

EEPROM Emulation Reference Manual

REVISION

Generated by Doxygen 1.4.6

Wed Sep 23 16:49:23 2009

Contents

Chapter 1

EEPROM Emulation Module Documentation

1.1 Definitions

Defines

- #define [CMD_BUF_SIZE](#) 64
- #define [ERROR_UNHANDLED](#) -3

Functions

- int [Cmd_clear](#) (int argc, char *argv[])
- int [Cmd_dump](#) (int argc, char *argv[])
- int [Cmd_help](#) (int argc, char *argv[])
- int [Cmd_read](#) (int argc, char *argv[])
- int [Cmd_write](#) (int argc, char *argv[])
- void [OutputErrorAndQuit](#) (unsigned long ulError)

Variables

- static char [g_cCmdBuf](#) [CMD_BUF_SIZE]
- tCmdLineEntry [g_sCmdTable](#) []

1.1.1 Define Documentation

1.1.1.1 `#define CMD_BUF_SIZE 64`

The size of the buffer that holds the command line entry.

1.1.1.2 `#define ERROR_UNHANDLED -3`

The error code returned when an error occurs that does not need to be handled other than notifying the user. -1 and -2 are already taken by the command line processor.

1.1.2 Function Documentation

1.1.2.1 `int Cmd_clear (int argc, char * argv[])`

Implements the clear command.

This function implements the "c" (clear) command. It clears the contents of the soft EEPROM. An argument count (argc) of zero is expected.

Parameters:

argc is the argument count from the command line.

argv is a pointer to the argument buffer from the command line.

Returns:

A value of 0 indicates that the command was successful. A non-zero value indicates a failure.

1.1.2.2 `int Cmd_dump (int argc, char * argv[])`

Implements the dump command.

This function implements the "d" (dump) command. It dumps the contents of the soft EEPROM. An argument count (argc) of zero is expected.

Parameters:

argc is the argument count from the command line.

argv is a pointer to the argument buffer from the command line.

Returns:

A value of 0 indicates that the command was successful. A non-zero value indicates a failure.

1.1.2.3 int Cmd_help (int *argc*, char * *argv* [])

Implements the help command.

This function implements the "help" command. It prints a simple list of the available commands with a brief description. An argument count (*argc*) of zero is expected.

Parameters:

argc is the argument count from the command line.

argv is a pointer to the argument buffer from the command line.

Returns:

This function always returns a value of 0.

1.1.2.4 int Cmd_read (int *argc*, char * *argv* [])

Implements the read command.

This function implements the "r" (read) command. It reads the data for the specified ID from the soft EEPROM and displays the value to the user. An argument count (*argc*) of one is expected.

Parameters:

argc is the argument count from the command line.

argv is a pointer to the argument buffer from the command line.

Returns:

A value of 0 indicates that the command was successful. A non-zero value indicates a failure.

1.1.2.5 int Cmd_write (int *argc*, char * *argv* [])

Implements the write command.

This function implements the "w" (write) command. It writes the specified data for the specified ID to the soft EEPROM. An argument count (*argc*) of two is expected.

Parameters:

argc is the argument count from the command line.

argv is a pointer to the argument buffer from the command line.

Returns:

A value of 0 indicates that the command was successful. A non-zero value indicates a failure.

1.1.2.6 void OutputErrorAndQuit (unsigned long *ulError*)

Outputs the error and quits.

This function will output an error message for the given error code and enter an infinite loop. The message is sent out via UART0.

Parameters:

ulError is the error code returned by the soft EEPROM drivers.

Returns:

None.

1.1.3 Variable Documentation

1.1.3.1 char [g_cCmdBuf](#)[CMD_BUF_SIZE] [static]

The buffer that holds the command line entry.

1.1.3.2 tCmdLineEntry [g_sCmdTable](#)[]

Initial value:

```

{
    { "help",    Cmd_help,    " : Display list of commands" },
    { "h",       Cmd_help,    " : alias for help" },
    { "?",       Cmd_help,    " : alias for help" },
    { "w",       Cmd_write,   " : Write soft EEPROM - Usage: w <ID> <Value>" },
    { "r",       Cmd_read,    " : Read soft EEPROM - Usage: r <ID>" },
    { "c",       Cmd_clear,   " : Clear soft EEPROM - Usage: c" },
    { "d",       Cmd_dump,    " : Dump soft EEPROM - Usage: d" },
    { 0, 0, 0 }
}

```

The table that holds the command names, implementing functions, and brief description.

1.2 Definitions

Defines

- #define `EEPROM_BOUNDARY` 0x1000
- #define `ERASED_WORD` 0xFFFFFFFF
- #define `ERR_ACTIVE_PG_CNT` 0x0005
- #define `ERR_AVAIL_ENTRY` 0x0007
- #define `ERR_ILLEGAL_ID` 0x0002
- #define `ERR_NOT_INIT` 0x0001
- #define `ERR_PG_ERASE` 0x0003
- #define `ERR_PG_WRITE` 0x0004
- #define `ERR_RANGE` 0x0006
- #define `ERR_SWAP` 0x8000
- #define `ERR_TWO_ACTIVE_NO_FULL` 0x0008
- #define `NUM_IDS` 255
- #define `NUM_VECTOR_BITS` (NUM_IDS + 8 - (NUM_IDS % 8))
- #define `NUM_VECTOR_BYTES` (NUM_VECTOR_BITS / 8)

Functions

- static unsigned char `GetActivePageCount` (void)
- static unsigned char * `GetMostRecentlyUsedPage` (void)
- static unsigned char * `GetNextAvailEntry` (void)
- static unsigned char `GetUsedPageCount` (void)
- static long `PageDataWrite` (unsigned long *pulData, unsigned char *pucPgAddr, unsigned char ucByteCount)
- static long `PageErase` (unsigned char *pucPageAddr)
- static tBoolean `PageIsActive` (unsigned char *pucPageAddr)
- static tBoolean `PageIsUsed` (unsigned char *pucPageAddr)
- static long `PageSwap` (unsigned char *pucFullPageAddr)
- long `SoftEEPROMClear` (void)
- long `SoftEEPROMInit` (unsigned long ulStart, unsigned long ulEnd, unsigned long ulSize)
- long `SoftEEPROMRead` (unsigned char ucID, unsigned short *pusData, t-Boolean *pbFound)
- long `SoftEEPROMWrite` (unsigned short ucID, unsigned short usData)

Variables

- static tBoolean [g_bEEPROMInitialized](#) = false
- static unsigned char * [g_pucActivePage](#)
- static unsigned char * [g_pucEEPROMEnd](#)
- static unsigned char * [g_pucEEPROMStart](#)
- static unsigned char * [g_pucNextAvailEntry](#)
- static unsigned long [g_ulEEPROMPgSize](#)

1.2.1 Define Documentation

1.2.1.1 #define EEPROM_BOUNDARY 0x1000

The EEPROM pages must be aligned on a 4K boundary. This definition is used for parameter checking and is not intended to be modified.

1.2.1.2 #define ERASED_WORD 0xFFFFFFFF

The value that is read when the status words and entries are in the erased state. This definition is not intended to be modified.

1.2.1.3 #define ERR_ACTIVE_PG_CNT 0x0005

The error code returned by [SoftEEPROMInit\(\)](#) when more than two active pages are found upon initialization.

1.2.1.4 #define ERR_AVAIL_ENTRY 0x0007

The error code returned by [SoftEEPROMWrite\(\)](#) if a page swap occurs and the next available entry is outside of the new page. If a 1-K page size is used ($1024/4 - 2 = 254$ entries) and all 255 IDs are used (0-254), then it is possible to swap to a new page and not have any available entries in the new page. This can be avoided by configuring the EEPROM appropriately. The actual returned value is `ERR_AVAIL_ENTRY | ERR_SWAP`.

1.2.1.5 #define ERR_ILLEGAL_ID 0x0002

The error code used to indicate that a read/write request has been made from/to an illegal identifier. This error code can be returned by [SoftEEPROMWrite\(\)](#) and [SoftEEPROMRead\(\)](#).

1.2.1.6 #define ERR_NOT_INIT 0x0001

The error code used to indicate that an EEPROM operation has been requested without the EEPROM being initialized. This error code can be returned by [SoftEEPROMWrite\(\)](#), [SoftEEPROMRead\(\)](#), and [SoftEEPROMClear\(\)](#).

1.2.1.7 #define ERR_PG_ERASE 0x0003

The error code used to indicate that an error occurred while erasing an EEPROM page. This error code can be returned by [SoftEEPROMInit\(\)](#), [SoftEEPROMClear\(\)](#), and [SoftEEPROMWrite\(\)](#). When returned by [SoftEEPROMWrite\(\)](#), the actual returned value is `ERR_PG_ERASE | ERR_SWAP`.

1.2.1.8 #define ERR_PG_WRITE 0x0004

The error code used to indicate that an error occurred while writing to an EEPROM page. This can occur while writing EEPROM page status words and EEPROM entries. This error code can be returned by [SoftEEPROMInit\(\)](#), [SoftEEPROMClear\(\)](#), and [SoftEEPROMWrite\(\)](#). [SoftEEPROMWrite\(\)](#) can return both `ERR_PG_WRITE` and `ERR_PG_WRITE | ERR_SWAP`.

1.2.1.9 #define ERR_RANGE 0x0006

The error code returned by [SoftEEPROMInit\(\)](#) when the EEPROM region is specified to be outside the Flash range.

1.2.1.10 #define ERR_SWAP 0x8000

Part of the error code returned by [SoftEEPROMWrite\(\)](#) if an error occurs during the swap portion of the write operation. `ERR_SWAP` is ORed in with the error code encountered during the swap operation (`ERR_PG_ERASE`, `ERR_PG_WRITE`, or `ERR_AVAIL_ENTRY`). This is provided since the user never calls the swap operation directly and otherwise would not know that the error occurred during a swap operation.

1.2.1.11 #define ERR_TWO_ACTIVE_NO_FULL 0x0008

The error code returned by [SoftEEPROMInit\(\)](#) when 2 active pages are found but neither of them are full. The only time there should be 2 active pages is when a reset or power-down occurs at a specific time interval during [PageSwap\(\)](#). However, [PageSwap\(\)](#) should only be called when a page is full.

1.2.1.12 `#define NUM_IDS 255`

The number of EEPROM identifiers allowed. The identifier in each EEPROM entry is 8 bits. 0xFF is not allowed since it indicates an empty entry. Therefore, the maximum number of identifiers allowed is 255 (0 - 254).

1.2.1.13 `#define NUM_VECTOR_BITS (NUM_IDS + 8 - (NUM_IDS % 8))`

The number of bits needed for the bit vector to store the status needed to determine if data for a given identifier has already been copied to the new page during [PageSwap\(\)](#). This needs to be a multiple of 8 so that we can store the bit vector in a byte array. This definition is not intended to be modified.

1.2.1.14 `#define NUM_VECTOR_BYTES (NUM_VECTOR_BITS / 8)`

The number of bytes needed for the bit vector to store the status needed to determine if data for a given identifier has already been copied to the new page during [PageSwap\(\)](#). This definition is not intended to be modified.

1.2.2 Function Documentation

1.2.2.1 `static unsigned char GetActivePageCount (void)` `[static]`

Gets the number of active pages.

This function searches the EEPROM pages counting the number of pages marked as active.

Returns:

The number of EEPROM pages marked as active.

1.2.2.2 `static unsigned char* GetMostRecentlyUsedPage (void)` `[static]`

Gets the most recently used page.

This function searches the EEPROM pages to find the most recently used page.

Returns:

A pointer to the beginning address of the page that is most recently used. If no used pages are found, the pointer points to 0xFFFFFFFF.

1.2.2.3 static unsigned char* GetNextAvailEntry (void) [static]

Gets the next available entry.

The function finds the next available EEPROM entry in the currently active page. If the active page is full, then it returns the address just outside of the active EEPROM page. This will be detected by the next write operation and [PageSwap\(\)](#) will be called. The `g_pucActivePage` variable must be initialized prior to calling this function.

Returns:

The address of the next available EEPROM entry.

1.2.2.4 static unsigned char GetUsedPageCount (void) [static]

Gets the number of used pages.

This function searches the EEPROM pages counting the number of pages marked as used.

Returns:

The number of EEPROM pages marked as used.

1.2.2.5 static long PageDataWrite (unsigned long * *pulData*, unsigned char * *pucPgAddr*, unsigned char *ucByteCount*) [static]

Writes a word to an EEPROM page.

This function is called to write a word (32 bits) to an EEPROM page. It verifies that the write was successful by reading the location and comparing against the value written.

Parameters:

pulData is the pointer to the data to be written.

pucPgAddr is the pointer to the target address.

ucByteCount is the number of bytes to be written.

Returns:

A value of 0 indicates that the write was successful. A non-zero value indicates a failure.

1.2.2.6 static long PageErase (unsigned char * *pucPageAddr*) [static]

Erases an EEPROM page.

This function is called to erase an EEPROM page. It verifies that the page has been erased by reading the status words of the page and confirming that they read as 0xFFFFFFFF.

Parameters:

pucPageAddr is the beginning address of the EEPROM page to erase.

Returns:

A value of 0 indicates that the erase was successful. A non-zero value indicates a failure.

1.2.2.7 static tBoolean PageIsActive (unsigned char * *pucPageAddr*) [static]

Checks if the page is active.

This function is called to determine if the given EEPROM page is marked as active.

Parameters:

pucPageAddr is the starting address of the EEPROM page to check for the active status.

Returns:

A value of true indicates that the page is marked as active. A return value of false indicates that the page is not marked as active.

1.2.2.8 static tBoolean PageIsUsed (unsigned char * *pucPageAddr*) [static]

Checks if the page is used.

This function is called to determine if the given EEPROM page is marked as used.

Parameters:

pucPageAddr is the starting address of the EEPROM page to check for the used status.

Returns:

A value of true indicates that the page is marked as used. A return value of false indicates that the page is not marked as used.

1.2.2.9 static long PageSwap (unsigned char * *pucFullPageAddr*) [static]

Copies the most recent data from the full page to the new page.

This function is called when an EEPROM page is full and is responsible for copying the most recent data for each ID to the next page. The global variables containing the beginning address of the active page and the next available entry are updated in this function.

Order of operations:

1. Erase the next page.
2. Move the current data for each ID to the next page.
3. Mark the next page as the active page (increment the counter).
4. Mark the full page as used.

It is possible that a power loss or reset could occur during the swap process. [Soft-EEPROMInit\(\)](#) is called upon the subsequent boot up and will attempt to detect and take the appropriate corrective actions if necessary.

Parameters:

pucFullPageAddr is a pointer to beginning of the full page.

Returns:

A value of 0 indicates that the page swap was successful. A non-zero value indicates a failure.

1.2.2.10 long SoftEEPROMClear (void)

Clears the EEPROM contents.

This function clears the EEPROM contents. Order of operations:

1. Mark the current page as used.
2. Erase the next page.
3. Mark the erased page as the active page (increment the counter).

It is possible that a power loss or reset could occur during the clear process. [Soft-EEPROMInit\(\)](#) is called upon the subsequent boot up and will attempt to detect and take the appropriate corrective actions if necessary.

Returns:

A value of 0 indicates that the clear operation was successful. A non-zero value indicates a failure.

1.2.2.11 **long SoftEEPROMInit (unsigned long *ulStart*, unsigned long *ulEnd*, unsigned long *ulSize*)**

Initializes the emulated EEPROM.

This function initializes the EEPROM emulation area within the Flash. This function must be called prior to using any of the other functions in the API. It is expected that SysCtlClockSet() is called prior to calling this function due to SysCtlClockGet() being used by this function.

Parameters:

ulStart is the start address for the EEPROM region. This address must be aligned on a 4K boundary.

ulEnd is the end address for the EEPROM region. This address is not inclusive. That is, it is the first address just after the EEPROM emulation region. It must be aligned on a 4K boundary. It can be the first location after the end of the Flash array if the last Flash page is used for EEPROM emulation.

ulSize is the size of each EEPROM page. This must be evenly divisible into the total EEPROM emulation region. The size must be specified such to allow for at least two EEPROM emulation pages.

Returns:

A value of 0 indicates that the initialization was successful. A non-zero value indicates a failure.

1.2.2.12 **long SoftEEPROMRead (unsigned char *ucID*, unsigned short * *pusData*, tBoolean * *pbFound*)**

Reads the most recent EEPROM entry for the given ID.

This function reads the data for the specified ID. It will search the active EEPROM page from the end to get the most recent data for the given ID. 0xFF is not a valid ID as it is equivalent to being erased.

Parameters:

ucID is the ID to search for.

pusData is the address to store the data for the specified ID. The address is written with 0xFFFF if the ID is not found, thereby emulating a real EEPROM being erased at that address.

pbFound is the address to store the boolean found value. If the ID is found, this address will be set to true. If the ID is not found, this address will be set to false.

Returns:

A value of 0 indicates that the read was successful. A non-zero value indicates a failure.

1.2.2.13 long SoftEEPROMWrite (unsigned short *ucID*, unsigned short *usData*)

Writes an EEPROM entry.

This function writes the specified ID and data to the next entry available in the active EEPROM page. If the page is full, [PageSwap\(\)](#) will be called.

Parameters:

ucID is the identifier associated with the data.

usData is the data to be saved for the given ID.

Returns:

A value of 0 indicates that the write was successful. A non-zero value indicates a failure.

1.2.3 Variable Documentation

1.2.3.1 tBoolean [g_bEEPROMInitialized](#) = false [static]

Boolean variable that is set to true after the EEPROM emulation region has been initialized. Initially set to false at start-up.

1.2.3.2 unsigned char* [g_pucActivePage](#) [static]

The beginning address of the active EEPROM emulation page.

1.2.3.3 unsigned char* [g_pucEEPROMEnd](#) [static]

The ending address of the EEPROM emulation region. This address is not inclusive. That is, it is the first address just after the EEPROM emulation region. It must be aligned on a 4K boundary. It can be the first location after the end of the Flash array if the last Flash page is used for EEPROM emulation.

1.2.3.4 unsigned char* [g_pucEEPROMStart](#) [static]

The starting address of the EEPROM emulation region. This must be aligned on a 4K boundary.

1.2.3.5 unsigned char* [g_pucNextAvailEntry](#) [static]

The address of the next available EEPROM entry in the active page.

1.2.3.6 unsigned long [g_ulEEPROMPgSize](#) [static]

The size of the EEPROM emulation pages in bytes. This must be evenly divisible into the total EEPROM emulation region. The size must be specified such to allow for at least two EEPROM emulation pages.

1.3 Example_list

This software is intended to provide a simple example of using the EEPROM emulation drivers. UART0 is used as a user interface to allow the user to read and write emulated EEPROM variables. In addition, the user can dump and clear the emulated EEPROM contents.

Software Emulation of EEPROM (softeeprom)

This software is intended to provide example drivers for EEPROM emulation.

Chapter 2

EEPROM Emulation Page Documentation

Chapter 3

Introduction

This document describes the functions, variables, structures, and defines in the EEPROM emulation software. The software contains the EEPROM drivers as well as an example which uses these drivers. The drivers use the internal Flash on the Stellaris family of microcontrollers to emulate EEPROM.

The Application Programmer's Interface (API) provides the user the ability to initialize the emulated EEPROM region, write and read from the emulated EEPROM, and clear the emulated EEPROM contents.

Extract the source code for the EEPROM emulation example and drivers from the ZIP file into the directory that contains the Stellaris Firmware Development Package. For example, if the Firmware Development Package was extracted to `C:\` (meaning there is a `C:\StellarisWare` directory), then the ZIP file for the source should be extracted to `C:\StellarisWare`. It will create a `StellarisWare\AppNotes\sw01267` directory that contains the source code for the application.

Once extracted, the code can be built just like any other StellarisWare example; either by typing "make" from the command line or by using the Keil RV-MDK, IAR EW-ARM, or Code Red red_suite project files.

Chapter 4

EEPROM Emulation Example App

softEEPROM_example_introIntroduction
group softeepprom_exampleDefinitions

4.1 Introduction

This module provides a simple example using the EEPROM emulation drivers. UART0 is used as a user interface to allow the user to read and write up to 16 emulated EEPROM variables. In addition, the user can dump and clear the emulated EEPROM contents. The example is configured for a board with an 8-MHz crystal. Change the appropriate parameter in the call to SysCtlClockSet() if a different crystal is used on the board.

To run the example:

1. Start Hyperterminal.
2. Select the COM port associated with the board.
3. Select 115200 bits per second.
4. Select 8 data bits.
5. Select no parity.
6. Select 1 stop bit.
7. Select no flow control.

Once the example is running, you should see a prompt in the Hyperterminal window. IDs 0 - 15 are supported. The data size is 16 bits.

Available commands:

help : Display list of commands

- h : alias for help
- ? : alias for help
- w : Write soft EEPROM - Usage: w <ID>
- r : Read soft EEPROM - Usage: r <ID>
- c : Clear soft EEPROM - Usage: c
- d : Dump soft EEPROM - Usage: d

Example:

> r 0

ID 0: 0xffff (65535) - The variable with this ID has not been initialized.

> r 1

ID 1: 0xffff (65535) - The variable with this ID has not been initialized.

> w 0 0xbeef

> w 1 0xdead

> r 0

ID 0: 0xbeef (48879)

> r 1

ID 1: 0xdead (57005)

> c

> r 0

ID 0: 0xffff (65535) - The variable with this ID has not been initialized.

> r 1

ID 1: 0xffff (65535) - The variable with this ID has not been initialized.

4.2 Definitions

Defines

- `#define CMD_BUF_SIZE 64`
- `#define ERROR_UNHANDLED -3`

Functions

- `int Cmd_clear (int argc, char *argv[])`
- `int Cmd_dump (int argc, char *argv[])`
- `int Cmd_help (int argc, char *argv[])`
- `int Cmd_read (int argc, char *argv[])`
- `int Cmd_write (int argc, char *argv[])`
- `void OutputErrorAndQuit (unsigned long ulError)`

Variables

- `static char g_cCmdBuf [CMD_BUF_SIZE]`
- `tCmdLineEntry g_sCmdTable []`

4.2.1 Define Documentation

4.2.1.1 `#define CMD_BUF_SIZE 64`

The size of the buffer that holds the command line entry.

4.2.1.2 `#define ERROR_UNHANDLED -3`

The error code returned when an error occurs that does not need to be handled other than notifying the user. -1 and -2 are already taken by the command line processor.

4.2.2 Function Documentation

4.2.2.1 `int Cmd_clear (int argc, char * argv[])`

Implements the clear command.

This function implements the "c" (clear) command. It clears the contents of the soft EEPROM. An argument count (argc) of zero is expected.

Parameters:

argc is the argument count from the command line.

argv is a pointer to the argument buffer from the command line.

Returns:

A value of 0 indicates that the command was successful. A non-zero value indicates a failure.

4.2.2.2 int Cmd_dump (int argc, char * argv[])

Implements the dump command.

This function implements the "d" (dump) command. It dumps the contents of the soft EEPROM. An argument count (argc) of zero is expected.

Parameters:

argc is the argument count from the command line.

argv is a pointer to the argument buffer from the command line.

Returns:

A value of 0 indicates that the command was successful. A non-zero value indicates a failure.

4.2.2.3 int Cmd_help (int argc, char * argv[])

Implements the help command.

This function implements the "help" command. It prints a simple list of the available commands with a brief description. An argument count (argc) of zero is expected.

Parameters:

argc is the argument count from the command line.

argv is a pointer to the argument buffer from the command line.

Returns:

This function always returns a value of 0.

4.2.2.4 int Cmd_read (int argc, char * argv[])

Implements the read command.

This function implements the "r" (read) command. It reads the data for the specified ID from the soft EEPROM and displays the value to the user. An argument count (argc) of one is expected.

Parameters:

argc is the argument count from the command line.

argv is a pointer to the argument buffer from the command line.

Returns:

A value of 0 indicates that the command was successful. A non-zero value indicates a failure.

4.2.2.5 int Cmd_write (int argc, char * argv[])

Implements the write command.

This function implements the "w" (write) command. It writes the specified data for the specified ID to the soft EEPROM. An argument count (argc) of two is expected.

Parameters:

argc is the argument count from the command line.

argv is a pointer to the argument buffer from the command line.

Returns:

A value of 0 indicates that the command was successful. A non-zero value indicates a failure.

4.2.2.6 void OutputErrorAndQuit (unsigned long ulError)

Outputs the error and quits.

This function will output an error message for the given error code and enter an infinite loop. The message is sent out via UART0.

Parameters:

ulError is the error code returned by the soft EEPROM drivers.

Returns:

None.

4.2.3 Variable Documentation**4.2.3.1 char g_cCmdBuf[CMD_BUF_SIZE] [static]**

The buffer that holds the command line entry.

4.2.3.2 tCmdLineEntry g_sCmdTable[]

Initial value:

```
{
    { "help",    Cmd_help,      " : Display list of commands" },
    { "h",       Cmd_help,      " : alias for help" },
    { "?",       Cmd_help,      " : alias for help" },
    { "w",       Cmd_write,     " : Write soft EEPROM - Usage: w <ID> <Value>" },
    { "r",       Cmd_read,      " : Read soft EEPROM - Usage: r <ID>" },
    { "c",       Cmd_clear,     " : Clear soft EEPROM - Usage: c" },
    { "d",       Cmd_dump,      " : Dump soft EEPROM - Usage: d" },
    { 0, 0, 0 }
}
```

The table that holds the command names, implementing functions, and brief description.

Chapter 5

EEPROM Emulation Drivers

softeprom_introIntroduction
group_softeprom_apiDefinitions

5.1 Introduction

This module contains the EEPROM emulation user API along with functions used by the API that the user does not have access to.

The user API consists of the following functions:

- [SoftEEPROMInit\(\)](#) - Used to initialize the emulated EEPROM.
- [SoftEEPROMWrite\(\)](#) - Used to store a variable in the emulated EEPROM.
- [SoftEEPROMRead\(\)](#) - Used to read a variable from the emulated EEPROM.
- [SoftEEPROMClear\(\)](#) - Used to clear the contents of the emulated EEPROM.

[SoftEEPROMInit\(\)](#) must be called and must return without error before calling [SoftEEPROMWrite\(\)](#), [SoftEEPROMRead\(\)](#), and [SoftEEPROMClear\(\)](#).

The other functions are static functions used by the API. These static functions are not intended to be used by the user.

The internal Flash of the Stellaris family of microcontrollers is used to emulate real EEPROM. The region of the Flash to be used for EEPROM emulation is specified as parameters when [SoftEEPROMInit\(\)](#) is called. The size of the region to be used must be a multiple of 4K bytes and the starting address of the region must be aligned on a 4K byte boundary. This is due to the Flash architecture of the Stellaris microcontrollers. The EEPROM region must consist of at least two EEPROM pages. The size of each

EEPROM page is specified as a parameter when [SoftEEPROMInit\(\)](#) is called. The internal Flash of the Stellaris microcontrollers is divided into 1K byte erasable blocks. Therefore, the size of the EEPROM pages must be a multiple of 1K bytes. In addition, the size of the EEPROM pages must be evenly divisible into to total EEPROM region.

The beginning of each EEPROM page consists of two 32-bit status words used by the emulation software. The remainder of the EEPROM page is divided into EEPROM entries. The Flash for Stellaris microcontrollers has the restriction that each 32-bit word can only be programmed one time between erase operations. Due to this, each EEPROM entry is 32-bits. Each entry consists of an 8-bit identifier and 16-bits of data. The EEPROM emulation software could be modified to support 24-bit data and not have 8 bits wasted for each entry.

The code for this module is contained in the `softeeprom.c` file, with the `softeeprom.h` file containing definitions and functions exported to the rest of the application.

5.2 Definitions

Defines

- `#define` [EEPROM_BOUNDARY](#) 0x1000
- `#define` [ERASED_WORD](#) 0xFFFFFFFF
- `#define` [ERR_ACTIVE_PG_CNT](#) 0x0005
- `#define` [ERR_AVAIL_ENTRY](#) 0x0007
- `#define` [ERR_ILLEGAL_ID](#) 0x0002
- `#define` [ERR_NOT_INIT](#) 0x0001
- `#define` [ERR_PG_ERASE](#) 0x0003
- `#define` [ERR_PG_WRITE](#) 0x0004
- `#define` [ERR_RANGE](#) 0x0006
- `#define` [ERR_SWAP](#) 0x8000
- `#define` [ERR_TWO_ACTIVE_NO_FULL](#) 0x0008
- `#define` [NUM_IDS](#) 255
- `#define` [NUM_VECTOR_BITS](#) (NUM_IDS + 8 - (NUM_IDS % 8))
- `#define` [NUM_VECTOR_BYTES](#) (NUM_VECTOR_BITS / 8)

Functions

- static unsigned char [GetActivePageCount](#) (void)
- static unsigned char * [GetMostRecentlyUsedPage](#) (void)
- static unsigned char * [GetNextAvailEntry](#) (void)
- static unsigned char [GetUsedPageCount](#) (void)

- static long [PageDataWrite](#) (unsigned long *pulData, unsigned char *pucPgAddr, unsigned char ucByteCount)
- static long [PageErase](#) (unsigned char *pucPageAddr)
- static tBoolean [PageIsActive](#) (unsigned char *pucPageAddr)
- static tBoolean [PageIsUsed](#) (unsigned char *pucPageAddr)
- static long [PageSwap](#) (unsigned char *pucFullPageAddr)
- long [SoftEEPROMClear](#) (void)
- long [SoftEEPROMInit](#) (unsigned long ulStart, unsigned long ulEnd, unsigned long ulSize)
- long [SoftEEPROMRead](#) (unsigned char ucID, unsigned short *pusData, t-Boolean *pbFound)
- long [SoftEEPROMWrite](#) (unsigned short ucID, unsigned short usData)

Variables

- static tBoolean [g_bEEPROMInitialized](#) = false
- static unsigned char * [g_pucActivePage](#)
- static unsigned char * [g_pucEEPROMEnd](#)
- static unsigned char * [g_pucEEPROMStart](#)
- static unsigned char * [g_pucNextAvailEntry](#)
- static unsigned long [g_ulEEPROMPgSize](#)

5.2.1 Define Documentation

5.2.1.1 #define EEPROM_BOUNDARY 0x1000

The EEPROM pages must be aligned on a 4K boundary. This definition is used for parameter checking and is not intended to be modified.

5.2.1.2 #define ERASED_WORD 0xFFFFFFFF

The value that is read when the status words and entries are in the erased state. This definition is not intended to be modified.

5.2.1.3 #define ERR_ACTIVE_PG_CNT 0x0005

The error code returned by [SoftEEPROMInit\(\)](#) when more than two active pages are found upon initialization.

5.2.1.4 **#define ERR_AVAIL_ENTRY 0x0007**

The error code returned by [SoftEEPROMWrite\(\)](#) if a page swap occurs and the next available entry is outside of the new page. If a 1-K page size is used ($1024/4 - 2 = 254$ entries) and all 255 IDs are used (0-254), then it is possible to swap to a new page and not have any available entries in the new page. This can be avoided by configuring the EEPROM appropriately. The actual returned value is `ERR_AVAIL_ENTRY | ERR_SWAP`.

5.2.1.5 **#define ERR_ILLEGAL_ID 0x0002**

The error code used to indicate that a read/write request has been made from/to an illegal identifier. This error code can be returned by [SoftEEPROMWrite\(\)](#) and [SoftEEPROMRead\(\)](#).

5.2.1.6 **#define ERR_NOT_INIT 0x0001**

The error code used to indicate that an EEPROM operation has been requested without the EEPROM being initialized. This error code can be returned by [SoftEEPROMWrite\(\)](#), [SoftEEPROMRead\(\)](#), and [SoftEEPROMClear\(\)](#).

5.2.1.7 **#define ERR_PG_ERASE 0x0003**

The error code used to indicate that an error occurred while erasing an EEPROM page. This error code can be returned by [SoftEEPROMInit\(\)](#), [SoftEEPROMClear\(\)](#), and [SoftEEPROMWrite\(\)](#). When returned by [SoftEEPROMWrite\(\)](#), the actual returned value is `ERR_PG_ERASE | ERR_SWAP`.

5.2.1.8 **#define ERR_PG_WRITE 0x0004**

The error code used to indicate that an error occurred while writing to an EEPROM page. This can occur while writing EEPROM page status words and EEPROM entries. This error code can be returned by [SoftEEPROMInit\(\)](#), [SoftEEPROMClear\(\)](#), and [SoftEEPROMWrite\(\)](#). [SoftEEPROMWrite\(\)](#) can return both `ERR_PG_WRITE` and `ERR_PG_WRITE | ERR_SWAP`.

5.2.1.9 **#define ERR_RANGE 0x0006**

The error code returned by [SoftEEPROMInit\(\)](#) when the EEPROM region is specified to be outside the Flash range.

5.2.1.10 #define ERR_SWAP 0x8000

Part of the error code returned by [SoftEEPROMWrite\(\)](#) if an error occurs during the swap portion of the write operation. ERR_SWAP is ORed in with the error code encountered during the swap operation (ERR_PG_ERASE, ERR_PG_WRITE, or ERR_AVAIL_ENTRY). This is provided since the user never calls the swap operation directly and otherwise would not know that the error occurred during a swap operation.

5.2.1.11 #define ERR_TWO_ACTIVE_NO_FULL 0x0008

The error code returned by [SoftEEPROMInit\(\)](#) when 2 active pages are found but neither of them are full. The only time there should be 2 active pages is when a reset or power-down occurs at a specific time interval during [PageSwap\(\)](#). However, [PageSwap\(\)](#) should only be called when a page is full.

5.2.1.12 #define NUM_IDS 255

The number of EEPROM identifiers allowed. The identifier in each EEPROM entry is 8 bits. 0xFF is not allowed since it indicates an empty entry. Therefore, the maximum number of identifiers allowed is 255 (0 - 254).

5.2.1.13 #define NUM_VECTOR_BITS (NUM_IDS + 8 - (NUM_IDS % 8))

The number of bits needed for the bit vector to store the status needed to determine if data for a given identifier has already been copied to the new page during [PageSwap\(\)](#). This needs to be a multiple of 8 so that we can store the bit vector in a byte array. This definition is not intended to be modified.

5.2.1.14 #define NUM_VECTOR_BYTES (NUM_VECTOR_BITS / 8)

The number of bytes needed for the bit vector to store the status needed to determine if data for a given identifier has already been copied to the new page during [PageSwap\(\)](#). This definition is not intended to be modified.

5.2.2 Function Documentation**5.2.2.1 static unsigned char GetActivePageCount (void) [static]**

Gets the number of active pages.

This function searches the EEPROM pages counting the number of pages marked as active.

Returns:

The number of EEPROM pages marked as active.

5.2.2.2 static unsigned char* GetMostRecentlyUsedPage (void) [static]

Gets the most recently used page.

This function searches the EEPROM pages to find the most recently used page.

Returns:

A pointer to the beginning address of the page that is most recently used. If no used pages are found, the pointer points to 0xFFFFFFFF.

5.2.2.3 static unsigned char* GetNextAvailEntry (void) [static]

Gets the next available entry.

The function finds the next available EEPROM entry in the currently active page. If the active page is full, then it returns the address just outside of the active EEPROM page. This will be detected by the next write operation and [PageSwap\(\)](#) will be called. The `g_pucActivePage` variable must be initialized prior to calling this function.

Returns:

The address of the next available EEPROM entry.

5.2.2.4 static unsigned char GetUsedPageCount (void) [static]

Gets the number of used pages.

This function searches the EEPROM pages counting the number of pages marked as used.

Returns:

The number of EEPROM pages marked as used.

5.2.2.5 static long PageDataWrite (unsigned long * *pulData*, unsigned char * *pucPgAddr*, unsigned char *ucByteCount*) [static]

Writes a word to an EEPROM page.

This function is called to write a word (32 bits) to an EEPROM page. It verifies that the write was successful by reading the location and comparing against the value written.

Parameters:

pulData is the pointer to the data to be written.

pucPgAddr is the pointer to the target address.

ucByteCount is the number of bytes to be written.

Returns:

A value of 0 indicates that the write was successful. A non-zero value indicates a failure.

5.2.2.6 static long PageErase (unsigned char * *pucPageAddr*) [static]

Erases an EEPROM page.

This function is called to erase an EEPROM page. It verifies that the page has been erased by reading the status words of the page and confirming that they read as 0xFFFFFFFF.

Parameters:

pucPageAddr is the beginning address of the EEPROM page to erase.

Returns:

A value of 0 indicates that the erase was successful. A non-zero value indicates a failure.

5.2.2.7 static tBoolean PageIsActive (unsigned char * *pucPageAddr*) [static]

Checks if the page is active.

This function is called to determine if the given EEPROM page is marked as active.

Parameters:

pucPageAddr is the starting address of the EEPROM page to check for the active status.

Returns:

A value of true indicates that the page is marked as active. A return value of false indicates that the page is not marked as active.

5.2.2.8 static tBoolean PageIsUsed (unsigned char * *pucPageAddr*) [static]

Checks if the page is used.

This function is called to determine if the given EEPROM page is marked as used.

Parameters:

pucPageAddr is the starting address of the EEPROM page to check for the used status.

Returns:

A value of true indicates that the page is marked as used. A return value of false indicates that the page is not marked as used.

5.2.2.9 static long PageSwap (unsigned char * *pucFullPageAddr*) [static]

Copies the most recent data from the full page to the new page.

This function is called when an EEPROM page is full and is responsible for copying the most recent data for each ID to the next page. The global variables containing the beginning address of the active page and the next available entry are updated in this function.

Order of operations:

1. Erase the next page.
2. Move the current data for each ID to the next page.
3. Mark the next page as the active page (increment the counter).
4. Mark the full page as used.

It is possible that a power loss or reset could occur during the swap process. [Soft-EEPROMInit\(\)](#) is called upon the subsequent boot up and will attempt to detect and take the appropriate corrective actions if necessary.

Parameters:

pucFullPageAddr is a pointer to beginning of the full page.

Returns:

A value of 0 indicates that the page swap was successful. A non-zero value indicates a failure.

5.2.2.10 long SoftEEPROMClear (void)

Clears the EEPROM contents.

This function clears the EEPROM contents. Order of operations:

1. Mark the current page as used.
2. Erase the next page.
3. Mark the erased page as the active page (increment the counter).

It is possible that a power loss or reset could occur during the clear process. [SoftEEPROMInit\(\)](#) is called upon the subsequent boot up and will attempt to detect and take the appropriate corrective actions if necessary.

Returns:

A value of 0 indicates that the clear operation was successful. A non-zero value indicates a failure.

5.2.2.11 long SoftEEPROMInit (unsigned long *ulStart*, unsigned long *ulEnd*, unsigned long *ulSize*)

Initializes the emulated EEPROM.

This function initializes the EEPROM emulation area within the Flash. This function must be called prior to using any of the other functions in the API. It is expected that SysCtlClockSet() is called prior to calling this function due to SysCtlClockGet() being used by this function.

Parameters:

ulStart is the start address for the EEPROM region. This address must be aligned on a 4K boundary.

ulEnd is the end address for the EEPROM region. This address is not inclusive. That is, it is the first address just after the EEPROM emulation region. It must be aligned on a 4K boundary. It can be the first location after the end of the Flash array if the last Flash page is used for EEPROM emulation.

ulSize is the size of each EEPROM page. This must be evenly divisible into the total EEPROM emulation region. The size must be specified such to allow for at least two EEPROM emulation pages.

Returns:

A value of 0 indicates that the initialization was successful. A non-zero value indicates a failure.

5.2.2.12 **long SoftEEPROMRead (unsigned char *ucID*, unsigned short * *pusData*, tBoolean * *pbFound*)**

Reads the most recent EEPROM entry for the given ID.

This function reads the data for the specified ID. It will search the active EEPROM page from the end to get the most recent data for the given ID. 0xFF is not a valid ID as it is equivalent to being erased.

Parameters:

ucID is the ID to search for.

pusData is the address to store the data for the specified ID. The address is written with 0xFFFF if the ID is not found, thereby emulating a real EEPROM being erased at that address.

pbFound is the address to store the boolean found value. If the ID is found, this address will be set to true. If the ID is not found, this address will be set to false.

Returns:

A value of 0 indicates that the read was successful. A non-zero value indicates a failure.

5.2.2.13 **long SoftEEPROMWrite (unsigned short *ucID*, unsigned short *usData*)**

Writes an EEPROM entry.

This function writes the specified ID and data to the next entry available in the active EEPROM page. If the page is full, [PageSwap\(\)](#) will be called.

Parameters:

ucID is the identifier associated with the data.

usData is the data to be saved for the given ID.

Returns:

A value of 0 indicates that the write was successful. A non-zero value indicates a failure.

5.2.3 Variable Documentation

5.2.3.1 **tBoolean [g_bEEPROMInitialized](#) = false** [static]

Boolean variable that is set to true after the EEPROM emulation region has been initialized. Initially set to false at start-up.

5.2.3.2 unsigned char* [g_pucActivePage](#) [static]

The beginning address of the active EEPROM emulation page.

5.2.3.3 unsigned char* [g_pucEEPROMEnd](#) [static]

The ending address of the EEPROM emulation region. This address is not inclusive. That is, it is the first address just after the EEPROM emulation region. It must be aligned on a 4K boundary. It can be the first location after the end of the Flash array if the last Flash page is used for EEPROM emulation.

5.2.3.4 unsigned char* [g_pucEEPROMStart](#) [static]

The starting address of the EEPROM emulation region. This must be aligned on a 4K boundary.

5.2.3.5 unsigned char* [g_pucNextAvailEntry](#) [static]

The address of the next available EEPROM entry in the active page.

5.2.3.6 unsigned long [g_ulEEPROMPgSize](#) [static]

The size of the EEPROM emulation pages in bytes. This must be evenly divisible into the total EEPROM emulation region. The size must be specified such to allow for at least two EEPROM emulation pages.

This document describes the functions, variables, structures, and defines in the EEPROM emulation software. The software contains the EEPROM drivers as well as an example which uses these drivers. The drivers use the internal Flash on the Stellaris family of microcontrollers to emulate EEPROM.

The Application Programmer's Interface (API) provides the user the ability to initialize the emulated EEPROM region, write and read from the emulated EEPROM, and clear the emulated EEPROM contents.

Extract the source code for the EEPROM emulation example and drivers from the ZIP file into the directory that contains the Stellaris Firmware Development Package. For example, if the Firmware Development Package was extracted to C:\ (meaning there is a C:\StellarisWare directory), then the ZIP file for the source should be extracted to C:\StellarisWare. It will create a StellarisWare\AppNotes\sw01267 directory that contains the source code for the application.

Once extracted, the code can be built just like any other StellarisWare example; either by typing "make" from the command line or by using the Keil RV-MDK, IAR EW-ARM, or Code Red red_suite project files.

5.3 Introduction

This module provides a simple example using the EEPROM emulation drivers. UART0 is used as a user interface to allow the user to read and write up to 16 emulated EEPROM variables. In addition, the user can dump and clear the emulated EEPROM contents. The example is configured for a board with an 8-MHz crystal. Change the appropriate parameter in the call to SysCtlClockSet() if a different crystal is used on the board.

To run the example:

1. Start Hyperterminal.
2. Select the COM port associated with the board.
3. Select 115200 bits per second.
4. Select 8 data bits.
5. Select no parity.
6. Select 1 stop bit.
7. Select no flow control.

Once the example is running, you should see a prompt in the Hyperterminal window. IDs 0 - 15 are supported. The data size is 16 bits.

Available commands:

help : Display list of commands

- h : alias for help
- ? : alias for help
- w : Write soft EEPROM - Usage: w <ID>
- r : Read soft EEPROM - Usage: r <ID>
- c : Clear soft EEPROM - Usage: c
- d : Dump soft EEPROM - Usage: d

Example:

```
> r 0
```

ID 0: 0xffff (65535) - The variable with this ID has not been initialized.

> r 1

ID 1: 0xffff (65535) - The variable with this ID has not been initialized.

> w 0 0xbeef

> w 1 0xdead

> r 0

ID 0: 0xbeef (48879)

> r 1

ID 1: 0xdead (57005)

> c

> r 0

ID 0: 0xffff (65535) - The variable with this ID has not been initialized.

> r 1

ID 1: 0xffff (65535) - The variable with this ID has not been initialized.

5.4 Introduction

This module contains the EEPROM emulation user API along with functions used by the API that the user does not have access to.

The user API consists of the following functions:

- [SoftEEPROMInit\(\)](#) - Used to initialize the emulated EEPROM.
- [SoftEEPROMWrite\(\)](#) - Used to store a variable in the emulated EEPROM.
- [SoftEEPROMRead\(\)](#) - Used to read a variable from the emulated EEPROM.
- [SoftEEPROMClear\(\)](#) - Used to clear the contents of the emulated EEPROM.

[SoftEEPROMInit\(\)](#) must be called and must return without error before calling [SoftEEPROMWrite\(\)](#), [SoftEEPROMRead\(\)](#), and [SoftEEPROMClear\(\)](#).

The other functions are static functions used by the API. These static functions are not intended to be used by the user.

The internal Flash of the Stellaris family of microcontrollers is used to emulate real EEPROM. The region of the Flash to be used for EEPROM emulation is specified as parameters when [SoftEEPROMInit\(\)](#) is called. The size of the region to be used must be a multiple of 4K bytes and the starting address of the region must be aligned on a 4K byte boundary. This is due to the Flash architecture of the Stellaris microcontrollers. The EEPROM region must consist of at least two EEPROM pages. The size of each EEPROM page is specified as a parameter when [SoftEEPROMInit\(\)](#) is called. The internal Flash of the Stellaris microcontrollers is divided into 1K byte erasable blocks. Therefore, the size of the EEPROM pages must be a multiple of 1K bytes. In addition, the size of the EEPROM pages must be evenly divisible into the total EEPROM region.

The beginning of each EEPROM page consists of two 32-bit status words used by the emulation software. The remainder of the EEPROM page is divided into EEPROM entries. The Flash for Stellaris microcontrollers has the restriction that each 32-bit word can only be programmed one time between erase operations. Due to this, each EEPROM entry is 32-bits. Each entry consists of an 8-bit identifier and 16-bits of data. The EEPROM emulation software could be modified to support 24-bit data and not have 8 bits wasted for each entry.

The code for this module is contained in the `softeeprom.c` file, with the `softeeprom.h` file containing definitions and functions exported to the rest of the application.