AN2010

PSoC® 1 Best Practices and Recommendations

Author: Jeff Dahlin, Rajiv Badiger

Associated Project: No

Associated Part Family: CY8C29x66, CY8C28xxx, CY8C27x43, CY8C24x94, CY8C24x23, CY8C24x33, CY8C23x33, CY8C22x45, CY8C21x45, CY8C21x34, CY8C21x23

Software Version: PSoC Designer™

Related Application Notes: AN75320, AN32200

AN2010 provides the guidelines on best practices for developing PSoC 1 systems, and it exposes some common mistakes designers make. It also contains a brief introduction to the PSoC 1 devices, software, and support collateral.

Contents

Introduction 1
Getting Started 1
- Understanding PSoC 1 device 1
- Install PSoC Designer 1
- Examine Example Projects 1
- Download Latest Documentation 2
- Explore Other Resources 2

Best Practices 2
- CPU and Clocks 2
- Digital Blocks 3
- Analog Blocks 4
- GPIOs 5
- Flash 6
- Firmware 7
- Debug and Test 10
- Common Mistakes 10

Summary 11
Related Application Notes 11
Appendix A 12
Worldwide Sales and Design Support 14

Introduction

This application note is organized into sections according to the earliest and most likely period of development during which it will be needed: getting started, user module placement, interconnection and pin-out, software implementation, testing, and debug. Each of these sections may contain information that is also applicable to other sections, so care should be taken to read and understand the document as a whole.

Getting Started

Understanding PSoC 1 device

For PSoC 1 beginners, it is recommended to read application note Getting Started with PSoC® 1 – AN75320. This application note provides the first level basic information on the PSoC 1 device.

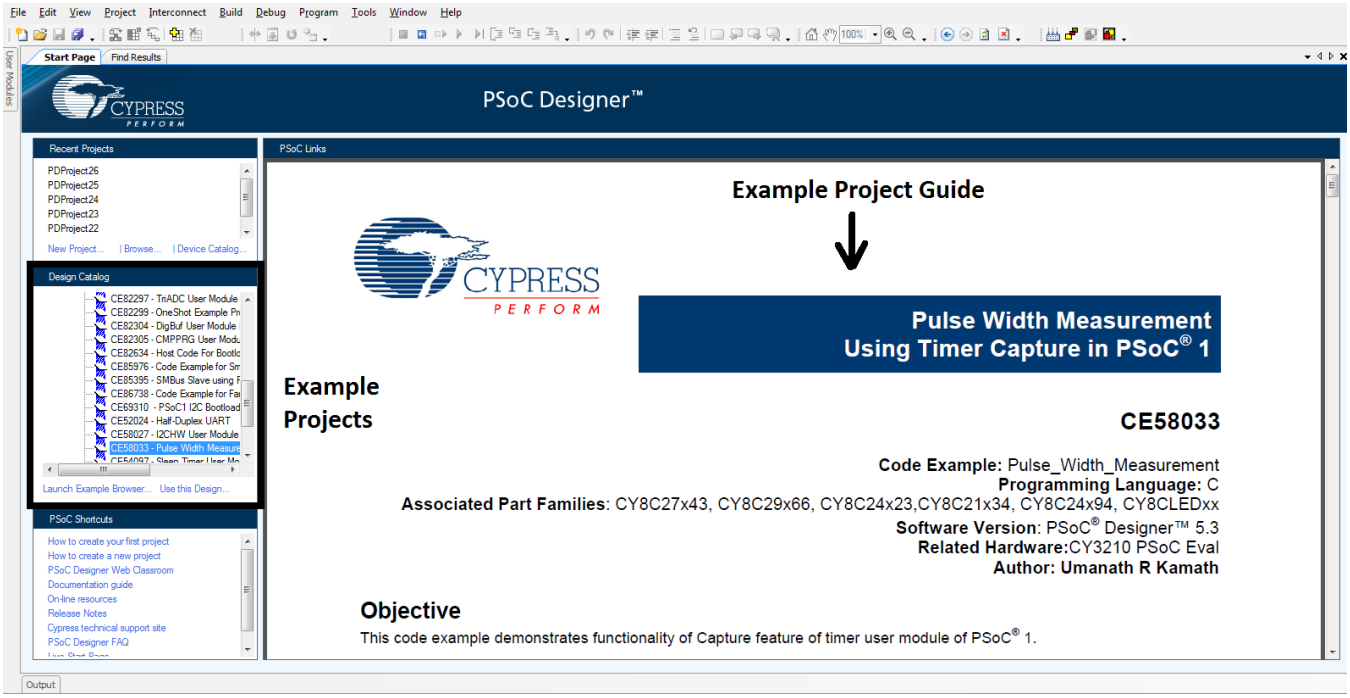
Install PSoC Designer

PSoC 1 projects are created using PSoC Designer IDE tool. Ensure that you have the most recent version of PSoC Designer software installed on your PC. You can check for the latest revision at www.cypress.com/psocdesigner.

Examine Example Projects

After installing PSoC Designer, go through the example projects included with the software. These projects can be accessed from **Start Page** of PSoC Designer as Figure 1 shows. Along with their associated documentation, working through them provides the best method for becoming familiar with PSoC 1 devices and the PSoC Designer software.

Figure 1. Example Projects in PSoC Designer



Download Latest Documentation

The PSoC Designer Documentation Suite, the PSoC 1 product data sheets, and other valuable files are available under the `…\Documentation` directory where the installation of PSoC Designer resides.

When working with the PSoC 1 family of devices, download the latest family data sheets, silicon errata, software release notes, and software errata from PSoC 1 landing Page.

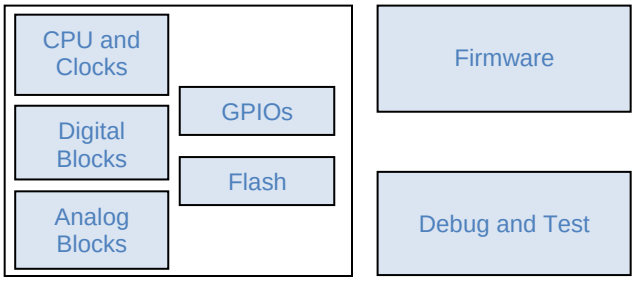
Explore Other Resources

- Latest releases of PSoC Designer software
- User forums enabling free information exchange between Cypress PSoC users
- An online technical support system
- Knowledge base articles
- Application notes
- Technical Reference Manuals

You can also refer to www.cypress.com/support for more details.

Best Practices

This section presents best practices and recommendations for some of the important sections in the PSoC device and the IDE.



CPU and Clocks

CPU Clock Speed Considerations

Maximum CPU clock frequency limit is depended on the operating voltage (VDD). Figure 2 shows the operating region of CY8C29x66 device (CPU clock frequency Vs VDD). If the required CPU clock frequency is greater than 24 MHz, VDD should always be between 4.75 V to 5.25 V. For 12 MHz and lower CPU frequencies, VDD can be as low as 3.0 V. For similar graph for other PSoC 1 devices, refer device datasheets.

If there are chances of VDD getting below 4.75 V, operate the CPU with 12 MHz clock or below.

Figure 2. CPU Frequency Vs VDD

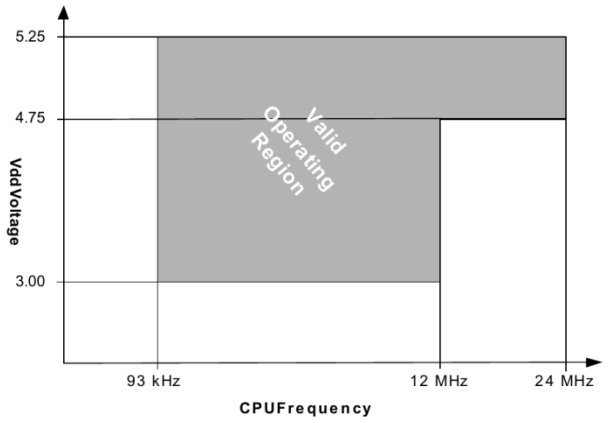
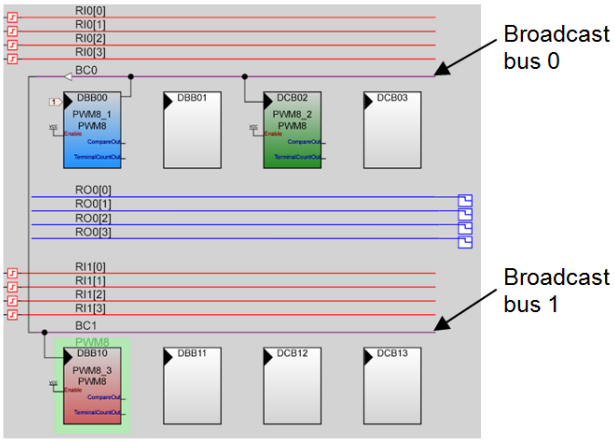


Figure 3. Broadcast Bus



External Clock as SYSCLK Source

When an external clock source is used as a source for SysClk (system clock), ensure that –

- Clock is glitch free; there is no glitch protection in PSoC 1 for external clock. The duty cycle / pulse width (high and low) of the external clock should adhere to the external clock specifications given in the device datasheet.
- IMO is always enabled; reserved bit OSC_CR2[1] should always be 0.

These two conditions are necessary for proper operation of the device when using an external clock.

Global Bus Clock Rate Limitations

The Global Input and Global Output busses are not guaranteed to operate properly above 12 MHz. This includes pulses at a width equivalent to 12 MHz. Beware of digital blocks configured as timers, counters or PWMs with a terminal count (TC) output. The TC output is simply the input clock gated to the output. Therefore, if a timer has 48 MHz as the clock source, the TC output is a single pulse from the 48 MHz clock, and must not be routed to a Global Output bus.

Clock Routing

In the PSoC device, there is a broadcast bus for each digital block row. Any block in a row can take the input from its broadcast bus or connect the output line. A broadcast bus can also take input from the broadcast bus of another row.

As Figure 3 shows, PWM8_1 output is routed to broadcast bus 0, which is taken by PWM8_2 as clock input. Broadcast bus 0 and bus 1 are connected to facilitate signal routing to PWM8_3 which is placed in next row.

Digital Clocks

In most of the PSoC device families, there are several clock sources derived from system clock – VC1, VC2 and VC3. It is always recommended to judiciously configure these sources (configured in global resources of PSoC Designer) so as to eliminate use of digital blocks for generating particular clock frequencies. This is particularly useful for configuring communication user modules such as UART and SPI.

For more details on Clocks, you can refer Application Note Clocks and Global Resources – AN32200.

Digital Blocks

User Module Clock Limitations

Digital blocks have the following clocking limitations:

- Counter user modules that use the enable function cannot operate above 24 MHz.
- Timer user modules that use the capture function cannot operate above 24 MHz.
- The CRC user module cannot operate above 24 MHz.
- Digital-block based serial communication user modules cannot operate above 24 MHz.

Note Legacy PSoC users (25/26xxx parts) refer to Appendix A for additional information.

N+1 Based Digital Clocks

All timing parameters (period and pulse width) for digital user modules are n+1 values. Therefore, the value entered as the parameter or passed to software APIs must be 1 less than the desired value. This is rooted in the fact that the counter and timers count down to zero.

False "Starts" with UARTs

Before the UART or RX8 user modules are started, it is important to ensure that the RxD line is in the mark state (1). The UART start bit detector is level sensitive. Therefore, if it is started with a 0 on the input, it repeatedly detects start bits with data of 0x00 (and detect a framing error) until the input is set to 1.

SPI ~SS Enable

When set to mode 0 or mode 1, the SPI slave requires the SS_ signal to be toggled at least once to correctly receive data. For modes 2 and 3, the ~SS signal does not need to be toggled and can be tied low. For details of SPI communication, you can refer application note Getting Started with SPI in PSoC® 1 – AN51234.

Analog Blocks

Clocking Multi-Block User Modules

Some multi-block user modules require individual input clocks to be set to the same clock source. A specific example is the ADCINC. The ADCINC requires that the same clock be routed to the user module's analog block and to the digital blocks associated with the ADCINC.

The clock for the digital blocks is selected in the User Module Parameters (see Figure 4) and the clock for the analog block is set as the analog column clock (Figure 5).

Figure 4. ADC Digital Block Clock Selection

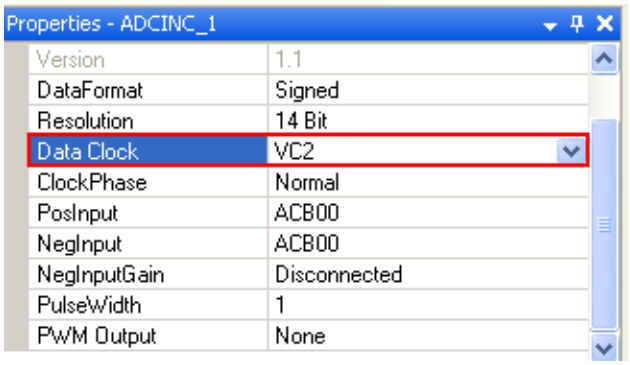
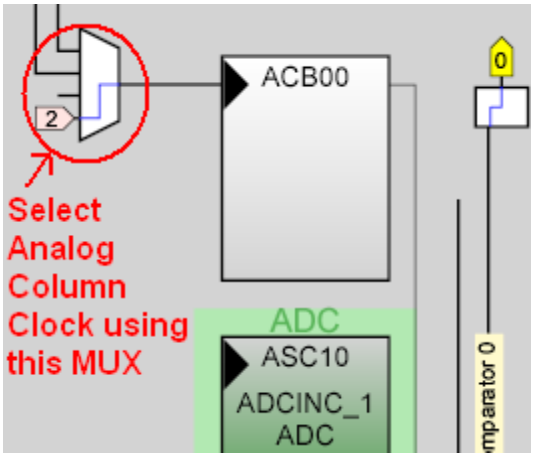


Figure 5. ADC Analog Column Clock Selection



This requirement is called out in the user module data sheet, but is sometimes overlooked.

Analog Column Clock Speed

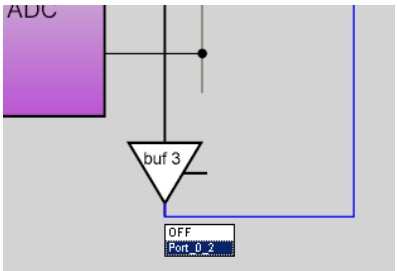
The maximum analog column clock frequency can be 8 MHz. Review each analog user module data sheet to make sure that the analog column clock selection meets the clock requirements for the user module. A very fast clock frequency can result in the internal capacitors not fully charging and a very slow clock frequency can result in a voltage droop on the capacitor.

The faster the clock to an analog block, the higher the power setting must be. The power setting affects the drive current limit for the operational amplifier inside the analog block. With a higher drive current, the capacitors are charged more quickly and the clock frequency is higher.

Analog Output Buffers

When routing analog outputs make sure that the output buffer associated with the analog column is turned on. To do this, click the symbol for the analog output buffer at the bottom of the Module Placement view of PSoC Designer (see Figure 6).

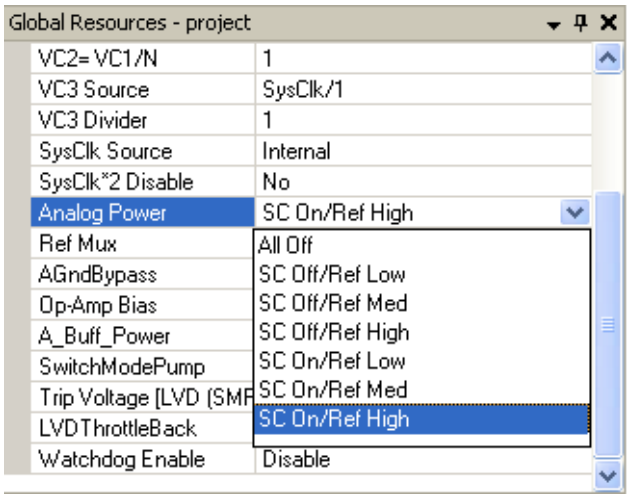
Figure 6. Enabling analog output buffer 0



Analog Power Setting

The power setting for the analog references must be set to the highest power setting used in the placed analog user modules (see Figure 7). For example, if there are two PGAs set to low power and a DAC set to high power, then the reference setting selected should be “SC On/Ref High.” The number of user modules does not impact the power setting because the issue is not total current supplied, but rather the charging rate of the switched capacitors. The capacitor must be fully charged or discharged at the time the switch is closed.

Figure 7. Setting the Analog Reference Power



DAC Clock Rates and Ripple

If a ripple is observed on the output of a DAC, it is possible that the clock for the analog column in which the DAC is placed is either too fast or too slow. A clock that is too fast causes the capacitors in the block to not fully charge, and a clock that is too slow can cause the voltage on the caps to droop. See the DAC User Module Data Sheet for the limitations of the clock rates.

For more on the analog section, you can refer following application notes:

PSoC® 1 Selecting Analog Ground and Reference – AN2219

PSoC® 1 Using Correlated Double Sampling to Reduce Offset, Drift and Low Frequency Noise – AN2226.

GPIOs

Pin Types

- There are eight types of configurable pins in the 21/22/24/27/28/29xxx PSoC families:
- **Digital I/O Only** – These pins are the most plentiful and all other configurable pins can become digital I/O pins. The digital I/O pins are connected to the Global Input and Global Output busses. They can also act as GPIO (General Purpose Input Output) pins by using the StdCPU configuration.
- **Digital I/O or Analog Input** – These pins can act as digital I/O pins or can be connected to the analog input muxes.
- **Digital I/O or Analog I/O** – These pins can act as digital I/O pins, analog inputs, or analog outputs. These should be used last among the choices for analog inputs. The number of pins available for analog output is less than that for analog inputs and should be valued.
- **Digital I/O or Direct Analog Input** – These pins can act as digital I/O pins or as inputs directly into switched capacitor (SC) analog blocks. These inputs do not route through the analog muxes but route directly to specific SC analog blocks. These connections can be used to get additional analog inputs, or can be used to bypass the continuous time analog blocks.
- **Digital I/O or External Reference** – These pins can act as digital I/O pins or can be used as inputs for an external AGND reference voltage, and for an external V_{REF} reference voltage. These external references provide added flexibility for analog designs.
- **Digital I/O or I^2C** – These pins can act as digital I/O pins or can be used for I^2C communication. These pins are the preferred location for I^2C connection because the alternate I^2C pins are also used for ISSP programming, which without proper precautions can interfere with I^2C communication.
- **Digital I/O or External System Clock** – These pins are used as digital I/O pins or can be used as the input from an external clock source (EXTCLK). The external clock sources the CPU system clock and all other external and internal signals are synchronized to this signal.
- **Digital I/O, External Crystal I/O or Alternate I^2C** – These pins can act as digital I/O pins, used to connect a 32.678 kHz crystal to the External Crystal Oscillator (ECO), or can be used as alternate I^2C communication pins in PSoC 1. The ECO is used to generate a highly accurate timing source. These pins are also used for ISSP programming.

Table 1 lists the port and pin numbers and the available functionality and suggested order of allocation for the 22/24/27/28/29xxx family of parts.

Table 1. CY8C22xxx, CY8C24xxx, CY8C27xxx, CY8C28xxx, and CY8C29xxx Pin Types

	Digital I/O	Analog In	Analog Out	Dir Analog In	External Ref.	Ext. Crystal	I2C comm	
Port1[2:3]	☆							Digital I/O only, use these pins first for GPIO, Global In and Global Out signals.
Port1[6]	☆							
Port2[5]	☆							
Port2[7]	☆							
Port3[0:7]	☆							
Port4[0:7]	☆							
Port5[0:3]	☆							
Port1[4]	☆							Digital I/O or External System Clock
Port1[5]	☆						☆	Digital I/O or I2C
Port1[7]	☆						☆	
Port0[0:1]	☆	☆						Digital I/O or Analog Input, use these analog in's first
Port0[6:7]	☆	☆						
Port0[2:5]	☆	☆	☆					Digital I/O or Analog I/O
Port2[0:3]	☆			☆				Digital I/O or Direct Analog Inputs
Port2[4]	☆				☆			Digital I/O or External references
Port2[6]	☆				☆			
Port1[0:1]	☆					☆	☆	Digital I/O, External Crystal connection or alternate I2C

Pin Allocation Best Practices

The best approach for allocating pins is to first assign analog pins to functions that require their use. After the analog pins have been allocated, assign digital functions to pins that are digital I/O only. Pins with multiple functions must be allocated as digital pins last, so they can be used for extra functions if there is an additional requirement later in the design.

ECO Pin Drive Modes

The drive modes for the crystal oscillator pins must be set to High Z if the ECO is used.

For more on GPIOs of PSoC 1, you can refer application note PSoC® 1 Getting Started with GPIO – AN2094.

GPIO Interrupts

The interrupt request signals for all GPIO pins are ORed together to create one GPIO interrupt signal. This can lead to some unexpected behavior. The interrupt controller generates an interrupt when it detects a rising edge on the GPIO interrupt signal. The interrupt request signal for each pin uses combinatorial logic and can output a constant high if the input to the pin matches the interrupt case. Because the interrupt request signals of all the pins are ORed together, a continuous interrupt request from one pin can mask an interrupt request from anther pin. The most common case is when a pin that is set to detect a rising edge interrupt has a signal that goes high and stays high. In this case, no future GPIO interrupts are recognized. The equivalent condition can occur with falling edge interrupts and a signal that stays low. A less obvious condition can occur when one GPIO pin has a signal that goes high and a second GPIO pin's signal goes high before the first one goes low. In this case, the overlapping high on the first pin blocks the second pin's interrupt from being recognized.

One obvious solution is to make sure that signals that generate interrupts do not remain at a level that blocks interrupts. Another workaround is to use the Change-from-Previous-Read interrupt selection and read the pin in the ISR. Reading the pin changes the interrupt level. If this is done, then extra interrupts from the opposite edge of the signal must be ignored.

Another less obvious workaround is to route signals to unused digital blocks and analog comparator buses to generate interrupts. The digital block can be configured as a timer with a period of 0. With its interrupt enabled, it generates an interrupt on each rising edge.

An analog CT block can be loaded with a comparator and routed to the analog comparator bus, which can be used to generate an interrupt.

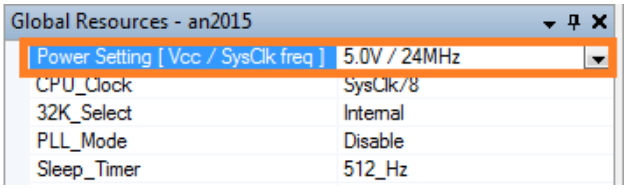
Flash

Endurance

Flash Endurance is measured in terms of the maximum write/erase cycles guaranteed for the flash memory. Endurance of PSoC 1 flash is limited to 50K cycles. It should be ensured in the system that the number of flash writes, occurring in a particular block, should be less than 50K. If there is a possibility of exceeding 50K write cycles, it is recommended to switch over to a different flash block when the cycle count reaches 50K. User needs to maintain a counter which needs to be stored in the flash block along with other data, to retain the cycle count information even if the device power is turned off.

Voltage Setting

PSoC 1 device uses calibration values stored in hidden flash memory to calculate the pulse width for writing the data into flash. The pulse width is depended on voltage setting, temperature and the device characteristics itself. Pulse width is more if voltage is less and temperature is more and vice versa. PSoC device doesn't measure the VDD voltage for calculating the pulse width. It needs to be set during project stage itself. PSoC Designer has a setting related to VDD in Global Resources.



Several options are provided in the Power Setting depending on the device selected. User needs to select the setting depending on the voltage VDD in the system.

POR setting

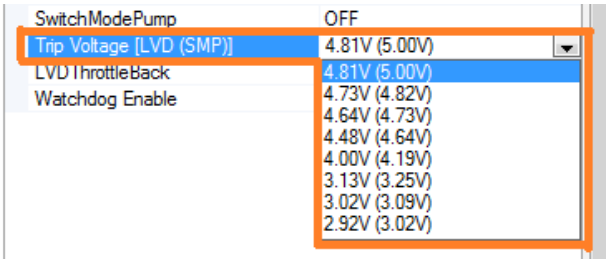
PSoC 1 device gets reset if VDD drops below a particular voltage known as POR (power on reset) voltage. The code located in boot.asm (an automatically generated file) properly sets the correct POR level based on the CPU's operating frequency and the VDD. For example, the POR level is set to 4.55 V when CPU clock is set to 24 MHz and VDD to 5.0 V. Please refer device datasheets for POR levels for other voltage and CPU clock settings It is possible to change the POR level in firmware, but it is not recommended.

VDD Ramp

If the VDD ramp is slower, then it is recommended not to initiate flash write in the beginning of main.c. Provide sufficient delay in the beginning of main.c to allow supply voltage to stabilize and then initiate flash write. Boot.asm code and device POR circuit will, however, block the entry to main.c till supply reaches POR voltage by causing multiple device resets. User must avoid the flash write till voltage ramps from POR level to the final value. This is important as a lower voltage flash programming can cause retention problems.

Low Voltage Detect (LVD) interrupt

PSoC 1 has LVD circuitry which halts the CPU when voltage drops below the circuitry. This helps to avoid low voltage flash programming and wrong CPU execution. VDD threshold for LVD interrupt can be set in PSoC Designer using "Trip Voltage" setting in Global Resources.



It is recommended that the LVD threshold is as close to the VDD as possible. On LVD interrupt, CPU is halted. Halt instruction is written in vector table which can be found in boot.asm file. LVD interrupt is enabled by writing '1' into bit 0 of INT_MSK0 register.

Flash Security setting

Four security modes are provided for flash. Full protection mode (W) should be put for the flash blocks which are not written during run time. This is to avoid unintentional writes, resulting in program corruption.

Firmware

This section is focused on the basics of placing and configuring user modules. It also discusses issues with user module parameterization and Global Resources settings, which are usually set at the same time as user modules are placed.

Initial User Module Parameter Selection

It is recommended that an initial value be chosen for all parameters associated with a user module. Even if particular parameters don't seem to apply (for example: the Interrupt Type in a Counter8 that is not using interrupts), or if parameters are later established using software APIs, the parameter should be set to something other than the initial "*?*" value set by PSoC Designer.

All Global Resources should be reviewed, understood, and configured in a manner that makes sense in the context of the project.

Design Rule Checker (DRC)

PSoC Designer provides a DRC tool that should be run to check for common placement and configuration mistakes.

Generated Code Conventions

A number of software conventions are followed in the code generated by the PSoC Designer tools. These are summarized in this section.

Register vs. Memory Access in Assembly

A fairly common mistake when accessing a register in assembly is to use the memory mnemonic instead of the register mnemonic. When accessing the INT_MSK0 register, for example, the following instruction should be used:

```
mov A, reg[INT_MSK0]
```

It is easy to accidentally type the following instead:

```
mov A, [INT_MSK0]
```

Both instructions compile without error. The difference is that the first command loads `A` with the content of the register at address 0xE0, whereas the second instruction loads `A` with the content of the memory at 0xE0.

Register Bank Selection

The PSoC 1 I/O registers are located in two different address banks. The design of the PSoC 1 intentionally places registers that might commonly be modified during operation in Register Bank 0. However, it is sometimes necessary to access registers in Register Bank 1.

Because it takes time and code space to continually switch banks for each register access, Cypress follows a convention aimed at reducing the number of required bank switches. User module APIs, library functions, and C code generated by PSoC Designer assume that Register Bank 0 is selected when functions are called and leave Register Bank 0 selected when the function is exited.

For this coding convention to work, programs that call Cypress APIs and Library functions must guarantee that Register Bank 0 is selected before they are called. You can ensure this by following the same convention and immediately setting the Register Bank back to 0 after every Register Bank 1 access.

The C compilers that accompany PSoC Designer use the address of the register to determine in which bank that register is located. The register definitions found in *m8c.h* have 0x100 added to the address for all Bank 1 registers. Because there are only 256 addresses per bank, the C compiler drops the 0x100 from the addresses and generates the correct address for the register. But the 0x100 tells the compiler that it is a Bank 1 register and before accessing the register, the compiler automatically switches to Bank 1 and switches back to Bank 0 after the access. Therefore, it is not necessary to switch banks when writing in C.

Interrupt Service Routines (ISRs) automatically follow this convention because of the nature of the interrupt hardware. The bit that controls the register bank selection is contained in the Flag register. When an interrupt is serviced, the content of the Flag register is pushed on the stack and the register is cleared. Clearing the Flag register selects Register Bank 0.

Therefore, Bank 0 is selected at the beginning of an ISR. When RETI is executed to return from the interrupt, the Flag register is restored by popping it off the stack, which restores the register bank selection to the state before the interrupt was processed.

When changing Register banks, selection should be done through the M8C_SetBank0 and M8C_SetBank1 macros provided by Cypress in *m8c.inc* and *m8c.h*.

Address Equates in M8C.INC and M8C.H

An include file for assembly language programming and a header file for C programming are provided by PSoC Designer. They provide equates for all the register addresses. Common mask values and some system macros are also provided. These assignments should be used whenever possible. The M8C files also contain macros that should be used.

CPU Register Contents

The CPU registers A and X should be treated as volatile and may not contain the same value after a function call as it did before the call. It is the responsibility of the designer to preserve any important register contents through a call to an API or library function. Any register may be changed within an API or library function and the designer should not assume that the contrary is true.

In fact, even if A and/or X are currently preserved through a function call, there is no guarantee that they will continue to be preserved in future versions of the function that are distributed as part of PSoC Designer.

Because the A and X registers are volatile, they are not pushed and popped within a standard assembly function (however, they need to be pushed and popped if they are used within an ISR). This leads to more efficient code.

The A and X registers are only pushed and popped (by the designer's code) if they need to be preserved, rather than pushing and popping every time.

Reserved and Unused bits

In some of the PSoC 1 control registers there are bits that have reserved values or are unused. When writing to registers that contain reserved or unused bits, the value of those bits should be set to 0. This enables the designer's code to be backward compatible. Cypress internal best practices are to designate the default state of any new function or feature that currently use reserved bits to be 0. This is a design goal, and not a guarantee of future implementations, but it should be assumed to be true when dealing with reserved or unused bits.

Currently, all reserved bits read 0, so read-modify-write functions should use the following masks when dealing with reserved or unused bits.

- XOR – use 0 in the reserved bit position
- OR – use 0 in the reserved bit position

- TST – use 0 in the reserved bit position
- AND – use 1 in the reserved bit position

The AND mask setting may be unexpected. The initial state of the bit is 0, so ANDing it with 1 results in 0. However, if a new feature is added and the designer wants to take advantage of the new feature, making it a 1 in the AND mask enables the bit to stay on when the AND is executed. The designer does not have to hunt down all the places where the register is being ANDed.

All Char are Unsigned in PSoC Designer

PSoC Designer's C compiler defines "char" as "unsigned char." This is opposed to some other conventions like Microsoft C, and can confuse first time PSoC users.

User Module Start Functions are Required

All user module APIs include a _Start function. The `usermodulename_Start()` API function must be called to start the PSoC 1 blocks used by the user module. Analog user modules must be called with a power setting passed to the Start function.

User Module Starting Sequence

It is recommended that user modules connected in a chain of functions have the user module at the end of the chain started first and the user module at the start of the chain started last. This leads to more predictable behavior.

This is a requirement for a chain of user modules that use the 48 MHz clock (SysClk *2) as the source for the first block in the chain.

Interrupt Latency

When using interrupts, interrupt latency (the time from an event that triggers an interrupt to when the code in the ISR starts execution), is a factor that must be looked at carefully.

The interrupt latency for PSoC 1 is made of three components; interrupt recognition, instruction completion, and interrupt response.

Interrupt Recognition*:* Interrupts are not recognized when globally disabled (M8C_DisableGInt). All interrupts that occur when Global Interrupts are disabled, are queued up as pending interrupts and the highest priority interrupt is serviced first when interrupts are globally enabled (M8C_EnableGInt). Therefore, if the foreground program disables interrupts for a period of time, this time adds to the interrupt latency. Interrupts are also automatically disabled within interrupt routines. So, any time spent in interrupt routines adds to interrupt latency. (M8C_EnableGInt can be used within an interrupt routine to decrease latency if the design enables it).

Instruction Completion*:* When an interrupt occurs, the instruction currently being executed is completed before the interrupt is serviced. The longest instruction in the PSoC 1 instruction set is 13 CPU cycles. If an interrupt occurs just as one such instruction began execution, 12+ CPU cycles are added to the interrupt latency. Because you cannot tell when an interrupt occurs, consider the worst-case instruction when reviewing interrupt latency.

Interrupt Response*:* The time to "execute" the interrupt is 13 CPU clocks. After 13 CPU clocks, the program is at the interrupt vector in *boot.asm* where a LJMP instruction (7 CPU clocks) takes control to the ISR. This results in a total of 20 CPU clocks to reach the ISR where the designer's code is located.

If interrupts occur at a rate that is faster than the interrupts can be serviced, no foreground instructions are executed. This is because global interrupts are enabled by the RETI instruction and the next interrupt is serviced when the RETI instruction has been completed.

Because the timing of the interrupt enables and disables, enabling global interrupts and immediately disabling interrupts results in only the highest priority interrupt being executed. However, if an instruction is inserted between the enable and disable, it results in all pending interrupts being serviced.

Interrupt Errata

There are two conditions associated with interrupts that can lead to unexpected resets. Both conditions are through the same mechanism.

Clearing pending interrupts by writing to the INT_VC register, just as an interrupt occurs, results in the interrupt process being started. But when it is time to jump to the interrupt vector, it is zero (the boot vector) because the INT_VC register now contains 0x00.

The other case is when an individual interrupt is disabled just as that interrupt occurs and there are no other pending interrupts. This too, results in the interrupt process being started with 0x00 in the INT_VC register. Both these cases are listed in the Silicon Errata document. The second case is avoided by using the macro provided in *M8C.inc* and *M8C.h* to disable individual interrupts. The macro contains a workaround.

Changing boot.asm through boot.tpl

There are times when *boot.asm* must be modified. Cypress's goal is to architect PSoC Designer so that users do not have to edit *boot.asm*. But in some cases it cannot be avoided. If *boot.asm* must be edited, it is important to know that *boot.asm* is a generated file that is overwritten every time "Generate Application" is executed. The *boot.asm* file is generated from a template called *boot.tpl*. Therefore, it is necessary to edit *boot.tpl* when changes to *boot.asm* are wanted.

Archiving Projects

When a project is completed, it is common to archive the source files. With PSoC Designer, it is important to also archive the version of PSoC Designer used to create the project. Each new version of PSoC Designer can have changes in the template files, user modules, and in the compiler. All these prevent recreating the original project if that is needed.

Dynamic Reconfiguration

When using the dynamic reconfiguration capability of PSoC Designer, it is important to stop all the user modules in a configuration before it is unloaded. In the process of unloading a configuration, unexpected conditions and/or glitches could occur, as different parts of the configuration are unloaded. It is also recommended to globally disable interrupts (M8C_DisableGInt) before unloading a configuration.

Watch Dog Timer (WDT) Duration

The Watch Dog Timer duration is not three times the Sleep Timer period. The WDT times out after the Sleep Timer reaches Terminal Count three times between WDT refreshes. This means, that if the WDT is refreshed just one 32 k clock tick before the Sleep Timer reaches TC, then the WDT can expire two sleep periods plus one 32 k clock period later. This must be taken into account when designing with the WDT. If the period of the Sleep Timer is not critical (for example, there is no real time clock running from the Sleep Timer), then use the ClearWDTandSleep function to refresh the WDT. This results in some Sleep Timer periods being shortened, but also results in the WDT duration at three times the Sleep Timer period.

Debug and Test

Power Settings for Analog User Modules

For analog user modules, begin testing by setting them to Full_Power. After the initial functionality is confirmed, changes can be made to reduce power consumption, if needed, and the performance compared. This helps avoid dealing with the effects of an underpowered user module when first bringing the project up.

Halting ADCs during Debug

Halting the PSoC 1 in the Debugger stops the M8C CPU. It does not stop the digital clocks and it does not “pause” the analog. Therefore, when “Run” is pressed, the state of the user modules may be unpredictable. A specific case of this is the ADCINC. When the CPU is halted, the counter (which accumulates data from that analog portion) continues to run. This means that when the Debugger is started again, the counter probably rolled over many times and contains numbers that are not related to the input to the ADC. Therefore, if a designer halts the CPU every time through a loop that collects ADC data, then that data gets corrupted.

There are a number of ways to check the ADC data in the Debugger. One way is to store data in an array so that multiple samples can be viewed at once. Another way is to make a loop to collect multiple samples per loop and then halt outside that loop. This causes the most recent data to be “good” and the corrupted data to not be seen. A third method is to use Events to halt every other time an address is executed. This is the same as looping multiple times, but does not require the project code to be changed. This condition holds true for all the ADC User Modules in PSoC Designer.

Missed Break Points

If there is a breakpoint in the foreground code of a program and it is not hit by the Debugger (even though there are no branches and it appears that there is no way to miss it), then it is possible that there is a problem with interrupts. If there is an ISR that takes too long to execute, then there is likely another interrupt pending before the ISR has been completed. In this case, the CPU would spend all its time in the ISR and will not execute any foreground code.

Control Register Writes

If writing to a Control register does not seem to have the expected effect, make sure that the correct bank is selected at the time the register write occurs.

Reducing Power Use

A few simple things can be done to reduce power consumption, as described below:

- Set VC1 and VC2 both to 16 if they are not used. If they are used, set them to the largest values that can be used, starting with VC1.
- Set the CPU clock speed as slow as possible for the project. From experience, 3 MHz seems to be the speed where the most benefit is found. Going slower than 3 MHz does not save as much power as changing from 6 MHz to 3 MHz.
- Be sure not to let the analog output buffers float. The analog output buffers (P0[2], P0[3], P0[4] and P0[5]) should not be enabled without a source to drive them. If this is done, it can lead to a large current drain (10s of milliamps).

Common Mistakes

Remember these points to avoid some simple mistakes:

- VC1 divides the 24 MHz clock, not the CPU clock.
- VC1 and VC2 do not have 50 percent duty cycle outputs. The outputs are pulses that are half cycles from the 24 MHz clock.

- Supervisor calls use RAM locations 0xF8-0xFF. Therefore, if supervisor functions are used (FlashWriteBlock and the E2PROM User Module use supervisor functions), those memory locations must be left unused.
- The ICE can only supply 5 V power. If 3.3 V is needed, then the power must be supplied from the target board.
- The C compiler generates big endian code – the Most Significant Byte (MSB) comes first in RAM.
- In the loadable configurations for dynamic reconfiguration, each overlay must contain the items that are different from the base configuration. PSoC Designer generates load and unload routines based on the differences between the configuration and the base configuration.
- **ClockSync** parameter in Digital block based user modules should be configured with a clock whose frequency is at least double as compared to clock input to the digital block. Use **SysClk Direct** option overrides the setting for Clock source and instead uses SysClk directly as the input clock to the user module.

Summary

This application note provided guidelines to avoid the issues and to simplify creation and debugging of PSoC Designer projects.

Related Application Notes

Getting Started with PSoC® 1 – AN75320

PSoC® 1 Getting Started with GPIO – AN2094

PSoC® 1 Interrupts – AN90833

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

Debugging with PSoC® 1 – AN73212

About the Author

Name:	Jeffrey Dahlin
Title:	Applications Engineer Principal
Name:	Rajiv Vasanth Badiger
Title:	Applications Engineer Staff

Appendix A

This appendix contains all relevant information from this document that pertains to the legacy 25/26xxx PSoC 1 parts. Although the 25/26xxx PSoC part family is now obsolete, the information contained in this appendix may prove useful for designers still using this part.

User Module Clock Limitations

In addition to the user module clock limitations listed previously in this document, Digital blocks in the legacy 25xxx/26xxx PSoC family of parts have additional limitations on their maximum clock frequencies:

- User modules greater than 16 bits in length have a maximum frequency of 24 MHz.
- At 3.3 V, the maximum digital block frequency is also 24 MHz.

Clock Routing

Unlike the 22/24/27/29xxx part families, which have a broadcast bus for each digital block row, the legacy 25/26xxx parts only have Digital Block DCA03 as a broadcast block that can be used as a clock source for any of the other blocks.

Pin Allocation

In the legacy 25xxx/26xxx PSoC family of parts, there are fewer pin types and configuration options, as shown in Table 2.

Table 2. CY8C25xxx and CY8C26xxx Pin Types

	Digital I/O	Analog In	Analog Out	Dir Analog In	External Ref.	Ext. Crystal	
Port1[2:7]	☆						Digital I/O only, use these pins first for GPIO, Global In and Global Out signals.
Port2[5]	☆						
Port2[7]	☆						
Port3[0:7]	☆						
Port4[0:7]	☆						
Port5[0:3]	☆						
Port0[0:1]	☆	☆					Digital I/O or Analog Input, use these analog in's first
Port0[6:7]	☆	☆					
Port0[2:5]	☆	☆	☆				Digital I/O or Analog I/O
Port2[0:3]	☆			☆			Digital I/O or Direct Analog Inputs
Port2[4]	☆				☆		Digital I/O or External references
Port2[6]	☆				☆		
Port1[0:1]	☆					☆	Digital I/O or External Crystal connection

Document History

Document Title: AN2010 - PSoC® 1 Best Practices and Recommendations

Document Number: 001-40401

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1532004	JVY	10/02/07	1. New publication of existing application note.
*A	2642011	JVY	01/21/09	1. Added Appendix A 2. Added CY8C25xxx and CY8C26xxx Pin Table 3. Removed CY8C25xxx and CY8C26xxx and added CY8C29xxx in Associated Part Family
*B	3172764	GIR	02/14/11	1. Updated title. 2. Updated abstract. 3. Extensive formatting and content update throughout every section of the document.
*C	3232507	GIR	04/18/11	1. Added associated application notes 2. Minor template updates 3. Minor updates in Register Bank Selection, and Changing boot.asm through boot.tpl section
*D	3276246	GIR	06/07/11	No change.
*E	4463968	RJVB	08/02/2014	Categorized the sections. Added section on flash. Updated Summary. Updated in new template. Completing Sunset Review.
*F	5713593	AESATMP9	04/24/2017	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, 2007-2017. 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.