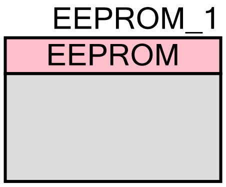
EEPROM

1.10



Features

- 512B to 2 KB
- 1,000,000 cycles, 20 year retention
- Read byte at a time
- Program 16 bytes at a time

General Description

The EEPROM component provides an API to write a row (16 bytes) of data to the EEPROM. The term write implies Erase and then program in one operation.

When to use an EEPROM

You can use an EEPROM component:

- For off-chip storage of data (freeing up on-chip RAM)
- For read-only (or rarely-changing) program data
- For data that must survive power cycles (e.g., calibration tables or device configuration)

Input/Output Connections

There are no IO connections for the EEPROM component. It is an API only.

Parameters and Setup

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

PRELIMINARY

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.

To read from the EEPROM or FLASH, no APIs are needed. The entire contents of the EEPROM are mapped into memory space and can be read directly. The following defines can be used for reading EEPROM:

- CYDEV_EE_BASE – the base pointer of the EEPROM memory
- CYDEV_EE_SIZE – The size of the EEPROM memory space in bytes

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).

To write to the EEPROM, you must first acquire the die temperature. You only need to acquire the temperature once to use the write functions. If the application will be used in an environment where the die temperature changes 20°C or more, the temperature should be refreshed to allow the Smart Write Algorithm to work correctly.

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".

Function	Description
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_QueryWrite	Checks the state of a write to EEPROM

cystatus EEPROM_EraseSector(uint8 sectorNumber)

Description: Erases a sector (64 rows) of memory. This function blocks until the operation is complete.

Parameters: Uint8 sector. Sector number to erase.

Return Value: CYRET_SUCCESS if the operation was successful.
CYRET_BAD_PARAM if the parameters were invalid.
CYRET_LOCKED if the SPC is busy.
CYRET_TIMEOUT if the operation timed out.
CYRET_UNKNOWN if there was an SPC error.

Side Effects: None

PRELIMINARY

cystatus EEPROM_Write(uint8 * rowData, uint8 rowNumber)

Description: Writes a row (16 bytes) of data to the EEPROM. This is a blocking call. It will not return until the function succeeds or fails.

Parameters: uint8 * rowData. Address of the data to write to the EEPROM.

uint8 rowNumber. EEPROM row number to program.

Return Value: CYRET_SUCCESS if the operation was successful.
CYRET_BAD_PARAM if the parameters were invalid.
CYRET_LOCKED if the SPC is busy.
CYRET_TIMEOUT if the operation timed out.
CYRET_UNKNOWN if there was an SPC error.

Side Effects: None

cystatus EEPROM_StartWrite(uint8 * rowData, uint8 rowNumber)

Description: Starts the SPC write function. This function does not block, it returns once the command has begun the SPC write function. Once this function has been called the SPC will be locked until EEPROM_QueryWrite() does not return CYRET_STARTED. To abandon the write, call CySpcUnlock() to unlock the SPC.

Parameters: uint8 * rowData. Address of the data to write to the EEPROM.

uint8 rowNumber. EEPROM row number to program.

Return Value: CYRET_STARTED if the SPC command to write was successfully started.
CYRET_BAD_PARAM if the parameters were invalid.
CYRET_LOCKED if the SPC is busy.
CYRET_TIMEOUT if the operation timed out.
CYRET_UNKNOWN if there was an SPC error.

Side Effects: None

cystatus EEPROM_QueryWrite(void)

Description: Checks the state of a write to EEPROM. This function must be called until the return value is not CYRET_STARTED.

Parameters: void

Return Value: CYRET_SUCCESS if the operation was successful.
CYRET_BAD_PARAM if the parameters were invalid.
CYRET_LOCKED if the SPC is busy.
CYRET_STARTED if the SPC command to write was successfully started.
CYRET_TIMEOUT if the operation timed out.
CYRET_UNKNOWN if there was an SPC error.

Side Effects: None

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the EEPROM component. This example assumes the component has been placed in a design with the default name "EEPROM_1."

Note If you rename your component you must also edit the example code as appropriate to match the component name you specify.

```
#include <DEVICE.H>

void main(void)
{
    reg8 * eeProm;
    uint16 index;
    cystatus status;

    /* Erase all 2k. */
    EEPROM_1_EraseSector(0);
    EEPROM_1_EraseSector(1);

    /* Check to see if entire EEPROM is set to zeros. */
    eeProm = (reg8 *) CYDEV_EE_BASE;
    for(index = 0; index < CYDEV_EE_SIZE; index++)
    {
        if(eeProm[index] != 0)
        {
            /* Error condition. */
        }
    }

    /* Acquire die temperature before using the write functions. */
    CySetTemp();

    /* Blocking method to write "0123456789ABCDEF" to EEPROM row 0  */
    status = EEPROM_1_Write("0123456789ABCDEF", 0);
    if(status != CYRET_SUCCESS)
    {
        /* Error condition. */
    }

    /* Polling method to write "0123456789ABCDEF" to EEPROM row 0  */
    status = EEPROM_1_StartWrite("0123456789ABCDEF", 0);
    if(status != CYRET_STARTED)
    {
        /* Error condition. */
    }

    do
    {
        status = EEPROM_1_QueryWrite();
        /* Do something else. */

    } while(status == CYRET_STARTED);
```

```
        /* Check if an '0' was written to the first byte of the EEPROM. */
        if(*eeProm != '0')
        {
            /* Do something? */
        }
        if(status == CYRET_SUCCESS)
        {
            /* Data was written. */
        }
        else
        {
            /* Error condition. */
        }
    }
```

Functional Description

Not applicable.

References

Refer also to the Die Temperature component data sheet.

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.