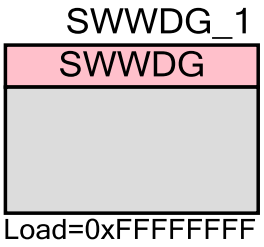
Software Watchdog (PDL_SWWDG) 1.0



Features

- Provides window watchdog mode
- Used for a count clock
- Counts cycles while CPU program is operating
- Retains count value after returning from standby mode
- Stopped by all resets

General Description

The Peripheral Driver Library (PDL) Software Watchdog Timer (PDL_SWWDG) component is a function to detect runaway of a user program. If the watchdog timer is not reloaded within the specified interval time, it judges that a user program is out of control, and outputs either a system reset request or an interrupt request to the CPU.

During watchdog timer operation, it is required to continually and periodically “feed” the watchdog before the specified interval time has elapsed. If an abnormal operation of a user program, such as hanging up, prevents it from being periodically reloaded, the timer continues counting down, underflows, and outputs a watchdog interrupt request or a watchdog reset request.

This component uses firmware drivers from the PDL_SWWDG module, which is automatically added to your project after a successful build.

When to Use a PDL_SWWDG Component

Use the SWWDG to detect runaway condition in your firmware. The watchdog timers can be reloaded by calling function Swwdg_Feed() . When the first underflow of the watchdog counter is generated, an interrupt request is generated. When the second underflow is generated without clearing the interrupt request, a reset request is generated.

Quick Start

1. Drag a PDL_SWWDG component from the Component Catalog FMx/System/Software Watchdog folder onto your schematic. The placed instance takes the name SWWDG_1.
2. Double-click to open the component’s Configure dialog.

3. On the **Basic** tab set the following parameters:
   - load value
   - enable reset request if needed
   - pfnSwwdgIrqCb - specify the interrupt callback function or clear it if not used

   Note The SWWDG interrupt fires whether the callback is declared or not. The interrupt is enabled when watchdog timer is started
4. Build the project to verify the correctness of your design. This will add the required PDL modules to the Workspace Explorer and generate configuration data for the SWWDG_1 instance.
5. In the *main.c* file, initialize the peripheral and start the application:

```
Swwdg_Init(&SWWDG_1_Config));
Swwdg_Start();
Swwdg_Feed();/* Call this API to clear Irq and Reset Timer */
```

6. Build and program the device.

Component Parameters

The PDL_SWWDG component Configure dialog allows you to edit the configuration parameters for the component instance.

General Tab

This tab contains the component parameters used in the general peripheral initialization settings.

Parameter Name	Description
bResetEnable	Enable/disable the software watchdog reset
u32LoadValue	Timer interval – number of clock cycles before reset
bWinWdgEnable	Enable/disable timing window mode
bWinWdgResetEnable	Reset or interrupt on timeout
pfnSwwdgIrqCb	Callback function for software watchdog. Note: this generates a declaration only - USER must implement the function.
u8TimingWindow	Allow reload at less than or equal to a percentage of WDOGLOAD

Component Usage

After a successful build, firmware drivers from the PDL_SWWDG module are added to your project in the pdl/drivers/swwdg folder. Pass the generated data structures to the associated PDL functions in your application initialization code to configure the peripheral.

Generated Data

The PDL_SWWDG component populates the following peripheral initialization data structure(s). The generated code is placed in C source and header files that are named after the instance of the component (e.g. *SWWDG _1_config.c*). Each variable is also prefixed with the instance name of the component.

Data Structure Type	Name	Description
stc_swwdg_config_t	SWWDG _1_Config	Configuration structure

Once the component is initialized, the application code should use the peripheral functions provided in the referenced PDL files. Refer to the PDL documentation for the list of provided API functions. To access this document, right-click on the component symbol on the schematic and choose "**Open API Documentation…**" in the drop-down menu.

Data in RAM

The generated data may be placed in flash memory (const) or RAM. The former is the more memory-efficient choice if you do not wish to modify the configuration data at run-time. Under the **Built-In** tab of the Configure dialog set the parameter CONST_CONFIG to make your selection. The default option is to place the data in flash.

Interrupt Support

The PDL_SWWDG component always generates an interrupt on a timer underflow. The IRQ handler calls a callback function if it is defined. If the name of a callback function is specified in pfnSwwdgIrqCb the function definition will be generated. The user is required to write the definition (implementation). If an empty string is provided to pfnSwwdgIrqCb the callback declaration is not generated and the handler does not make a function call. The component generates the following function declarations.

Function Callback	Description
SWWDG_1_SwwdgIrqCb	Interrupt callback function. Note: this generates a declaration only - USER must implement the function. If you don't need to use callback function declared by the component, clear pfnSwwdgIrqCb parameter in the component's configuration dialog.

Code Examples and Application Notes

There are numerous code examples that include schematics and example code available online at the Cypress Code Examples web page.

Cypress also provides a number of application notes describing how FMx devices can be integrated into your design. You can access the Cypress Application Notes search web page at www.cypress.com/appnotes.

Resources

The PDL_SWWDG component uses the Software Watchdog peripheral block.

References

- FM0+ Family of 32-bit ARM® Cortex®-M0+ Microcontrollers Peripheral Manuals
- Cypress FM0+ Family of 32-bit ARM® Cortex®-M0+ Microcontrollers

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.0	Initial Version	

© Cypress Semiconductor Corporation, 2016. 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.