

AN2026b

Author: Max Kingsbury

Associated Project: No

Associated Part Family: CY8C24794, CY8C24894, CY8C24994/CY8C29x66

[GET FREE SAMPLES HERE](#)

Associated Application Notes: [AN2014](#), [AN2026](#), [AN2026a](#)

Application Note Abstract

This application note describes how to program a PSoC[®] device using the In System Serial Programming (ISSP) protocol.

Introduction

The emulator pod and individual chips can be programmed in-system through the ICE-Cube or MiniProg, using PSoC Designer or an external factory programmer. You may need in-circuit programming, either from a manufacturing or an in-system, field-update standpoint.

This application note provides the programming timing and vectors so that developers and programmer vendors can create their own in-system programming solutions for a PSoC device.

There are two participants in the programming procedure: the programmer and the target device. The programmer communicates serially with the target. The programmer supplies the clocking and sends commands to the target. The

target receives data from the programmer and supplies data upon a read request. The target drives the data line only upon request from the programmer. The programmer programs the target with the program image contained in the <PROJECT NAME> .hex file, which is generated by PSoC Designer. Refer to [Appendix B on page 11](#) for more information regarding the Intel .hex file format.

The critical pin connections between the programmer and the target are listed in [Table 1](#). This includes: 2 signal pins, a reset pin, a power pin and a ground pin. Other pins must be left floating. The pin naming conventions and drive strength requirements are also listed in [Table 1](#).

Table 1. Pin Names and Drive Strength Requirements

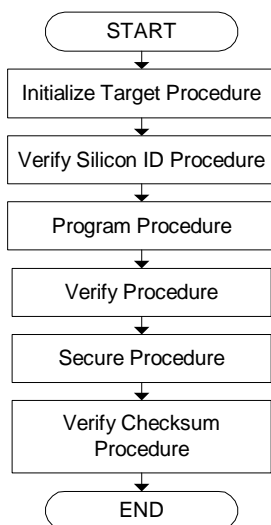
Pin Name	Function	Programmer HW Pin Requirements
P1[0]	SDATA - Serial Data In/Out	Drive TTL Levels, Read TTL, High Z
P1[1]	SCLK - Serial Clock	Drive TTL Level Clock Signal
X _{res}	Reset	Drive TTL Levels
V _{ss}	Power Supply Ground Connection	Low Resistance Ground Connection
V _{DD}	Power Supply Positive Voltage	3.3 or 5V, 30 mA Capability

For more information on the physical requirements of connections between the programmer and target, refer to application note [AN2014](#) "Design for In-System Serial Programming (ISSP)."

Programming Flow

Successful target programming depends on adherence to the programming flow shown in [Figure 1](#). Each procedure is explained in detail in the following sections. Failure to complete these steps can result in incorrectly programmed Flash.

Figure 1. Target Programming Flow



Vectors

Vectors are the binary representation of the commands necessary to perform the various operations involved in the programming flow. Each procedure in the programming flow has many individual vectors associated with it ([Appendix A on page 9](#)). Each vector is 22 bits long and any number of zeros can be sent between sequential vectors. The target ignores the zero padding and any subsequent '0' on the SDATA line. This continues until the target receives a '1', which is the first bit in the next vector in the vector-set.

Clocking

During the programming flow, there is one clocking mode: Data Transfer. Data Transfer mode is used while the programmer communicates with the target, either by sending or receiving data. During this time, the programmer can drive the SCLK signal at any frequency that enables reliable data transfer with a maximum transmit frequency of 8 MHz (F_{sclk}, [Table 3 on page 8](#)). The SCLK rate in data transfer mode can be controlled with less precision. Clocks are not required during the "Wait and Poll" steps described in the procedure flowcharts.

Command Format

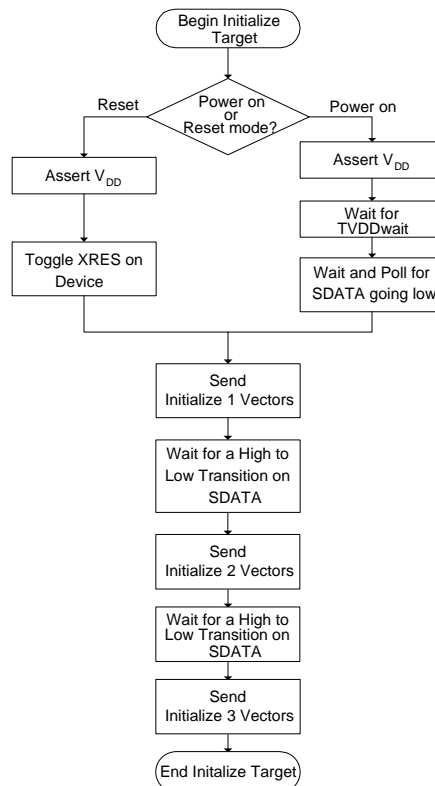
During programming, only the programmer drives the SCLK line. The programmer and target can drive the SDATA line, although the target drives SDATA only upon a read request from the programmer. The programmer always writes and reads SDATA on the rising edge of SCLK, while the target writes and reads on the falling edge. After the programmer requests a read from the target ([Appendix A on page 9](#), Read Byte, [Figure 6 on page 4](#), Last Portion of READ-ID From Target), it releases the SDATA to a High Z state and resumes driving the line only after the byte is sent by the target. The programmer supplies clocks even when it has released (High Z) the SDATA line. During the "Wait for a High to Low Transition on SDATA" phase of programming, the programmer must release (High Z) the SDATA line.

Initialize Target Procedure

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

Reset mode is the preferred method for initiating communication with the target. However, in the case of the CY8C24794, there is no X_{res} pin so power on mode is the only option. It must be noted that because power on mode involves cycling power to the target, in-circuit field programming may involve PCB layout considerations in the design phase.

Figure 2. Initialize Target Procedure



Reset Mode

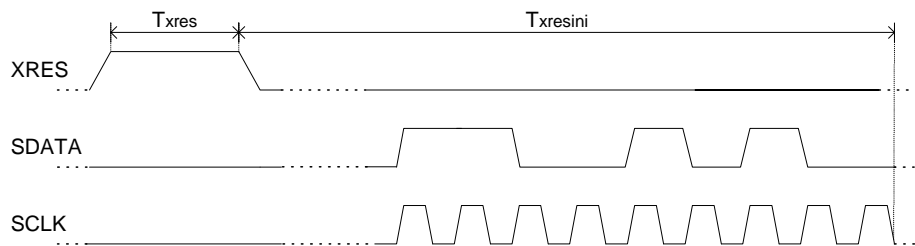
The timing to enter programming mode with Reset is shown in Figure 3. To initialize the part using the X_{res} line, first wait until V_{DD} is stable, and then assert the X_{res} line for the time specified by T_{xres} (Table 3 on page 8). After X_{res} is driven low, there is a window of time specified by $T_{xresini}$ (Table 3 on page 8) in which the first 8 bits of the Initialize 1 vector-set must be transmitted.

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

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

The programmer must sense the system supply and decide which Initialize 3 vectors to supply. If $V_{DD} \leq 3.6$ volts, use one set; if $V_{DD} > 3.6$ volts, use the other. See Appendix A on page 9.

Figure 3. Using Reset to Initialize



Power on Mode

To initiate communication with the target using power on, apply V_{DD} to the target as is shown in Figure 4. This causes the target to attempt to drive the SDATA line High. The programmer then waits and polls for a HIGH to LOW transition on the SDATA line, which is the signal from the target that V_{DD} has stabilized. Note that until V_{DD} stabilizes, the SDATA signal is noisy and a false edge could be detected. As a result, the programmer must wait for the time specified by $T_{VDDwait}$ (Table 3 on page 8) before beginning to wait and poll. In addition, the programmer must not drive the SCLK signal until the $T_{VDDwait}$ time period has passed.

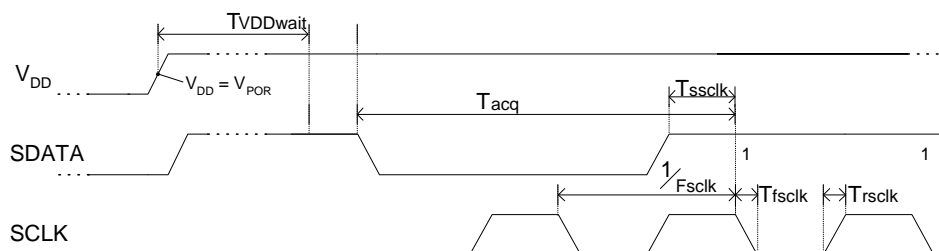
After the SDATA transition is detected, the programmer must transmit the Initialize 1 vectors in T_{acq} seconds (Table 3 on page 8).

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

The pin drive modes vary during the programming operation. When the PSoC drives the SDATA line to indicate it has started up completely, or to send data back to the host, SDATA is in a strong drive configuration. When it waits for or receives data, SDATA is in a pull down configuration. It is important to design the external pin drive mode circuitry such that a strong high to resistive low transition can be detected, and also so that the pin can be driven both high and low when it is in pull down mode.

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

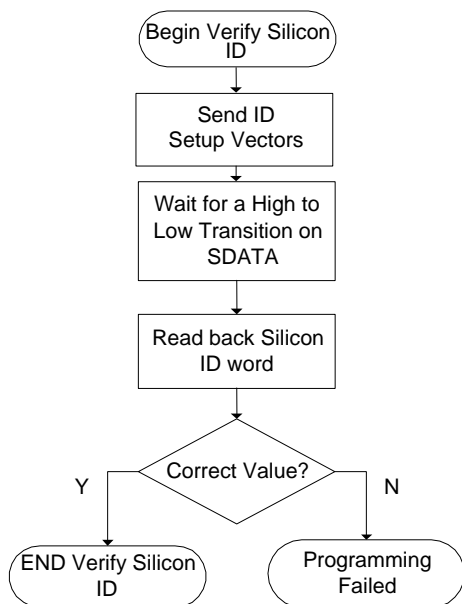
Figure 4. Using Power Cycling to Initialize



Verify Silicon ID Procedure

The Verify Silicon ID Procedure (shown in [Figure 5](#)) returns the package-specific silicon ID value from the target. This is used by the programmer to verify the package type of the target.

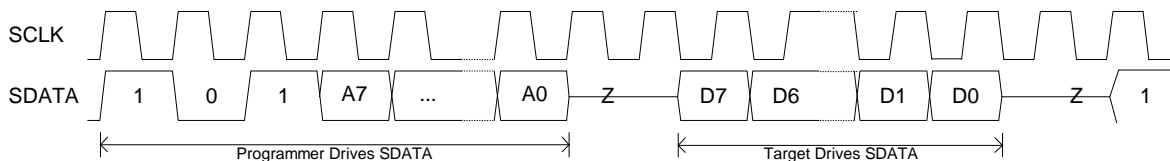
Figure 5. Verify Silicon ID Procedure



The first step in the Verify Silicon ID Procedure is for the programmer to send the ID-Setup vector-set. The programmer then drives the SDATA line into a High Z state. It waits and polls the SDATA line for a HIGH to LOW transition, which signifies that the target has executed the operation. The silicon ID value can then be read back by using the READ-ID vector-set. The sequence for a READ-ID command is shown in [Figure 6](#). Two bytes must be read to obtain a complete Silicon ID word.

The vectors in [Appendix A on page 9](#) under READ-ID show the device-specific values read from the target, that is, a LLLLLLLL LLLHHHLH denotes a 0x00ID hex read back from CY8C24794. The programmer must compare the value in the READ-ID ([Appendix A](#)) and the value returned by the target. If these values do not match, the programmer must terminate the programming flow.

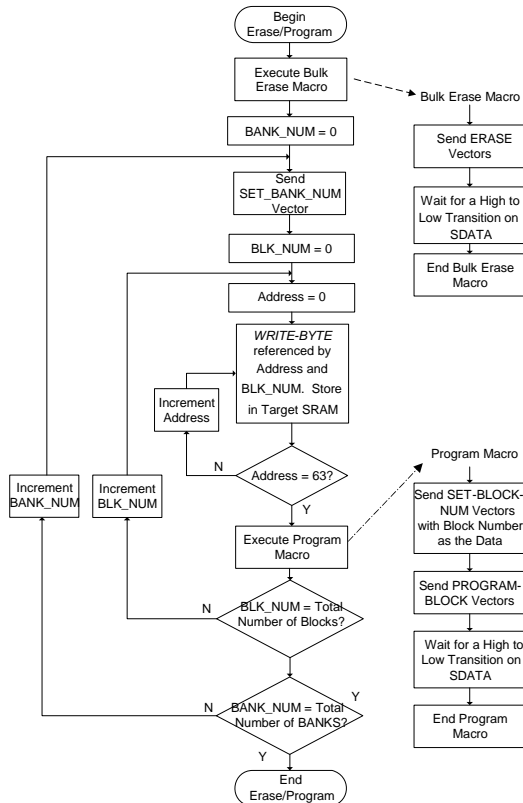
Figure 6. Last Portion of READ-ID From Target



Program Procedure

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

Figure 7. Program Procedure



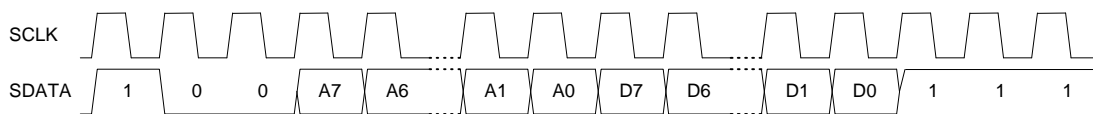
A Bulk Erase operation must be executed to prepare the Flash for programming.

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

To place the actual program image into the Flash, the program portion of the .hex (Appendix B on page 11) is read by the programmer in 64-byte blocks. This is written into the SRAM of the target one byte at a time using the WRITE-BYTE vector.

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

Figure 8. WRITE-BYTE to Target



Verify Procedure

The Verify Procedure (shown in Figure 10 on page 6), is responsible for verification of the programmed Flash.

Flash must be verified to ensure program integrity. This procedure uses a loop to read back the same number of blocks programmed into the Flash. To verify a block of Flash, the SET-BLOCK-NUM vector (Appendix A on page 9) is first sent with the 'ddddddd' in the vector replaced with the block number to be read from Flash.

The programmer sends the VERIFY-SETUP vector-set and then waits and polls. Each Read Block operation reads a 64-byte block from Flash and stores the data in the target's SRAM. The programmer must then use the READ-BYTE vector (Figure 9 on page 6) to individually read each byte in the block. After the programmer reads the block, the programming software must compare it with the block written to the Flash. Data mismatch denotes an incorrect Flash write and the programmer must terminate the programming flow as a failure.

Figure 9. READ-BYTE From Target

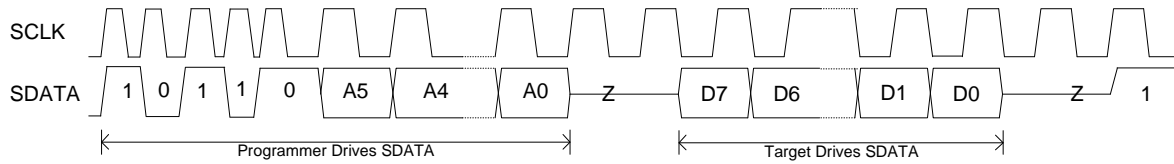
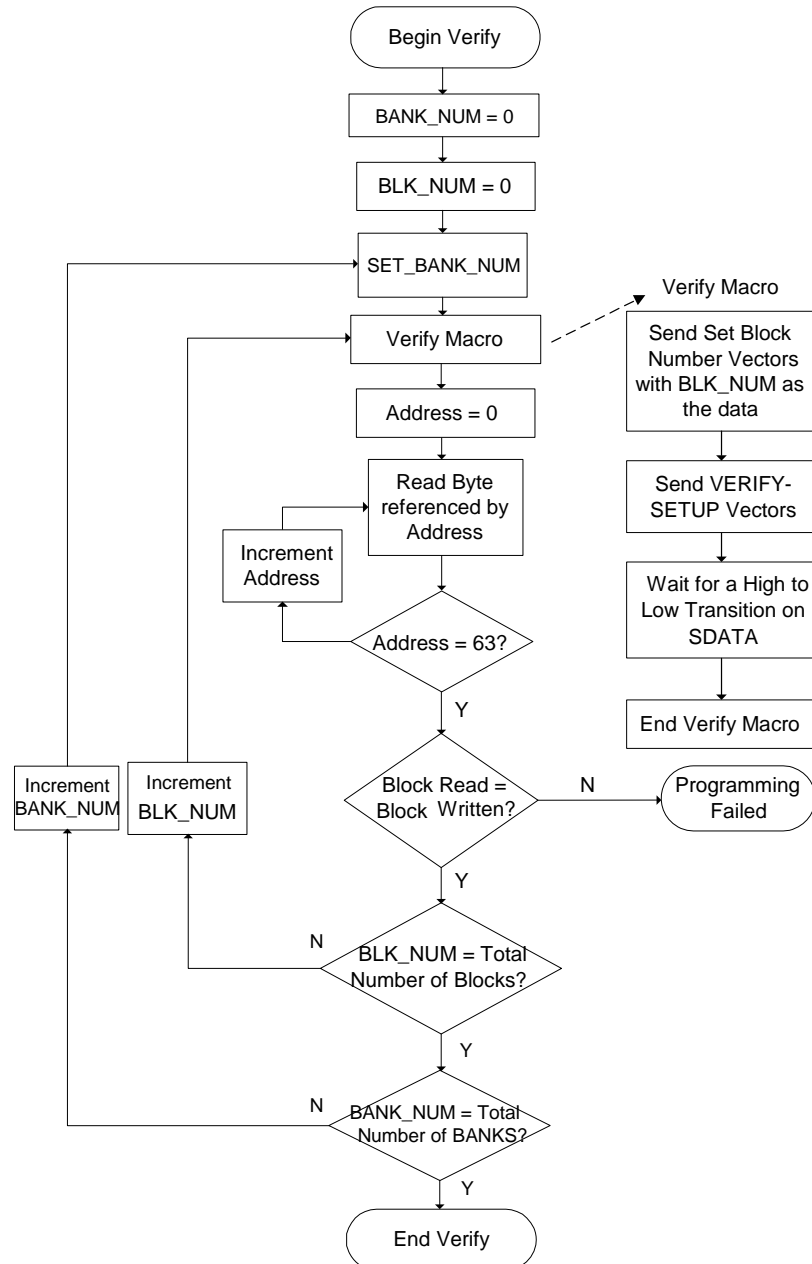


Figure 10. Verify Procedure

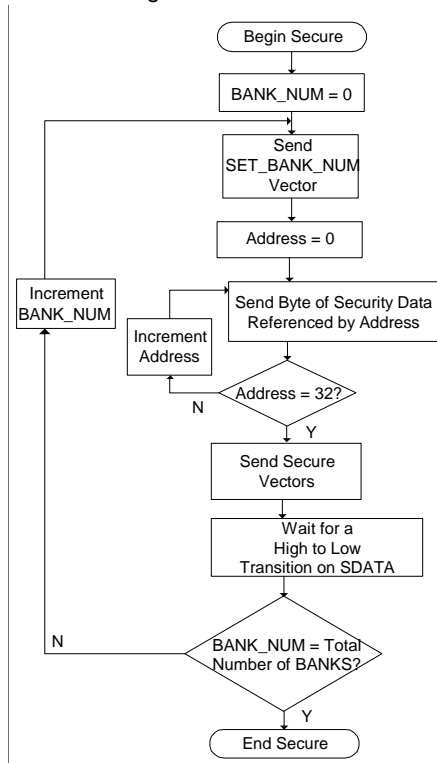


Secure Procedure

The Secure Procedure (shown in [Figure 11](#)), writes the user-determined security values to the target for each block.

After the Flash is programmed and verified, each bank of the Flash must be secured separately. Each 4K bank is secured by 32 bytes of security data. The 32 bytes of security data are written to the target SRAM using the WRITE-BYTE vector. This block defines the access modes for each 64-byte block of the program image. After the 32 bytes are written to the target, the appropriate SECURE vector-set is sent to the target and the programmer waits and polls SDATA while the operation executes. The security data is located in the .hex file ([Appendix B on page 11](#)).

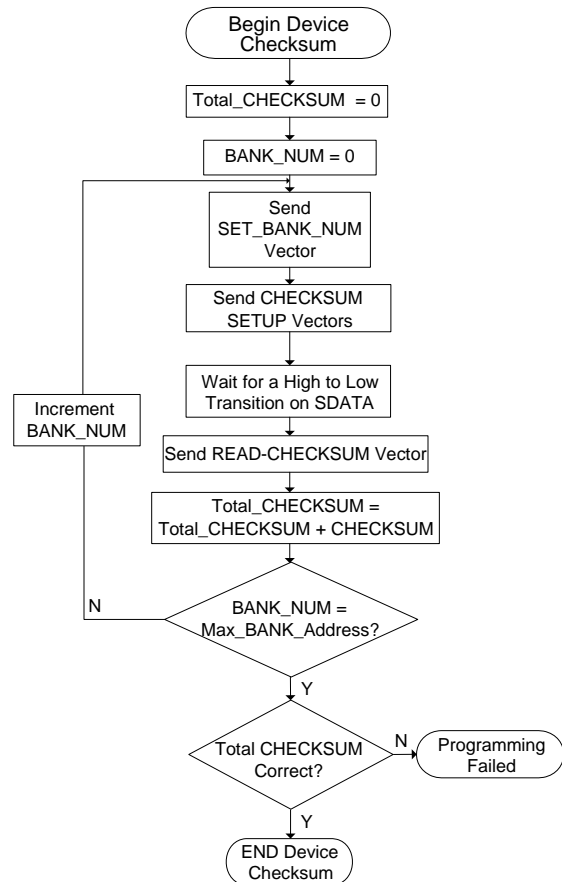
Figure 11. Secure Procedure



Verify Checksum Procedure

The Verify Checksum Procedure (shown in [Figure 12](#)), causes the target to generate a Checksum Value for the data in Flash.

Figure 12. Verify Checksum Procedure



To get the Checksum Value from the target, the programmer sends the appropriate CHECKSUM-SETUP vector to the target. The programmer releases the SDATA line, then waits and polls. After the target signals that the operation is complete, the READ-CHECKSUM vector is used to read back the two-byte Checksum Value from the target. This value from the target is compared to the Device Checksum Value from the .hex file ([Appendix B on page 11](#)). If the values are not equal, a programming error has occurred.

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

Specifications and Definitions

DC Programming Specifications

Table 2. DC Programming Specifications

DC Programming Specifications	Minimum	Maximum
I_{DDP} (Supply Current During Programming or Verify)		30 mA
V_{ilp} (Input Low Voltage During Programming or Verify)	-0.3 V	0.8 V
V_{ihp} (Input High Voltage During Programming or Verify)	2.2 V	$V_{DD}+0.3$ V
I_{ilp} (Input Current when Applying V_{ilp} to P1[0] or P1[1] During Programming or Verify)		0.20 ^a mA
I_{ihp} (Input Current when Applying V_{ihp} to P1[0] or P1[1] During Programming or Verify)		1.5 ¹ mA
V_{olv} (Output Low Voltage During Programming or Verify IOL=0.1 mA)		$V_{SS}+0.75$ V
V_{ohv} (Output High Voltage During Programming or Verify IOH=5 mA)	$V_{DD}-1.0$ V	V_{DD}
V_{ddp} (V_{DD} for Programming and Erase)	3.0 V	5.5 V
V_{dd} (V_{DD} for Verify)	3.0 V	5.5 V
V_{ipor} (Power On Reset Trip)	1.6 V	2.30 V

a. Driving internal pull down resistor.

AC Programming Specifications

Table 3. AC Programming Specifications

AC Programming Specifications	Minimum	Maximum
T_{rclk} (Rise Time of SCLK)	1 ns	20 ns
T_{fclk} (Fall Time of SCLK)	1 ns	20 ns
T_{ssclk} (Data Set Up Time to Falling Edge of SCLK)	40 ns	
T_{hsclk} (Data Hold Time From Falling Edge of SCLK)	40 ns	
F_{sclk} (Frequency of SCLK)	0 MHz	8 MHz ^a
T_{dsclk} (Data-Out Delay from Falling Edge of SCLK)		45 ns ¹
$T_{vddwait}$ (V_{DD} Stable to WAIT-AND-POLL Hold Off ^b)	0.1 ms	1 ms
T_{poll} (SDATA High Pulse Time ^c)	10 μ s	100 ms
T_{acq} (Delay from WAIT-AND-POLL to Initialize-1 ^d)		3 ms
T_{xres} (Duration of External Reset)	10 μ s	
$T_{xresini}$ (Programming Mode Acquisition Window)		125 μ s

a. With capacitance ≤ 30 pF.

b. Until V_{DD} stabilizes, SDATA is noisy and the falling edge must not be searched for. Therefore, a delay of $T_{vddwait}$ is needed after V_{DD} is applied and before WAIT-AND-POLL.

c. This applies to WAIT-AND-POLL mnemonic. The SDATA remains high for T_{poll} time.

d. The Initialize-1 bitstream data must not be delayed more than T_{acq} from the end of the WAIT-AND-POLL (measured from SDATA's falling edge).

Device Address and Block Definitions

Table 4. Device Address and Block Definitions

Device Part Numbers	CY8C24794, CY8C24894, CY8C24994	CY8C29xxx
Byte Addresses within a Block	0-63	0-63
Max_byte_address	63	63
Block Addresses within a Flash Bank	0-127	0-127
Max_block_address	127	127
Flash Bank Addresses	0-1	0-3
Max_bank_address	1	3

Appendix A

Programming Vectors for CY8C24794, CY8C24894, CY8C24994, and CY8C29xxx

Table 5. Programming Vectors

[illegible]

Table 5. Programming Vectors (Continued)

Name	Data...
PROGRAM-BLOCK	1001111110001010100111100111111001010110111101110111000000001111011110110000000011110 011111000001110101111001111100100000011111101111010100000000111101111010000000011111001 111101110000000111110111100100110000111101111010100100000001111011110000000001011110111 1100000000000111110111111100010010111
VERIFY-SETUP	1101111011100000000111101111010000000001110011111000001110101111001111001000000111111 011110101000000001111011110100000000111110011110111000000011110111100100110000111101 11110100100000011110111100000000001111101111000000000011110111111100010010111
READ-BYTE	10110aaaaaZDDDDDDDDZ1 where DDDDDDD= data out, aaaaa=address (6 bits)
SECURE	1001111110001010100111100111111001010110111101110111000000001111011110110000000011110 011111000001110101111001111100100000011111101111010100000000111101111010000000011111001 11110111000000011110111110010011000011110111101010010000001111011110000000010011110111 1100000000000111110111111100010010111
CHECKSUM-SETUP	1101111011100000000111101111010000000001110011111000001110101111001111001000000111111 01111010100000000111101111010000000011111001111101110000000011110111100100110000111101 11110100100000011110011110101000000011110111100000000111111011111000000000011110111 1111100010010111
READ-CHECKSUM	10111111001ZDDDDDDDDZ110111111000ZDDDDDDDDZ1 where DDDDDDDDDDDDD= Device Checksum data out

Notes1=Logic high= V_{ihp} 0=Logic low= V_{ilp}

Z=High Z (floating)

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

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

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

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

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

Other Mnemonics**WAIT-AND-POLL:**

The programmer clocks in a “Z” to the device (with enough set up time for the device SDATA pin to drift low to V_{ilp} by the device’s internal pull down resistor -- typically 1 μ S). The SCLK is then held low.

The device outputs a logic high on the SDATA pin and then switches to output a logic low. The programmer must WAIT-AND-POLL the SDATA pin for the HIGH to LOW transition.

WAIT-AND-POLL uses AC timing specification T_{poll} (from [Table 3 on page 8](#)).

After the transition to low is observed, the programmer must apply a bit stream of 0000000000000000000000000000000000 (40 zero bits) to the SDATA pin of the device and then continue to the next step.

Appendix B

IntelHex File Format for CY8C24794, CY8C24894, CY8C24994, and CY8C29xxx

IntelHex file records are a text representation of Hexadecimal coded binary data. Only ASCII characters are used, so the format is portable across virtually all computer platforms.

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

Each line in an IntelHex file is called a 'record'. The flash program data and end data are made up of a single record. The security data and checksum data are made up of multiple records. These data each have an extended linear address record and one or more records. Records always begin with a colon (:), followed by the number of data bytes in each record. For the devices, Flash program data records always use 64 bytes of data so the Hexadecimal value in the file is always \$40 for that type.

For flash programming data records, the next pair of numbers represent the 16-bit starting address of the data in the record. This is the absolute location in the Flash memory. This number must be a multiple of 64 (\$00, \$40, \$80, \$C0,...) for Flash program data records because each record contains 64 bytes.

The starting address is followed by a byte representing the record type. If this is \$00, the next bytes are the actual program data to be stored in Flash. A \$01 indicates that this is the end of the file. A \$04 indicates an “Extended Linear Address Record” and is used for security data and device checksum data storage (see the following examples).

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

Following the record type are the Hexadecimal representations of the data to be stored. The last byte is a two's-complement checksum of all of the bytes in the record, not including the colon – this is called the record checksum. Note that this value is derived from the binary values of the bytes rather than the ASCII representation.

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

Example Flash Program Data Record

[illegible]

Broken down, it is as follows:

```
:      - Colon, indicates that this is IntelHex
40     - Number of data bytes to follow = $40(40 hex)
00C0   - Starting address in the FLASH for record.
00     - This is the record type -- $00 = Data
```

[illegible]

These are 64 bytes of data in hex as noted above. The first byte (\$50) will be stored at \$00C0, with the remaining bytes following in sequence.

E8 - This is record checksum. If you add all of successive bytes (note that the address is treated as two individual bytes), and truncate it to the lowest eight bits, the result is \$18. The two's complement of \$18 is \$E8. (This may be derived by subtracting \$18 from \$100, or by inverting the bits and adding one to the result.)

(CR/LF) - End of this record.

Example Device Checksum Data Records

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

:      - Colon, indicates that this is IntelHex
02      - Number of data bytes - 2 bytes of data
0000      - Address - zero
04      - This is the record type -- $04 indicates
          Extended Linear Address record
0020      - 2 hex data bytes used - here byte 1 has $00,
          byte 2 has $20 data.
          This indicates that indicates that the checksum
          data is offset in memory space ($0020 is use
          for checksum data).
da      - The record checksum, calculated as above.
(CR/LF)      - End of this record.

:      - Colon, indicates that this is IntelHex
02      - Number of data bytes - 2 bytes of data
0000      - Address - zero
00      - Record type -- $00 indicates data record
253a      - 2 hex data bytes used - here byte 1 has $25, byte 2 has $3a data. The data is a 2
          byte checksum of all of the data stored in flash.
9f      - The record checksum, calculated as above.
(CR/LF)      - End of this record.
```

Additional Notes on Device Checksum Data Records

The device checksum data must be in the file after all security data records are specified.

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

End Record (End of File)

```
:00000001FF(CR/LF)

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

Device Address and Block Definitions

The least significant 6 bits in the IntelHex address define the byte address (0 to 63) within a block. The most significant bits in the IntelHex address define the block number. See [Table 4 on page 8](#).

About the Author

Name: Max Kingsbury
Title: Applications Engineer
 B.S. Electrical Engineering, Washington
 State University
Background:
Contact: maxk@cypress.com

Document History Page

Document Title: In-System Serial Programming (ISSP) Protocol for CY8C24794, CY8C24894, CY8C24994, CY8C29x66
Document Number: 001-15239

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1483026	HMT	09/17/2007	Converted to new template.
*A	2546104	MAXK/AESA	07/31/2008	Updated Power on Mode on page 3 . Updated IntelHex File Format for CY8C24794, CY8C24894, CY8C24994, and CY8C29xxx on page 11 . Converted to latest application note template.

In March of 2007, Cypress recataloged all of its Application Notes using a new documentation number and revision code. This new documentation number and revision code (001-xxxxx, beginning with rev. **), located in the footer of the document, will be used in all subsequent revisions.

PSoC is a registered trademark of Cypress Semiconductor Corp. "Programmable System-on-Chip," PSoC Designer, and PSoC Express are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2005-2008. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.