In-System Serial Programming (ISSP) Protocol for CY8C21xxx/CY8C22xxx/CY8C24xxx/CY8C24xxxA/CY8C27xxx

AN2026a

Author: Max Kingsbury
Associated Project: No
Associated Part Family: CY8C21xxx/CY8C22xxx/CY8C24xxx/CY8C24xxxA/CY8C27xxx
GET FREE SAMPLES HERE
Associated Application Notes: AN2014, AN2026, AN2026b

Application Note Abstract

This application note describes how to program CY8C21/22/24/24A/27xxx PSoC® devices using the In-System Serial Programming (ISSP) protocol (the CY8C24x94 device family is not included).

Introduction

The emulator pod and individual chips can be programmed in-system through 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 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 13 for more information on 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

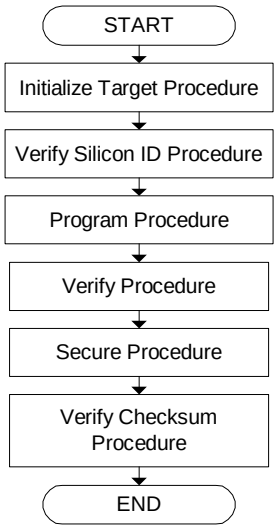
Pin Name	Function	Programmer HW Pin Requirements
P1[0]	SDATA - Serial Data In/Out	Drive TTL Levels, Read TTL, High Z
P1[1]	SCLK - Serial Clock	Drive TTL Level Clock Signal
X_{res}	Reset	Drive TTL Levels
V_{SS}	Power Supply Ground Connection	Low Resistance Ground Connection
V_{DD}	Power Supply Positive Voltage	3.3 or 5V, 25 mA Capability

For more information on the physical requirements of connections between the programmer and the 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 various operations involved in the programming flow. Each procedure in the programming flow has many individual vectors associated with it (Appendix A on page 10). 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 when 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 (Fsclk, Table 3 on page 8). The SCLK rate in data transfer mode can be controlled with less precision. Clocks are not required for the "Wait and Poll" steps described in the procedure flowcharts. Data Transfer clocking can be used at all other times.

Command Format

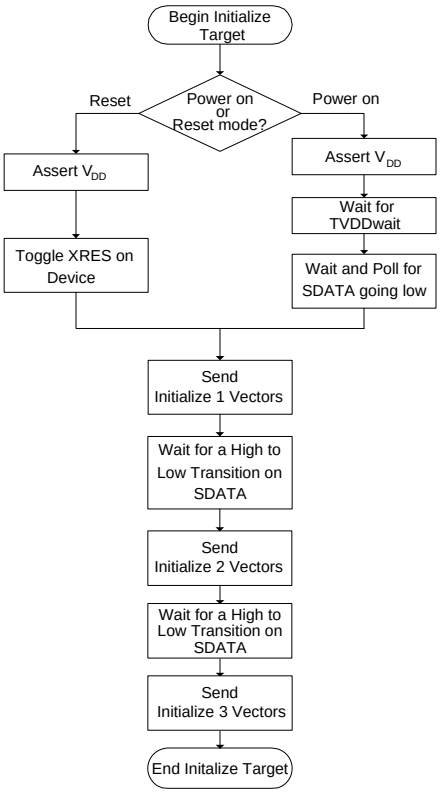
During programming, only the programmer drives the SCLK line. The programmer and target can drive the SDATA line, although the target drives SDATA only upon a read request from the programmer. The programmer always writes and reads SDATA on the rising edge of SCLK, while the target writes and reads on the falling edge. After the programmer requests a read from the target (Appendix A on page 10, 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 Reset mode or Power on mode.

Reset mode is the preferred method for initiating communication with the target. However, in the case of the 8-pin DIP package, 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

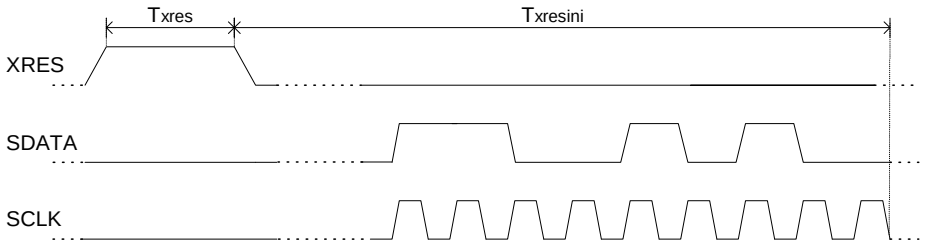
The timing to enter programming mode with Reset is shown in Figure 3. To initialize the part using the X_{res} line, first wait until V_{DD} is stable, and then assert the X_{res} line for the time specified by T_{xres} (Table 3 on page 8). After X_{res} is driven low, there is a window of time specified 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 10.

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 can 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.

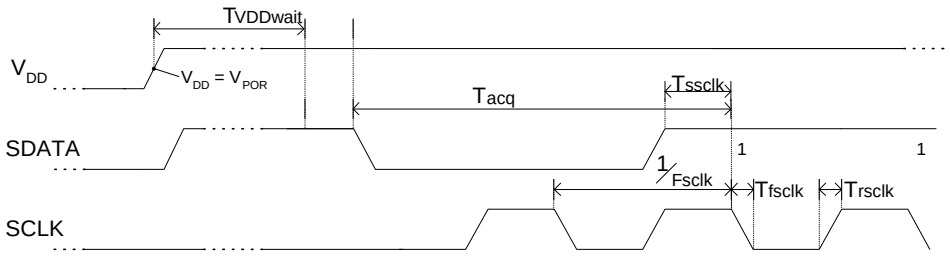
After the SDATA transition is detected, the programmer must transmit the Initialize 1 vectors in T_{acq} seconds (Table 3 on page 8).

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.

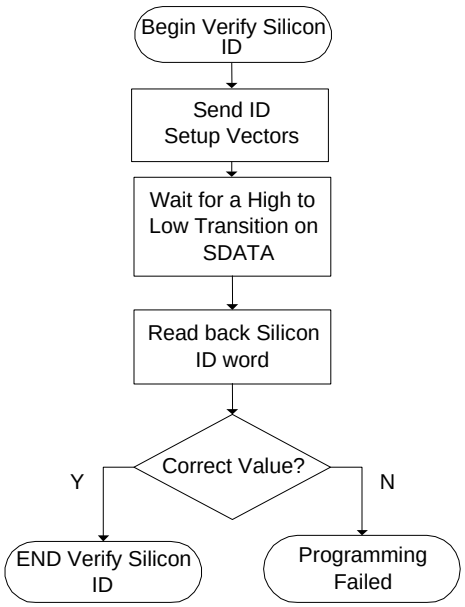
Figure 4. Using Power Cycling to Initialize



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.

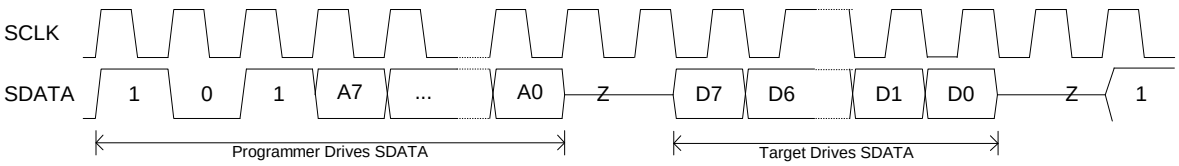
Figure 5. Verify Silicon ID Procedure



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 10 under READ-ID show the package-specific values read from the target, that is, a LLLLHLLH denotes a 0x09 hex read back from an 8-pin target (CY8C27143). The programmer must compare the value in the READ-ID (Appendix A on page 10) and the value returned by the target. If these values do not match, the programmer must terminate the programming flow.

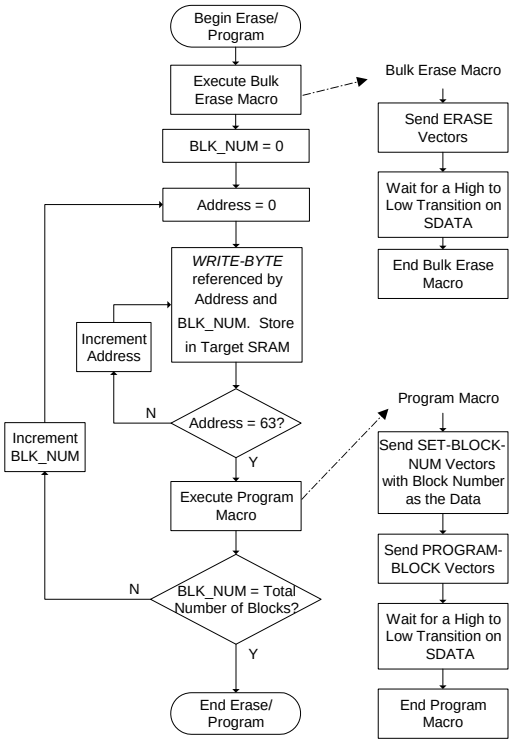
Figure 6. Last Portion of READ-ID from Target



Program Procedure

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

Figure 7. Program Procedure



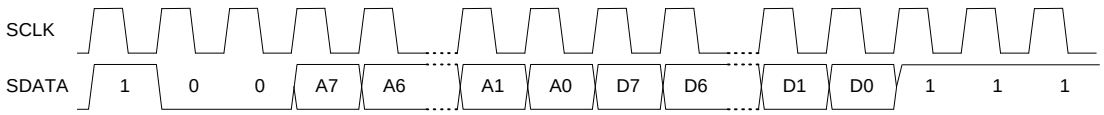
A Bulk Erase operation must be executed to prepare the Flash for programming.

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 13) 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 10) is first sent with the 'ddddddddd' 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 starting at address 0x80. The programmer must then use the READ-BYTE vector with an offset of 0x80 (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 must terminate the programming flow as a failure.

Figure 9. READ-BYTE From Target

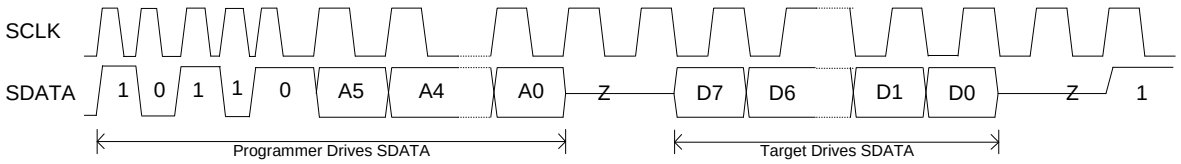
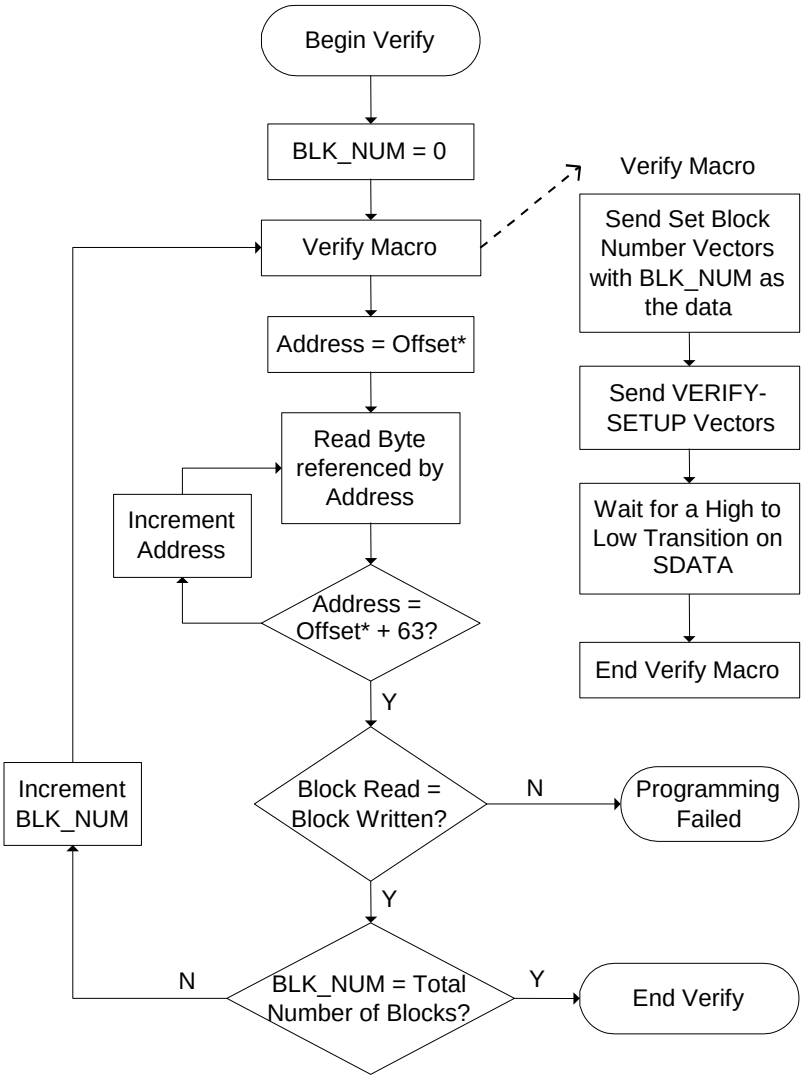


Figure 10. Verify Procedure

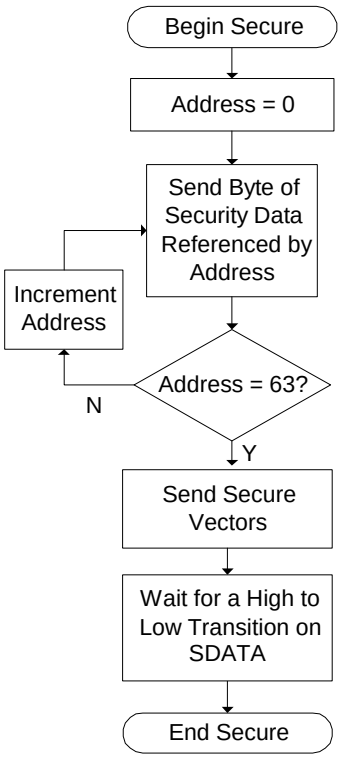


Secure Procedure

The Secure Procedure (shown in Figure 11 on page 7), writes the user-determined security values to the target for each block.

After the flash is programmed and verified, each byte of the 64-byte security block is written to the target SRAM using the WRITE-BYTE vector. This block defines the access modes for each 64-byte block of the program image. After the 64-byte block is written to the target, the appropriate SECURE vector-set is sent to the target and the programmer waits and polls SDATA for the operation to execute. There are different SECURE vectors for different part numbers. The correct vector must be specified for the PSoC being programmed. The security data is located in the .hex file (Appendix B on page 13).

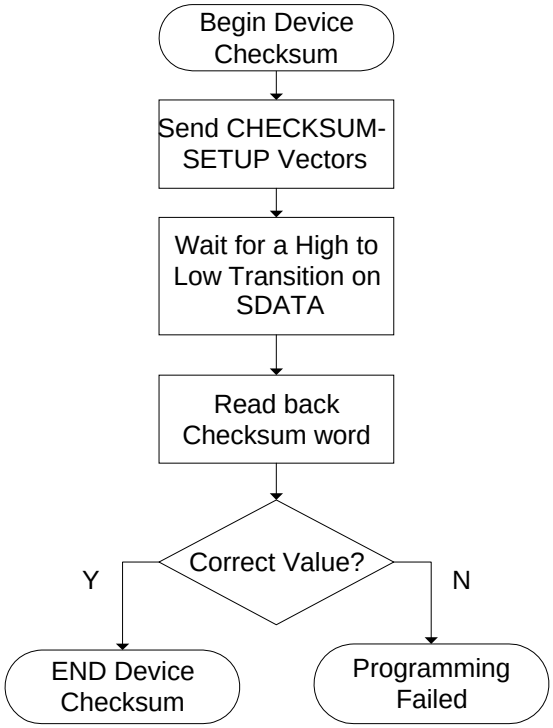
Figure 11. Secure Procedure



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-set to the target. There are different CHECKSUM-SETUP vectors for different part numbers. The correct vector must be specified for the PSoC being programmed. The programmer releases the SDATA line then waits and polls. After the target signals that the operation is complete, the READ-CHECKSUM vector-set is used to read back the two-byte Checksum Value from the target. This value from the target is compared to the Device Checksum Value from the .hex file (Appendix B). 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)		25 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[a] 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)[b]	2.7 V	5.5 V
V_{dd} ($V_{\mathbf{DD}}$ 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.
b. The minimum voltage of 2.7 only applies to devices that support this value.

AC Programming Specifications

Table 3. AC Programming Specifications

AC Programming Specifications	Minimum	Maximum
T_{rsclk} (Rise Time of SCLK)	1 ns	20 ns
T_{fsclk} (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 [a]
$T_{vddwait}$ (V_{DD} Stable to WAIT-AND-POLL Hold Off[b])	0.1 ms	1 ms
T_{poll} (SDATA High Pulse Time[c])	10 µs	100 ms
T_{acq} (Delay from WAIT-AND-POLL to Initialize-1[d])		3 ms
T_{xres} (Duration of External Reset)	10 µs	
$T_{xresini}$ (Programming Mode Acquisition Window)		125 µs

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	Address Numbers (Bytes Within a Block)	Block Numbers (Program Data)	max_data_block
CY8C21123	0-63	0-63	63
CY8C21223	0-63	0-63	63
CY8C21323	0-63	0-63	63
CY8C21234	0-63	0-127	127
CY8C21334	0-63	0-127	127
CY8C21434	0-63	0-127	127
CY8C21534	0-63	0-127	127
CY8C21634	0-63	0-127	127
CY8C22113	0-63	0-31	31
CY8C22213	0-63	0-31	31
CY8C24123[a]	0-63	0-63	63
CY8C24223[a]	0-63	0-63	63
CY8C24423[a]	0-63	0-63	63
CY8C27143	0-63	0-255	255
CY8C27243	0-63	0-255	255
CY8C27443	0-63	0-255	255
CY8C27543	0-63	0-255	255
CY8C27643	0-63	0-255	255

a. Also applies to CY8C24xxxA devices.

Appendix A

Programming Vectors for CY8C21/22/24/24A/27xxx

Table 5. Programming Vectors

Name	Data
Vector	Bit Stream (Executed From Left Bit to Right)
Initialize-1	1100101000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000001101111011100000000111110111101100000000001111001 11110000011101011110011111001000000111111101111010100000000111110111101000000000111111100111 110111000000011111011111001001100001111101111101001000000111110111100000000100111111011111 000000000001111101111111100010010111
Initialize-2	11011110111000000000111110111101100000000011110011111000001110101111001111100100000011111110111101010000 0000111110111101000000000111111001111101110000000111110111110010011000011111011111010010000001111001111110 100000000111111011110000000000110111110111110000000000001111101111111100010010111
Initialize-3 3V	110111101110000000001111101111010000000001111111011110101000000000111110111101100000100011111011111100000101 000111111011111001111110001111101111101000110000111110111111110001001011100000000000000000000000110111101 11000000001111101111010000000001111111011110101000000000111110111101100000100011111011111000011000001111 011111001111010101111101111101000110000111110111101110001000011111011111111000100101110000000000000000000 000001101111011100000000011111011110100000000011111110111101010000000001111101111011000001000111110111100 001010001111110111110011111100111111011111010001100001111101111111100010010111000000000000000000000000110 111101110000000001111101111010000000001111111011110101000000000111110111101100000100011111011111000011000 0111110111110011110100011111011111010001100001111101111011100010000111110111111110001001011100000000000000 0000000000
Initialize-3 5V	110111101110000000001111101111010000000001111111011110101000000000111110111101100000100011111011111100000101 000111111011111001111110011111011111010001100001111101111111100010010111000000000000000000000000110111101 11000000001111101111010000000001111111011110101000000000111110111101100000100011111011111000011000001111 011111001111010101111101111101000110000111110111101110001000011111011111111000100101110000000000000000000 000001101111011100000000011111011110100000000011111110111101010000000001111101111011000001000111110111100 00101000111111011111001111110111111011111010001100001111101111111100010010111000000000000000000000000110 111101110000000001111101111010000000001111111011110101000000000111110111101100000100011111011111000011000 0111110111110011110100011111011111010001100001111101111011100010000111110111111110001001011100000000000000 0000000000
ID-SETUP	11011110111000100001111101110000000000000101111101111011100000000111110111101100000000011110 0111110000011101011110011111001000000111111101111010100000000111110111101000000000111111001 11110111000000011111011111001001100001111101111101001000000111100111110100000000000111110111 100000000011011111011111000000000000011111011111111000100100111
	READ-ID-WORD (CY8C27143) 10111111000ZLLLLLLLLZ110111111001ZLLLLHLLHZ1
	READ-ID-WORD (CY8C27243) 10111111000ZLLLLLLLLZ110111111001ZLLLLHLHLZ1
	READ-ID-WORD (CY8C27443) 10111111000ZLLLLLLLLZ110111111001ZLLLLHLHHZ1
	READ-ID-WORD (CY8C27543) 10111111000ZLLLLLLLLZ110111111001ZLLLLHHLLZ1
	READ-ID-WORD (CY8C27643) 10111111000ZLLLLLLLLZ110111111001ZLLLLHHLHZ1
	READ-ID-WORD (CY8C24123)[a] 10111111000ZLLLLLLLLZ110111111001ZLLLHLLHLZ1
	READ-ID-WORD (CY8C24223)[1] 10111111000ZLLLLLLLLZ110111111001ZLLLHLLHHZ1
	READ-ID-WORD (CY8C24423)[1] 10111111000ZLLLLLLLLZ110111111001ZLLLLHLHLLZ1
	READ-ID-WORD (CY8C22113) 10111111000ZLLLLLLLLZ110111111001ZLLLLHHHHZ1
	READ-ID-WORD (CY8C22213) 10111111000ZLLLLLLLLZ110111111001ZLLLHLLLLZ1
	READ-ID-WORD (CY8C21123) 10111111000ZLLLLLLLLZ110111111001ZLLLHLHHHZ1
	READ-ID-WORD (CY8C21223) 10111111000ZLLLLLLLLZ110111111001ZLLLHHLLLZ1

Table 5. Programming Vectors (Continued)

Name	Data
	READ-ID-WORD (CY8C21323) 10111111000ZLLLLLLLLZ110111111001ZLLLHHLLHZ1
	READ-ID-WORD (CY8C21234) 10111111000ZLLLLLLLLZ110111111001ZLLHHLHHLZ1
	READ-ID-WORD (CY8C21334) 10111111000ZLLLLLLLLZ110111111001ZLLHHLHHHZ1
	READ-ID-WORD (CY8C21434) 10111111000ZLLLLLLLLZ110111111001ZLLHHHLLLZ1
	READ-ID-WORD (CY8C21534) 10111111000ZLLLLLLLLZ110111111001ZLHLLLLLLZ1
	READ-ID-WORD (CY8C21634) 10111111000ZLLLLLLLLZ110111111001ZLHLLHLLHZ1
SET-BLOCK-NUM	10011111010dddddddd111 where dddddddd=block #
ERASE	10011111100000101011111001111111001010110111110111101110000000001111101111011000000000011110 01111100000111010111100111110010000001111111011110101000000000111110111101000000000111111001 11110111000000011111011111001001100001111101111101001000000011111011110000000001011111110111 11000000000001111101111111100010010111
WRITE-BYTE	10010aaaaaadddddddd111 where dddddddd= data in, aaaaaa=address (6 bits)
PROGRAM-BLOCK (CY8C21xxx/CY8C22xxx/CY8C24xxx/CY8C24xxxA)	10011111100010101001111001111111001010110111110111101110000000001111101111011000000000011110 01111100000111010111100111110010000001111111011110101000000000111110111101000000000111111001 11110111000000011111011111001001100001111101111101001000000011111011110000000001011111110111 11000000000001111101111111100010010111
PROGRAM-BLOCK (CY8C27xxx)	10011111100000101011111001111111001010110111110111101110000000001111101111011000000000011110 01111100000111010111100111110010000001111111011110101000000000111110111101000000000111111001 11110111000000011111011111001001100001111101111101001000000011111011110000000001011111110111 11000000000001111101111111100010010111
VERIFY-SETUP	11011110111000000001111101111011000000000111100111111000001110101111001111100100000001111111 01111010100000000111110111101000000000111111001111101110000000111110111110010011000011111101 11110100100000011111011110000000000001111110111110000000000001111101111111100010010111
READ-BYTE	10110aaaaaaZDDDDDDDDZ1 where DDDDDDDD= data out, aaaaaa=address (6 bits)
SECURE (CY8C21xxx/CY8C22xxx/CY8C24xxx/CY8C24xxxA)	10011111100010101001111001111111001010110111110111101110000000001111101111011000000000011110 01111100000111010111100111110010000001111111011110101000000000111110111101000000000111111001 11110111000000011111011111001001100001111101111101001000000011111011110000000001001111110111 11000000000001111101111111100010010111
SECURE (CY8C27xxx)	10011111100000101011111001111111001010110111110111101110000000001111101111011000000000011110 01111100000111010111100111110010000001111111011110101000000000111110111101000000000111111001 11110111000000011111011111001001100001111101111101001000000011111011110000000001001111110111 11000000000001111101111111100010010111
CHECKSUM-SETUP (CY8C27xxx)	11011110111000000001111101111011000000000111100111111000001110101111001111100100000001111111 01111010100000000111110111101000000000111111001111101110000000111110111110010011000011111101 11110100100000011110011111010000000000111110111100000000001111111101111100000000000111110111 1111100010010111
CHECKSUM-SETUP (CY8C21x23/CY8C24xxx/CY8C24xxxA)	11011110111000000001111101111011000000000111100111111000001110101111001111100100000001111111 01111010100000000111110111101000000000111111001111101110000000111110111110010011000011111101 11110100100000011110011111010010000001111101111000000000011111111011111000000000000111110111 1111100010010111
CHECKSUM-SETUP (CY8C21x34	11011110111000000001111101111011000000000111100111111000001110101111001111100100000001111111 01111010100000000111110111101000000000111111001111101100000000111110111110010011000011111101 11110100100000011110011111010100000000111110111100000000001111111101111100000000000111110111 1111100010010111
CHECKSUM-SETUP (CY8C22xxx)	11011110111000000001111101111011000000000111100111111000001110101111001111100100000001111111 01111010100000000111110111101000000000111111001111101110000000111110111110010011000011111101 11110100100000011110011111010001000001111101111000000000011111111011111000000000000111110111 1111100010010111
READ-CHECKSUM	10111111001ZDDDDDDDDZ110111111000ZDDDDDDDDZ1 where DDDDDDDDDDDDDDDD= Device Checksum data out

a. Also applies to CY8C24xxxA devices.

Notes

1=Logic high=Vihp

0=Logic low=Vilp

Z=High Z (floating)

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

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

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

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

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

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

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 µS). 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 Tpoll (from Table 3 on page 8).

After the transition to low is observed, the programmer must apply a bit stream of 0000000000000000000000000000000000000000 (40 zero bits) to the SDATA pin of the device and then continue to the next step.

Appendix B

IntelHex File Format for CY8C21/22/24/24A/27xxx

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 data records. Records always begin with a colon (:), followed by the number of data bytes in each record. For the devices, Flash program data records always use 64 bytes of data so the Hexadecimal value in the file is always $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 the 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

```
:4000C000505152535455565758595A5B5C5D5E5F000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000E8(CR/LF)
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
505152535455565758595A5B5C5D5E5F000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000
          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 Security Data Records

```
:020000040010ea(CR/LF)
:40000000555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555
55555555555555555555555555555555555555555555580(CR/LF)

:         - Colon, indicates IntelHex
02        - Number of data bytes – 2 bytes of data
0000      - Address - zero
04        - This is the record type -- $04 indicates
                 Extended Linear Address record
0010      - 2 hex data bytes used – here byte 1 has $00,
                 byte 2 has $10 data.
                 This indicates that the
                 security data is offset in memory space
                 ($0010 is used for security data).
ea        - The record checksum, calculated as above.
(CR/LF)   - End of this record.

:         - Colon, indicates that this is IntelHex
40        - Number of data bytes – 64 bytes
0000      - Address – zero
00        - Record type - $00 indicates data record
55555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555
55555555555555555555555555555555
          - 64 data bytes – here bytes have $55 data
80        - The record checksum, calculated as above.
(CR/LF)   - End of this record.
```

Additional Notes on Security Records

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

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

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

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 $39 data. The data is a 2
            byte checksum of all of the data stored in flash.
9f        - The record checksum, calculated as above.
            (CR/LF)- End of this record.
```

Additional Notes on Device Checksum Data Records

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

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

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 9.

About the Author

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

Document History Page

Document Title: In-System Serial Programming (ISSP) Protocol for CY8C21xxx/CY8C22xxx/CY8C24xxx/CY8C24xxxA /CY8C27xxx

Document Number: 001-13617

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1772947	FSU	11/26/2007	Converted to new template.
*A	2546104	MAXK/AESA	07/31/2008	Updated Power on Mode on page 3. Updated IntelHex File Format for CY8C21/22/24/24A/27xxx on page 13. Converted to latest application note template. Modified author details.

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.

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