AN73617

PSoC® Designer Boot Process, from Reset to Main

Author: Chris Keeser
Associated Project: No
Associated Part Family: 29x66, 28xxx, 27x43, 24x94, 24x23A, 22x45, 21x45, 21x43, 21x23, 20xxx
Software Version: PSoC® Designer™ 5.4
Related Application Notes: For a complete list of the application notes, click here.

AN73617 describes the PSoC® Designer initialization process of PSoC 1, from the release of reset to the start of C code execution in main. The application note includes instructions on how to modify the interrupt vector table to execute custom interrupt service routines.

Contents

Introduction 2
Boot Process Overview 2
Boot Process Details 4
Reset (Four Scenarios) [1] 4
IPOR (Device Power-up) 4
PPOR (Power Brownout Detect) 4
XRES (External Reset) 4
WDR (Watchdog Reset) 5
SROM Boot [2] 5
Execution Begins at 0x0000 [3] 5
Interrupt Vector Table 5
Start of Execution 5
Switch Mode Pump (SMP) Initialization [4] 5
Related Configuration Options 6
FLASH Bank Initialization [5] 6
Watchdog Enable [6] 6
Related Configuration Options 6
Unlock / Lock ECO Operation [7] 6
Related Configuration Options 6
Trim the IMO and VBG Based on the Selected Operating Condition and Enable AGndBypass [8] 7
Related Configuration Options 7
Initialize the 32-kHz Xtal [9] 7
Related Configuration Options 7
PLL Lock [10] 7
Related Configuration Options 8
External Clock Pin Configuration [11] 8
Related Configuration Options 8
Close the CT Leakage Path [12] 8
Large Memory Model Initialization [13] 8
Related Configuration Options 8
Load the Base Device Configuration [14] 9
Related Configuration Options 9
Initialize the C Run-Time Environment [15] 9
Voltage Stabilization for the SMP [16] 9
Related Configuration Options 9
Set the Power-On Reset (POR) Level [17] 10
Related Configuration Options 10
Wrap Up and Invoke Main [18] [19] 10
Glossary of Terms 11
Summary 12
Related Application Notes 12
Appendix: Example *Boot.asm* for CY8C29xxx Devices 13
Worldwide Sales and Design Support 27

Introduction

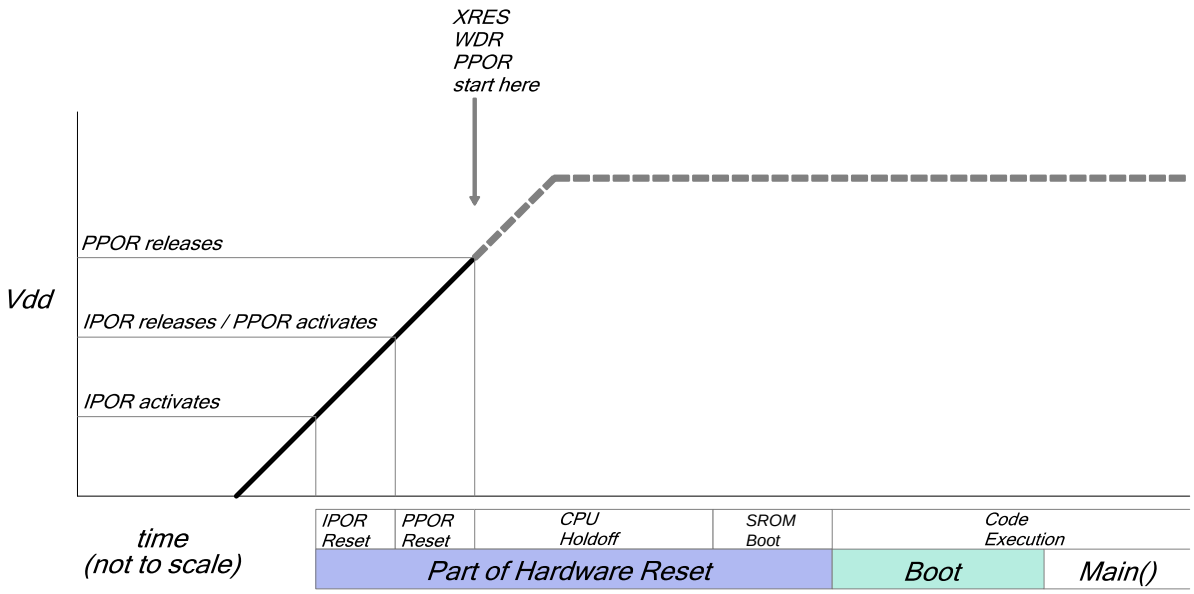
Because PSoC 1 offers billions of configuration options, the PSoC device must be properly initialized after reset to fulfill its potential. The PSoC Designer boot process performs these necessary initialization tasks before entering main to provide an optimal working environment. This application note explains the device initialization procedure, but this document is not required reading for users of PSoC 1 or the PSoC Designer integrated development environment (IDE). The information is for users seeking a deeper understanding of PSoC and PSoC Designer initialization before main is executed.

Most of this application note applies to all PSoC 1 device families, with allowances for small variations between device capabilities. Examples of such differences include devices with only 256 bytes of RAM, devices with USB, or devices without a switch mode pump. Boot time cannot be specified because it varies depending on the device family and PSoC Designer project configuration.

Boot Process Overview

Figure 1 shows the sequence of events before the boot process is initiated. The reset system holds the device in reset while the power supply ramps up (IPOR Reset, PPOR Reset). After the supply is stable, the CPU is held in reset for an additional period depending on the reset source (CPU Holdoff). After the CPU reset is released, the device performs an initial hardware controlled trim of vital systems (SROM Boot), and then instructs the CPU to start executing code.

Figure 1. Simple Timing Diagram of Reset Behavior



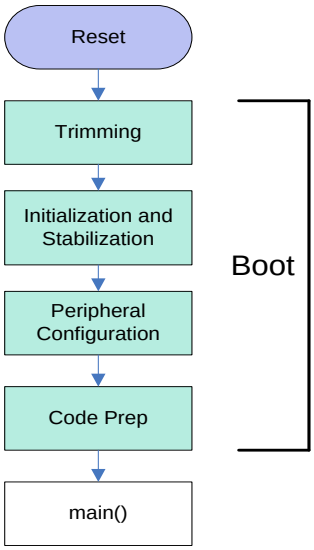
Once reset is released, the boot process can begin. For a simplified flow chart of the process, see Figure 2.

From a high level, the boot process must perform these key tasks:

- Trim key features, such as the IMO and VBG for the operating voltage set in Global Resources.
- Wait for peripherals, such as the ECO, to stabilize after initialization
- Configure analog and digital hardware, such as Amplifiers, ADCs, PWMs and UARTs, based on your design settings
- Prepare the device to execute C or assembly code

The rest of the application note describes the boot process in more detail, elaborating on each step, and directing you to related documentation.

Figure 2. Simplified Overview of Key Tasks Performed during Boot



Boot Process Details

This section divides the boot process into steps in the order of execution. A numbered list details each step of the boot code. The numbers of each step correspond to information later in the app note, and to locations in an example *boot.asm* file in the appendix that gives more details of the code. An asterisk (*) indicates an optional step and may be skipped depending on the configuration options selected in PSoC Designer.

1. Device reset, which has four possible sources, is released. The device delays for a variable time after the device reset is released and before CPU reset is released.
2. The hardwired supervisory ROM (SROM) code automatically loads trim for critical systems.
3. The CPU begins to execute code beginning at address 0x0000.
4. *The SMP is enabled and the desired output voltage is set.
5. The FLASH bank initialization is performed
6. *The watchdog timer is enabled.
7. The option to allow 32-kHz crystal for sleep timer is applied.
8. *Specific trims are loaded for the IMO and the VBG based on the operating parameters specified in PSoC Designer. The AGND bypass option is applied
9. *The 32 kHz crystal is initialized, with a 1-second wait.
10. *The PLL option is initialized, with a 16-ms wait.
11. *If the external bus clock option is enabled, initialization of external clock pin occurs.
12. The continuous time (CT) analog block leakage path is removed.
13. *Preparation is made for the Large Memory Model.
14. The configuration registers are loaded with the configuration data based on the global settings and placed user modules.
15. C environment and variable initialization occurs.
16. *SMP voltage stabilization. Before enabling the PPOR, the PSoC waits for 2 ms to ensure that the SMP has reached the desired final voltage.
17. The PPOR voltage is set to the desired value.
18. Set the final values for the sleep timer source, sleep timer interval, PLL mode, and CPU clock speed.
19. Jump to main.

Small variations exist among the devices in the PSoC 1 family. Some devices may have a section removed or added (that is, no SMP code for devices that do not have an SMP, or added USB config for devices with USB). However, for all devices in the PSoC 1 family, the flow and intent will follow the outline above.

Reset (Four Scenarios) [1]

IPOR (Device Power-up)

When the device first receives power, the imprecise power-on reset (IPOR) holds the system in reset until the voltage reaches a level (~2.2 volts) at which the precise power-on reset (PPOR) can function[1]. The PPOR then holds the system in reset until the voltage reaches a safe operating value[2], ~2.9 volts[3].

After the PPOR is released, the CPU is kept in reset for 512 untrimmed ILO clocks[4]. The worst-case untrimmed ILO frequency is 5 kHz[5]. As a result, the CPU can be held in reset for a maximum of ~100 ms before it begins the SROM boot process[6].

During this hold-off time, pins P1[0] and P1[1] change from the default state of Hi-Z[7]. These pins will toggle between strong high (1) and resistive low (0). The pin toggles are used to synchronize a device programmer with the PSoC immediately after a reset event.

After the CPU reset has been deasserted, the SROM boot function begins to execute (skip to SROM Boot).

PPOR (Power Brownout Detect)

If the device is already running and a voltage dip causes a PPOR, when the PPOR releases[3], the CPU is held in reset for 1 untrimmed ILO clock[4] before beginning the SROM boot[6].

After the CPU reset has been deasserted, the SROM boot function begins to execute (skip to SROM Boot).

XRES (External Reset)

When the XRES pin is deasserted, the CPU is held in reset for 8 untrimmed ILO clocks[4] before beginning the SROM boot process[6].

After the CPU reset has been deasserted, the SROM boot function begins to execute (skip to SROM Boot).

---

[1] TRM Section 29.4.1 Power On Reset

[2] TRM Section 31.1 POR and LVD

[3] Refer to the device datasheet, DC Electrical Characteristics, DC POR specification for your device's specific PPOR release voltage

[4] TRM section 29.4.4 Reset Details

[5] Refer to the device datasheet, AC Electrical Characteristics, F32k_u for your device's specific minimum untrimmed ILO frequency

[6] TRM Section 3.1.2.1 SWBootReset Function

[7] TRM section 29.2.1 GPIO behavior on Power Up

WDR (Watchdog Reset)

When the watchdog reset occurs[8], the system reset asserts for 1 ILO clock, and then the CPU is held in reset

for 1 untrimmed ILO clock[4] before beginning the SROM boot[6].

After the CPU reset has been deasserted, the SROM boot function begins to execute (skip to SROM Boot).

SROM Boot [2]

When the CPU reset is deasserted, the SROM function SWBootReset[6] executes, loading trim for the ECO, ILO, IMO (5 volts), and VBG (5 volts). RAM locations in page zero also are cleared based on table 3-4 (SRAM Map Post SWBootReset) in the TRM under section 3.1.2.1 SWBootReset Function. The execution time of the SROM boot is ~2.2 ms, after which the CPU begins to execute code beginning at address 0x0000[4].

Execution Begins at 0x0000 [3]

This is the point where PSoC Designer takes over and begins the next phase of the boot. Code for this process resides in *boot.asm*.

Note In PSoC Designer, the file *boot.tpl* (boot template) is used to generate the *boot.asm* file whenever the project is generated. Since boot.asm is regenerated every time the PSoC Designer project is generated, any changes that you want to preserve in the *boot.asm* file need to be made in the *boot.tpl* file. If you make changes directly to *boot.asm*, your changes will be lost the next time you generate your application.

At the beginning of the executable portion of boot.asm, the linker places a "jump to __Start" instruction at instruction address `0x0000`. The jump is needed because the interrupt vector table begins at instruction address `0x0004`. If you did not jump away, you would have only 4 bytes of memory from address `0x0000` to `0x0003` before you need to worry about overlap with the interrupt vector table. Once you jump away from the interrupt vector table, you have as much room as you need for boot code.

Interrupt Vector Table

The interrupt vector table, located at the beginning of flash memory[9], provides 4 instruction bytes for each interrupt vector. This table[10] allows different interrupt sources to vector directly to different interrupt service routines.

In the majority of designs, you do not have to modify the interrupt table directly. Typically, user modules in your design change the table for you. However, if you want to write your own interrupt handler, modify the interrupt vector table to `ljmp` to your custom interrupt handler code. In the ImageCraft compiler, a `pragma` keyword identifies a function as an interrupt handler[11]. Here is an example of declaring a function as an interrupt handler:

```
#pragma interrupt_handler FooHandler
…
void FooHandler()
{
    …
}
```

In *boot.tpl*, you would add an `ljmp` to `_FooHandler` (note the underscore) at the appropriate interrupt vector location. Let's say you intend to handle the sleep timer interrupt with your FooHandler:

```
org  64h ;Sleep Timer Interrupt Vector
    ljmp _FooHandler
reti
```

The compiler automatically adds a `reti` at the end of your ISR function (FooHandler in this example) and handles the stack appropriately

Start of Execution

After the interrupt vector table, the linker is told to locate "__Start" at an address away from the interrupt vector table. At this point, device initialization begins.

Switch Mode Pump (SMP) Initialization [4]

Out of reset, the SMP is enabled by default and maintains a voltage of ~3 volts. If you decide not to use the SMP through the switch mode pump configuration (Figure 3), then the SMP is disabled here. If you use the SMP, it remains enabled, and your desired output voltage (Figure 3) is set here. The PPOR voltage is left at its lowest trip voltage to allow time for the SMP to stabilize.

[8] TRM Section 12 Sleep and Watchdog

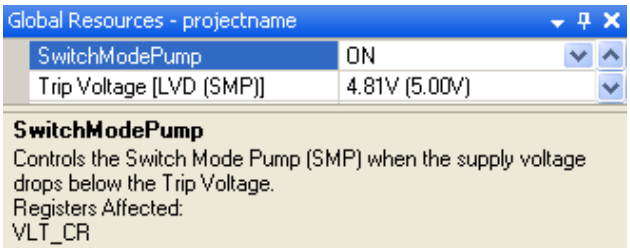
[9] Refer to your device's *boot.tpl* to determine the exact addresses and the number of interrupt vectors for your device.

[10] TRM Section 5 Interrupt Controller

[11] C Language Compiler User Guide section 6.6 Interrupts

Related Configuration Options

Figure 3. Switch Mode Pump Enable and Drive Voltage Setting



FLASH Bank Initialization [12] [5]

For devices with more than one flash bank, the first read from a flash bank other than bank 0 could return corrupt the data. The initialization performs a dummy read on all the flash banks and waits at least 5 µs for the system to stabilize before trying to make a valid read from a bank other than bank 0.

The boot.asm code does not explicitly wait for 5 µs when initializing the flash banks, because the rest of *boot.asm* takes significantly longer than 5 µs to execute, and all of the instructions for boot.asm reside in bank 0 of the flash.

Watchdog Enable [6]

If you enable the watchdog timer (Figure 4), it will be initialized here. The watchdog timer generates a system reset if it is not "fed" before three rollover events of the sleep timer[13]. The watchdog period is determined by setting the sleep timer period (Figure 5). To "feed" the watchdog, write any value to the `RES_WDT` register. If you write a value of `0x38` to the `RES_WDT` register, it will reset both the sleep timer and the watchdog timer[14] in one instruction.

Related Configuration Options

Figure 4. Watchdog Enable

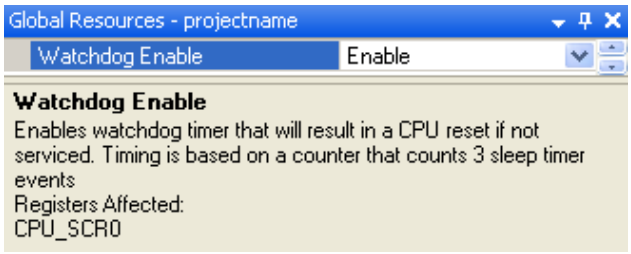
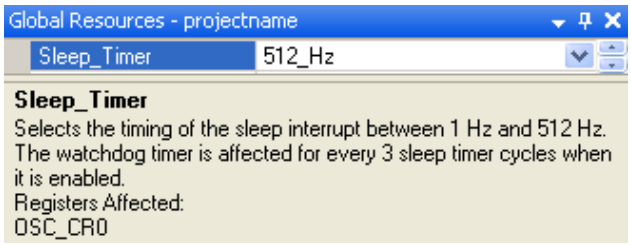


Figure 5. Sleep Timer Configuration

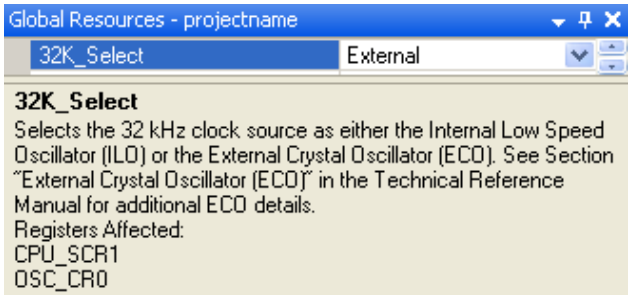


Unlock / Lock ECO Operation [7]

If the external 32-kHz option is enabled (Figure 6), this section will write a 1 to the `CPU_SCR1_ECO_ALLOWED` bit in the `CPU_SCR1` register. If the Internal 32-kHz option is selected, a 0 will be written to this bit. Writing to this bit with either a 1 or a 0 will lock out any further writes to this bit[15]. Note that this code does not enable the ECO; the code merely indicates to the PSoC if an external 32-kHz crystal can be used in the system. The code is written early in the boot process to prevent a spurious command from accidentally enabling or disabling this feature.

Related Configuration Options

Figure 6. External 32-kHz Crystal Enable



[12] This erratum applies to the CY8C29x66 parts. Check your specific device's errata for workarounds related to your device, if applicable.

[13] TRM Section 12.3.5 OSC_CR0 Register table 12-1 Seep Interval Selections

[14] TRM section 12.3.2 RES_WDT Register

[15] TRM Section 10.3.1 CPU_SCR1 Register

Trim the IMO and VBG Based on the Selected Operating Condition and Enable AGndBypass [8]

After Reset, the SROM automatically trims the IMO and VBG for 5-volt / 24-MHz operation. This code trims the device based on your specific settings (Figure 7) and sets the AGNDBYP bit[16] if it is enabled (Figure 8).

Here is a general overview of the device trimming procedure:

1. The supervisory command to read the manufacturing trim tables is issued to read one of four, 8-byte trim tables[17].
2. The 8 bytes of the specified trim table are loaded into RAM.
3. The relevant trim is read from RAM and stored in the appropriate trim register.

The bootup code adds logic for trimming the IMO and VBG and for setting the AGNDBYP bit based on your settings in the Global Resources if those settings differ from those in the default.

Related Configuration Options

Figure 7. System Voltage and IMO Speed Configuration

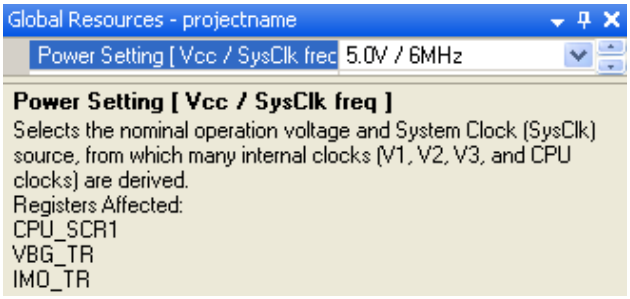
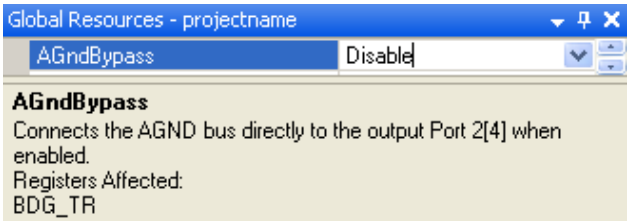


Figure 8. Analog Ground Bypass Enable Option



[16] TRM Section 13.3.41 BDG_TR, AN2219 – PsoC 1 Selecting Analog Ground and Reference

[17] TRM Section 3.1.2.6, TableRead Function

Initialize the 32-kHz Xtal [9]

If you enable the 32-kHz crystal operation (see Figure 9), then the crystal needs about 1 second[18] to stabilize before you can us it effectively.

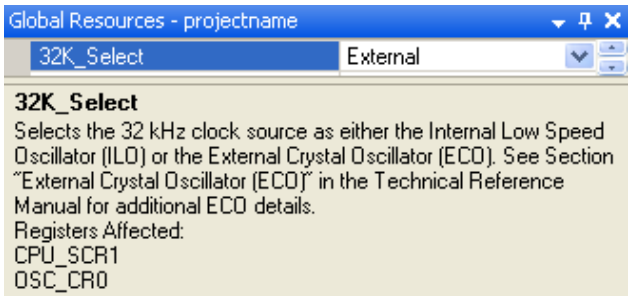
This code is responsible for enabling the 32-kHz Xtal and waiting for 1 second to ensure the crystal has started up properly. You can bypass the 1-second delay by changing the WAIT_FOR_32K equate at the beginning of the boot.tpl file from '1' to '0'. However, that change is not recommended unless you plan to initialize the crystal yourself later in the main code. The 1-second delay blocks all code execution until that time has elapsed. Therefore this feature significantly delays the startup of your device.

Related Configuration Options

Near the beginning of *Boot.tpl*:

```
WAIT_FOR_32K: equ 1
```

Figure 9. External 32 kHz Crystal Enable



PLL Lock [10]

If you enable the PLL (Figure 10), it needs about 16 ms to stabilize before it can be used to drive the system clock.

This code is responsible for enabling the PLL and waiting for 16 ms to allow the PLL to stabilize. The external 32-kHz Xtal must be enabled (Figure 11) for the PLL to lock on to, and you must leave WAIT_FOR_32K at its default of '1' to ensure that the 32-kHz XTAL is stable before the PLL tries to lock.

Because the PLL requires that the 32-kHz Xtal is stable before enabling the PLL, you will generate an error if you bypass the WAIT_FOR_32K delay and have the PLL enabled.

[18] Refer to the device datasheet, AC Electrical Characteristics, Toscacc for your device's specific maximum 32-kHz Xtal startup time, if applicable.

Related Configuration Options

Near the beginning of *Boot.asm*:

```
WAIT_FOR_32K: equ 1
```

Figure 10. PLL Enable

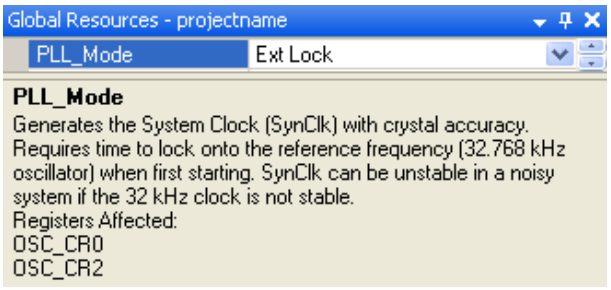
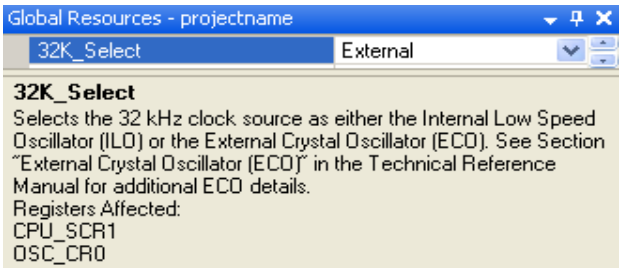


Figure 11. External 32-kHz Crystal Enable



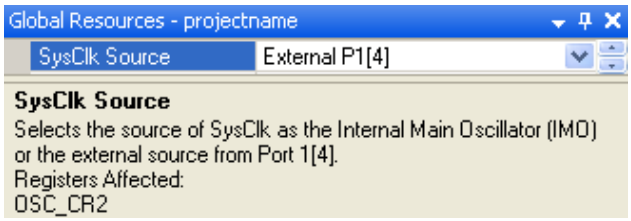
External Clock Pin Configuration [11]

Out of Reset, the default drive mode for every GPIO is Analog Hi-Z[19]. If the system clock (SYSCLK) is set to be sourced from an external signal, then you must change the drive mode for pin 1[4] from the default Analog Hi-Z to a digital input (Digital Hi-Z). This code configures the drive mode of pin 1[4] to be a digital input (Digital Hi-Z mode). At this point, the PSoC is still running off the internal clock source (IMO or PLL). The switch to the external clock source occurs in the "load base device configuration" section (the `LoadConfigInit` call). Pin 1[4] is configured at this point in the boot process because the switch from internal to external clock occurs before the pins are configured in the `LoadConfigInit` function.

[19] TRM Sections 13.3.1 PRTxDM0, 13.3.2 PRTxDM1, 13.2.4 PRTxDM2. Reset state of these registers. Table 6-1 under Section 6, General Purpose IO (GPIO)

Related Configuration Options

Figure 12. External Clock Enable



Close the CT Leakage Path [12]

This code resets a high-resistance path between V_{dd} and the bottom of a resistor matrix for adjacent CT blocks.

After Reset, the default configuration of unused CT blocks could affect the performance of adjacent CT blocks. Specifically, a high-resistance leakage path goes from V_{dd} to the bottom of the resistor matrix for the adjacent CT blocks. This code eliminates that potential leakage path.

Large Memory Model Initialization [13]

Some PSoC devices have more than one page of RAM. Each RAM page has 256 bytes and can be used to store variables, static data, and the processor's stack. Many instructions in the m8c can operate on multiple RAM pages. These instructions use page pointer registers to indicate which RAM page they will use. For example, the stack page pointer (`STK_PP`) sets the page on which the stack operations occur. PSoC has a variety of paging modes that affect how page pointer registers work. This section of code initializes those page pointer registers based on your chosen configuration (Figure 13,

Figure 14) to allow the code to take advantage of all the RAM in your device.

For more information on the large memory model and how it can affect your code, refer to the C users guide and the Assembly users guide.

Related Configuration Options

Figure 13. Menu Location for Enabling / Disabling the LMM

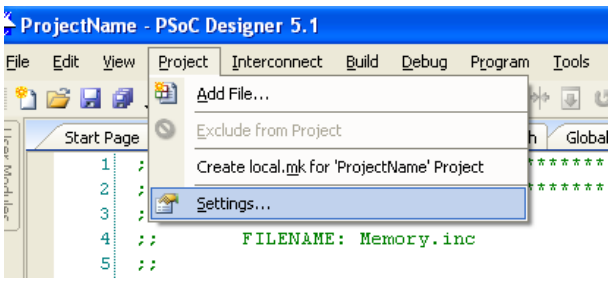
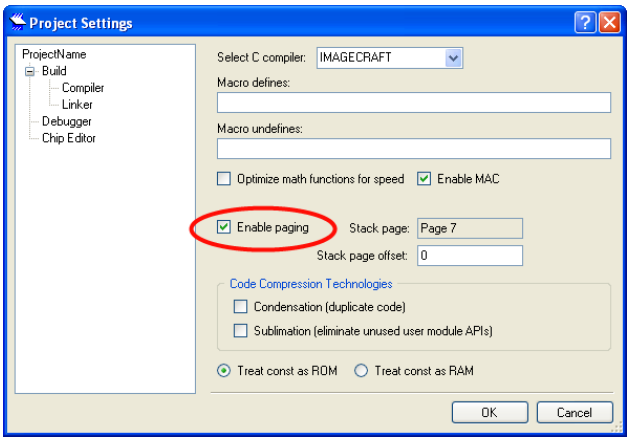


Figure 14. RAM Paging Enable



Load the Base Device Configuration [14]

This code loads all of the configuration registers with their initial values. The values are based on the placed user modules and global settings (for example, GPIO drive modes, initial states, digital and analog routing, user module configuration, and clock divider values).

All of the configuration information is stored in the *PSoCConfigTBL.asm* file. The data can be stored in one of two ways. The first method uses a flash-efficient table of addresses and values that are unpacked and written into the corresponding registers by `LoadConfigInit`. The second method, a faster direct-write table, is composed of individual `mov` instructions for each register. To choose one of these two methods, refer to Figure 15 and Figure 16.

Related Configuration Options

Figure 15. Menu Location for Configuring Device Configuration Data Storage

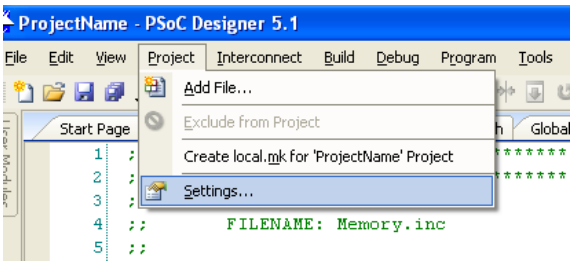
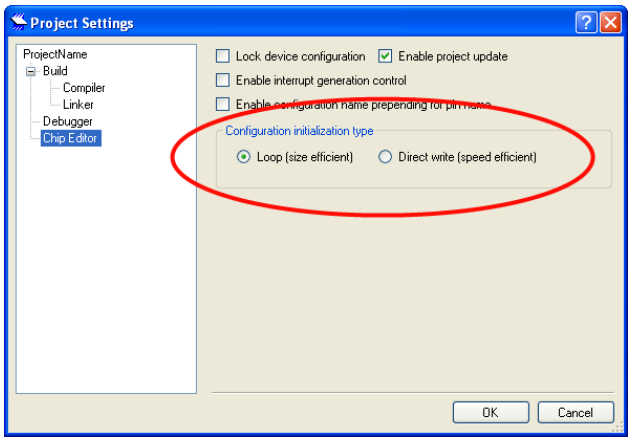


Figure 16. Device Configuration Storage Method



Initialize the C Run-Time Environment [15]

This code sets up and initializes variables in the C environment.

Voltage Stabilization for the SMP [16]

This code is responsible for waiting 2 ms to allow the SMP to settle to 5 volts before setting the PPOR level.

Related Configuration Options

Figure 17. System Voltage Setting

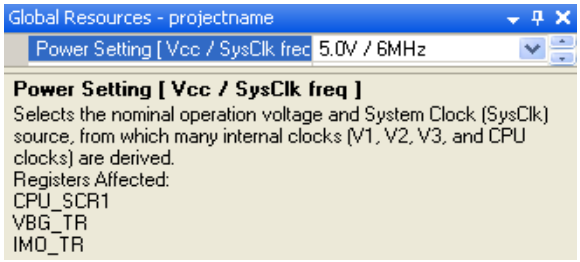
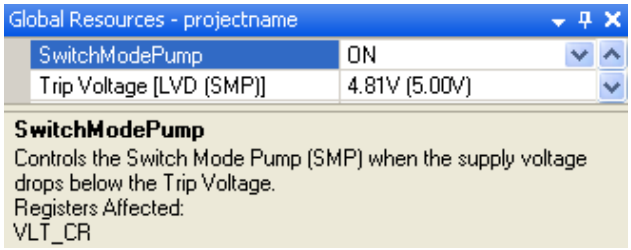


Figure 18. PPOR Voltage Level

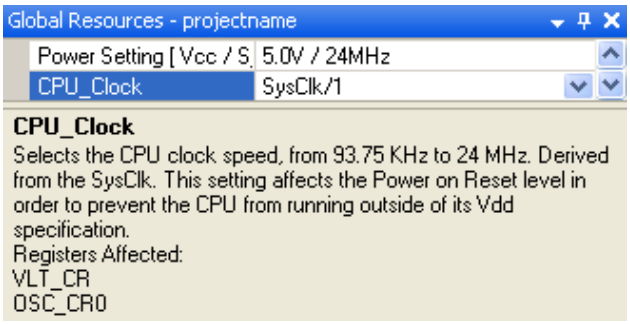


Set the Power-On Reset (POR) Level [17]

In this code, the PPOR level is set to its highest voltage to protect operation of the m8c core. This protection is only required if you set the device to run at 5 volts with the 24-MHz IMO and SysClk is set to run at SysClk/1 (24 MHz) (Figure 19). The m8c core cannot operate at 24 MHz if the voltage dips below 5 V[20].

Related Configuration Options

Figure 19. System Voltage, IMO and SysClk Setting



Wrap Up and Invoke Main [18] [19]

This step is the final one before main is called. Because the sleep interrupt was used as a timer for certain boot tasks, the sleep timer interrupt mask is cleared. Then, your final selections for CPU speed (Figure 20) and sleep timer period (Figure 21) are written. All pending interrupts are cleared, and, finally, the jump to main is executed.

Related Configuration Options

Figure 20. SysClk Setting

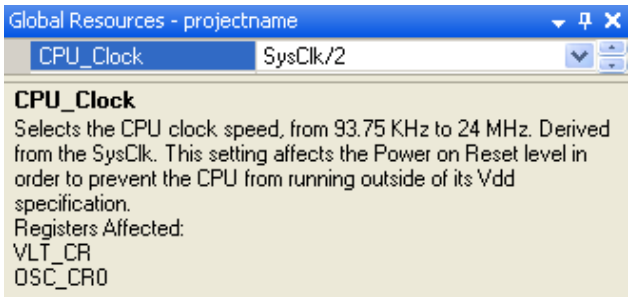
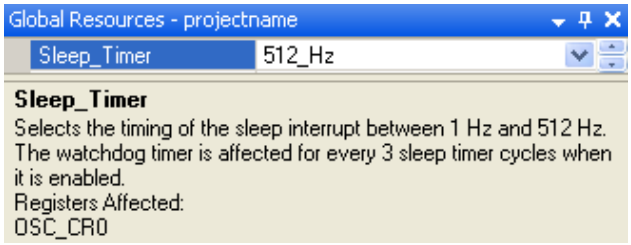


Figure 21. Sleep Timer Period



[20] Device Datasheet, Electrical Specifications, Figure 9, Voltage versus CPU Frequency

Glossary of Terms

- **AGND**: Analog Ground is an internal voltage used as a ground reference for analog resources.
- **CT Blocks** (continuous time analog blocks): refers analog resources with configuration options for continuous time operation.
- **ECO** (external crystal oscillator): a different term for XTAL.
- **ILO (**internal low-speed oscillator): an internal low-power and low-accuracy 32-kHz oscillator.
- **IMO** (internal main oscillator): an internal high-frequency oscillator for clocking the entire device.
- **IPOR (**imprecise power-on reset): holds the device in reset from power up until the Precise Power on Reset is functional (~2.2 volts).
- **LMM (**Large Memory Model): a RAM paging structure for devices with more than 256 bytes of RAM.
- **PLL (**phase-locked loop): an internal frequency multiplier that can drive the device clock if an external 32-kHz crystal is used.
- **PPOR (**precision power-on reset): resets the device if the voltage falls below an adjustable threshold.
- **Sleep Timer**: a timer driven from either the XTAL or the ILO.
- **SMP (**switch mode pump): a voltage boost circuit that can drive the PSoC from sources as low as 1 volt.
- **SROM** (Supervisory ROM): dedicated functionality built into the PSoC device to perform key hardware-related tasks.
- **VBG** (Bandgap Voltage Reference): an internal temperature-compensated voltage reference.
- **WDR** (watchdog reset): generates a device reset if the watchdog timer is not reset within a specific time.
- **WDT** (watchdog timer): responsible for generating a WDR.
- **XRES (**external reset): resets the device when the signal on the XRES pin is driven to a logic high.
- **XTAL (**external crystal): an optional external 32-kHz crystal.

Summary

From the moment reset is released to the beginning of main(), the PSoC boot process automatically handles a variety of initialization and configuration tasks to simplify programming.

Related Application Notes

- Technical Reference Manual - TRM
- C Language Compiler User Guide
- M8C Assembly guide
- CY8C29x66 Device Datasheet

About the Author

Name:	Christopher Keeser
Title:	Applications Engineer Staff
Contact:	kees@cypress.com

Appendix: Example *Boot.asm* for CY8C29xxx Devices

```
;  Generated by PSoC Designer 5.1.2309
;
;@Id: boot.tpl#897 @
;=============================================================================
;  FILENAME:   boot.asm
;  Version:    4.21
;
;  DESCRIPTION:
;  M8C Boot Code for CY8C29xxx microcontroller family.
;
;  Copyright (c) Cypress Semiconductor 2011. All Rights Reserved.
;
; NOTES:
; PSoC Designer's Device Editor uses a template file, BOOT.TPL, located in
; the project's root directory to create BOOT.ASM. Any changes made to
; BOOT.ASM will be overwritten every time the project is generated; therefore,
; changes should be made to BOOT.TPL, not BOOT.ASM. Care must be taken when
; modifying BOOT.TPL so that replacement strings (such as @PROJECT_NAME)
; are not accidentally modified.
;
;=============================================================================

include ".\lib\GlobalParams.inc" ;File generated by PSoC Designer (Project dependent)
include "m8c.inc"                ;Part specific file
include "m8ssc.inc"              ;Part specific file
include "memory.inc"                    ;File generated by PSoC Designer (Project dependent)

;-----------------------------------
; Export Declarations
;-----------------------------------

export __Start
IF     (TOOLCHAIN & HITECH)
ELSE
export __bss_start
export __data_start
export __idata_start
export __func_lit_start
export __text_start
ENDIF
export  _bGetPowerSetting
export   bGetPowerSetting


;-----------------------------------
; Optimization flags
;-----------------------------------
;
; To change the value of these flags, modify the file boot.tpl, not
; boot.asm. See the notes in the banner comment at the beginning of
; this file.

; Optimization for Assembly language (only) projects and C-language projects
; that do not depend on the C compiler to initialize the values of RAM variables.
;   Set to 1: Support for C Run-time Environment initialization
;   Set to 0: Support for C not included. Faster start up, smaller code space.
;
```

```
IF     (TOOLCHAIN & HITECH)
; The C compiler will customize the startup code - it's not required here

C_LANGUAGE_SUPPORT:              equ 0
ELSE
C_LANGUAGE_SUPPORT:              equ 1
ENDIF


; The following equate is required for proper operation. Resetting its value
; is discouraged. WAIT_FOR_32K is effective only if the crystal oscillator is
; selected. If the designer chooses not to wait, then stabilization of the ECO
; and PLL_Lock must take place within user code. See the family data sheet for
; the requirements of starting the ECO and PLL lock mode.
;
;   Set to 1: Wait for XTAL (and PLL if selected) to stabilize before
;                invoking main.
;   Set to 0: Boot code does not wait; clock may not have stabilized by
;               the time code in main starts executing.
;
WAIT_FOR_32K:                    equ 1


; For historical reasons, by default the boot code uses an lcall instruction
; to invoke the user's _main code. If _main executes a return instruction,
; boot provides an infinite loop. By changing the following equate from zero
; to 1, boot's lcall will be replaced by a ljmp instruction, saving two
; bytes on the stack that are otherwise required for the return address. If
; this option is enabled, _main must not return. (Beginning with the 4.2
; release, the C compiler automatically places an infinite loop at the end
; of main, rather than a return instruction.)
;
ENABLE_LJMP_TO_MAIN:             equ 0


;-----------------------------------------------------------------------------
; Interrupt Vector Table
;-----------------------------------------------------------------------------
;
; Interrupt vector table entries are 4 bytes long. Each one contains
; a jump instruction to an ISR (interrupt service routine), although
; very short ISRs could be encoded within the table itself. Normally,
; vector jump targets are modified automatically according to the user
; modules selected. This occurs when the 'Generate Application' opera-
; tion is run, causing PSoC Designer to create boot.asm and the other
; configuration files. If you need to hard-code a vector, update the
; file boot.tpl, not boot.asm. See the banner comment at the beginning
; of this file.
;-----------------------------------------------------------------------------
```

[3]

```
    AREA TOP (ROM, ABS, CON)

    org   0                      ;Reset Interrupt Vector
IF     (TOOLCHAIN & HITECH)
;   jmp   __Start                ;C compiler fills in this vector
ELSE
    jmp   __Start                ;First instruction executed following a Reset
ENDIF
```

[Interrupt Vector Table]

```
    org   04h                   ;Low-Voltage Detect (LVD) Interrupt Vector
    halt                        ;Stop execution if power falls too low

    org   08h                   ;Analog Column 0 Interrupt Vector
    ljmp      _COMP_1_ISR
    reti

    org   0Ch                   ;Analog Column 1 Interrupt Vector
    // call   void_handler
    reti

    org   10h                   ;Analog Column 2 Interrupt Vector
    // call   void_handler
    reti

    org   14h                   ;Analog Column 3 Interrupt Vector
    // call   void_handler
    reti

    org   18h                   ;VC3 Interrupt Vector
    // call   void_handler
    reti

    org   1Ch                   ;GPIO Interrupt Vector
    // call   void_handler
    reti

    org   20h                   ;PSoC Block DBB00 Interrupt Vector
    // call   void_handler
    reti

    org   24h                   ;PSoC Block DBB01 Interrupt Vector
    // call   void_handler
    reti

    org   28h                   ;PSoC Block DCB02 Interrupt Vector
    // call   void_handler
    reti

    org   2Ch                   ;PSoC Block DCB03 Interrupt Vector
    // call   void_handler
    reti

    org   30h                   ;PSoC Block DBB10 Interrupt Vector
    // call   void_handler
    reti

    org   34h                   ;PSoC Block DBB11 Interrupt Vector
    // call   void_handler
    reti

    org   38h                   ;PSoC Block DCB12 Interrupt Vector
    // call   void_handler
    reti

    org   3Ch                   ;PSoC Block DCB13 Interrupt Vector
    // call   void_handler
    reti
```

```
    org   40h                       ;PSoC Block DBB20 Interrupt Vector
    // call  void_handler
    reti

    org   44h                       ;PSoC Block DBB21 Interrupt Vector
    // call  void_handler
    reti

    org   48h                       ;PSoC Block DCB22 Interrupt Vector
    // call  void_handler
    reti

    org   4Ch                       ;PSoC Block DCB23 Interrupt Vector
    // call  void_handler
    reti

    org   50h                       ;PSoC Block DBB30 Interrupt Vector
    // call  void_handler
    reti

    org   54h                       ;PSoC Block DBB31 Interrupt Vector
    // call  void_handler
    reti

    org   58h                       ;PSoC Block DCB32 Interrupt Vector
    // call  void_handler
    reti

    org   5Ch                       ;PSoC Block DCB33 Interrupt Vector
    // call  void_handler
    reti

    org   60h                       ;PSoC I2C Interrupt Vector
    // call  void_handler
    reti

    org   64h                       ;Sleep Timer Interrupt Vector
    // call  void_handler
    reti

;-------------------------------------------------------------------------
;  Start of Execution.
;-------------------------------------------------------------------------
;  The Supervisory ROM SWBootReset function has already completed the
;  calibrate1 process, loading trim values for 5-volt operation.
;

IF     (TOOLCHAIN & HITECH)
       AREA PD_startup(CODE, REL, CON)
ELSE
    org 68h
ENDIF
__Start:
```

[4]

```
    ; initialize SMP values for voltage stabilization, if required,
    ; leaving power-on reset (POR) level at the default (low) level, at
    ; least for now.
    ;
    M8C_SetBank1
```

```
    mov reg[0FAh], 0                            ;Reset flash location
    mov   reg[VLT_CR], SWITCH_MODE_PUMP_JUST | LVD_TBEN_JUST | TRIP_VOLTAGE_JUST
    M8C_SetBank0
```

[5]

```
    ; %53%20%46%46% Apply Erratum 001-05137 workaround
    mov   A, 20h
    romx
    mov   A, 40h
    romx
    mov   A, 60h
    romx
    ; %45%20%46%46% End workaround
```

[6]

```
M8C_ClearWDTAndSleep                        ; Clear WDT before enabling it.
IF ( WATCHDOG_ENABLE )              ; WDT selected in Global Params
    M8C_EnableWatchDog
ENDIF
```

[7]

```
IF ( SELECT_32K )
    or    reg[CPU_SCR1],  CPU_SCR1_ECO_ALLOWED  ; ECO will be used in this project
ELSE
    and   reg[CPU_SCR1], ~CPU_SCR1_ECO_ALLOWED  ; Prevent ECO from being enabled
ENDIF

    ;---------------------------
    ; Set up the Temporary stack
    ;---------------------------
    ; A temporary stack is set up for the SSC instructions.
    ; The real stack start will be assigned later.
    ;
_stack_start:          equ 80h
    mov   A, _stack_start           ; Set top of stack to end of used RAM
    swap  SP, A                     ; This is only temporary if going to LMM
```

[8]

```
    ;-----------------------------------------------
    ; Set Power-related Trim & the AGND Bypass bit.
    ;-----------------------------------------------
M8C_ClearWDTAndSleep ; Clear WDT before enabling it.

IF ( POWER_SETTING & POWER_SET_5V0)             ; *** 5.0-Volt operation   ***
 IF ( POWER_SETTING & POWER_SET_SLOW_IMO)       ; *** 6-MHZ Main Oscillator ***
    or  reg[CPU_SCR1], CPU_SCR1_SLIMO
    M8SSC_Set2TableTrims 2, SSCTBL2_TRIM_IMO_5V_6MHZ, 1, SSCTBL1_TRIM_BGR_5V,
AGND_BYPASS_JUST
 ELSE                                           ; *** 12-MHZ Main Oscillator ***
  IF ( AGND_BYPASS )
    ;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    ; The 5-V trim has already been set, but we need to update the AGNDBYP
    ; bit in the write-only BDG_TR register. Recalculate the register
    ; value using the proper trim values.
    ;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
    M8SSC_SetTableVoltageTrim 1, SSCTBL1_TRIM_BGR_5V, AGND_BYPASS_JUST
  ENDIF
 ENDIF
ENDIF ; 5.0-V Operation

IF ( POWER_SETTING & POWER_SET_3V3)            ; *** 3.3-Volt operation   ***
 IF ( POWER_SETTING & POWER_SET_SLOW_IMO)      ; *** 6-MHZ Main Oscillator ***
    or  reg[CPU_SCR1], CPU_SCR1_SLIMO
    M8SSC_Set2TableTrims 2, SSCTBL2_TRIM_IMO_3V_6MHZ, 1, SSCTBL1_TRIM_BGR_3V,
AGND_BYPASS_JUST
 ELSE                                          ; *** 12-MHZ Main Oscillator ***
    M8SSC_SetTableTrims  1, SSCTBL1_TRIM_IMO_3V_24MHZ, SSCTBL1_TRIM_BGR_3V, AGND_BYPASS_JUST
 ENDIF
ENDIF ; 3.3-Volt Operation

    mov  [bSSC_KEY1],  0           ; Lock out Flash and Supervisory operations
    mov  [bSSC_KEYSP], 0
```

[9]

```
    ;---------------------------------------
    ; Initialize Crystal Oscillator and PLL
    ;---------------------------------------

IF ( SELECT_32K & WAIT_FOR_32K )
    ; If the user has requested the External Crystal Oscillator (ECO), then turn it
    ; on and wait for it to stabilize and the system to switch over to it. The PLL
    ; is left off. Set the SleepTimer period set to 1 sec to time the wait for
    ; the ECO to stabilize.
    ;
    M8C_SetBank1
    mov   reg[OSC_CR0], (SELECT_32K_JUST | OSC_CR0_SLEEP_1Hz | OSC_CR0_CPU_12MHz)
    M8C_SetBank0
    M8C_ClearWDTAndSleep                  ; Reset the sleep timer to get a full second
    or    reg[INT_MSK0], INT_MSK0_SLEEP   ; Enable latching of SleepTimer interrupt
    mov   reg[INT_VC],   0                ; Clear all pending interrupts
.WaitFor1s:
    tst   reg[INT_CLR0], INT_MSK0_SLEEP   ; Test the SleepTimer Interrupt Status
    jz    .WaitFor1s                      ; Interrupt will latch but will not dispatch
                                          ;   since interrupts are not globally enabled
ELSE ; !( SELECT_32K & WAIT_FOR_32K )
    ; Either no ECO, or waiting for stable clock is to be done in main
    M8C_SetBank1
    mov   reg[OSC_CR0], (SELECT_32K_JUST | PLL_MODE_JUST | SLEEP_TIMER_JUST |
OSC_CR0_CPU_12MHz)
    M8C_SetBank0
    M8C_ClearWDTAndSleep          ; Reset the watch dog

ENDIF ;( SELECT_32K & WAIT_FOR_32K )
```

[10]

```
IF ( PLL_MODE )
    ; Crystal is now fully operational (assuming WAIT_FOR_32K was enabled).
    ; Now startup PLL if selected, and wait 16 ms for it to stabilize.
    ;
    M8C_SetBank1
    mov   reg[OSC_CR0], (SELECT_32K_JUST | PLL_MODE_JUST | OSC_CR0_SLEEP_64Hz |
OSC_CR0_CPU_3MHz)
```

```
    M8C_SetBank0
    M8C_ClearWDTAndSleep                ; Reset the sleep timer to get full period
    mov   reg[INT_VC], 0                ; Clear all pending interrupts

.WaitFor16ms:
    tst   reg[INT_CLR0],INT_MSK0_SLEEP  ; Test the SleepTimer Interrupt Status
    jz    .WaitFor16ms
    M8C_SetBank1                        ; continue boot at CPU Speed of SYSCLK/2
    mov   reg[OSC_CR0], (SELECT_32K_JUST | PLL_MODE_JUST | OSC_CR0_SLEEP_64Hz |
OSC_CR0_CPU_12MHz)
    M8C_SetBank0

IF      ( WAIT_FOR_32K )
ELSE ; !( WAIT_FOR_32K )
    ; Option settings (PLL-Yes, ECO-No) are incompatible - force a syntax error
    ERROR_PSoC Disabling WAIT_FOR_32K requires that the PLL_Lock must be enabled in user
code.
ENDIF ;(WAIT_FOR_32K)
ENDIF ;(PLL_MODE)
```

[11]

```
      ;-----------------------------------------------------
    ; Initialize Proper Drive Mode for External Clock Pin
    ;-----------------------------------------------------

    ; Change EXTCLK pin from Hi-Z Analog (110b) drive mode to Hi-Z (010b) drive mode

IF (SYSCLK_SOURCE)
    and reg[PRT1DM2],  ~0x10        ; Clear bit 4 of EXTCLK pin's DM2 register
ENDIF
    ; EXTCLK pin is now in proper drive mode to input the external clock signal
```

[12]

```
    ;-------------------------
    ; Close CT leakage path.
    ;-------------------------
    mov   reg[ACB00CR0], 05h
    mov   reg[ACB01CR0], 05h
    mov   reg[ACB02CR0], 05h
    mov   reg[ACB03CR0], 05h
```

[13]

```
IF     (TOOLCHAIN & HITECH)
    ;-------------------------------------------
    ; HI-TECH initialization: Enter the Large Memory Model, if applicable
    ;-------------------------------------------
      global        __Lstackps
      mov     a,low __Lstackps
      swap    a,sp

IF ( SYSTEM_LARGE_MEMORY_MODEL )
    RAM_SETPAGE_STK SYSTEM_STACK_PAGE      ; relocate stack page ...
    RAM_SETPAGE_IDX2STK            ; initialize other page pointers
    RAM_SETPAGE_CUR 0
    RAM_SETPAGE_MVW 0
    RAM_SETPAGE_MVR 0
```

```
    IF ( SYSTEM_IDXPG_TRACKS_STK_PP ); Now enable paging:
    or    F, FLAG_PGMODE_11b       ; LMM w/ IndexPage<==>StackPage
    ELSE
    or    F, FLAG_PGMODE_10b       ; LMM w/ independent IndexPage
    ENDIF ;  SYSTEM_IDXPG_TRACKS_STK_PP
ENDIF ;  SYSTEM_LARGE_MEMORY_MODEL
ELSE
    ;-------------------------------------------
    ; ImageCraft Enter the Large Memory Model, if applicable
    ;-------------------------------------------
IF ( SYSTEM_LARGE_MEMORY_MODEL )
    RAM_SETPAGE_STK SYSTEM_STACK_PAGE      ; relocate stack page ...
    mov   A, SYSTEM_STACK_BASE_ADDR        ;   and offset, if any
    swap  A, SP
    RAM_SETPAGE_IDX2STK            ; initialize other page pointers
    RAM_SETPAGE_CUR 0
    RAM_SETPAGE_MVW 0
    RAM_SETPAGE_MVR 0

  IF ( SYSTEM_IDXPG_TRACKS_STK_PP ); Now enable paging:
    or    F, FLAG_PGMODE_11b       ; LMM w/ IndexPage<==>StackPage
  ELSE
    or    F, FLAG_PGMODE_10b       ; LMM w/ independent IndexPage
  ENDIF ;  SYSTEM_IDXPG_TRACKS_STK_PP
ELSE
    mov   A, __ramareas_end        ; Set top of stack to end of used RAM
    swap  SP, A
ENDIF ;  SYSTEM_LARGE_MEMORY_MODEL
ENDIF ;      TOOLCHAIN
```

[14]

```
    ;-------------------------
    ; Load Base Configuration
    ;-------------------------
    ; Load global parameter settings and load the user modules in the
    ; base configuration. Exceptions: (1) Leave CPU speed as fast as possible
    ; to minimize startup time; (2) You might still need to play with the
    ; sleep timer.
    ;
    lcall LoadConfigInit
```

[15]

```
    ;-----------------------------------
    ; Initialize C Run-Time Environment
    ;-----------------------------------
IF ( C_LANGUAGE_SUPPORT )
IF ( SYSTEM_SMALL_MEMORY_MODEL )
    mov  A,0                          ; clear the 'bss' segment to zero
    mov  [__r0],<__bss_start
BssLoop:
    cmp  [__r0],<__bss_end
    jz   BssDone
    mvi  [__r0],A
    jmp  BssLoop
BssDone:
    mov  A,>__idata_start             ; copy idata to data segment
    mov  X,<__idata_start
    mov  [__r0],<__data_start
```

```
IDataLoop:
    cmp  [__r0],<__data_end
    jz   C_RTE_Done
    push A
    romx
    mvi  [__r0],A
    pop  A
    inc  X
    adc  A,0
    jmp  IDataLoop

ENDIF ; SYSTEM_SMALL_MEMORY_MODEL

IF ( SYSTEM_LARGE_MEMORY_MODEL )
    mov   reg[CUR_PP], >__r0          ; force direct addr mode instructions
                                      ; to use the Virtual Register page.

    ; Dereference the constant (flash) pointer pXIData to access the start
    ; of the extended idata area, "xidata." Xidata follows the end of the
    ; text segment and may have been relocated by the Code Compressor.
    ;
    mov   A, >__pXIData               ; Get the address of the flash
    mov   X, <__pXIData               ;   pointer to the xidata area.
    push  A
    romx                              ; Get the MSB of xidata's address.
    mov   [__r0], A
    pop   A
    inc   X
    adc   A, 0
    romx                              ; Get the LSB of xidata's address
    swap  A, X
    mov   A, [__r0]                   ; pXIData (in [A,X]) points to the
                                      ;   XIData structure list in flash
    jmp   .AccessStruct

    ; Unpack one element in the xidata "structure list" that specifies the
    ; values of C variables. Each structure contains 3 member elements.
    ; The first is a pointer to a contiguous block of RAM to be initial-
    ; ized. Blocks are always 255 bytes or less in length, and they never cross
    ; RAM page boundaries. The list terminates when the MSB of the pointer
    ; contains 0xFF. There are two formats for the struct, depending on the
    ; value in the second member element, an unsigned byte:
    ; (1) If the value of the second element is non-zero, it represents
    ; the 'size' of the block of RAM to be initialized. In this case, the
    ; third member of the struct is an array of bytes of length 'size' and
    ; the bytes are copied to the block of RAM.
    ; (2) If the value of the second element is zero, the block of RAM is
    ; to be cleared to zero. In this case, the third member of the struct
    ; is an unsigned byte containing the number of bytes to clear.

.AccessNextStructLoop:
    inc   X                           ; pXIData++
    adc   A, 0
.AccessStruct:                        ; Entry point for first block
    ;
    ; Assert: pXIData in [A,X] points to the beginning of an XIData struct.
    ;
    M8C_ClearWDT                      ; Clear the watchdog for long inits
    push  A
    romx                              ; MSB of RAM addr (CPU.A <- *pXIData)
    mov   reg[MVW_PP], A              ;   for use with MVI write operations
```

```
    inc   A                             ; End of Struct List? (MSB==0xFF?)
    jz    .C_RTE_WrapUp                 ;   Yes, C runtime environment complete
    pop   A                             ; restore pXIData to [A,X]
    inc   X                             ; pXIData++
    adc   A, 0
    push  A
    romx                                ; LSB of RAM addr (CPU.A <- *pXIData)
    mov   [__r0], A                     ; RAM Addr now in [reg[MVW_PP],[__r0]]
    pop   A                             ; restore pXIData to [A,X]
    inc   X                             ; pXIData++ (point to size)
    adc   A, 0
    push  A
    romx                                ; Get the size (CPU.A <- *pXIData)
    jz    .ClearRAMBlockToZero          ; If Size==0, then go clear RAM
    mov   [__r1], A                     ;             else downcount in __r1
    pop   A                             ; restore pXIData to [A,X]

.CopyNextByteLoop:
    ; For each byte in the structure's array member, copy from flash to RAM.
    ; Assert: pXIData in [A,X] points to previous byte of flash source;
    ;         [reg[MVW_PP],[__r0]] points to next RAM destination;
    ;         __r1 holds a non-zero count of the number of bytes remaining.
    ;
    inc   X                             ; pXIData++ (point to next data byte)
    adc   A, 0
    push  A
    romx                                ; Get the data value (CPU.A <- *pXIData)
    mvi   [__r0], A                     ; Transfer the data to RAM
    tst   [__r0], 0xff                  ; Check for page crossing
    jnz   .CopyLoopTail                 ;   No crossing, keep going
    mov   A, reg[ MVW_PP]               ;   If crossing, bump MVW page reg
    inc   A
    mov   reg[ MVW_PP], A
.CopyLoopTail:
    pop   A                             ; restore pXIData to [A,X]
    dec   [__r1]                        ; End of this array in flash?
    jnz   .CopyNextByteLoop             ;   No, more bytes to copy
    jmp   .AccessNextStructLoop         ;   Yes, initialize another RAM block

.ClearRAMBlockToZero:
    pop   A                             ; restore pXIData to [A,X]
    inc   X                             ; pXIData++ (point to next data byte)
    adc   A, 0
    push  A
    romx                                ; Get the run length (CPU.A <- *pXIData)
    mov   [__r1], A                     ; Initialize downcounter
    mov   A, 0                          ; Initialize source data

.ClearRAMBlockLoop:
    ; Assert: [reg[MVW_PP],[__r0]] points to next RAM destination and
    ;         __r1 holds a non-zero count of the number of bytes remaining.
    ;
    mvi   [__r0], A                     ; Clear a byte
    tst   [__r0], 0xff                  ; Check for page crossing
    jnz   .ClearLoopTail                ;   No crossing, keep going
    mov   A, reg[ MVW_PP]               ;   If crossing, bump MVW page reg
    inc   A
    mov   reg[ MVW_PP], A
    mov   A, 0                          ; Restore the zero used for clearing
.ClearLoopTail:
    dec   [__r1]                        ; Was this the last byte?
```

```
    jnz   .ClearRAMBlockLoop          ;   No, continue
    pop   A                           ;   Yes, restore pXIData to [A,X] and
    jmp   .AccessNextStructLoop       ;        initialize another RAM block

.C_RTE_WrapUp:
    pop   A                           ; balance stack

ENDIF ; SYSTEM_LARGE_MEMORY_MODEL

C_RTE_Done:

ENDIF ; C_LANGUAGE_SUPPORT
```

[16]

```
    ;-------------------------------
    ; Voltage Stabilization for SMP
    ;-------------------------------

IF ( POWER_SETTING & POWER_SET_5V0)    ; 5.0V Operation
IF ( SWITCH_MODE_PUMP ^ 1 )            ; SMP is operational
    ;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    ; When using the SMP at 5 V, please wait for Vdd to slew from 3.1 V to
    ; 5 V before enabling the Precision Power-On Reset (PPOR).
    ;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    or   reg[INT_MSK0],INT_MSK0_SLEEP
    M8C_SetBank1
    and   reg[OSC_CR0], ~OSC_CR0_SLEEP
    or    reg[OSC_CR0],  OSC_CR0_SLEEP_512Hz
    M8C_SetBank0
    M8C_ClearWDTAndSleep                  ; Restart the sleep timer
    mov   reg[INT_VC], 0                  ; Clear all pending interrupts
.WaitFor2ms:
    tst   reg[INT_CLR0], INT_MSK0_SLEEP   ; Test the SleepTimer Interrupt Status
    jz   .WaitFor2ms                      ; Branch fails when 2 ms has passed
ENDIF ; SMP is operational
ENDIF ; 5.0-V Operation
```

[17]

```
    ;-------------------------------
    ; Set Power-On Reset (POR) Level
    ;-------------------------------

    ;  The writes to the VLT_CR register below include setting the POR to VLT_CR_POR_HIGH,
    ;  VLT_CR_POR_MID, or VLT_CR_POR_LOW. Correctly setting this value is critical to the
proper
    ;  operation of the PSoC. The POR protects the M8C from misexecuting when Vdd falls low.
These
    ;  values should not be changed from the settings here. Failure to follow this
instruction could
    ;  lead to corruption of PSoC flash.

    M8C_SetBank1

IF (POWER_SETTING & POWER_SET_5V0)          ; 5.0-V Operation?
 IF (POWER_SETTING & POWER_SET_SLOW_IMO)    ; and Slow Mode?
 ELSE                                       ;    No, fast mode
  IF ( CPU_CLOCK_JUST ^ OSC_CR0_CPU_24MHz ) ;    As fast as 24 MHz?
```

```
                                            ;       No, set midpoint POR in user code, if
desired
  ELSE ; 2 MHz                              ;
    or    reg[VLT_CR],  VLT_CR_POR_HIGH     ;       yes, highest POR trip point required
  ENDIF ; 24 MHz
 ENDIF ; Slow Mode
ENDIF ; 5.0-V Operation

    M8C_SetBank0
```

[18]

```
    ;-----------------------------
    ; Wrap up and invoke "main"
    ;-----------------------------

    ; Disable the sleep interrupt used for timing above. In fact,
    ; no interrupts should be enabled now; therefore, clear the register.
    ;
    mov  reg[INT_MSK0],0

    ; Everything has started OK. Now select requested CPU and sleep frequency.
    ; And put decimator in full mode so that it does not consume too much current.
    ;
    M8C_SetBank1
    mov  reg[OSC_CR0],(SELECT_32K_JUST | PLL_MODE_JUST | SLEEP_TIMER_JUST | CPU_CLOCK_JUST)
    or   reg[DEC_CR2],80h                    ; Put decimator in full mode.
    M8C_SetBank0
```

[19]

```
    ; Global Interruptsare NOT enabled; this should be done in main().
    ; LVD is set but will not occur unless global interrupts are enabled.
    ; Global interrupts should be enabled as soon as possible in main().
    ;
    mov  reg[INT_VC],0              ; Clear any pending interrupts that may
                                    ; have been set during the boot process.
IF     (TOOLCHAIN & HITECH)
      ljmp  startup                   ; Jump to C compiler startup code.
ELSE
IF ENABLE_LJMP_TO_MAIN
    ljmp  _main                     ; Go to main (no return).
ELSE
    lcall _main                     ; Call main.
.Exit:
    jmp  .Exit                      ; Wait here after return till power off or reset.
ENDIF
ENDIF ; TOOLCHAIN

    ;---------------------------------
    ; Library Access to Global Params
    ;---------------------------------
    ;
 bGetPowerSetting:
_bGetPowerSetting:
    ; Returns value of POWER_SETTING in the A register.
    ; No inputs. No side effects.
    ;
    mov   A, POWER_SETTING
    ret
```

```
IF     (TOOLCHAIN & HITECH)
ELSE
    ;---------------------------------
    ; Order Critical RAM and ROM AREAs
    ;---------------------------------
    ;  'TOP' is all that has been defined so far...

    ;  ROM AREAs for C CONST, static & global items
    ;
    AREA lit                (ROM, REL, CON)    ; 'const' definitions
    AREA idata              (ROM, REL, CON)    ; Constants for initializing RAM
__idata_start:

    AREA func_lit           (ROM, REL, CON)    ; Function Pointers
__func_lit_start:

IF ( SYSTEM_LARGE_MEMORY_MODEL )
    ; We use the func_lit area to store a pointer to extended initialized
    ; data (xidata) area that follows the text area. Func_lit isn't
    ; relocated by the code compressor, but the text area may shrink, and
    ; that moves xidata around.
    ;
__pXIData:          word __text_end           ; ptr to extended idata
ENDIF

    AREA psoc_config        (ROM, REL, CON)    ; Configuration Load and Unload
    AREA UserModules        (ROM, REL, CON)    ; User Module APIs

    ; CODE segment for general use
    ;
    AREA text (ROM, REL, CON)
__text_start:

    ; RAM area usage
    ;
    AREA data               (RAM, REL, CON)    ; initialized RAM
__data_start:

    AREA virtual_registers (RAM, REL, CON)    ; Temp vars of C compiler
    AREA InterruptRAM      (RAM, REL, CON)    ; Interrupts, on Page 0
    AREA bss               (RAM, REL, CON)    ; general use
__bss_start:

ENDIF ; TOOLCHAIN

; end of file boot.asm
```

Document History

Document Title: AN73617 – PSoC® Designer Boot Process, from Reset to Main

Document Number: 001-73617

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3473512	KEES	12/22/2011	New Application Note
*A	3807180	KEES	11/09/2012	Updated in new template.
*B	4605712	GRAA	12/23/2014	Updated the overview section, with reference to the sequence of events shown in Figure 1.
*C	5701830	AESATMP9	04/19/2017	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, 2011-2017. 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.