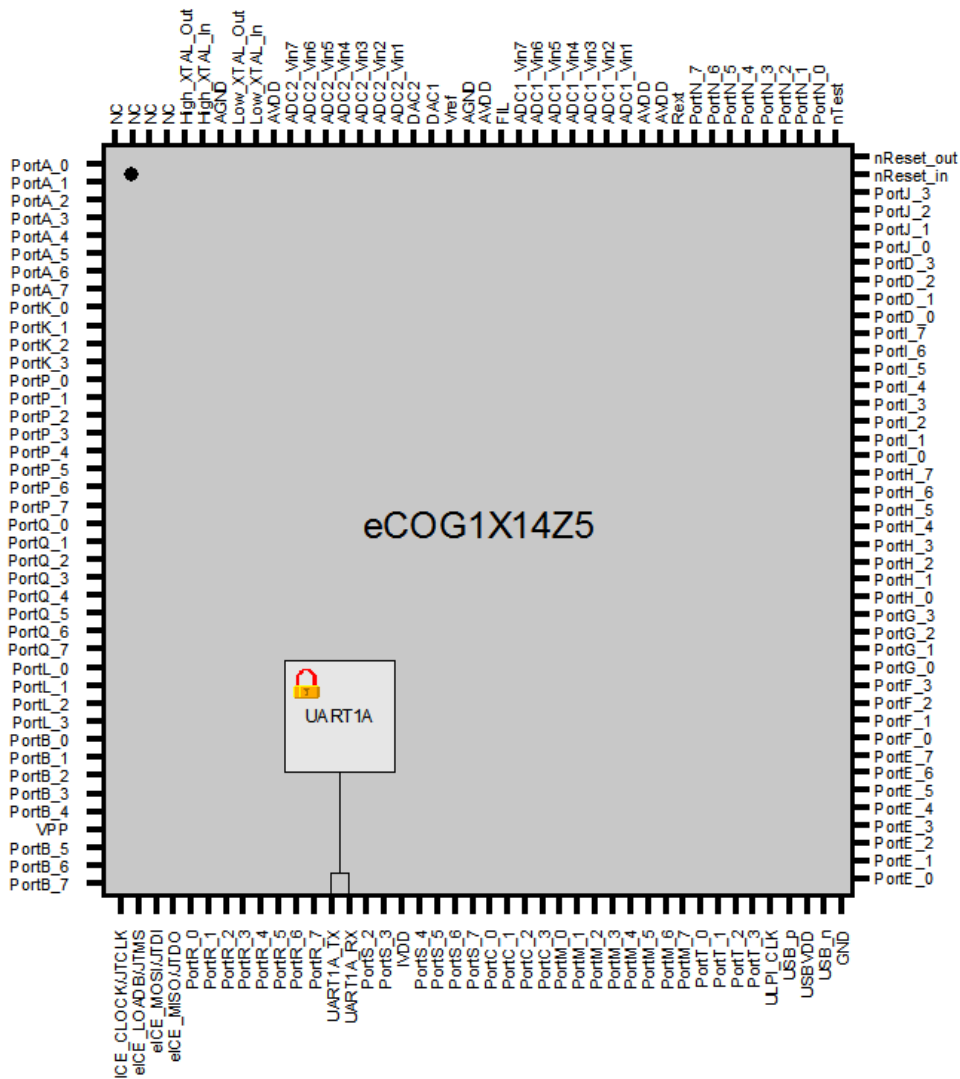




AN064 – eCOG1X Internal Flash Programming

Version 1.0

This application note describes Read, Write and Erase operations for the internal flash in the eCOG1X microcontrollers. It also describes an example application project that copies data from one flash sector to another.



Confidential and Proprietary Information

©Cyan Technology Ltd, 2008

This document contains confidential and proprietary information of Cyan Technology Ltd and is protected by copyright laws. Its receipt or possession does not convey any rights to reproduce, manufacture, use or sell anything based on information contained within this document.

Cyan Technology™, the Cyan Technology logo and Max-eICE™ are trademarks of Cyan Holdings Ltd. CyanIDE® and eCOG® are registered trademarks of Cyan Holdings Ltd. Cyan Technology Ltd recognises other brand and product names as trademarks or registered trademarks of their respective holders.

Any product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by Cyan Technology Ltd in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Cyan Technology Ltd shall not be liable for any loss or damage arising from the use of any information in this guide, any error or omission in such information, or any incorrect use of the product.

This product is not designed or intended to be used for on-line control of aircraft, aircraft navigation or communications systems or in air traffic control applications or in the design, construction, operation or maintenance of any nuclear facility, or for any medical use related to either life support equipment or any other life-critical application. Cyan Technology Ltd specifically disclaims any express or implied warranty of fitness for any or all of such uses. Ask your sales representative for details.



Revision History

Version	Date	Notes
V1.0	29/08/2007	First release

Contents

1	Introduction	5
2	Glossary	5
3	Internal Flash Operations	6
3.1	Read Memory Operations	6
3.2	Write Memory Operations.....	7
3.3	Sector Erase Operation	8
3.4	Operation Status.....	8
4	Example.....	9
4.1	Overview.....	9
4.2	Program Flow	10
4.3	Installing the Application Software.....	11
Appendix A	Software Functions	11
A.1	flashAPIs.c.....	11
	<i>flash_modify_enable()</i>	11
	<i>erase_sector()</i>	12
	<i>flash_write()</i>	12
	<i>flash_write_buffer()</i>	13
	<i>flash_read()</i>	13
	<i>flash_read_buffer()</i>	13
	<i>Program_Heap()</i>	14
	<i>Free_Heap()</i>	14

1 Introduction

This application note describes READ, WRITE and ERASE operations for the internal flash memory of the eCOG1X. These flash processes are provided as APIs in the software bundle. A simple example demonstrates a procedure for copying data from one flash sector to another.

2 Glossary

A table of abbreviations used in this document.

API	Application Programming Interface
CyanIDE	Cyan's integrated development environment
eCOG1X	Cyan target 16-bit microcontroller
eCOG1X14Z5	Product option of eCOG1X 16-bit microcontroller
IRAM	Internal Random Access Memory
IROM	Internal Read Only Memory
MCU	Micro-Controller Unit
MMU	Memory Management Unit
SA	Flash sector address

3 Internal Flash Operations

The eCOG1X contains up to 512KB embedded flash memory for program and data storage. The flash memory has 11 flash sectors of various sizes from 8KB to 64KB. The eCOG1X User Manual gives more details on the organisation and sector sizes of the internal flash.

This section describes the internal flash programming operations, which are:

Operation	Function Name	Description
<i>READ WORD</i>	flash_read()	Read one word from the target data address in the embedded flash.
<i>READ BUFFER</i>	flash_read_buffer()	Read 32 successive words from the target data address range in the embedded flash.
<i>WRITE WORD</i>	flash_write()	Write one word to the target data address in the embedded flash.
<i>WRITE BUFFER</i>	flash_write_buffer ()	Write 32 successive words to the target data address range in the embedded flash.
<i>SECTOR ERASE</i>	erase_sector()	Erase a flash sector.

Table 1. Internal flash operations

3.1 Read Memory Operations

The *READ WORD* operation reads one data word from the target address in flash memory.

The *READ BUFFER* operation iterates the *READ WORD* operation to get several successive data words starting at the specified target address in flash memory.

READ WORD

Below is the C code statement corresponding to the *READ WORD* operation:

```
data = *(unsigned int *) (FLASH_DATAx_BASE + offset);
```

FLASH_DATAx_BASE: The logical base address in data space for the internal flash mapping. The flash sector to be read should be mapped to this region.

offset: Any valid offset address within the selected flash sector relative to *FLASH_DATAx_BASE*. This indicates the target read position.

data: A 16-bit variable stores the content of the offset address specified in *flash_offset*.

READ BUFFER

The C code statements corresponding to the *READ BUFFER* operation are:

```
for (i = 0; i < 32; i++)
{
    data[i] = *(unsigned int *) (FLASH_DATAx_BASE + offset + i);
}
```

FLASH_DATAx_BASE: The logical base address in data space for the internal flash mapping. The flash sector to be read should be mapped to this region.

offset: Any valid offset address within the selected flash sector relative to *FLASH_DATAx_BASE*. This indicates the start reading position of the target region.

data[]: An array of 16-bit data words stores the contents read from the internal flash memory starting from the offset address *offset*.

i: Indicates the number of words read.

In the demonstration software, the *READ BUFFER* operation is programmed to read 32 successive words from the internal memory.

3.2 Write Memory Operations

The *WRITE WORD* operation programs one data word to the target address in flash memory.

The *WRITE BUFFER* operation programs up to 32 successive data words to the target address range in flash memory.

Note that flash memory bits can be programmed only from a '1' to a '0'. Changing bits back from a '0' to a '1' requires an erase operation, either a sector erase or a chip erase.

WRITE WORD

The command sequence for the *WRITE WORD* operation consists of four write cycles.

Address	Data
FLASH_DATAx_BASE + 0x555	0xAA
FLASH_DATAx_BASE + 0x2AA	0x55
FLASH_DATAx_BASE + 0x555	0xA0
FLASH_DATAx_BASE + offset	data[0]

Table 2. Write Word Operation

FLASH_DATAx_BASE: The logical base address in data space for the internal flash mapping. The flash sector to be programmed should be mapped to this region.

offset: Any valid offset address within the selected flash sector relative to *FLASH_DATAx_BASE*. This indicates the target program position.

data[0]: A 16-bit data to be programmed to the content of the offset address specified in *offset*.

WRITE BUFFER

The command sequence for the *WRITE BUFFER* operation consists of six or more write cycles.

Address	Data
FLASH_DATAx_BASE + 0x555	0xAA
FLASH_DATAx_BASE + 0x2AA	0x55
FLASH_DATAx_BASE + sectaddr	0x25
FLASH_DATAx_BASE + sectaddr	i-1
FLASH_DATAx_BASE + offset + 0	data[0]
FLASH_DATAx_BASE + offset + 1	data[1]
:	:
FLASH_DATAx_BASE + offset + (i-2)	data[i-2]
FLASH_DATAx_BASE + offset + (i-1)	data[i-1]
FLASH_DATAx_BASE + sectaddr	0x29

Table 3. Write Buffer Operation

FLASH_DATAx_BASE: The logical base address in data space for the internal flash mapping. The flash sector to be erased should be mapped to this region.

sectaddr: Any valid address within the target flash sector.

offset: Any valid offset address within the selected flash sector relative to *FLASH_DATAx_BASE*. This indicates the start programming position of the target region.

data[]: An array of 16-bit data word contains the data to be written to the internal flash.

i: Indicates the number of data words to be programmed.

3.3 Sector Erase Operation

Flash memory bits can be programmed only from a '1' to a '0'. Changing bits back from a '0' to a '1' requires an erase operation, either a sector erase or a chip erase.

The *SECTOR ERASE* function allows one selected flash sector to be erased in a single operation. Single word erase operation is not available in the eCOG1X internal flash.

The following table shows the command sequence for the *SECTOR ERASE* operation, consisting of six write cycles.

Address	Data
FLASH_DATAx_BASE + 0x555	0xAA
FLASH_DATAx_BASE + 0x2AA	0x55
FLASH_DATAx_BASE + 0x555	0x80
FLASH_DATAx_BASE + 0x555	0xAA
FLASH_DATAx_BASE + 0x2AA	0x55
FLASH_DATAx_BASE + offset	0x30

Table 4. Sector erase operation

FLASH_DATAx_BASE: The logical base address in data space for the internal flash mapping. The flash sector to be erased should be mapped to this region.

offset: Any valid address within the selected flash sector.

3.4 Operation Status

The result of the above flash operations can be accessed by reading the contents of the flash at the data base address:

```
status = *(unsigned int *) (FLASH_DATAx_BASE);
```

FLASH_DATAx_BASE: The logical base address in data space for the internal flash mapping. The flash sector to be read should be mapped to this region.

The least significant byte indicates the status of the operation.

Here is the summary of the status bits:

Bit	Bit Name	Description
DB7	Data Polling Bit	Inverse of data bit if operation is in progress or fails. Outputs actual data bit when the operation succeeds.
DB6	Toggle Bit 1	Continues to toggle on successive reads if an operation is in progress or fails. Outputs actual data bit when operation succeeds.
DB5	Internal Timeout	Set if internal timeout occurs during a Program or Erase operation.
DB4	Erase Phase Indicator	Additional information to DB5 = 1 when the Erase operation fails. DB4 = 1 indicates the Erase Phase of the Erase operation failed. DB4 = 0 indicates the Program Phase of the Erase operation failed.
DB3	Sector Erase Timer	Set when the Sector Erase operation starts. Clear when the Erase Hold Timer is triggered and counting down.
DB2	Toggle Bit 2	Toggles on successive read cycles during the Erase operation. Outputs actual data bit when the Erase operation succeeds.
DB1	Write to Buffer Abort	Set when a Write Buffer operation has been aborted.
DB0	Not used	-

Table 5. Flash memory status bits

4 Example

4.1 Overview

The software example demonstrates these operations on the internal flash. The target device is an eCOG1X14Z5, which contains 512KB embedded flash. The software example defines flash sector 0 (SA0) as the code area. Flash sector 1 (SA1) is assigned as the application data area. Flash sector 2 (SA2), which has the same size as sector 1, is used as a temporary storage area when the content of SA1 is being erased and reprogrammed.

The purpose of the example is to modify the application data content in SA1. In order to modify the data in SA1, it is necessary first to erase this flash sector area and then to program it with the new contents. However, code cannot be executed from anywhere in internal flash while any part of it is being erased or programmed. One solution is to copy the *SECTOR ERASE*, *WRITE WORD*, and *WRITE BUFFER* routines to internal RAM, such that the flash erase and program operations can be executed from internal RAM while the internal flash is being modified. Note that the *READ* operation on the internal flash does not require code to be executed from RAM.

The following diagram shows the internal memory maps for code space and data space in the eCOG1X. Addresses in physical memory areas such as internal flash and internal RAM are physical addresses; addresses shown in code space and data space areas are logical addresses. All addresses are in hexadecimal.

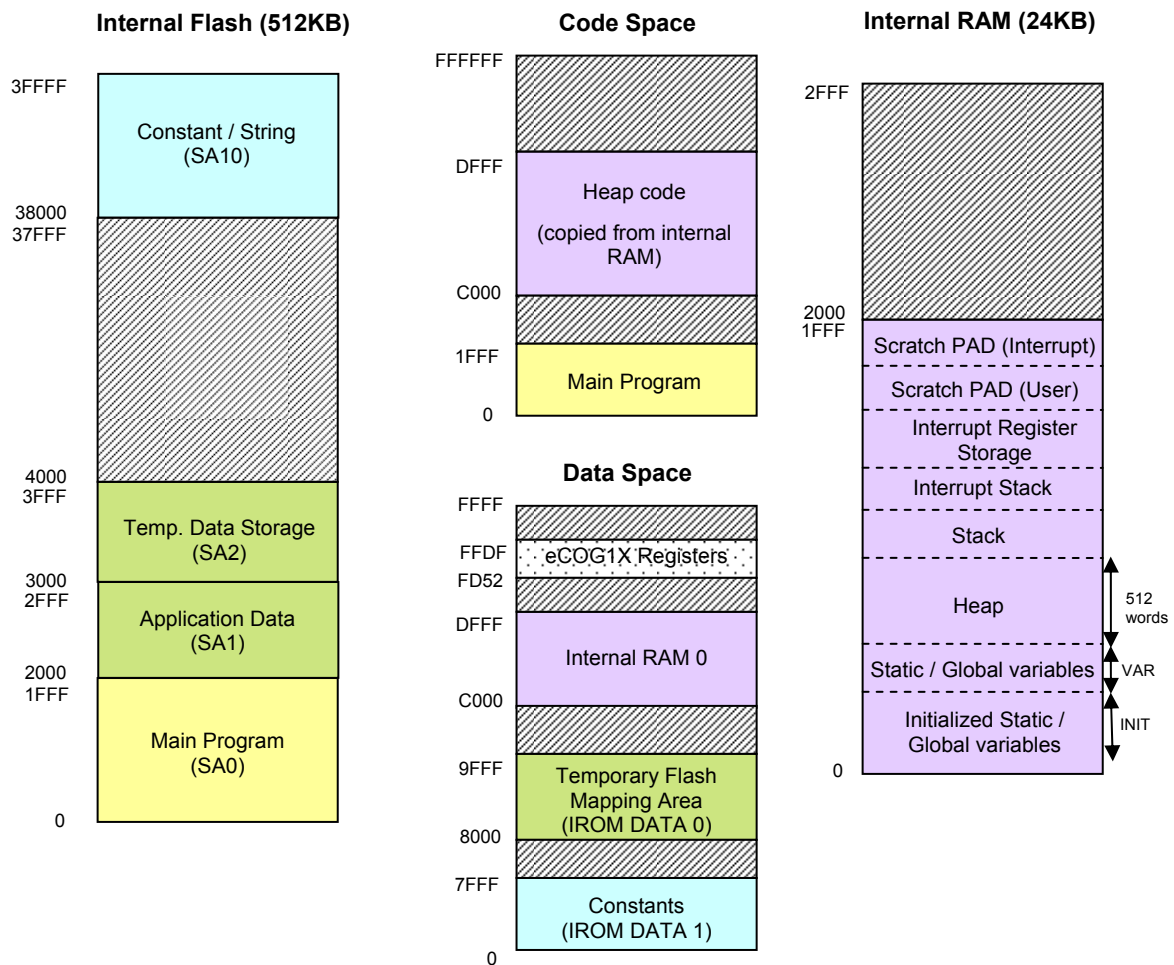


Figure 1. Memory map of the software example

4.2 Program Flow

The following steps describe the operation of the software example.

Erase flash sector 2

1. Allocate 256 words in heap area.
2. Map erase_sector() to flash data 0.
3. Copy erase_sector() from flash data 0 to IRAM in heap area.
4. Map IRAM (heap) to code space.
5. Map SA2 to flash data 0.
6. Execute erase_sector() from IRAM.
7. Free space in heap and restore flash data 0 translator configuration.

Read flash sector 1

8. Map SA1 to flash data 0.
9. Read 32 words from SA1 by flash_read_buffer() function.
10. Restore flash data 0 translator configuration.

Copy flash sector 1 to flash sector 2

11. Allocate 256 words in heap area.
12. Map flash_write_buffer() to flash data 0.
13. Copy flash_write_buffer() from flash data 0 to IRAM in heap area.
14. Map IRAM (heap) to code space.
15. Map SA2 to flash data 0.
16. Execute flash_write_buffer() from IRAM to program 32 words of data.
17. Free space in heap and restore flash data 0 translator configuration.
18. Repeat steps 8-17 until all data contents are copied from SA1 to SA2.

Erase flash sector 1

19. Allocate 256 words in heap area.
20. Map erase_sector() to flash data 0.
21. Copy erase_sector() from flash data 0 to IRAM in heap area.
22. Map IRAM (heap) to code space.
23. Map SA1 to flash data 0.
24. Execute erase_sector() from IRAM.
25. Free space in heap and restore flash data 0 translator configuration.

Read flash sector 2

26. Map SA2 to flash data 0.
27. Read 32 words from SA2 by flash_read_buffer() function.
28. Restore flash data 0 translator configuration.

Modify and restore data content to flash sector 1

29. Allocate 256 words in heap area.
30. Map flash_write_buffer() to flash data 0.
31. Copy flash_write_buffer() from flash data 0 to IRAM in heap area.
32. Map IRAM (heap) to code space.
33. Map SA1 to flash data 0.
34. Execute flash_write_buffer() from IRAM to program 32 words of modified data to SA1.
35. Free space in heap and restore flash data 0 translator configuration.
36. Repeat steps 26-35 until all data contents are restored to SA1.

Since the code area for internal RAM is changed at run-time, with the same logical addresses mapped to different sections of code, it is essential to turn off the code cache while executing these routines so that the executed code is always loaded from the internal RAM, not the cache.

A function pointer is declared to point to a block of memory reserved from the heap, which has been mapped to data space. The address of the function pointer must be in the logical address range 0x0 to 0xFFFF, because the eCOG1X supports only a 16-bit address range in data space. To execute code in this heap area, it is necessary also to map the heap to code space; it is simplest to use the same logical address in code space and in data space, copying the logical mapping address from `mmu.ram0_data_log` to `mmu.ram_code_log`. This allows the code to be executed from the same function pointer address with no recalculation of the function pointer offset. For the same reason, the logical mapping address for the code in RAM should be in the range 0x0 to 0xFFFF.

4.3 Installing the Application Software

Unpack the application software to the CyanIDE projects directory, usually `<My Documents\CyanIDE Projects>`. Launch the application project by double-clicking the `InternalFlashProg.cyp` file in the `AN064SW` subdirectory.

Select **Build** from the **Build** menu. CyanIDE compiles and builds the application project, and generates a `<project>.rom` output file. Select **Download** from the **Debug** menu. The Output window shows the progress as the software is downloaded to the eCOG1X and programmed into flash memory.

If the project fails to run, check the following:

Heap size

The heap area is reserved in `cstartup.asm`. In this application software, it is necessary to reserve at least 512 words to store and run code in the heap memory area.

```

$??ISTACK_SIZE   =      H'0040
$??STACK_SIZE   =      H'0100
$??HEAP_SIZE     =      H'0200      ; Reserved 512 words - to execute flash API in Heap

```

Instruction cache

The instruction cache of the eCOG1X captures instruction fetches from both internal flash and internal RAM to achieve faster performance and lower power consumption. Since the code stored in the internal flash does not change during execution, the instruction cache does allow faster instruction execution. However, the code in the internal RAM may change during execution. This may cause a problem when code in the internal RAM is captured in the instruction cache. When the processor discovers the next code address is already stored in the instruction cache, it executes the respective instruction from cache instead of running (possibly changed) code from internal RAM. As a result, the processor may execute incorrect or stale code as the code being stored in the instruction cache is not updated when the RAM is modified.

It is essential to turn off the instruction cache in this example software. The instruction cache is turned off as follows:

1. Right-click the eCOG1X chip in `ecog1x.cfg` and select **eCOG1X Properties**.
2. Choose the **Cache** tab, and set the **Main Cache Operation** to **Disabled** from the drop-down list box.

Appendix A Software Functions

A.1 flashAPIs.c

flash_modify_enable()

NAME: Internal flash configuration function

SYNOPSIS: void flash_modify_enable(void)

DESCRIPTION: Set internal flash configuration parameters to allow Erase/Write operations.

RETURN: None

erase_sector()

NAME: Erase sector function

SYNOPSIS: unsigned int erase_sector(unsigned long address, unsigned int offset,
unsigned int *data)

PARAMETERS: address - 24-bit flash physical address of the target sector
offset – Not used
data[] – Not used

DESCRIPTION: Erase the target internal flash sector.

RETURN: FALSE - if erase operation failed
Others - program success

flash_write()

NAME: Write flash function

SYNOPSIS: unsigned int flash_write(unsigned long address, unsigned int offset,
unsigned int *data)

PARAMETERS: address - 24-bit flash physical address of the target sector
flash_offset – indicates the target program position
flash_data[0] - 16-bit data word to be programmed
flash_data[1-31] – Not used

DESCRIPTION: Program a data word to internal flash.

RETURN: FALSE – if write operation failed
Others - program success

flash_write_buffer()

NAME: Write flash buffer function

SYNOPSIS: unsigned int flash_buffer_write(unsigned long address, unsigned int offset, unsigned int *data)

PARAMETERS: address - 24-bit flash physical address of the target sector
offset – indicates the target program position
data[0-31] – An array of 16-bit data words to be programmed

DESCRIPTION: Write 32 data words to internal flash

RETURN: FALSE - if write operation failed
Others - program success

flash_read()

NAME: Read flash function

SYNOPSIS: unsigned int flash_read(unsigned long address, unsigned int offset, unsigned int data)

PARAMETERS: address - 24-bit flash physical address of the target sector
offset – indicates the target read position
data – temporary storage for the 16-bit read data

DESCRIPTION: Read one data word from internal flash.

RETURN: TRUE

flash_read_buffer()

NAME: Read flash buffer function

SYNOPSIS: unsigned int flash_read_buffer(unsigned long address, unsigned int offset, unsigned int *data)

PARAMETERS: address - 24-bit flash physical address of the target sector
flash_offset – indicates the target read position
flash_data[0-31] – temporary storage for the 32-words of read data.

DESCRIPTION: Read 32 data words from internal flash

RETURN: TRUE

Program_Heap()

NAME: Program heap function

SYNOPSIS: void Program_Heap(void)

DESCRIPTION: Copy the target function to internal RAM heap area, and map the internal ram to code space for execution while the internal flash is being erased or programmed.

RETURN: None

Free_Heap()

NAME: Free heap function

SYNOPSIS: void Free_Heap(void)

DESCRIPTION: Free heap space and restore MMU translator parameters.

RETURN: None