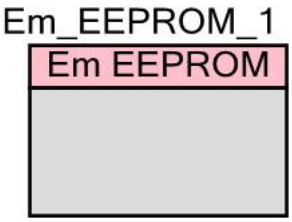
Emulated EEPROM
1.10



Features

- Provides EEPROM-like non-volatile storage
- Supports PSoC 3, PSoC 4, and PSoC 5LP devices

General Description

The Emulated EEPROM component emulates an EEPROM device in the flash memory of a PSoC, providing simplified access to non-volatile memory.

When to Use a Emulated EEPROM

The Emulated EEPROM component should be used when:

- Data needs to be preserved across power cycles, and the target device does not have dedicated EEPROM memory.
- The target device has dedicated EEPROM memory, but the data that needs to be preserved is too large for the embedded EEPROM array.

Input/Output Connections

There are no I/O connections for the Emulated EEPROM component. It is an API only.

Component Parameters

The EEPROM has no configurable parameters other than standard **Name** and **Built-in** parameters.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "Em_EEPROM_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "Em_EEPROM."

NOTE: Data (variables or arrays) used with this component must be pre-initialized with some values. Not following this requirement will cause incorrect data placement and broken component functionality.

Functions

Function	Description
Em_EEPROM_Start()	Empty function, included for consistency with other components.
Em_EEPROM_Stop()	Empty function, included for consistency with other components.
Em_EEPROM_Write()	Writes the specified number of bytes from the source buffer in SRAM to the emulated EEPROM array in flash, without modifying other data in flash.

void Em_EEPROM_Start(void)

Description: Empty function. Included for consistency with other components.

Parameters: None

Return Value: None

Side Effects: None

void Em_EEPROM_Stop(void)

Description: Empty function. Included for consistency with other components.

Parameters: None

Return Value: None

Side Effects: None

cystatus Em_EEPROM_Write (const uint8 srcBuf[], const uint8 eepromPtr[], uint16/uint32 byteCount)

Description: Writes the specified number of bytes from the source buffer in SRAM to the emulated EEPROM array in flash, without modifying other data in flash.

Parameters: srcBuf: Pointer to the SRAM buffer holding the data to write.

eepromPtr: Pointer to the array or variable in flash representing the emulated EEPROM.

byteCount: Number of bytes to write from srcBuf to eepromPtr (uint16 on PSoC 3, uint32 on PSoC 4 and PSoC 5LP).

Return Value:

Value	Description
CYRET_SUCCESS	Successful write.
CYRET_BAD_PARAM	Request to write outside the flash boundary.
CYRET_UNKNOWN	Other error in writing flash.

Side Effects: This function flushes instruction cache after writing into the emulated EEPROM is complete.

Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Example Project" topic in the PSoC Creator Help for more information.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined: project deviations – deviations that are applicable for all PSoC Creator components and specific deviations – deviations that are applicable only for this component. This section provides information on component specific deviations. The project deviations are described in the MISRA Compliance section of the System Reference Guide along with information on the MISRA compliance verification environment.

The Emulated EEPROM component does not have any specific deviations.

API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.

The measurements have been done with associated compiler configured in Release mode with optimization set for Size. For a specific design the map file generated by the compiler can be analyzed to determine the memory usage.

Configuration	PSoC 3 (Keil_PK51)		PSoC 4 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Emulated EEPROM	593	0	128	0	168	0

Functional Description

The Emulated EEPROM component is purely a firmware abstraction of the flash memory. It consumes no other resources.

The Emulated EEPROM component does not correspond to a single instance of a virtual EEPROM. Instead, it provides a write function to safely interact with variables and arrays defined in flash. The Em_EEPROM_Write() function is a simple way to modify user variables, defined in flash, so that variable values will be saved between power cycles.

Flash memory is readable by byte, like other memory spaces. Read access to the emulated EEPROM is achieved by directly reading the underlying flash variable or array.

The Em_EEPROM_Write() API handles writes that are not aligned to the beginning/end of a flash row. Data that resides in the same flash row as the location of the emulated EEPROM data is not affected by writes to the emulated EEPROM data. This is accomplished by first reading the contents of a shared row into a buffer on the stack, modifying the data targeted by the Write(), and writing the buffer back to flash.

In designs that use the ECC memory for data storage (on devices that have ECC memory), the ECC memory is not affected by writes to the emulated EEPROM.

In order to force the compiler to locate data in Flash, variables/arrays should be declared as "*static const (CYCODE for PSoC 3)*" and initialized with some value at the same time. Reads from Flash are recommended to be done using "*(volatile)*" type qualifier to prevent the compiler from optimizing reads from static variables. For details on component usage please refer to Emulated EEPROM example project.

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted.
Specifications are valid for 1.71 V to 5.5 V, except where noted.

DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
	Erase and program voltage	V_{DDD} pin	1.71	-	5.5	V

AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
T_{WRITE}	Row write time		15	-	-	ms
	Flash data retention time, retention period measured from last erase cycle	Average ambient temp. TA ≤ 55 °C, 100 K erase/program cycles	20	-	-	years
		Average ambient temp. TA ≤ 85 °C, 10 K erase/program cycles	10	-	-	

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.10.b	Minor datasheet update.	
1.10.a	Datasheet update.	Added note about pre-initializing variables or arrays.
1.10	Fixed incorrect flash array id and row id calculation for devices that have more than one flash array id.	Em_EEPROM_Write() function fails if the eepromPtr points to an address in flash that corresponds to an array id other than zero.
	Added flush instruction cache after flash write into emulated EEPROM complete.	Reading back the flash data that was just written might return cached data instead of reading flash content when instruction cache is enabled.
1.0.a	Added DC/AC Electrical Characteristics section.	To provide component operating characteristics.
1.0	First component version.	First component version.

© Cypress Semiconductor Corporation, 2013-2016. 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.