of all fields except fields 1 and 6 (Start code ‘:’ byte and two hex digits of the Checksum)

Examples for the different record types used in the hex file generated by PSoC Creator are as follows.

Consider that these three records are placed in consecutive lines of the hex file.

:0200000490006A

:0420000000000005F7

:00000001FF

The first record (*:0200000490006A*) is an extended linear address record as indicated by the value in the Record Type field (04). The address field is 0000, the byte count is 02. This means that there are two data bytes in this record. These data bytes (9000) specify the upper 16-bits address of the 32-bit address of data bytes. In this case, all the data records that follow this record are assumed to have their upper 16-bit address as 0x9000 (in other words, the base address is 0x90000000). 6A is the checksum byte for this record.

The next record (*:0420000000000005F7*) is a data record, as indicated by the value in the Record Type field (00). The byte count is 04 indicating that there are four data bytes in this record (00000005). The 32-bit starting address for these data bytes is at address 90002000. The upper 16-bit address (9000) is derived from the extended linear address record in the first line;

the lower 16-bit address is specified in the address field of this record as 2000. F7 is the checksum byte for this record.

The last record (*:00000001FF*) is the end of file record, as indicated by the value in the Record Type field (01). This is the last record of the hex file.

Note The data records of the following multi-bytes region in the hex file are in big-endian format (MSB in lower address): Checksum data at address 0x9030 0000 of hex file; metadata at address 0x9050 0000. The data records of the rest of the multi-byte regions in hex file are all in little-endian format (LSB in lower address).

A.1.1 Organization of Hex File Data

The hex file generated by PSoC Creator contains different types of data, which includes the flash code data, flash configuration data, flash protection data, EEPROM data, customer nonvolatile latch, and write once latch data. Apart from this, the hex file also contains metadata. Metadata is information that is not used for programming the device memory. It is used to maintain data integrity of the hex file and store silicon revision and device ID information. All information including metadata are stored at specific addresses. This allows the programmer to identify which data is meant for what purpose. The address map is explained here and summarized in Figure A-2.

0x0000 0000 – Flash Code Region Data: The flash code data starts at address 0x0000 0000 of the hex file. Each record in the hex file contains 64 bytes of actual data; arrange these into rows of 256 bytes. This is because each flash row of device is of length 256 code bytes (not including the 32 configuration bytes, which are stored in another region). The last address of this section depends on the flash memory size of the device for which the hex file is intended. As an example, for a device with a flash memory capacity of 256 KB, the end address is 0x0003FFFF. See the respective device datasheet or the Device Selector menu in PSoC Creator to know the flash memory size of different part numbers.

0x8000 0000 – Flash Configuration Data: There are 32 bytes of configuration data for each row of flash. This data needs to be appended with the main flash data during the flash programming step. For every 256 code bytes in Program Flash, 32 bytes from this section are appended. The last address of this section depends on the device flash memory capacity. A device with 256 KB flash memory has 32 KB of configuration memory. So in this case, the last address is 0x80007FFF.

0x9000 0000 – Device Configuration NV Latch Data: The 4-byte data in this region is not used anywhere in PSoC 5 programming flow and can be ignored.

0x9010 0000 – Secured Device Mode Configuration Data: This section contains four bytes of the write-once nonvolatile latch data that is used to enable device security. **Warning:** Programming the write-once NV latch with the correct 32-bit key locks the device; perform this step only if all previous steps are passed without errors. PSoC Creator generates all four bytes as zero if the device security feature has not been enabled to ensure that there is no accidental programming of the latch with correct key. Failure analysis support may be lost on units after this step is performed with correct key. Refer to Appendix B of the PSoC 5 TRM for details on this device security feature.

0x9020 0000 – EEPROM: PSoC 5 devices have on-chip EEPROM memory and the data to be programmed into the EEPROM is stored in this region. EEPROM is programmed row wise where each row contains 16 bytes. Because each record in the EEPROM region of the hex file contains 64 bytes of data, each record has the data corresponding to four contiguous EEPROM rows.

0x9030 0000 – Checksum Data: This 2-byte checksum data is the checksum computed from the entire flash memory of the device (main code and configuration data). This 2-byte checksum is compared with the checksum value read from the device to check if correct data has been programmed. Though the CHECKSUM command sent to the device returns a 4-byte value, only the lower two bytes of the returned value are compared with the checksum data in the hex file. The 2-byte checksum in the data record is in Big-endian format (MSB byte is first byte).

0x9040 0000 – Flash Protection Data: This section contains data to be programmed to configure the protection settings of flash memory. Arrange data in this section in a single row to match the internal flash memory architecture. Because there are two bits of protection data for each main flash row, a 256 KB flash (with 1024 rows, 256 rows in each of four 64K Flash arrays) has 256 bytes of protection data.

0x9050 0000 – Metadata: The data in this section of the hex file is not programmed into the target device. It is used to check the data integrity of hex file, silicon revision for which the hex file is intended, and so on. The different data in this section is tabulated as follows.

Table A-1. Metadata Organization in Hex FIle

Starting Address	Data Type	Number of Bytes
0x9050 0000	Hex file version	2 (big-endian)
0x9050 0002	Device ID	4(big-endian)
0x9050 0006	Silicon revision	1
0x9050 0007	Debug Enable	1
0x9050 0008	Internal use	4

Hex File Version: This 2-byte data (big-endian format) is used to differentiate between different hex file versions. For example, if new metadata information or EEPROM data is added to the hex file generated by PSoC Creator, there is a need to distinguish between the different versions of hex files. By reading these two bytes you can ascertain which version of the hex file is going to be programmed. At present, PSoC Creator generates only one type of hex file and this field always has a constant value of 0x0001. The only value that this field accepts is 0x0001 because there is only one version of the hex file.

Device ID: This field has the 4-byte device ID (big-endian format), which is unique to each part number. Compare the device ID read from the device with the device ID present in this field to make sure the correct device for which the hex file is intended is programmed. See the device datasheet for information on the device IDs of different part numbers.

Silicon Revision: This 1-byte value is for different revisions of the silicon. This data is not used anywhere in the PSoC 5 programming sequence. For PSoC 5, the revision IDs are as follows:

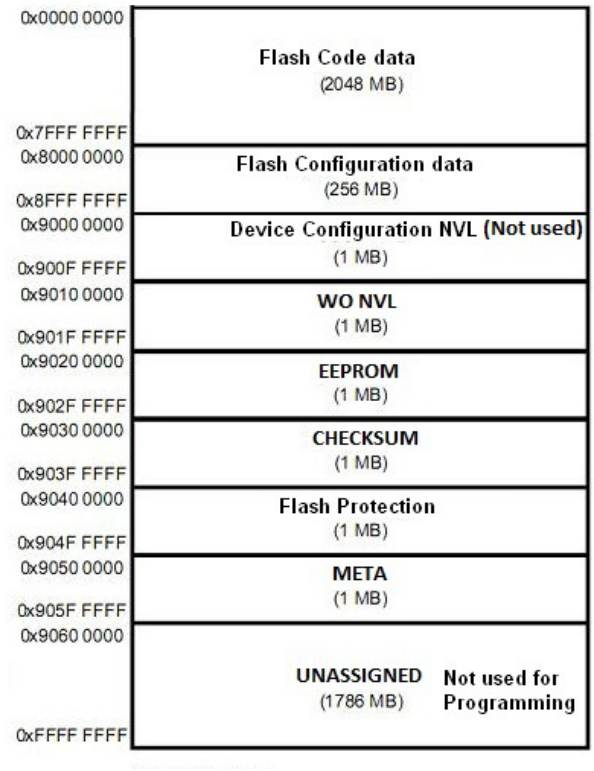
1 – ES1

Debug Enable: This 1-byte data stores a Boolean value indicating if debugging is enabled for the program code. This is also not used in programming. The possible values for this byte are:

0 – Debugging Disabled, 1 – Debugging Enabled

Internal Use: The 4-byte data is used internally by PSoC Programmer software. It is not related to actual device programming and need not be used by programmers of third party vendors.

Figure A-2. PSoC 5 Hex File Address Map



A.2 Nonvolatile Memory Organization in PSoC 5

PSoC 3 and PSoC 5 devices have three types of nonvolatile memory: Flash, Electronically Erasable Programmable Read Only Memory (EEPROM), and Nonvolatile Latch (NVL). This section gives a quick overview of the interface used to program the nonvolatile memory. It also discusses nonvolatile memory organization. EEPROM memory is not explained in this section because programming of EEPROM using external programmer is not defined in the device programming specification currently. Refer to the "Memory" section of the PSoC 5 TRM for detailed information on these topics.

A.2.1 Nonvolatile Memory Programming

All nonvolatile memory programming operations are done through a simple command/status register interface summarized in Table A-2.

Table A-2. SPC Command and Status Registers

Register	Size (Bits)	Description
SPC_CPU_DATA	8	Data to/from the CPU
SPC_DMA_DATA	8	Data to/from the DMAC
SPC_SR	8	Status – ready, data available, status code

Commands and data are sent as a series of bytes to either SPC_CPU_DATA or SPC_DMA_DATA, depending on the source of the command. The programming procedure in this document always uses the SPC_CPU_DATA register. Response data is read via the same register to which the command is sent. The status register, SPC_SR, indicates whether a new command can be accepted, when data is available for the most recent command, and a success/failure response for the most recent command.

A.2.2 Commands

Before sending a command to the SPC_CPU_DATA or SPC_DMA_DATA register, the SPC_Idle bit in SPC_SR[1] must be '1'. SPC_Idle will go to '0' when the first byte of a command (0xB6) is written to a data register, and go back to '1' when command execution is complete or an error is detected. Commands sent to either data register while SPC_Idle is '0' are ignored. All commands must adhere to the following format:

- Key byte #1 – always 0xB6
- Key byte #2 – 0xD3 plus the command code (ignore overflow)
- Command code byte
- Command parameter bytes
- Command data bytes

Refer to the "Nonvolatile Memory Programming" chapter in the PSoC 5 TRM for a list of command codes and the explanation, parameters, and return values for each command.

A.2.3 Command Status

The status register, SPC_SR, indicates whether a new command can be accepted, when data is available for the most recent command, and a success/failure response for the most recent command. The bit-field definitions of the SPC_SR register is given in Figure A-3.

Figure A-3. SPC_SR Status Register Bit Field Definitions

Bits	7	6	5	4	3	2	1	0
SW Access:Reset	R:000000						R:1	R:0
HW Access	R/W						R/W	R/W
Name	Status_Code						SPC_Idle	Data_Ready

Data_Ready bit: This bit (Bit [0] of SPC_SR) indicates whether the SPC has data that is ready to be read from the SPC CPU or DMA Data Register.

SPC_Idle bit: This bit (Bit [1] of SPC_SR) indicates whether the SPC is currently executing an instruction. The bit transitions low as soon as the first byte of the 2-byte command key (0xB6) is written into the SPC CPU or DMA Data Register. The bit transitions high as soon as an instruction completes or if the second byte of the command key is invalid.

Status_code (5-bit status code): The Status Code (Bits [7:2] of SPC_SR) represents the exit status of the last executed SPC instruction. The values of this field are given in Table A-3.

Table A-3. Status Codes for an SPC Command

SPC Status Code (Bits[7:2] in SPC_SR register)	Meaning
0x00	Operation Successful
0x01	Invalid Array ID for given command
0x02	Invalid 2-byte key
0x03	Addressed Nonvolatile memory array is Asleep
0x04	External Access Failure (SPC is not in external access mode)
0x05	Invalid 'N' Value for given command
0x06	Test Mode Failure (SPC is not in programming mode)
0x07	Smart Write Algorithm Checksum Failure
0x08	Smart Write Parameter Checksum Failure
0x09	Protection Check Failure: Flash protection settings are in a state which prevents the given command from executing
0x0A	Invalid Address parameter for the given command
0x0B	Invalid Command Code
0x0C	Invalid Row ID parameter for given command
0x0D	Invalid input value for Get Temp & Get ADC commands
0x0E	Tempsensor Vbe is currently driven to an external device
0x0F	Invalid SPC state
0x10 – 0x3F	Smart Write return codes (only when using Smart Write algorithm)
0x20	PEP Program Failure (only when using PEP algorithm): Data Verification Failure (row latch checksum!= programmed row checksum)

A.2.4 Nonvolatile Memory Organization

A.2.4.1 Flash Program Memory

PSoC 5 flash memory has the following features:

- Organized in rows, where each row contains 256 main flash code bytes plus 32 bytes for configuration data storage. The size of each flash row is 288 bytes. Flash memory can be programmed in resolution of rows.
- Organized as either one array of 128 or 256 rows, or as multiple arrays of 256 rows each.
- For each flash row, protection bits control whether the flash can be read or written by external debug devices and whether it can be reprogrammed by a boot loader. For each flash array, flash protection bits are stored in a hidden row in that array. In the hidden row, two protection bits per row are packed into a byte, so each byte in the hidden row has protection settings for four flash rows of that array.

A.2.4.2 Write Once Nonvolatile Latches (WO NVL)

The user can write the key in WOL to lock out external access only if no flash protection is set. In the programming flow, programming of WOL is done before the flash protection bytes.

Note that when the WO NVL is programmed with the correct 32-bit key (0x50536F43) and the device is reset after programming, the part cannot be programmed further, and becomes an OTP (One Time Programmable) device. The Write Once NV latch locks the part out of Debug and Test modes; it also permanently gates off the ability to erase or alter the contents of the latch. This step should hence be exercised with extreme caution considering these effects.

A.3 Programming Procedure Differences Between PSoC 5 and PSoC 3

This section details the differences between the PSoC 5 and PSoC 3 programming procedures. This helps in the implementation of a programmer that can program both PSoC 5 and PSoC 3 silicon. It also enables the migration of PSoC 3 programmer to support PSoC 5 as well.

The differences in programming procedure between PSoC 5 and PSoC 3 silicon are listed here. The first three are hardware differences and the remaining are differences in the programming algorithm.

- **JTAG interface programming not supported:** PSoC 5 supports programming only through the SWD interface. JTAG interface is not supported. PSoC 3, on the other hand, supports programming through both SWD and JTAG interfaces.
- **SWD clock erratum when programming using USB SWD pins**: For PSoC 5, when using USB SWD pins (P15[6], P15[7]) to program or debug the device, the other P1[1] SWDCK pin must be pulled to ground using an external 100 KΩ pull-down resistor. This is required for the internal PSoC 5 device to acquire logic and see the SWDCK signal on USB SWD pin. This hardware condition is not applicable for PSoC 3.
- **Hardware limitations on WO NVL programming**: Programming of Write Once Nonvolatile Latch (WO NVL) is an optional step used only if the Device Security feature is required. If this step is included in the programming flow, there are conditions imposed on the PSoC 5 Vddd operating voltage (Vddd ≤ 3.3 V) and programming temperature (TJ = 25 °C ±15 °C). These conditions are not applicable for PSoC 3 WO NVL programming.
- **No device configuration NVL programming:** Unlike PSoC 3, PSoC 5 does not have Device Configuration NVL memory.
- **No Error Correction (ECC) feature in PSoC 5 flash**: Unlike PSoC 3, ECC feature is not available in PSoC 5 flash memory. This space is always used for storing the flash configuration data. As a result, the size of the flash row to be programmed is always 288 bytes in PSoC 5. For PSoC 3, the flash row size can be 256 bytes or 288 bytes, depending on whether the ECC feature is enabled.
- **Multiple flash arrays in PSoC 5**: The maximum size of PSoC 3 flash memory is 64 KB and only one flash array is present. PSoC 5 can have a flash memory of up to 256 KB capacity, resulting in a maximum of four flash arrays. All the flash memory related steps must be repeated for all flash arrays. This is taken care in the PSoC 5 programming algorithm provided in this specification.
- **ProgramRow() and GetTemp() SPC APIs in PSoC 5**: SPC_PROGRAM_ROW API is not functional in PSoC 5 silicon. Instead of using SPC_PROGRAM_ROW call, flash programming in PSoC 5 uses SPC_WRITE_ROW.

Internal device temperature is one of parameters required for programming Flash memory in PSoC 3 and PSoC 5. The SPC_GET_TEMP API is used to get the internal device temperature. The internal temperature sensor is not functional in the PSoC 5 silicon. It is not correct to use result of SPC_GET_TEMP API for temperature values in SPC_WRITE_ROW API. In PSoC 5 flash programming step, a hard-coded value of +25 °C is passed in SPC_WRITE_ROW API.

In the PSoC 3 silicon, both the APIs mentioned above are functional. SPC_PROGRAM_ROW API is used with the temperature value returned by SPC_GET_TEMP API for flash programming.

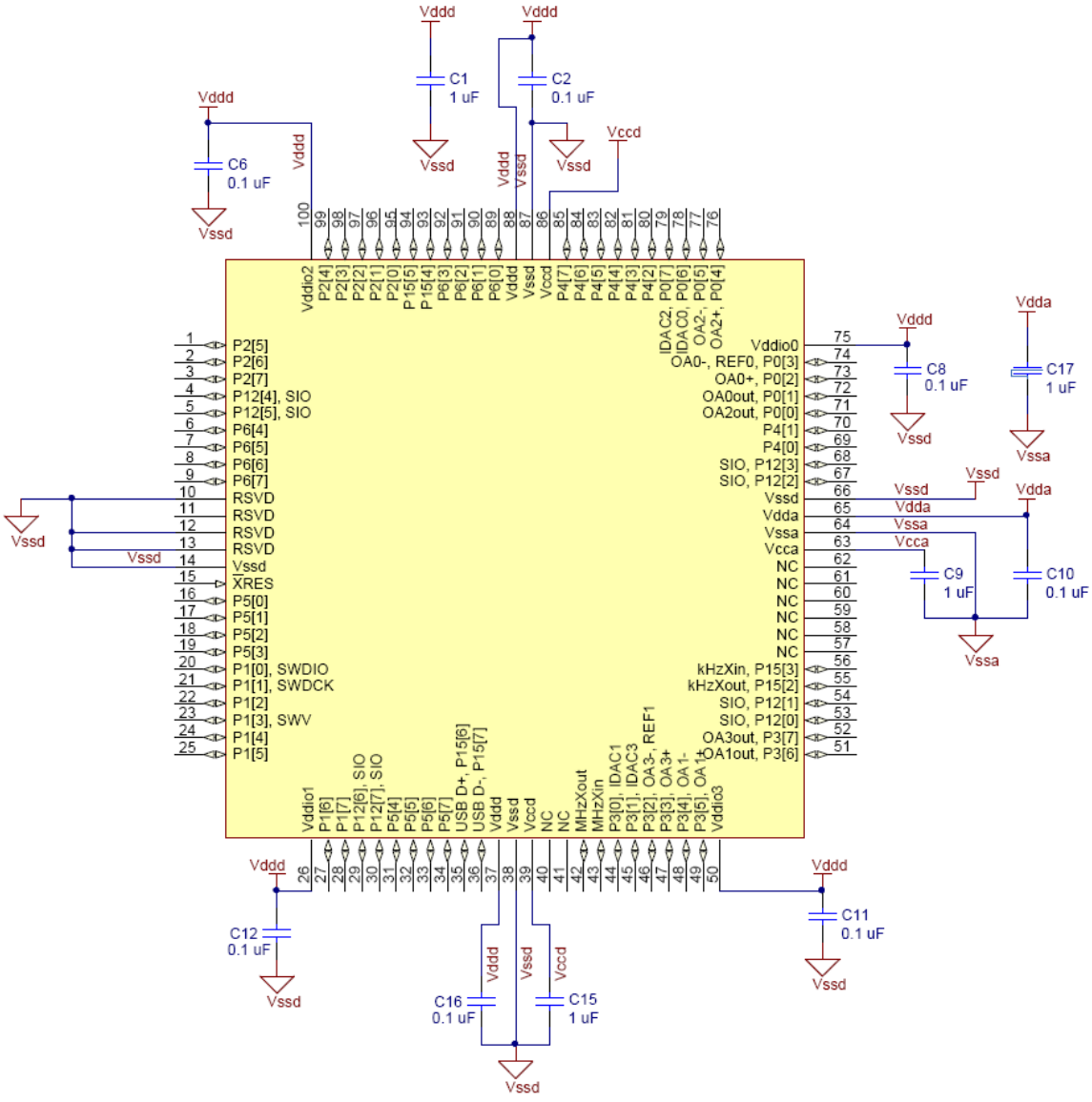
A.4 Example Schematic

The following figure shows an example reference schematic for the 100-pin TQFP part with the power connections. This can also be used for the other PSoC 5 packages; however, the pin out will vary for each package. See the PSoC 5 device datasheet for information on specific package pinout and for specifications on power supply pins. Note that Figure A-4 does not show the programming connections between the host programmer and PSoC 5. This is illustrated in Figure 1-1.

Figure A-4 shows that:

- The two pins labeled Vddd must be connected together.
- The two pins labeled Vccd must be connected together, with capacitance added. The trace between the two Vccd pins should be as short as possible.
- The two pins labeled Vssd must be connected together.

Figure A-4. 100-pin TQFP Part with Power Connections



Note The two Vccd pins must be connected together with as short a trace as possible. A trace under the device is recommended.