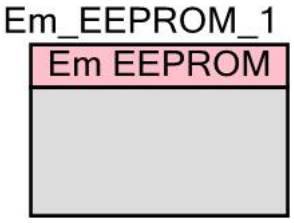
Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

Emulated EEPROM (Em_EEPROM)

2.20



Features

- EEPROM-Like Non-Volatile Storage
- Easy-to-use Read and Write API Functions
- Optional Wear Leveling
- Optional Redundant EEPROM Copy Storage

General Description

The Emulated EEPROM Component emulates an EEPROM device in the PSoC device flash memory.

On PSoC 6 devices, the Em_EEPROM Component is a graphical configuration entity built on top of the Cy_Em_EEPROM middleware library. This library operates on the top of a flash driver available in the Peripheral Driver Library (PDL). On non-PSoC 6 devices, the Em_EEPROM Component is the same graphical configuration entity built on top of the Em_EEPROM Dynamic design-wide resource Component.

When To Use Em_EEPROM Component

The Emulated EEPROM Component should be used to store nonvolatile data on a target device.

Quick Start

Note For PSoC 6 devices, the Em_EEPROM operates on top of the flash driver. The flash driver has some prerequisites for proper operation. Refer to the *Flash System Routine (Flash)* section of the *PDL API Reference Manual* (PSoC Creator **Help** menu **> Documentation > Peripheral Driver Library**).

1. Drag the Em_EEPROM Component from the Component Catalog *System/Emulated EEPROM* folder onto your schematic. The placed instance takes the name of Em_EEPROM_1.
2. Double-click to open the Configure dialog.
3. Select the desired parameter settings. For more details on the parameters, refer to the Component Parameters section of this datasheet.

4. For PSoC 6 devices, by default the Component allocates the required amount of memory for the Em_EEPROM storage in the Emulated EEPROM flash area. There is an option to switch between the Emulated EEPROM flash area and main flash (user flash).
   - If the **Use Emulated EEPROM** option is set to "Yes," the *Em_EEPROM_1_em_EepromStorage[]* is declared as the EEPROM storage and available for use. Pass the address of the storage to the *Cy_Em_EEPROM_Init()* function if you do not want to use the Component API functions.
   - If the **Use Emulated EEPROM** option is set to "No," follow the action described below for non-PSoC 6 devices.

   For non-PSoC 6 devices, you must statically allocate the memory that will be used for Em_EEPROM storage.

   To do this, declare an array in flash aligned to the size of the device flash row. The following is an example of such array declaration for GCC and MDK compilers:

   ```
   const uint8 emEeprom[Em_EEPROM_1_PHYSICAL_SIZE]
    __ALIGNED(CY_FLASH_SIZEOF_ROW) = {0u};
   ```

   The same for the IAR compiler:

   ```
   #pragma data_alignment = CY_FLASH_SIZEOF_ROW
   const uint8 emEeprom[Em_EEPROM_1_PHYSICAL_SIZE] = {0u};
   ```

   For PSoC 3 devices, the Keil compiler doesn't support data aligning. You must place the array to the flash memory address aligned to the flash row size. The following declaration is used:

   ```
   const uint8 CYCODE emEeprom[Em_EEPROM_1_PHYSICAL_SIZE] _at_ 0x1000u;
   ```

   The value "0x1000u" in the declaration defines the start address where `emEeprom[]` will be placed. The address can be any value aligned to 0x100 (the size of a flash row on PSoC 3).

   The `Em_EEPROM_1_PHYSICAL_SIZE` is calculated by the Component. This constant defines the amount of memory dedicated for Em_EEPROM based on the user configuration entered in the Configure dialog.

5. After Em_EEPROM storage is defined, pass the address to the Component. Use the following code for that purpose:

   ```
   cy_en_em_eeprom_status_t returnValue;
   returnValue = Em_EEPROM_1_Init(&emEeprom[0]);
   ```

   Note For PSoC 6 devices, when the **Use Emulated EEPROM** option is set to "Yes," the Em_EEPROM address will be overwritten with storage from Emulated EEPROM flash area. In this case, the address may be passed as zero.

Note If the Em_EEPROM is planned to be used on one specific core, you must modify the linker scripts. For more information, refer to the *Middleware/Cypress Em_EEPROM Middleware* section of the PDL documentation. To access this document, go to the PSoC Creator **Help** menu **> Documentation > Peripheral Driver Library**.

6. Build the project in order to verify the correctness of your design. For PSoC 6, add the required PDL modules to the Workspace Explorer, and generate the configuration data for the Em_EEPROM_1 instance.
7. Program the device.

Placing EEPROM Storage at Fixed Address

EEPROM storage can be allocated at a fixed address in flash. To do this, you must modify the linker control file (linker script). This requires a fundamental knowledge of the linker control file, because there is a risk of receiving a linker error while building the project if you make some improper modifications.

The Keil C51 compiler doesn't have a linker control file. However, it allows placement of data at a fixed address from source code as shown in the Quick Start section.

The steps below describe how to update the linker control file for GCC and MDK compilers for PSoC 4/PSoC 5LP devices and PSoC 6 devices. There is a separate process for the IAR compiler.

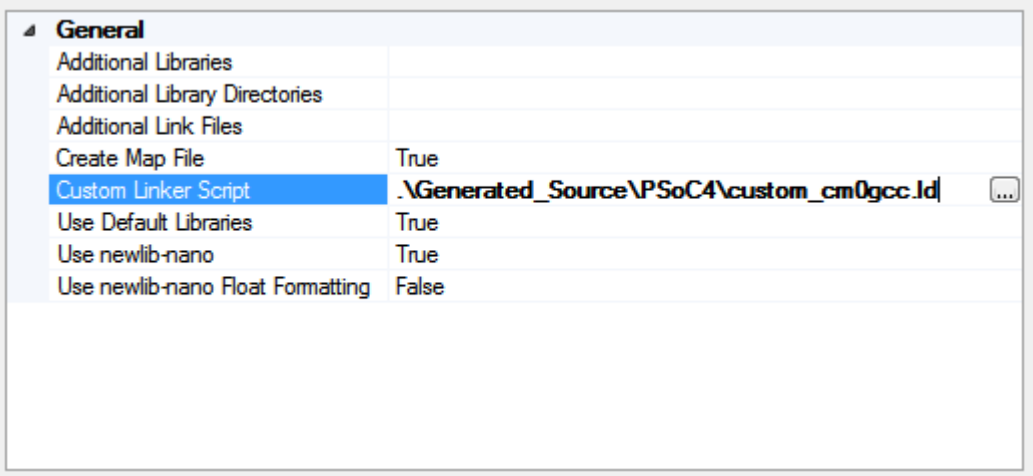
This approach demonstrates adding EEPROM storage reservation in flash after the application. You must calculate the application end address and select the address of the EEPROM storage so that the memory spaces of the storage and application won't overlap. You might also add some offset between the application end address and the EEPROM storage start address to ensure there is extra space in case the project code grows.

PSoC 4/PSoC 5LP

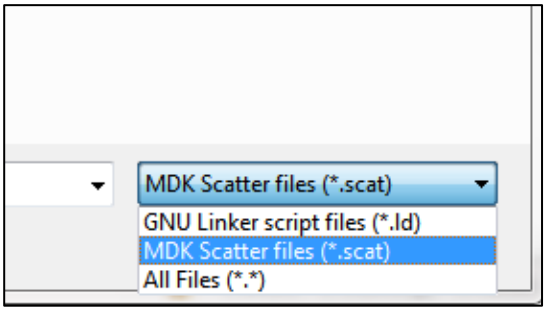
1. Drag the Em_EEPROM Component from the Component Catalog *System/Emulated EEPROM* folder onto your schematic.
2. Double-click to open the Configure dialog and select the desired parameter settings. For more details about the parameters, refer to the Component Parameters section of this datasheet.
3. Build the project. This is required for a linker script to be generated.
4. Go to the linker script directory. It is located at \<*em_eepromproject_name*>*.cydsn**Generated_Source**PSoC(4)(5)*\.
    - For the GCC compiler, the linker control file name is *cm0gcc.ld* / *cm3gcc.ld.*
    - For the MDK compiler, the linker control file name is *Cm0RealView.scat* / *Cm3RealView.scat.*

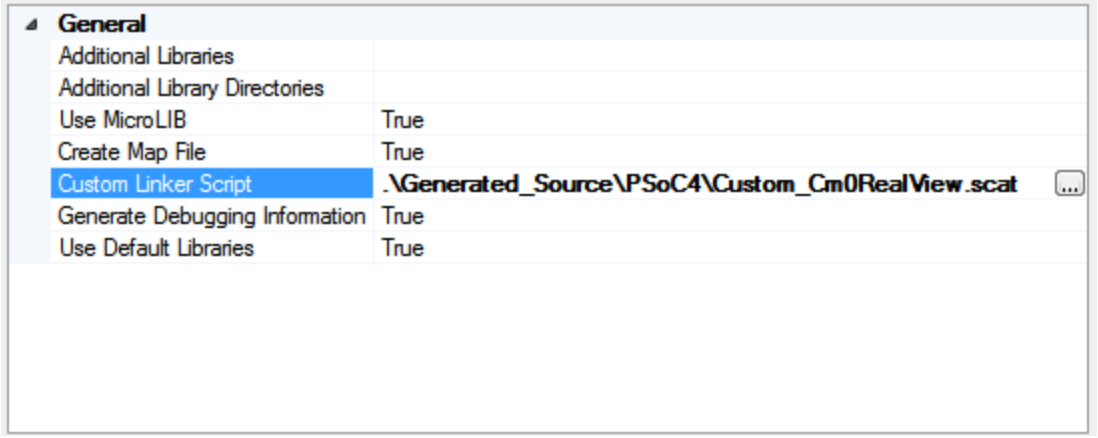
5. Select the required linker script, copy it, and rename it to the same directory. For example: *cm0gcc.ld -> custom_cm0gcc.ld (Cm0RealView.scat -> Custom_Cm0RealView.scat).*
6. Open the Build Settings dialog and select the **Linker** node under the selected compiler. Then, select the **Custom Linker Script** field and click the ellipsis **[…]** button to select the appropriate file.

   For **GCC**, select the "*custom_cm0gcc.ld*" file:



   For **MDK**, change the file type to *.scat, and then select the "*Custom_Cm0RealView.scat*" file:





Note If you want to switch between **Debug** and **Release** project configurations, you need to do these steps for both of the configurations.

7. For the GCC compiler, open the custom linker script (*custom_cm0gcc.ld*) and search for the following declaration:

```
.cy_checksum_exclude : { KEEP(*(.cy_checksum_exclude)) } >rom
```

   a. Paste the following code right after the declaration:

```
EM_EEPROM_START_ADDRESS      = <EEPROM Address>;
.my_emulated_eeprom EM_EEPROM_START_ADDRESS :
{
    KEEP(*(.my_emulated_eeprom))
} >rom
```

      - `<EEPROM Address>` – This is an absolute address in flash where the EEPROM should start. You must define the address value. The address should be aligned to the size of the device's flash row and should not overlap with the memory space used by the application.
      - `my_emulated_eeprom` – This is the name of the section where the EEPROM storage will be placed. The name can be changed to any name you choose.

   b. Save the changes and close the file.

8. For the MDK compiler, open the custom linker script (*Custom_Cm0RealView.scat*) and search for the following declaration:

```
APPLICATION APPL_START (CY_FLASH_SIZE - APPL_START)
{
    ...
}
```

a. Paste the following code right after the declaration:

```
#define EM_EEPROM_START_ADDRESS      <EEPROM Address>
EM_EEPROM (EM_EEPROM_START_ADDRESS)
{
    .my_emulated_eeprom+0
    {
        *(.my_emulated_eeprom)
    }
}
```

- `<EEPROM Address>` – This is an absolute address in flash where the EEPROM should start. You must define the address value. The address should be aligned to the size of the device's flash row and should not overlap with the memory space used by the application.
- `my_emulated_eeprom` – This is the name of the section where the EEPROM storage will be placed. The name can be changed to any name you choose.

b. Save the changes and close the file.

9. Similar to Step 4 of Quick Start section, declare EEPROM storage in the newly created section. To do this, declare an array in flash, aligned to the size of the flash row of the device you are using. An example of such array declaration is following:

```
const uint8 emEeprom[Em_EEPROM_1_PHYSICAL_SIZE]
CY_SECTION(".my_emulated_eeprom") __ALIGNED(CY_FLASH_SIZEOF_ROW) = {0u};
```

10. After Em_EEPROM storage is defined, pass the address to the Component. Use the following code for that purpose:

```
cy_en_em_eeprom_status_t returnValue;
returnValue = Em_EEPROM_1_Init(&emEeprom[0]);
```

11. Build the project in order to verify the correctness of the linker control file modifications.

PSoC 6

1. Drag the Em_EEPROM Component from the Component Catalog *System/Emulated EEPROM* folder onto your schematic.
2. Double-click to open the Configure dialog and select the desired parameter settings. For more details on the parameters, refer to the Component Parameters section of this datasheet.
3. Build the project. This is required for linker scripts to be generated.
4. Go to the linker script directory. It is located at \<*em_eeprom_project_name*>.cydsn\.
   - For the GCC compiler, the linker control file name is *cy8c6xx7_cm0plus.ld* or *cy8c6xx7_cm4_dual.ld.*

- For the MDK compiler, the linker control file name is *cy8c6xx7_cm0plus.scat or cy8c6xx7_cm4_dual.scat.*

The name depends on the core on which the Em_EEPROM code will be run:CM0+ or CM4.

5. For the GCC compiler, open the linker script and search the following declaration:

```
etext =  . ;
```

   a. Paste the following code right after the declaration:

```
EM_EEPROM_START_ADDRESS      = <EEPROM Address>;
.my_emulated_eeprom EM_EEPROM_START_ADDRESS :
{
    KEEP(*(.my_emulated_eeprom))
} > flash
```

      – `<EEPROM Address>` – This is an absolute address in flash where the EEPROM should start. You must define the address value. The address should be aligned to the size of the device's flash row and should not overlap with the memory space used by the application.

        Note Do not write to and read/execute from the same flash sector at the same time. This is true for all sectors. For more details, refer to the PDL reference for Flash System Routine (Flash).

      – `my_emulated_eeprom` – This is the name of the section where the EEPROM storage will be placed. The name can be changed to any name you choose.

   b. Save the changes and close the file.

6. For the MDK compiler, open the custom linker script and search for the following declaration:

```
LR_FLASH FLASH_START FLASH_SIZE
{
...
}
```

   a. Paste the following code right after the declaration:

```
#define EM_EEPROM_START_ADDRESS      <EEPROM Address>
EM_EEPROM (EM_EEPROM_START_ADDRESS)
{
    .my_emulated_eeprom+0
    {
        *(.my_emulated_eeprom)
    }
}
```

- `<EEPROM Address>` – This is an absolute address in flash where the EEPROM should start. You must define the address value. The address should be aligned to the size of the device's flash row and should not overlap with the memory space used by the application.

  Note Do not write to and read/execute from the same flash sector at the same time. This is true for all sectors. For more details, refer to the PDL reference for Flash System Routine (Flash).

- `my_emulated_eeprom` – This is the name of the section where the EEPROM storage will be placed. The name can be changed to any name you choose.

b. Save the changes and close the file.

7. Follow the same process from Step 9 through Step 11 under PSoC 4/PSoC 5LP to make appropriate changes and build the project.

IAR compiler

To use the IAR compiler, you must export your PSoC Creator design to the IAR IDE. Refer to the PSoC Creator Help "Integrating into 3rd Party IDEs" for information about how to do this.

After the design has been integrated with the IAR IDE, build the project in IAR to verify the correctness of the steps performed. Then, follow these steps to place the EEPROM Storage at a fixed address.

1. In the IAR IDE, open the IAR linker control file. Depending on the device used, the linker control file name should be *Cm0Iar.icf* (PSoC 4), *Cm3Iar.icf* (PSoC 5LP), *cy8c6xx7_cm0plus.icf* (PSoC 6 CM0+), or *cy8c6xx7_cm4_dual.icf* (PSoC 6 CM4).
2. Find the following record:
   - For PSoC 4/PSoC 5LP:

     ```
     "APPL" : place at start of APPL_region {block APPL};
     ```

   - For PSoC 6 CM0+:

     ```
     ".cy_app_header" : place at start of IROM1_region
     { section .cy_app_header };
     ```

   - For PSoC 6 CM4:

     ```
     ".cy_app_signature" : place at address
     (__ICFEDIT_region_IROM1_end__ - 0x200)
     { section .cy_app_signature };
     ```

3. Insert the following code after the record mentioned in the previous step:

   ```
   define symbol EM_EEPROM_START_ADDRESS = <EEPROM Address>;
   ".my_emulated_eeprom" : place at address (EM_EEPROM_START_ADDRESS) {
   section .my_emulated_eeprom };
   ```

- `<EEPROM Address>` – This is an absolute address in flash where the EEPROM should start. You must define the address value. The address should be aligned to the size of the device's flash row and should not overlap with the memory space used by the application.

  Note Do not write to and read/execute from the same flash sector at the same time. This is true for all sectors. For more details, refer to the PDL reference for Flash System Routine (Flash).

- `my_emulated_eeprom` – This is the name of the section where the EEPROM storage will be placed. The name can be changed to any name you choose.

4. Save the changes and close the file.
5. Similar to Step 4 of Quick Start section, declare EEPROM storage in the newly created section. For this, add the following declaration:

```
#pragma location = ".my_emulated_eeprom"
#pragma data_alignment = CY_FLASH_SIZEOF_ROW
const uint8 emEeprom[Em_EEPROM_1_PHYSICAL_SIZE] = {0u};
```

6. Follow the same process from Step 10 through Step 11 under PSoC 4/PSoC 5LP to make appropriated changes and build the project.

Adding EEPROM Storage to Checksum Exclude Section (PSoC 4/PSoC 5LP)

This section describes the actions required to add EEPROM Storage to the checksum exclude section when using the Em_EEPROM Component in a bootloader project. The section doesn't provide details on creating a bootloader/bootloadable. Refer instead to the Bootloader/Bootloadable Component datasheet.

To add the EEPROM Storage to the checksum exclude section:

1. In the Bootloadable project, follow the same process from Step 1 through Step 4 in the Quick Start section.
2. Modify the declaration of EEPROM Storage as follows:

   For GCC and MDK compilers:

   from:

```
const uint8 emEeprom[Em_EEPROM_1_PHYSICAL_SIZE]
__ALIGNED(CY_FLASH_SIZEOF_ROW) = {0u};
```

   to:

```
const uint8 emEeprom[Em_EEPROM_1_PHYSICAL_SIZE]
CY_SECTION(".cy_checksum_exclude")
```

```
__ALIGNED(CY_FLASH_SIZEOF_ROW) = {0u};
```

For the IAR compiler:

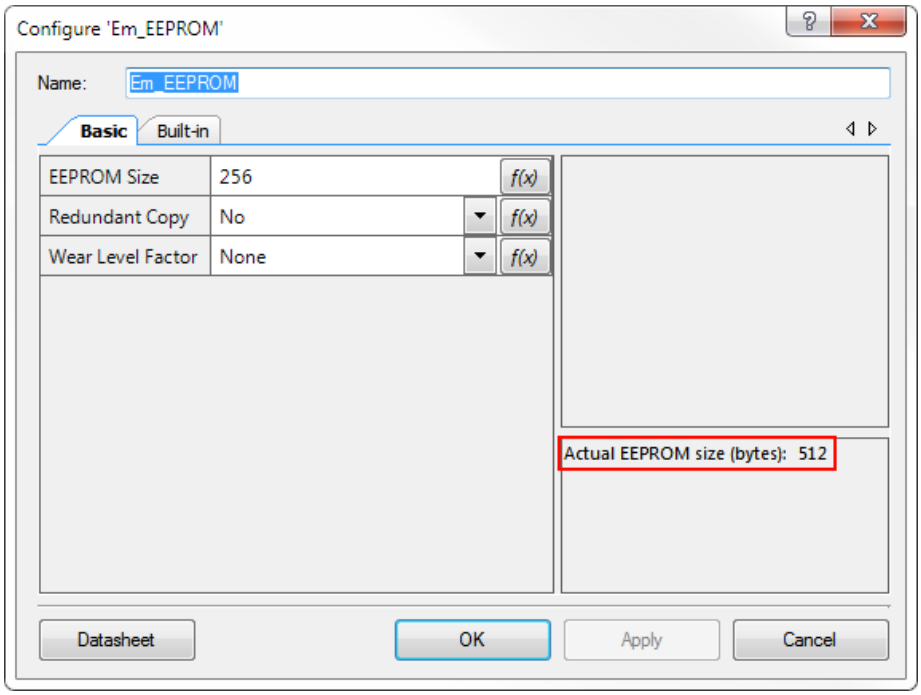
from:

```
#pragma data_alignment = CY_FLASH_SIZEOF_ROW
const uint8 emEeprom[Em_EEPROM_1_PHYSICAL_SIZE] = {0u};
```

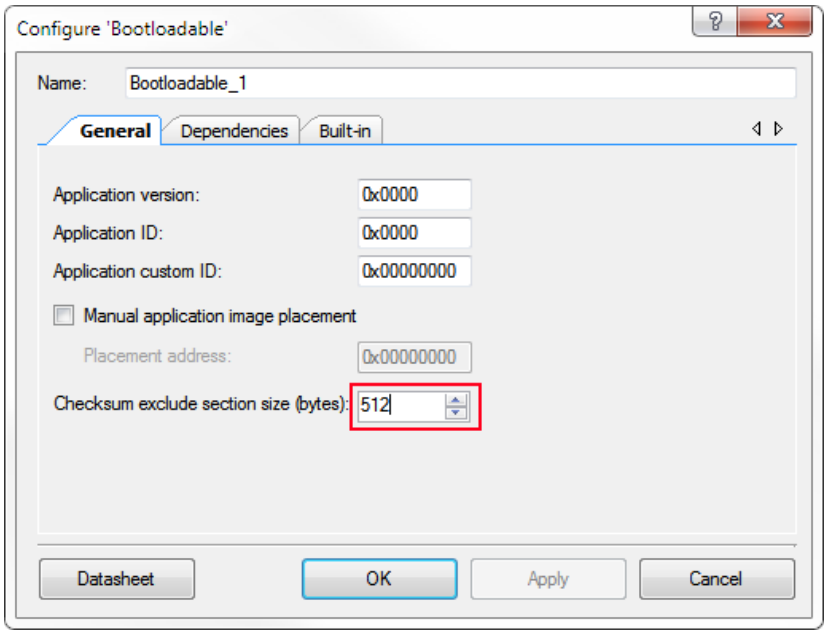
to:

```
#pragma location = ".cy_checksum_exclude"
#pragma data_alignment = CY_FLASH_SIZEOF_ROW
const uint8 emEeprom[Em_EEPROM_1_PHYSICAL_SIZE] = {0u};
```

3. Open the Em_EEPROM Configure dialog to see the size occupied by the EEPROM storage.



4. Open the Bootloadable Component Configure dialog, and enter the EEPROM storage size in the Checksum exclude section size field.



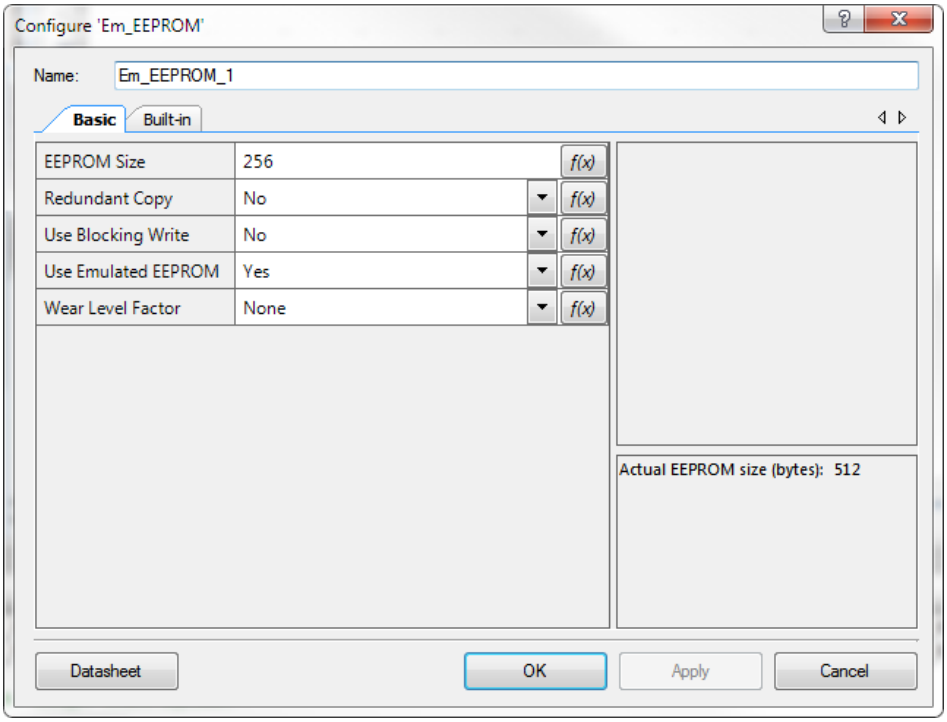
5. Follow the same process from Step 5 through Step 7 in the Quick Start section.

Component Parameters

The Em_EEPROM Component Configure dialog allows editing the configuration parameters for the Component instance.

Basic Tab

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



Parameter Name	Description
EEPROM Size	Sets an EEPROM size. The size should be rounded up to a full EEPROM page size. The page size is specific for a device family. For PSoC 4 the page size is 64 bytes, for PSoC 3/PSoC 5LP – 128 bytes, for PSoC 6 – 256 bytes. Min size is 1 page/row. Max Size = available flash.
Actual EEPROM Size	Shows the entire size of the EEPROM allocated. This parameter displays the actual size allocated including an overhead for wear levelling and redundant copy implementations.
Redundant Copy	If enabled, the checksum is calculated on each row of data (that checksum is stored in the row), and a redundant copy of the EEPROM is stored in another location. When data is read, the checksum is checked first. If the checksum is bad, the redundant copy is restored if its checksum is good.

Parameter Name	Description
Use Blocking Write	Applicable only for PSoC 6 devices. When selected, blocking writes to flash will be used in the design; otherwise, non-blocking flash writes will be used. The behavior of blocking and non-blocking writes are the same with the difference being that the non-blocking writes do not block interrupts. **Note** Do not write to and read/execute from the same flash sector at the same time. This is true for all sectors. For more details, refer to the PDL reference for Flash System Routine (Flash).
Use Emulated EEPROM	Applicable only for PSoC 6 devices. Selects if Emulated EEPROM flash area or User flash will be used for the EEPROM storage.
Wear Level factor	Selects how much wear leveling is required. The higher the factor is, the more flash is used, but a higher number of erase/write cycles can be done on the EEPROM. Multiply this number by the datasheet write endurance spec to determine the max of write cycles.

Em_EEPROM Dynamic

The Em_EEPROM Component is linked with a hidden design-wide Em_EEPROM Dynamic Component, which is always present in a design to support the placement of multiple instances of the Em_EEPROM for non-PSoC 6 devices. For PSoC 6, the source code that is included in the Em_EEPROM Dynamic is present in the PDL in the form of a middleware library.

Em_EEPROM Version and Updates

The Em_EEPROM_Dynamic version must be the same as the Em_EEPROM Component used in the design. Therefore, both the Em_EEPROM and the Em_EEPROM_Dynamic Components must be updated synchronously.

The Em_EEPROM_Dynamic Component is also shown in the Component Update Tool because of its nature as a design-wide Component. If you do not have an Em_EEPROM Component in your design, then no action is required; the Em_EEPROM_Dynamic Component is inactive and colored gray.

Application Programming Interface

The Application Programming Interface (API) routines allow configuring the Component using software.

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 PSoC 6, the Component uses the Cy_Em_EEPROM middleware library module from the PDL. The library is copied into the pdl\middleware\cy_em_eeprom\ directory of the application project after a successful build.

Refer to the *Middleware* section of the PDL documentation for a detailed description of a complete API. To access this document, go to the PSoC Creator **Help** menu **> Documentation > Peripheral Driver Library**.

For non-PSoC 6 devices. PSoC Creator doesn't provide the PDL library. The code base of the Cy_Em_EEPROM middleware library is included in the Em_EEPROM Dynamic DWR Component.

The Component generates the configuration structures described in the Global variables and Preprocessor Macros sections. Pass the generated data structure and the EEPROM storage address to the associated Cy_Em_EEPROM middleware library function in the application initialization code to configure the library. After the peripheral is initialized, the application code can perform run-time changes by referencing the provided base address in the driver API functions.

Global Variables

The Em_EEPROM Component populates peripheral initialization data structure(s) (see the below paragraph). The generated code is placed in the C source and header files that are named after the instance of the Component (e.g. Em_EEPROM_1.c). Each variable is also prefixed with the instance name of the Component.

cy_stc_eeprom_config_t Em_EEPROM_1_config

The instance-specific configuration structure. This should be used in the associated Em_EEPROM_1_Init() function.

const uint8_t Em_EEPROM_1_em_EepromStorage[]

The Component-defined emulated EEPROM storage. The storage is only defined for PSoC 6 devices in cases when the **Use Emulated EEPROM** option is set to "Yes."

Preprocessor Macros

The Em_EEPROM Component generates the following pre-processor macro. Note that each macro is prefixed with the instance name of the Component (e.g. "Em_EEPROM_1").

Em_EEPROM_1_PHYSICAL_SIZE

The actual size of flash used to implement the EEPROM with the configuration entered by the user.

Data Structures

struct cy_stc_eeprom_config_t

Emulated EEPROM configuration structure.

Data Fields

uint32 eepromSize
uint32 wearLevelingFactor
uint8 redundantCopy
uint8 blockingWrite
uint32 userFlashStartAddr

uint32 cy_stc_eeprom_config_t::eepromSize

The number of bytes to store in EEPROM

uint32 cy_stc_eeprom_config_t::wearLevelingFactor

The amount of wear leveling from 1 to 10. 1 means no wear leveling is used.

uint8 cy_stc_eeprom_config_t::redundantCopy

If not zero, a redundant copy of the Em_EEPROM is included.

uint8 cy_stc_eeprom_config_t::blockingWrite

If not zero, a blocking write to flash is used. Otherwise non-blocking write is used. This parameter is not used for non-PSoC 6 deices.

uint32 cy_stc_eeprom_config_t::userFlashStartAddr

The start address for the EEPROM memory in the user's flash.

enum cy_en_em_eeprom_status_t

Emulated EEPROM return enumeration type.

Enumerator

CY_EM_EEPROM_SUCCESS The function executed successfully
CY_EM_EEPROM_BAD_PARAM The input parameter(s) is(are) invalid.
CY_EM_EEPROM_BAD_CHECKSUM The data in EEPROM is corrupted.
CY_EM_EEPROM_BAD_DATA Failed to place the EEPROM in flash (Init).
CY_EM_EEPROM_WRITE_FAIL Write to EEPROM failed.

Component Functions

This Component includes a set of Component-specific wrapper functions that provide simplified access to the basic Cy_Em_EEPROM operation. These functions are generated during the build process and are all prefixed with the name of the Component instance.

Function	Description
Em_EEPROM_1_Init()	Fills the start address of the EEPROM to the Component configuration structure and invokes Cy_Em_EEPROM_Init() function. In case of PSoC 6 the function is located in Cy_Em_EEPROM middleware library which is part of PDL. In case if it is utilized in the design based on non-PSOC 6 device the function is located in internal design wide resource Component - Em_EEPROM_Dynamic.
Em_EEPROM_1_Read()	Invokes the Cy_Em_EEPROM_Read() function. In case of PSoC 6 the function is located in Cy_Em_EEPROM middleware library which is part of PDL. In case if it is utilized in the design based on non-PSOC 6 device the function is located in internal design wide resource Component - Em_EEPROM_Dynamic.
Em_EEPROM_1_Write()	Invokes the Cy_Em_EEPROM_Write() function. In case of PSoC 6 the function is located in Cy_Em_EEPROM middleware library which is part of PDL. In case if it is utilized in the design based on non-PSoC 6 device the function is located in internal design wide resource Component - Em_EEPROM_Dynamic.
Em_EEPROM_1_Erase()	Invokes the Cy_Em_EEPROM_Erase() function. In case of PSoC 6 the function is located in Cy_Em_EEPROM middleware library which is part of PDL. In case if it is utilized in the design based on non-PSOC 6 device the function is located in internal design wide resource Component - Em_EEPROM_Dynamic.
Em_EEPROM_1_NumWrites()	Invokes the Cy_Em_EEPROM_NumWrites() function. In case of PSoC 6 the function is located in Cy_Em_EEPROM middleware library which is part of PDL. In case if it is utilized in the design based on non-PSOC 6 device the function is located in internal design wide resource Component - Em_EEPROM_Dynamic.

cy_en_em_eeprom_status_t EmEEPROM_1_Init (uint32 startAddress)

Fills the start address of the EEPROM to the Component configuration structure and invokes Cy_Em_EEPROM_Init() function. In case of PSoC 6 the function is located in Cy_Em_EEPROM middleware library which is part of PDL. In case if it is utilized in the design based on non-PSOC 6 device the function is located in internal design wide resource Component - Em_EEPROM_Dynamic.

Parameters:

Start address of the EEPROM. For PSoC 6, if Emulated EEPROM flash area is selected for EEPROM storage, the start address will be overwritten to some address from that area.

Returns:

cy_en_em_eeprom_status_t A result of function execution of type cy_en_em_eeprom_status_t.

cy_en_em_eeprom_status_t EmEEPROM_1_Write (uint32 addr, void * eepromData, uint32 size)

Invokes the Cy_Em_EEPROM_Write() function. In case of PSoC 6 the function is located in Cy_Em_EEPROM middleware library which is part of PDL. In case if it is utilized in the design based on non-PSoC 6 device the function is located in internal design wide resource Component - Em_EEPROM_Dynamic.

cy_en_em_eeprom_status_t EmEEPROM_1_Read (uint32 addr, void * eepromDdata, uint32 size)

Invokes the Cy_Em_EEPROM_Read() function. In case of PSoC 6 the function is located in Cy_Em_EEPROM middleware library which is part of PDL. In case if it is utilized in the design based on non-PSOC 6 device the function is located in internal design wide resource Component - Em_EEPROM_Dynamic.

cy_en_em_eeprom_status_t EmEEPROM_1_Erase (void)

Invokes the Cy_Em_EEPROM_Erase() function. In case of PSoC 6 the function is located in Cy_Em_EEPROM middleware library which is part of PDL. In case if it is utilized in the design based on non-PSOC 6 device the function is located in internal design wide resource Component - Em_EEPROM_Dynamic.

uint32 EmEEPROM_1_NumWrites (void)

Invokes the Cy_Em_EEPROM_NumWrites() function. In case of PSoC 6 the function is located in Cy_Em_EEPROM middleware library which is part of PDL. In case if it is utilized in the design based on non-PSOC 6 device the function is located in internal design wide resource Component - Em_EEPROM_Dynamic.

Em_EEPROM_Dynamic Component Functions

Function	Description
Cy_Em_EEPROM_Init()	Initializes the Emulated EEPROM library.
Cy_Em_EEPROM_Read()	This function takes the logical EEPROM address, converts it to the actual physical address where the data is stored and returns the data to the user.
Cy_Em_EEPROM_Write()	This function takes the logical EEPROM address and converts it to the actual physical address and writes data there. If wear leveling is implemented, the writing process will use the wear leveling techniques. This is a blocking function and it does not return until the write operation is completed.

Function	Description
Cy_Em_EEPROM_Erase()	This function erases the entire contents of the EEPROM. Erased values are all zeros. This is a blocking function and it does not return until the write operation is completed.
Cy_Em_EEPROM_NumWrites()	Returns the number of the EEPROM writes completed so far.

cy_en_em_eeprom_status_t Cy_Em_EEPROM_Init (cy_stc_eeprom_config_t *config, cy_stc_eeprom_context_t *context)

Initializes the Emulated EEPROM library.

Parameters:

config	The pointer to a configuration structure. See cy_stc_eeprom_config_t.
context	The pointer to the EEPROM context structure cy_stc_eeprom_context_t.

Returns:

error / status code. See cy_en_em_eeprom_status_t.

Note:

The context structure should not be modified after it is filled with this function. Modifying the context structure may cause unexpected behavior of the Cy_Em_EEPROM functions that rely on it.

This function uses a buffer of the flash row size to perform a read operation. For the size of the row, refer to the specific PSoC device datasheet.

Note:

If the "Redundant Copy" option is used, the function performs a number of write operations to the EEPROM to initialize flash rows checksums. Therefore, Cy_Em_EEPROM_NumWrites(), when it is called right after Cy_Em_EEPROM_Init(), will return a non-zero value that identifies the number of writes performed by Cy_Em_EEPROM_Init().

cy_en_em_eeprom_status_t Cy_Em_EEPROM_Read (uint32 addr, void *eepromData, uint32 size, cy_stc_eeprom_context_t *context)

This function takes the logical EEPROM address, converts it to the actual physical address where the data is stored and returns the data to the user.

Parameters:

addr	The logical start address in EEPROM to start reading data from.
eepromData	The pointer to a user array to write data to.
size	The amount of data to read.
context	The pointer to the EEPROM context structure cy_stc_eeprom_context_t.

Returns:

This function returns cy_en_em_eeprom_status_t.

Note:

This function uses a buffer of the flash row size to perform read operation. For the size of the row refer to the specific PSoC device datasheet.

In case in if redundant copy option is enabled the function may perform writes to EEPROM. This is done in case if the data in the EEPPROM is corrupted and the data in redundant copy is valid based on CRC-8 integrity check.

cy_en_em_eeprom_status_t Cy_Em_EEPROM_Write (uint32 addr, void *eepromData, uint32 size, cy_stc_eeprom_context_t *context)

This function takes the logical EEPROM address and converts it to the actual physical address and writes data there. If wear leveling is implemented, the writing process will use the wear leveling techniques. This is a blocking function and it does not return until the write operation is completed. The user firmware should not enter Hibernate mode until write is completed. The write operation is allowed in Sleep and Deep-Sleep modes. During the flash operation, the device should not be reset, including the XRES pin, a software reset, and watchdog reset sources. Also, low-voltage detect circuits should be configured to generate an interrupt instead of a reset. Otherwise, portions of flash may undergo unexpected changes.

Parameters:

addr	The logical start address in EEPROM to start writing data from.
eepromData	Data to write to EEPROM.
size	The amount of data to write to EEPROM.
context	The pointer to the EEPROM context structure cy_stc_eeprom_context_t.

Returns:

This function returns cy_en_em_eeprom_status_t.

Note:

This function uses a buffer of the flash row size to perform write operation. For the size of the row refer to the specific PSoC device datasheet.

For PSoC 3/PSoC 5LP, this function calls the CySetTemp() function for a reliable write procedure to occur. The CySetTemp() function should be called once before executing a series of Flash writes in case the die temperature changes significantly (10 °C or more).

cy_en_em_eeprom_status_t Cy_Em_EEPROM_Erase (cy_stc_eeprom_context_t *context)

This function erases the entire contents of the EEPROM. Erased values are all zeros. This is a blocking function and it does not return until the write operation is completed. The user firmware should not enter Hibernate mode until erase is completed. The erase operation is allowed in Sleep and Deep-Sleep modes. During the flash operation, the device should not be reset, including the XRES pin, a software reset, and watchdog reset sources. Also, low-voltage detect circuits should be configured to generate an interrupt instead of a reset. Otherwise, portions of flash may undergo unexpected changes.

Parameters:

context	The pointer to the EEPROM context structure cy_stc_eeprom_context_t.

Returns:

This function returns cy_en_em_eeprom_status_t.

Note:

The erase operation is performed by clearing the EEPROM data using flash write. This affects the flash durability. So it is recommended to use this function in utmost case for prolongation of the flash life.

This function uses a buffer of the flash row size to perform erase operation. For the size of the row refer to the specific PSoC device datasheet.

For PSoC 3/PSoC 5LP, this function calls the CySetTemp() function for a reliable erase procedure to occur. The CySetTemp() function should be called once before executing a series of Flash erase operations in case the die temperature changes significantly (10 °C or more).

uint32 Cy_Em_EEPROM_NumWrites (cy_stc_eeprom_context_t *context)

Returns the number of the EEPROM writes completed so far.

Parameters:

context	The pointer to the EEPROM context structure cy_stc_eeprom_context_t.

Returns:

The number of writes performed to the EEPROM.

Data in RAM

The generated Component configuration structure 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.

Code Examples and Application Notes

This section lists the projects that demonstrate the use of this Component.

Code Examples

PSoC Creator provides access to code examples in the Code Example 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. In need, use the **Filter Options** in the dialog to narrow the list of projects available to select.

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

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

Application Notes

Cypress provides a number of application notes describing how PSoC can be integrated into your design. You can access the Cypress Application Notes search web page at www.cypress.com/appnotes. Application Notes related to this Component include:

- AN210781 – Getting started with PSoC 6 BLE
- AN215656 – PSoC 6 Dual-Core CPU system Design

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 an 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.

PSoC 3 (Keil_PK51)

Configuration	Flash Bytes	SRAM Bytes
Default	7447	296

PSoC 4 (GCC)

Configuration	Flash Bytes	SRAM Bytes
Default	2386	48

PSoC 5LP (GCC)

Configuration	Flash Bytes	SRAM Bytes
Default	2562	56

PSoC 6 (GCC)

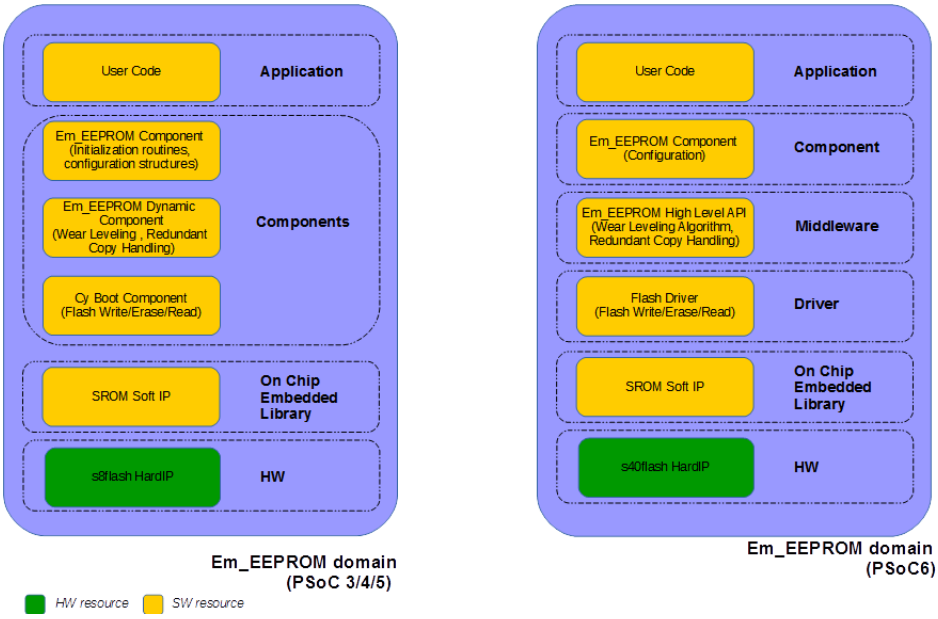
Configuration	Flash Bytes	SRAM Bytes
Default	2998	60

Functional Description

The Emulated EEPROM Component provides an interface to either Cy_Em_EEPROM Middleware library (PSoC 6) or to Em_EEPROM Dynamic DWR Component (non-PSoC 6).

Block Diagram and Configuration

A simplified diagram of the Em_EEPROM hardware is shown below:



For the PSoC 3/PSoC 4/PSoC 5LP devices, the Em_EEPROM Component itself includes the Component configuration structures, initialization routine and interface to Em_EEPROM Dynamic Component routines. The Em_EEPROM Dynamic design-wide Component includes the implementation of wear leveling and redundant copy algorithms. The Em_EEPROM Dynamic Component uses cy_boot Component that contains an API for operations with flash.

For PSoC 6, the Emulated EEPROM implements a middleware library on the top of existing Flash driver that creates an emulated EEPROM in flash and has the ability to do wear leveling and restore corrupted data from a redundant copy. Em_EEPROM Component itself is only responsible for configuring the Em_EEPROM middleware library and the Flash driver.

Wear leveling

Wear leveling allows to increase the flash endurance (the number of flash erase/write cycles) by using extra flash rows in EEPROM. For example, if the user selected wear leveling of 5x, that allows 5x more flash write/erase cycles to be performed to flash. However, the overall emulated EEPROM size will be 5 times greater comparing to the case where no wear leveling is used.

Redundant Copy

Redundant copy option allows recovering of damaged EEPROM data from the redundant EEPROM copy. This option, when it is used, doubles the overall flash size of the Emulated EEPROM to store the redundant EEPROM copy in it. The algorithm works as following. During the write operation the checksum of the active EEPROM row data is calculated and stored in the active row and in the corresponding row of redundant EEPROM copy. When the data from the EEPROM row is read - the checksum is calculated on the row data again and compared to the stored one. If the checksums don’t match the same actions are performed on the redundant EEPROM copy row. If the checksums are matching the corresponding EEPROM row is restored from the redundant copy and valid data is returned to user. If the checksums don’t match the error status is returned to the user and no data is read.

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 applicable to all PSoC Creator Components
- specific deviations – Deviations applicable only for this Component

This section provides information on the Component-specific deviations. Non-PSoC 6 project deviations are described in the “MISRA Compliance” section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

For PSoC 6, this Component uses the Cy_Em_EEPROM middleware library from Peripheral Driver Library (PDL) module. Refer to the PDL documentation for information on its MISRA compliance and specific deviations. Also refer to **PSoC Creator Help>Building a PSoC Creator Project>Generated Files (PSoC 6)** for information on MISRA compliance and deviations of a files generated by PSoC Creator.

For PSoC 6, the Em_EEPROM Component is MISRA Compliant, except for the following specific deviations:

Rule	Rule Class	Rule Description	Description of Deviation(s)
8.7	R	Objects shall be defined at block scope if they are only accessed from within a single function.	The Component defines the EEPROM storage array that is directly used only in 'Em_EEPROM_1_Init'. Indirectly, the EEPROM storage is used by every Component API.
14.2	R	All non-null statements shall either have at least one side-effect, however executed, or cause control flow to change.	To maintain common codebase some variables, unused for a specific device, are casted to void to prevent generation of an unused variable compiler warning.
19.7	A	A function shall be used in preference to a function-like macro.	A macro is used for performance reasons.

For non-PSoC 6 devices, the Em_EEPROM Component (including Em_EEPROM_Dynamic) is MISRA Compliant, except for the following specific deviations:

Rule	Rule Class	Rule Description	Description of Deviation(s)
5.6	A	No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure and union member names.	“config” is a field of CyLib structure, which is used for hardware configuration inside the CyLib module. There is no possibility of confusion with the internal “config” variable.
8.1	R	Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call.	Both functions memcpy() and memset() are defined within standard C-library and theirs declarations are present in the file string.h in the library. This header file is already included into file cy_em_eeprom.c No possible issues expected.
8.3	R	For each function parameter the type given in the declaration and definition shall be identical, and the return types shall also be identical.	The function memcpy() is function from the standard C-library and is declared as following: void * memcpy (void *restrict to, const void *restrict from, size_t size). The type of third parameter of this function (size_t) is platform specific and is defined in the standard C-library (stddef.h). For PSoC4 size_t is equal to uint32.
10.1	R	The value of an expression of integer type shall not be implicitly converted to a different underlying type under some circumstances.	For PSoC 3, the #10.1 violation occurs for the memset() and memcpy() because of a difference of the functions parameter types for PSoC 3 and PSoC 4/PSoC 5LP/PSoC 6.
11.4	A	The cast should not be performed between a pointer to the object type and a different pointer to the object type.	The cast from the object type and a different pointer to the object was used intentionally because of the performance reasons.

Rule	Rule Class	Rule Description	Description of Deviation(s)
14.2	R	All non-null statements shall either have at least one side-effect, however executed, or cause control flow to change.	To maintain common codebase, some variables, unused for a specific device, are casted to void to prevent generation of an unused variable compiler warning.
16.7	A	The object addressed by the pointer parameter is not modified and so the pointer could be of type “pointer to const”.	The warning is generated because of the pointer dereferencing to the address which makes the MISRA checker think the data is not modified.
17.4	R	The array indexing shall be the only allowed form of the pointer arithmetic.	The pointer arithmetic used in several places on the Cy_Em_EEPROM implementation is safe and preferred because it increases the code flexibility.
19.7	A	A function shall be used in preference to a function-like macro.	A macro is used for performance reasons.

DC and AC Electrical Characteristics

Refer to the Flash DC Specifications and Flash AC Specifications in the specific device datasheet.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
2.20.b	Updated datasheet.	Added rules 5.6, 8.1, and 8.8 to the non-PSoC 6 MISRA Compliance table.
2.20.a	Edited the datasheet.	Added notes to not write to and read/execute from the same flash sector at the same time. This is true for all sectors. For more details, refer to the PDL reference for Flash System Routine (Flash).
2.20	Updated Component version to fix an incorrect flash row number calculation for PSoC 4 devices with more than one flash array ID.	The Em_EEPROM_Write() function was always writing to the zero flash array.
2.10	The “EEPROM Size” and “Actual EEPROM Size” parameters were updated for larger base types.	Fixed an issue where you were unable to set the EEPROM Size larger than 32,767 bytes.
2.0.f	Updated the datasheet.	Changed API functions return type from “cy_en_eeprom_return_value_t” to “cy_en_em_eeprom_status_t”. Added a note about modifying the link scripts when using the Em_EEPROM on one specific core.

Version	Description of Changes	Reason for Changes / Impact
2.0.e	Updated the datasheet.	Added a note about prerequisites for Flash driver at the top of the Quick Start section. Added notes to the Cy_Em_EEPROM_Init() function description.
2.0.d	Updated the datasheet.	The proper code for declaring an array for IAR compiler was added to IAR compiler section.
2.0.c	Updated the datasheet.	Updated MISRA Compliance section. Updated Placing EEPROM Storage at Fixed Address section. Added description on how to use fixed placement of EEPROM Storage with IAR compiler.
2.0.b	Made the Em_EEPROM_em_EepromStorage[] array accessible to the user.	Passing the address of the emulated EEPROM storage to PDL _Init() function is mandatory. Making the Em_EEPROM_em_EepromStorage[] array accessible allows the removal of the Component wrapper API functions if one wants to use only the PDL API (PSoC 6).
2.0.a	Edited datasheet.	Added the section: Adding EEPROM Storage to Checksum Exclude Section (PSoC 4/PSoC 5LP).
2.0	This is a complete re-write of the Component to bring in line with customer expectations 1. Added Init, Read, Erase and NumWrites API functions. 2. Remove Start and Stop API functions. 3. Modified Write API to pass a logical EEPROM Address. 4. Added Wear Leveling. 5. Added Redundant Copy option. 6. Updated GUI to provide information on flash used by Component. 7. The datasheet was moved to a new template.	Improve the emulated EEPROM to match the industry norms for emulated EEPROMs by adding features such as wear leveling and redundant data copies.
1.10	Fixed the 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 a flush instruction cache after a flash write into the emulated EEPROM is complete.	Reading back just written flash data might return cached data instead of reading flash content when the instruction cache is enabled.
1.0.a	Added the “DC/AC Electrical Characteristics” section.	To provide the Component operating characteristics.
1.0	The first Component version.	The first Component version.

© Cypress Semiconductor Corporation (an Infineon company), 2018-2021. 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.

