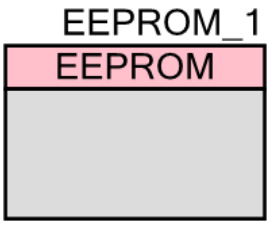
EEPROM
3.0



Features

- 512 B to 2 KB EEPROM memory
- 1,000,000 cycles, 20-year retention
- Read/Write 1 byte at a time
- Program 16 bytes (a row) at a time

General Description

The EEPROM component provides a set of APIs to erase and write data to non-volatile on-chip EEPROM memory. The term write implies that it will erase and then program in one operation.

An EEPROM memory in PSoC devices is organized in arrays. PSoC 3 and PSoC 5LP devices offer an EEPROM array of size 512 bytes, 1 KB or 2 KB depending on the device. EEPROM array can be divided into sectors that have up to 64 rows with a size of 16 bytes. The Application Programming Interface (API) routines allow you to modify a whole EEPROM row, individual EEPROM bytes, or erase a whole EEPROM sector in one operation.

The EEPROM memory is not initialized by the EEPROM component: the initial state of the memory is defined in the device datasheet. The default values can be changed in the PSoC Creator EEPROM Editor. For more details, refer to the PSoC Creator Help.

The EEPROM component is tightly coupled with various system elements contained within the cy_boot component. These elements are generated upon a successful build. Refer to the *System Reference Guide* for more information about the cy_boot component and its various elements.

When to use an EEPROM

An EEPROM component can be used for the below purposes:

- Non-volatile storage that must survive power cycles (for example, calibration tables or device configuration)
- For additional storage of data (freeing up on-chip RAM)
- For read-only (or rarely-changing) program data

Input/Output Connections

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

Component Parameters

The EEPROM has no configurable parameters other than standard Instance 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.

The entire contents of the EEPROM are mapped into memory space and can be read directly. EEPROM allows read access at the byte level. However, the component provides a separate API for reading data from EEPROM memory: EEPROM_ReadByte(). This API along with EEPROM_WriteByte() provide a simple interface to read and write individual EEPROM bytes. Erasing the EEPROM can be done either by sector number, or by writing some data to individual cells. The time that is takes to write an individual byte is equal to time of writing a whole EEPROM row.

The following defines are used for reading EEPROM:

- CYDEV_EE_BASE – The base address of the EEPROM memory in absolute address space
- CYDEV_EE_SIZE – The size of the EEPROM memory space in bytes
- CYDEV_EEPROM_ROW_SIZE – The size of a row of the EEPROM in bytes

Using the base pointer, individual bytes of the EEPROM memory can be read. To navigate to a specific row the base pointer must be incremented by the number of times of the size of the EEPROM row.

To read the data from EEPROM in PSoC 3 using direct addressing, the pointer to the EEPROM variable should be typecasted to (CYXDATA *) because the EEPROM memory is in an extended memory region of the 8051. For example, when a floating point integer is stored in EEPROM, the following should be used:

```
(volatile float CYXDATA *)CYDEV_EE_BASE
```

The SIZEOF_EEPROM_ROW define can be used instead of CYDEV_EEPROM_ROW_SIZE.

EEPROM_1_EEPROM_SIZE is also defined as the size of the EEPROM memory space in bytes (where EEPROM_1 is the instance name of the EEPROM component).

It is necessary to acquire the die temperature by calling the EEPROM_UpdateTemperature() function before a series of EEPROM write operations. The EEPROM_UpdateTemperature() function queries SPC for the die temperature and stores it in a global variable, which is used while performing EEPROM write operations. If the application is used in an environment where the die temperature changes 10° C or more, the temperature should be refreshed to adjust the write times for the optimal performance.

It can take as many as 20 ms to write to EEPROM. During this time, the device should not be reset, or unexpected changes may be made to portions of EEPROM or Flash. Reset sources include XRES pin, software reset, and watchdog. Make sure that these are not inadvertently activated. Also, the low voltage detect circuits should be configured to generate an interrupt instead of a reset.

By default, PSoC Creator assigns the instance name "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 "EEPROM"

Note that the contents of EEPROM memory will not be available until the corresponding block was started using the EEPROM_Start() API.

Functions

Function	Description
EEPROM_Enable()	Enables EEPROM block operation
EEPROM_Start()	Starts EEPROM
EEPROM_Stop()	Stops and powers down EEPROM
EEPROM_WriteByte()	Writes a byte of data to the EEPROM.
EEPROM_ReadByte()	Reads a byte of data from the EEPROM.
EEPROM_UpdateTemperature()	Updates store temperature value.
EEPROM_EraseSector()	Erases an EEPROM sector
EEPROM_Write()	Blocks while writing a row to EEPROM
EEPROM_StartWrite()	Starts writing a row of data to EEPROM
EEPROM_Query()	Checks the state of a write to EEPROM
EEPROM_ByteWritePos()	Writes a byte of data to EEPROM

void EEPROM_Enable(void)

Description: Enables EEPROM block operation.

Parameters: None

Return Value: None

Side Effects: A call to EEPROM_Start() calls EEPROM_Enable(). You can call either EEPROM_Start() or EEPROM_Enable() directly; both have the same effect. After calling this API, theEEPROM needs 5 µS to start, so no write requests should be performed.

void EEPROM_Start(void)

Description: Starts the EEPROM. This has to be called before using write/erase APIs and reading the EEPROM.

Parameters: None

Return Value: None

Side Effects: A call to EEPROM_Start() calls EEPROM_Enable(). You can call either EEPROM_Start() or EEPROM_Enable() directly; both have the same effect. After calling this API, the EEPROM needs 5 µS to start, so no write requests should be performed.

void EEPROM_Stop(void)

Description: Stops and powers down the EEPROM.

Parameters: None

Return Value: None

Side Effects: None

cystatus EEPROM_WriteByte(uint8 dataByte, uint16 address)

Description: Writes a byte of data to the EEPROM. This function blocks until the function is complete. For reliable write procedure to occur you should call EEPROM_UpdateTemperature() API if the temperature of the silicon has changed for more than 10°C since component was started.

Parameters: dataByte: Byte of data to write to the EEPROM

address: Address of data to be written. Maximum address is dependent on EEPROM size.

Return Value:

Value	Description
CYRET_SUCCESS	Successful completion
CYRET_BAD_PARAM	Row number or byte number out
CYRET_LOCKED	SPC locked by another operation
CYRET_UNKNOWN	Other error from the SPC

Side Effects: None

uint8 EEPROM_ReadByte(uint16 address)

Description: Reads a byte of data from the EEPROM. Although the data is present in one of the memory spaces, this provides an intuitive user interface, addressing EEPROM memory as a separate block with first EERPOM address 0x0000.

Parameters: address: Address array of data in EEPROM to be read. Maximum address is dependent on EEPROM size.

Return Value: Data located at address.

Side Effects: None

uint8 EEPROM_UpdateTemperature(void)

Description: Updates store temperature value. This should be called anytime the EEPROM is active and temperature may have changed by more than 10C.

Parameters: None

Return Value: Status of operation, 0 if operation complete, non-zero value if error was detected.

Side Effects: None

cystatus EEPROM_EraseSector(uint8 sectorNumber)

Description: Erases a sector (64 rows) of memory by making the bits zero. This function blocks until the operation is complete. Using this API helps to erase EEPROM a sector at a time. This is faster than using individual writes but affects cycle recourse of the whole row.

Parameters: sectorNumber: Sector number to erase

Return Value:

Value	Description
CYRET_SUCCESS	Successful completion
CYRET_BAD_PARAM	Sector number out of range
CYRET_LOCKED	SPC locked by another operation
CYRET_UNKNOWN	Other error from the SPC

Side Effects: None

cystatus EEPROM_Write(const uint8 *rowData, uint8 rowNumber)

Description: Writes a row (16 bytes) of data to the EEPROM. This function blocks until the function is complete. Compared to APIs that write one byte, this API allows you to write a whole row (16 bytes) at a same time.

Parameters: rowData: Address of the data to write to the EEPROM

rowNumber: Row number to write

Return Value:

Value	Description
CYRET_SUCCESS	Successful completion
CYRET_BAD_PARAM	Row number or byte number out
CYRET_LOCKED	SPC locked by another operation
CYRET_UNKNOWN	Other error from the SPC

Side Effects: None

cystatus EEPROM_StartWrite(const uint8 *rowData, uint8 rowNumber)

Description: Starts the write of a row (16 bytes) of data to the EEPROM. This function does not block. The function returns once the SPC has begun writing the data. This function must be used in combination with EEPROM_Query(). EEPROM_Query() must be called until it returns a status other than CYRET_STARTED. That indicates the write has completed. Until EEPROM_Query() detects that the write is complete the SPC is marked as locked to prevent another SPC operation from being performed. For reliable write procedure to occur you should call EEPROM_UpdateTemperature() API if the temperature of the silicon has changed for more than 10°C since component was started.

Parameters: rowData: Address of the data to write to the EEPROM

rowNumber: Row number to write

Return Value:

Value	Description
CYRET_SUCCESS	Successful completion
CYRET_BAD_PARAM	Row number or byte number out
CYRET_LOCKED	SPC locked by another operation
CYRET_UNKNOWN	Other error from the SPC

Side Effects: After calling this API, the device should not be powered down, reset, or switched to low power mode until the EEPROM operation is complete. Not following this recommendation may lead to data corruption or silicon unexpected behavior.

cystatus EEPROM_StartErase(uint8 sectorNumber)

Description: Starts the EEPROM sector erase. This function does not block. The function returns once the SPC has begun writing the data. This function must be used in combination with EEPROM_Query(). EEPROM_Query() must be called until it returns a status other than CYRET_STARTED. That indicates the erase has completed. Until EEPROM_Query() detects that the erase is complete the SPC is marked as locked to prevent another SPC operation from being performed.

Parameters: rowData: Address of the data to write to the EEPROM

rowNumber: Row number to write

Return Value:

Value	Description
CYRET_SUCCESS	Successful completion
CYRET_BAD_PARAM	Row number or byte number out
CYRET_LOCKED	SPC locked by another operation
CYRET_UNKNOWN	Other error from the SPC

Side Effects: After calling this API device should not be powered down, reset or switched to low power mode until EEPROM operation isn't complete. Not following this recommendation may lead to data corruption or silicon unexpected behavior.Not following this recommendation may lead to data corruption or silicon unexpected behavior.

cystatus EEPROM_Query(void)

Description: Checks the status of an earlier call to EEPROM_StartWrite() or EEPROM_StartErase(). This function must be called until it returns a value other than CYRET_STARTED. Once that occurs the write has been completed and the SPC is unlocked.

Parameters: None

Return Value:

Value	Description
CYRET_SUCCESS	Successful completion. Write has completed.
CYRET_STARTED	Write has not yet completed
CYRET_UNKNOWN	Other error from the SPC

Side Effects: None

cystatus EEPROM_ByteWritePos(unit8 dataByte, uint8 rowNumber, uint8 byteNumber)

Description: Writes a byte of data to EEPROM. Compared to EEPROM_WriteByte() this API allows to write a specific byte in a specified EEPROM row. This is a blocking call. It will not return until the function succeeds or fails.

Parameters: dataByte: Byte of data to write to the EEPROM

rowNumber: Row number to write

byteNumber: Byte number within the row to write

Return Value:

Value	Description
CYRET_SUCCESS	Successful completion
CYRET_BAD_PARAM	Row number or byte number out
CYRET_LOCKED	SPC locked by another operation
CYRET_UNKNOWN	Other error from the SPC

Side Effects: None

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
- specific deviations – deviations that are applicable only for this component

This section provides information on component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The 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 the 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 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Default	1228	0	796	0

References

Refer also to the Die Temperature component datasheet and the *System Reference Guide*.

Resources

The EEPROM component uses EEPROM capability of the device.

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		1.71	--	5.5	V

AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
T_{WRITE}	Single row erase/write cycle time		--	2	20	ms
	EEPROM data retention time, retention period measured from	Average ambient temp, T_A ≤ 25°C, 1M erase/program cycles	20	--	--	years

Parameter	Description	Conditions	Min	Typ	Max	Units
	last erase cycle	Average ambient temp, $T_A \le 55°C$, 100K erase/program cycles	20	--	--	
		Average ambient temp, $T_A \le 85°C$, 10K 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
3.0	Added APIs for non-blocking EEPROM access. Renamed EEPROM_QueryWrite() and EEPROM_ByteWrite() APIs to EEROM_Query and EEPROM_ByteWritePos() respectfully.	Updated component requirements.
2.10.b	Updated datasheet.	Removed references to obsolete PSoC 5 device.
2.10.a	Updated EEPROM_Start/EEPROM_Enable API descriptions, added clarification on accessing floating point integers on PSoC 3.	To clarify the component operation.
2.10	Added MISRA Compliance section.	The component does not have any specific deviations.
2.0.a	Updated AC and DC characteristics section.	Keeping AC and DC characteristics aligned with the device datasheet.
	Added information about EEPROM memory initial state.	Clarify the component operation and refer to the device datasheet for the initial state of the EEPROM memory.
2.0	Added support for PSoC 5LP silicon. Added new API EEPROM_ByteWrite().	To support the byte write capability.
	Codes changes in EEPROM_Write(), EEPROM_StartWrite(), EEPROM_QueryWrite() and EEPROM_EraseSector().	To support the SPC code changes.
	Removed CySetTemp() calls from EEPROM_Write() and EEPROM_StartWrite() APIs.	For better performance.
1.60	Minor code changes in the APIs EEPROM_Write(), EEPROM_StartWrite() and EERPOM_QueryWrite().	For better performance.
	Code changes in the EEPROM_ EaraseSector() API to support PSoC5.	PSoC 5 silicon supports the Erase Sector command.
	EEPROM_EraseSector() API description changes in the datasheet	

Version	Description of Changes	Reason for Changes / Impact
1.50.b	Added explanation of how to acquire temperature to datasheet.	Clarity
1.50.a	Added characterization data to datasheet	
	Noted in EEPROM_EraseSector() API in datasheet that it is only available for PSoC 3 Production or later.	
	Minor datasheet edits and updates	
1.50	Modified the *EEPROM.c* file to switch the include file from *cydevice.h* file to *cydevice_trm.h*.	The *cydevice.h* file is obsolete. So the generated source and APIs provided with PSoC Creator should use *cydevice_trm.h* instead.
	Updated the EEPROM_EraseSector() section of the example code.	The Erase Sector command does not function on EEPROM for PSOC 3 ES1 and ES2 silicon or on PSOC 5 silicon. This API can be used on EEPROM for PSOC 3 Production or later.
	Added EEPROM_Enable(), EEPROM_Start(), and EEPROM_Stop() APIs.	To support PSoC 3 Production silicon requirement that the EEPROM is powered off by default. A call to EEPROM_Start() will call EEPROM_Enable(). You can call either EEPROM_Start() or EEPROM_Enable() function directly; both have the same effect.
1.20.a	Moved component into subfolders of the component catalog.	
	Added information to the component that advertizes its compatibility with silicon revisions.	The tool reports an error/warning if the component is used on incompatible silicon. If this happens, update to a revision that supports your target device.
1.20	Updated the Configure dialog.	Digital Port was changed to Pins component in the schematic.

© Cypress Semiconductor Corporation, 2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.