AN60616

PSoC® 3 and PSoC 5LP Startup Procedure

Author: Max Kingsbury
Associated Project: No
Associated Part Family: All PSoC® 3 and PSoC 5LP Parts
Software Version: PSoC Creator™ 2.1 SP1 or higher
Related Application Notes: AN54439, AN60631, AN73854, System Reference Guide

AN60616 describes PSoC® 3 and PSoC 5LP startup procedures, from the application of device power until the execution of user code. It describes how to customize the startup procedure, and includes the reasons a designer might want to change the startup procedure.

Introduction

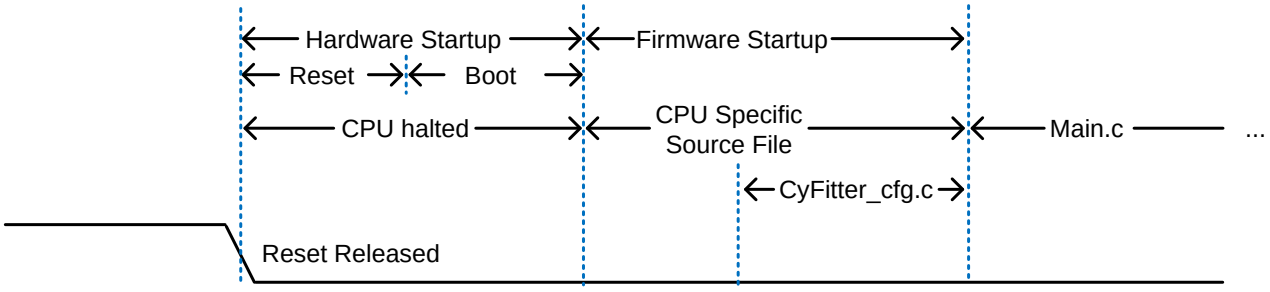
PSoC 3 and PSoC 5LP are incredibly powerful and complicated mixed-signal microcontrollers. Through careful configuration, they can be used to solve all kinds of technical problems. The PSoC Creator integrated design environment (IDE) generates the code that will configure the parts at startup, but it requires application-specific configuration details. Startup behavior can be manipulated to change the amount of time startup takes, what peripherals are configured, and much more. This application note describes the procedure of startup and also how it can be manipulated to best suit an application.

This application note describes PSoC 3 and PSoC 5LP, but does not describe PSoC 5. For a legacy version of this application note describing PSoC 5, see revision *C of this application note, available on the Cypress website here.

Startup Procedure

The PSoC 3 and PSoC 5LP startup procedure, shown in Figure 1, configures the parts to meet the PSoC 3 and PSoC 5LP device datasheet and PSoC Creator™ project specifications. Startup begins after the release of a reset source, or after the end of a power supply ramp. There are two main portions of startup: hardware startup and firmware startup. During hardware startup, the CPU is halted, and other resources configure the part. During firmware startup, the CPU runs code generated by PSoC Creator to configure the part. When startup ends, the PSoC 3 or PSoC 5LP device is fully configured, and its CPU begins execution of user-authored *Main.c* code.

Figure 1. PSoC 3 and PSoC 5LP Startup Procedure Overview



Hardware Startup

PSoC 3 and PSoC 5LP hardware startup configures the parts to meet the general performance specifications given in their datasheets. The hardware startup phase begins after a power supply ramp or reset event. There are two phases of hardware startup: reset and boot. After hardware startup ends, code execution from flash begins.

Startup begins after the release of all reset sources. Reset can be caused by the dedicated and optional reset pins, a watchdog timer, low-voltage detection, power-on reset (POR), or other sources. While reset is asserted, the I/O pins are in the analog high-Z drive mode.

If the chip is experiencing power-supply ramp, startup begins after the POR is initiated. The POR is released when all device power supplies meet the requirements shown in the datasheet. These requirements are listed in the datasheet table “Power on Reset (POR) with Brown Out DC Specifications”.

Once reset is released, the hardware-controlled portion of startup begins. Hardware startup can be split into two phases: reset and boot. In both phases, the CPU is halted. In the reset phase, the device is inactive, waiting for on-chip resources to stabilize enough to enter the boot phase. In the boot phase, a dedicated hardware state machine controls basic configuration and trim of the device using direct memory access (DMA). Executing the boot phase takes a fixed number of clock cycles.

Immediately after the part exits reset, the device is clocked by the fast-start output of the internal main oscillator (IMO). This fast-start output is generated using a fast reference. The normal IMO output becomes valid later, after the normal reference becomes stable. The IMO begins to source the master clock partway through the reset phase. Although the main IMO output is used during part of the reset phase, it is not trimmed until the boot phase.

At the completion of hardware startup, the 8051 or Cortex™-M3 CPU begins to execute code from flash, beginning at address 0.

Note Total hardware startup time is specified in the device datasheet as “$T_{STARTUP}$”.

Note For more information on hardware startup behavior, see the “Reset” section in the PSoC 3 or PSoC 5LP TRM.

Firmware Startup

Firmware startup configures the PSoC 3 or PSoC 5LP device to behave as described in the PSoC Creator project. It begins at the end of hardware startup. The CPU begins executing user-authored main() code after the completion of firmware startup.

The main task of firmware startup is to populate configuration registers such that the part behaves as designed in the PSoC Creator project. This includes configuring analog and digital peripherals, as well as system resources such as clocks and routing.

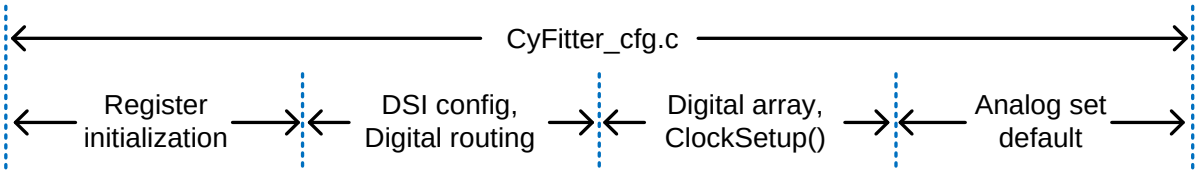
Note Some PSoC Creator components are completely operational after firmware startup, and some require additional code to activate. For example, pins are fully functional, but ADCs require additional API calls. Refer to individual component datasheets for details.

The firmware startup procedure is defined by the contents of two source files. A CPU-specific source file for the 8051 or Cortex M3 CPU contains assembly code that executes immediately at the beginning of firmware startup. For both CPUs, *CyFitter_cfg.c* contains C code that is called from the CPU-specific source code.

The resources configured in the CPU-specific code include debugging, bootloaders, and DMA endpoints.

The code in *CyFitter_cfg.c* configures the device’s registers so that its resources behave as designed in the PSoC Creator project. This includes configuring the analog and digital resources, as well as system resources such as the clock tree and routing. The procedure is shown in detail in Figure 2, which shows a more specific portion of the procedure that was shown in Figure 1.

Figure 2. CyFitter_cfg.c Execution Steps Overview



The *CyFitter_cfg.c* source file contains the function CyFitter_cfg(), which is called by the code in the CPU-specific source file. During the execution of CyFitter_cfg(), numerous device registers are populated.

The largest batch of register population occurs first, where the analog and digital resources are configured using register writes. This step can be performed using either CPU activity or DMA. If this step is performed using the CPU, the function cfg_write_bytes_code() is called. If it is performed using DMA, the function cfg_dma_init() is called.

The second largest batch of register population is the ClockSetup() API call. During clock setup, the part's clock sources and clock tree are configured to match the project's clock configuration. This configuration is altered in PSoC Creator using the clocks tab of the Design Wide Resources interface. During clock setup, the CPU's operating frequency changes from the initial, partially trimmed value to the final desired value.

Note See AN60631 for more details on the clock startup procedure.

After the completion of the firmware startup, the main() code in *main.c* is executed.

PSoC 3and 5LP Pin Behavior at Startup

PSoC 3 and 5LP have special nonvolatile latch (NVL) bits that can be used to set pin behavior immediately after reset is released. These I/O NVL behavior settings become effective during hardware startup. Pins and other resources begin to behave as desired following a set delay after reset is de-asserted or power supply ramp meets requirements. This delay is specified in the datasheet as "T_{IO_INIT}". This value is on the order of microseconds. NVL-controlled pin behavior is configured in the pin component customizer.

If the NVLs are not configured, the pins remain in the analog High-Z drive mode until they are reconfigured to their run-time state.

For more details on pin behavior, see AN72382.

Bootloader Startup

In projects with both bootloader and bootloadable application code, the two application codes run sequentially. This means that the bootloader executes its startup and main() code, followed by the bootloadable startup and main() code. For more details on bootloader startup, see AN73854.

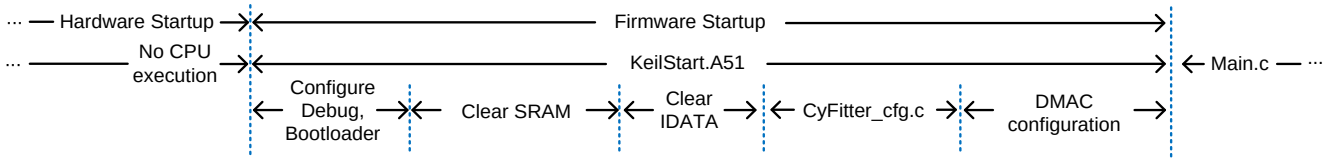
PSoC 3-Specific Startup Details

PSoC 3 Firmware Startup

PSoC 3-specific startup code is stored in the source file *KeilStart.A51* and is written in the 8051 assembler. The code begins at address 0 in flash with an instruction to jump to the STARTUP1 label. The *KeilStart.A51* code performs basic part configuration, and calls CyFitter_cfg() to perform more complex configuration. *KeilStart.A51* execution is shown in Figure 3, which shows a more specific portion of the procedure that was introduced in Figure 1.

The "clear IDATA" step writes zeros to the portion of program memory allocated for IDATA, usually used for variables. The "DMAC configuration" step configures the PSoC 3 part's DMA resources as specified in the PSoC Creator project.

Figure 3. KeilStart.A51 Execution Steps



PSoC 3 Resets

In PSoC 3, all non-software resets must be immediately followed by a software reset. PSoC Creator fulfils this requirement automatically. The resulting behavior is that when a non-software reset is executed, device startup firmware detects the reset type, and executes a software reset. Then, the software reset completes and user code is executed. This code for this firmware startup step is contained in the KeilStart.A51 code, shortly after the STARTUP1 entry point.

Compared to the time taken in firmware startup, the additional time taken to check for and perform software resets is minimal. In a typical application, hardware startup takes about 30 µs with the 48-MHz fast IMO output; therefore, performing it twice takes about 60 µs. Typical firmware startup takes hundreds or thousands of µs. The limited 8051 assembly code required to perform this step takes about 0.5 µs with the 48-MHz fast IMO output.

NVL IMO Frequency Selection

In PSoC 3, the initial IMO frequency can be set to two different values: 12 MHz and 48 MHz. This frequency applies just after the start of the boot portion of hardware startup, until the clock setup portion of *CyFitter_cfg.c* code execution, where the configuration from the clocks tab of the Design Wide Resources is applied. This frequency may be configured by making a selection in the PSoC Creator Design Wide Resources, described in the PSoC 3 and PSoC 5LP Design Wide Resources section. This selection is stored in the NVL bits in the device.

PSoC 5LP-Specific Startup Details

PSoC 5LP Firmware Startup

PSoC 5LP device-specific startup code is contained in the *Cm3Start.c* source file. This file contains two reset functions, one for each of the compilers. The source file uses conditional compilation to determine which compiler is being used, and this way only the applicable compiler's reset function is used. The comments in the functions describe which compiler they are for.

Although they are in a *.c* file, each of the two reset functions are written in in-line assembly. Each ends with a call to the sub-main function, also stored in *Cm3Start.c*. The sub-main function calls a function for PSoC initialization, aptly titled initialize_psoc(), and then calls the main() function in *main.c*. The initialize_psoc() function performs some configuration steps but leaves the majority of configuration to the code in cyfitter_cfg.c, which was discussed in the Firmware Startup section.

Identifying Reset Sources

After reset and startup, it may be of use to know what caused the device to reset. Soft resets, including the watchdog timer and software reset result in a status bit that can be read after the reset. Although the low-voltage interrupt (LVI) and high-voltage interrupt (HVI) sources may be configured to cause a reset, the status bits are cleared after reset so that a reset cause by LVI or HVI cannot be determined. There are also two general-purpose bits that are stored across resets, which may be used to track system status. These resets and general-purpose bits may be observed by examining the CyResetStatus variable, which is maintained by PSoC Creator-generated startup code. More details can be found in the PSoC Creator System Reference Guide, under the "Preservation of Reset Status" heading.

Note "Hard" resets, such as POR, precision reset (PRES), and reset pins (XRES), are not identifiable using this resource.

Clock Startup

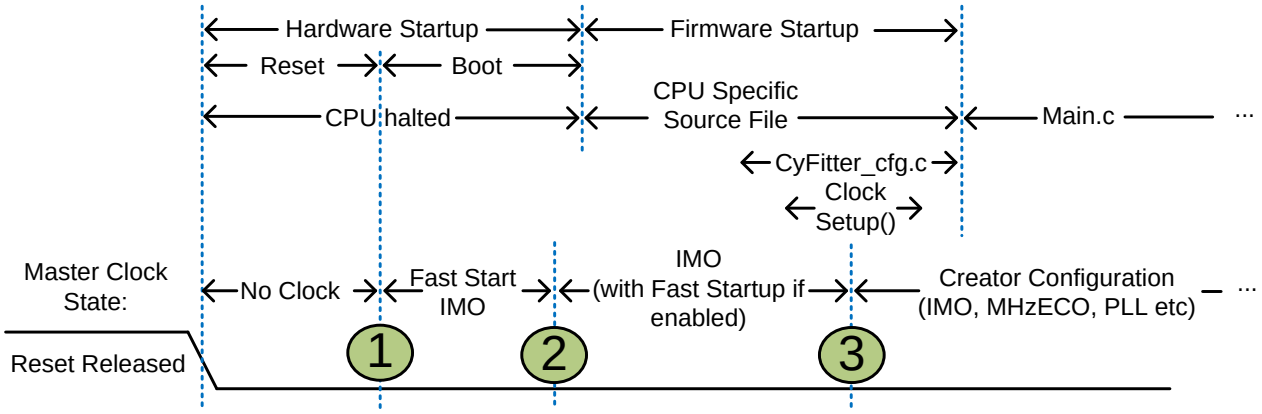
PSoC 3 and PSoC 5LP clocks are configured during startup, as described in previous sections. The clocking startup steps are laid out sequentially here for clarity. The sequence is shown in Figure 4. Immediately following the release of any reset source, the master clock is driven low. It is not until the end of the hardware reset phase, and the beginning of the boot phase, when the fast start IMO is routed to the master clock bus. This is marked as "1" in the figure. This clock is distinct from the normal IMO, and it starts up more quickly but is less accurate.

After a certain amount of time in the boot phase, the IMO has stabilized, and is used to source the master clock. This is marked as "2" in the figure. In PSoC 3, if the "fast startup" option has been selected, this IMO clock will be at the configured frequency.

The final master clock reconfiguration is performed during firmware startup, in the ClockStartup() function that is contained in *CyFitter_cfg.c*. This is marked in the figure as "3". This function performs additional configuration of the master clock and all other clock sources in the project. It is only after the completion of ClockStartup() that the configuration of clocks in the device will match the configuration in the PSoC Creator project.

For more details on clock configuration in PSoC 3 and PSoC 5LP, see AN60631.

Figure 4. PSoC 3 and PSoC 5LP Startup Clock Configuration Sequence



Modifying Startup

The startup procedure may be altered to better fit a specific application's needs. You can modify the device startup in the following ways: using the design wide resources (DWR) interface in the PSoC Creator GUI and modifying the PSoC 3 startup code stored in the CPU-specific source files and *CyFitter_cfg.c*.

Using the PSoC Creator GUI

The startup procedure may be modified in PSoC Creator in the "System" tab of the design wide resources. Because PSoC 3 and PSoC 5LP have different capabilities, they have different options available in this interface. These interfaces are shown in Figure 5 and Figure 6.

The only option that applies to startup for both devices is "Device Configuration Mode". This selection chooses how device registers are populated. "Compressed" and "uncompressed" modes both use the CPU to populate configuration registers, and store the data in flash. The "compressed" mode optimizes for flash space consumption rather than startup time. The difference in performance between these two modes is extremely modest. The "DMA" mode uses DMA to populate the registers. In "DMA" mode, the CPU's execution is blocked until DMA configuration is completed. In the PSoC 3, DMA population takes much less time than 8051 CPU population, as DMA transfers data from flash to registers at a much higher rate. However, in PSoC 5LP, the Cortex-M3 can populate registers more than twice as fast as DMA.

PSoC 3 and PSoC 5LP Design Wide Resources

The "Clear SRAM…" option determines whether or not SRAM should be cleared upon reset. The "Enable Fast IMO …" option selects between the two available NVL-selected IMO speeds: 12 MHz and 48 MHz nominal. The 12-MHz and 48-MHz options are referred to in the device datasheet as "slow boot mode" and "fast boot mode", respectively.

Note The "Enable Fast IMO During Startup" option is not available in some parts due to maximum operating frequency ratings.

Figure 5. System Tab of PSoC 3 DWR in PSoC Creator

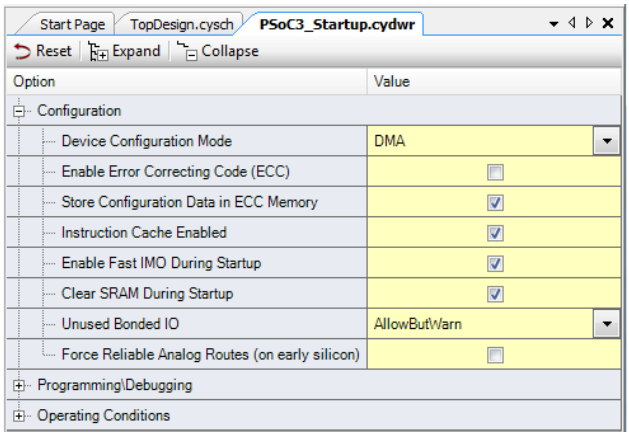
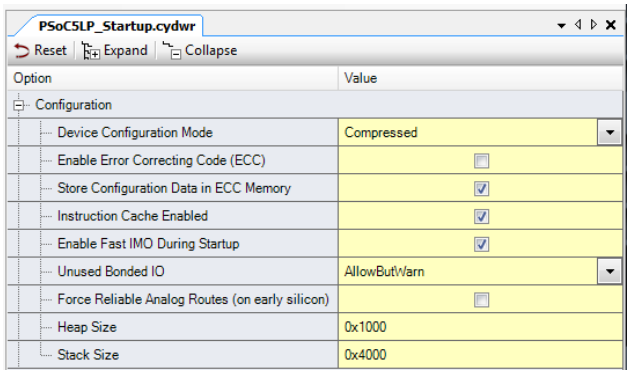


Figure 6. System Tab of PSoC 5LP DWR in PSoC Creator



Modifying Startup Code

PSoC 3 and PSoC 5LP startup code is modified depending upon the configuration of the PSoC Creator project. The code is re-generated every time a change is made to the PSoC Creator schematic or design wide resources. This means that any changes made to the CPU-specific source code and *CyFitter_cfg.c* files can easily be lost.

To edit the startup code, and have changes reflected in the HEX file that is generated for device programming, the files must only be edited when there is no need to perform a “generate” operation. This means that the configuration in automatically-generated source files must match the configuration of the project’s design wide resources and schematic. This may be ensured by first making schematic and DWR changes, performing a “clean and build” and only subsequently editing generated source files. At this point, the project may be built, and the resulting firmware reflects the edits. Every time a “clean and build” is performed, modifications to automatically generated source files are lost.

To attain a better understanding of the time spent performing various tasks during startup, pin toggles may be added in the firmware to indicate at what time certain operations are carried out. Pin toggles may be added in C and 8051 and Cortex-M3 assembly using the code shown in Code 1, Code 2, Code 3, and Code 4. These may be modified to use the desired ports and pins by changing the constants for port drive mode and data registers, and the constants for pin masks.

Note These pin toggles take some time to execute, and when calculating total startup time, they should be accounted for. Register and memory usage should also be considered.

Code 1. PSoC 3 and PSoC 5LP Pin 0.0 Toggle in C

```
/* Drive pin 0.0 to logic high        */
/* Set port-wide drive mode to strong */
CY_SET_REG8(CYDEV_IO_PRT_PRT0_PRT,0x0C);
/* Set pin 0.0 data register to 0     */
CY_SET_REG8(CYDEV_IO_PRT_PRT0_DR,0x01);
```

Code 2. PSoC 3 Pin 0.0 Toggle in Assembly

```
; Set port 0 drive mode
; Set A to strong drive mode
MOV   A, #0CH
; Load register address (PRT0_PRT)
MOV   DPTR, #510AH
; Load register value
MOVX  @DPTR, A
; Set p0.0 high
; Set A to desired register value(pin 0)
MOV   A, #01H
; Load register address (PRT0_DR)
MOV   DPTR, #5100H
; Load register value
MOVX  @DPTR, A
```

Code 3. PSoC 5LP Pin 0.0 Toggle in GCC Assembly

```
/* Load register address (PRT0_PRT) */
"ldr r2, =0x4000510A\n"
/* Set drive mode                   */
"mov r3, 0x0C\n"
/* Store drive mode in register     */
"str r3, [r2]\n"
/* Load register address (PRT0_DR)  */
"ldr r2, =0x40005100\n"
/* Set pin data                     */
"mov r3, 0x01\n"
/* Store pin data in register       */
"str r3, [r2]\n"
```

Code 4. PSoC 5LP Pin 0.0 Toggle in MDK Assembly

```
/* Load register address (PRT0_PRT) */
ldr r2, =0x4000510A
/* Set pin drive mode               */
mov r3, 0x0C
/* Store drive mode in register     */
str r3, [r2]
/* Load register address (PRT0_PRT) */
ldr r2, =0x40005100
/* Set drive mode                   */
mov r3, 0x01
/* Store drive mode in register     */
str r3, [r2]
```

Startup Time Optimization Tips

In some designs, a low startup time is essential. In these designs, there are a number of steps that can be taken to reduce startup time. It is worth noting that some of these steps increase current consumption.

Increase Trimmed IMO Frequency

Running the fully-trimmed IMO at a higher frequency also improves startup time. Because most startup occurs under partially-trimmed IMO, the benefits are not as significant as changing the partially trimmed IMO frequency. As with increasing the partially trimmed IMO frequency, this change increases current consumption.

Increase Fast-start IMO Frequency

Running the Fast-start IMO at 48 MHz instead of 12 MHz speeds up most portions of startup by a factor of 4. Much of startup is CPU or DMA limited, and these two resources operate at the speed of the IMO. The downside to increasing the speed of the Fast-start IMO is that device current consumption increases. The fast-start IMO frequency can be modified in the PSoC Creator Design Wide Resources, as described in the PSoC 3 and PSoC 5LP Design Wide Resources section.

Faster Power Supply Ramp

While the power supply ramp is not normally considered part of microcontroller startup, it does block the beginning of the startup procedure. The embedded designer should consider the possibility of increasing the speed of the VDD ramps, if it is taking a significant amount of time compared to device startup. Power supply ramp rates are discussed further in AN61290 - PSoC® 3 and PSoC 5LP Hardware Design Considerations.

Note PSoC 3 and PSoC 5LP have maximum and minimum power supply ramp ratings. They are listed as “Svdd” in the device datasheet.

Use CPU for Register Population in PSoC 5LP

In PSoC 5LP, the Cortex-M3 CPU is much faster than DMA at populating device registers. The CPU populates registers over twice as fast as DMA.

Use DMA for Register Population in PSoC 3

In PSoC 3, DMA is much faster than the 8051 CPU at populating device registers. Gains depend heavily upon the configuration of the part, but switching from CPU to DMA population may save on the order of 1-20 ms.

Do Not Clear SRAM in PSoC 3

Clearing SRAM takes approximately 4500 CPU clock cycles in an 8-KB SRAM PSoC 3 part. At the slow IMO speed of 12 MHz, this can be almost 400 µs. At 48 MHz, it can take 100 µs. As long as variables are initialized properly in code, not clearing SRAM has no effect upon firmware operation.

Conclusion

AN60616 has described the PSoC 3 and PSoC 5LP startup procedure and how it can be altered to better suit a variety of applications.

About the Author

Name:	Max Kingsbury
Title:	Applications Engineer Senior
Background:	Max holds a bachelors degree in electrical engineering from Washington State University. He specializes in technical composition and debugging embedded designs. In his spare time, he enjoys running, photography, and birding.

Document History

Document Title: AN60616 - PSoC® 3 and PSoC 5LP Startup Procedure

Document Number: 001-60616

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2901405	MAXK	03/30/2010	New application note.
*A	3207332	MAXK	03/27/2011	Updated details and readability. Updated PSoC Creator version. Updated abstract.
*B	3684897	MAXK	07/16/2012	Added PSoC 5. Added clock startup section. Updated for PSoC Creator Version 2.1.
*C	3700192	CFT	08/01/2012	Updated title in Document History section to match with the spec title.
*D	3818185	MAXK	11/21/2012	Updated for PSoC 5LP.
*E	4339856	MEH	04/10/2014	Fixed paragraph alignment on pages 1, 2, and 3 Minor grammatical changes.
*F	4693295	MEH	03/19/2015	Clarified that pin behavior is the same for both PSoC 3 and PSoC 5LP Clarified the LVI and HVI status bits are cleared after reset.
*G	5741437	AESATMP9	05/18/2017	Updated logo and copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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

©Cypress Semiconductor Corporation, 2010-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.