FreeEMS Memory Management Scheme

Version : 0.2
Date : 27/11/08

Purpose/Preamble

This document exists to summarise and explain the memory available on the XDP512 processor and how it is being used in the FreeEMS application firmware.

This document is believed to be true and correct at the time of writing. Obviously the firmware is under continuous active development and there will be changes to the way things are done as time passes. The code is the best reference, however I will endeavour to keep this up to date.

Overall Memory Layout

The XDP512 has a 16 bit data bus and main addressing scheme. This limits addressable space to a total of 64k (65536 Bytes).

The 64k is made up of the following parts :

• 2k of control registers	from 0x0000 to 0x07FF with length 0x0800
• 2k of addressable EEPROM	from 0x0800 to 0x0FFF with length 0x0800
• 12k of addressable RAM	from 0x1000 to 0x3FFF with length 0x3000
• 48k of addressable Flash	from 0x4000 to 0xFFFF with length 0xC000

The registers are permanently exposed and linearly addressed and therefore accessed directly at any time. The EEPROM, RAM and Flash, however, all use a paging scheme to expose a larger space than would otherwise be possible.

The total amount of each accessible using paging is as follows :

- 4k of EEPROM
- 32k of RAM
- 512k of Flash

EEPROM Memory Layout

EEPROM is not used in FreeEMS at this time, however the 2k is made up of the following parts :

- 1k of EEPROM page window from 0x0C00 to 0x0FFF with length 0x0400
- 1k of direct addressable EEPROM from 0x0800 to 0x0BFF with length 0x0400

The following are valid EPAGE values : 0xFF, 0xFE, 0xFD, 0xFC

The fixed 1K page from 0x0C00 to 0x0FFF is the page exposed by setting EPAGE to 0xFF. Doing that is pointless however as you then have two copies of the same EEPROM exposed at the same time.

The reset value of EPAGE is 0xFE and ensures that there is a linear EEPROM space available between addresses 0x0800 and 0x0FFF out of reset.

Therefore the useful set of EPAGE values is : 0xFE, 0xFD, 0xFC

RAM Memory Layout

This section details how the available RAM is laid out and used in FreeEMS.

The 12k of direct addressable RAM consists of the following parts :

• 4k of RAM page window	from 0x1000 to 0x1FFF of length 0x1000
• 8k of direct addressable RAM	from 0x2000 to 0x3FFF of length 0x2000

The following are valid RPAGE values : 0xFF, 0xFE, 0xFD, 0xFC, 0xFB, 0xFA, 0xF9, 0xF8

The fixed 4K page from 0x2000–0x2FFF is the page exposed by setting RPAGE to 0xFE. The fixed 4K page from 0x3000–0x3FFF is the page exposed by setting RPAGE to 0xFF. As with the EEPROM, doing that is pointless because you are just exposing the region twice at the same time.

The reset value of RPAGE is 0xFD and ensures that there is a linear RAM space available between addresses 0x1000 and 0x3FFF out of reset.

Therefore the useful set of RPAGE values is : 0xFD, 0xFC, 0xFB, 0xFA, 0xF9, 0xF8

In FreeEMS we use the 6 available pages as two triplets for tune switching (like table switching, but almost everything is switched). At any given time the firmware is using 3 pages that contain one copy of the following :

- 3 VE Tables
- 1 Lambda Target Table
- 2 Ignition Timing Tables
- 2 Injection Timing Tables
- 4 blocks of tunable items including all small tables etc.

Each of these 12 items is 1k in size totalling 12k which doubled (for tune switching) is 24k. These are available in 3 sets of four blocks at a time. VE and Lambda are together, Timing tables are together and the tunable items are together. The remaining 8k is directly addressable linear space.

The 8k of linear space is used as follows :

- 2k buffer for reception of communications data
- 2k buffer for transmission of communications data
- 4k of general purpose RAM

The 4k of general purpose RAM is used in the traditional way with all of the global variables at the low end and the stack at the high end growing down toward the globals. Currently there are approximately 1k of global variables in that region.

Flash Memory Layout

This section details how the available Flash is laid out and used in FreeEMS.

The 48k of direct addressable Flash consists of the following parts :

- 16k of direct addressable Flash from 0x4000 to 0x7FFF of length 0x4000
- 16k of Flash page window from 0x8000 to 0xBFFF of length 0x4000
- 16k of direct addressable Flash from 0xC000 to 0xFFFF of length 0x4000

The following are valid PPAGE values : 0xFF, 0xFE, 0xFD, 0xFC, 0xFB, 0xFA, 0xF9, 0xF8, 0xF7, 0xF6, 0xF5, 0xF4, 0xF3, 0xF2, 0xF1, 0xF0, 0xEF, 0xEE, 0xED, 0xEC, 0xEB, 0xEA, 0xE9, 0xE8, 0xE7, 0xE6, 0xE5, 0xE4, 0xE3, 0xE2, 0xE1, 0xE0

The fixed 16K page from 0x4000–0x7FFF is the page exposed by setting PPAGE to 0xFD. The fixed 16K page from 0xC000–0xFFFF is the page exposed by setting PPAGE to 0xFF. As with the RAM and EEPROM, doing that is pointless because you are just exposing the region twice at the same time.

The reset value of PPAGE is 0xFE and ensures that there is a linear Flash space available between addresses 0x8000 and 0xBFFF out of reset.

Therefore the useful set of PPAGE values is : 0xFE, 0xFC, 0xFB, 0xFA, 0xF9, 0xF8, 0xF7, 0xF6, 0xF5, 0xF4, 0xF3, 0xF2, 0xF1, 0xF0, 0xEF, 0xEE, 0xED, 0xEC, 0xEB, 0xEA, 0xE9, 0xE8, 0xE7, 0xE6, 0xE5, 0xE4, 0xE3, 0xE2, 0xE1, 0xE0

The 512k of available flash space is made up of four 128k blocks each with eight 16k pages. Currently due to limitations of hcs12mem and/or the serial monitor we can only use up to eight 16k pages totalling 128k of Flash. Two of those pages are permanently exposed in linear space (0xFF/text and 0xFD/text1) leaving six pages of Flash for dynamic use in the short to medium term.

The two linear regions are called text and text1, the six Flash pages are swapped in and out automatically by the compiler whenever needed. In them we store the following :

- Commonly used items and the serial monitor
- All Interrupt Service Routines
- Lookups and functions operating on them
- Fuel tables and functions operating on them
- Timing tables and functions operating on them
- Tunable items and functions operating on them
- General function storage area one
- General function storage area two

Where Things Are

The current break down of usage within those pages is as follows :

Commonly used items and the serial monitor :

- All functions without a page specified are stored in text region by default
- The serial monitor takes up 2k of space at the high end of text

All Interrupt Service Routines :

- All placed in text1 region as a way of tracking flash usage

Lookups and functions :

- 2k CHT lookup table
- 2k IAT lookup table
- 2k MAF lookup table
- Core variable generation function

Fuel tables and functions :

- 6 1k VE tables
- 2 1k Lambda tables
- Copy flash to ram function

Timing tables and functions :

- 4 1k Ignition timing tables
- 4 1k Injection timing tables

Tunable items and functions :

- 8 1k blocks of tunable items

General paged function space one :

- Assorted functions and data used purely within them

General paged function space two :

- Assorted functions and data used purely within them

Note : The ISR code MUST be kept in linear space such that they can be called asynchronously at any time.

How They Get There

During compilation directives are used to locate various pieces of data and code into specific sections. These sections are defined in the memory.x file which lays out the sizes and offsets from the codes point of view. Regions labeled as certain sections are directed and redirected into specific sections based on the contents of the linker script and also regions.x for FreeEMS specific regions.

The linker as instructed by the linker script then moves and splits and merges these sections to form final sections to be included in the output.

Finally, the object copy program takes these blocks and further relocates them into a loadable image according to rules passed on the command line in the Makefile.

There are many sections that come from the compilation process, however we only use a subset of those.

The following Flash sections are present in the final loadable image :

- rodata – Constants as declared with the const keyword.
- data – Static initialisation to non zero values.
- text – Code that must be present in linear space to function correctly.
- text1 – ISR code that must be present in linear space to be called asynchronously.
- vectors – The jump table of void function addresses for all interrupt handlers.
- fixedconf1 – The first block of non live tunable configuration.
- fixedconf2 – The second block of non live tunable configuration.
- ppageF8 – General purpose flash page.
- ppageF9 – General purpose flash page.
- dpageFA – Non-volatile storage of the fuel tables.
- fpageFA – Function to load the fuel tables.
- dpageFB1 – dpageFB8 – Non-volatile storage of the tunable tables.
- fpageFB – Function to load the tunable tables.
- dpageFC – Non-volatile storage of the timing tables.
- fpageFC – Function to load the timing tables.

The following RAM sections are referred to by the final loadable image :

- bss – All global variables including those initialised by the data section
- rpage – The window region to align a struct inside.
- rxbuf – The comms receive buffer.
- txbuf – The comms transmit buffer.

Additionally, the code uses a standard stack at the high end of ram (0x4000) which grows down towards the bss global variable section. When execution begins the gcc start up code initialises the stack frame pointer to the location defined using the symbol _stack. As each function runs it builds from this location downwards allocating the space it requires for local variables and temporary variables etc. When a function ends the stack frame pointer is restored to where it was before the call occurred. Because of this and the lack of a heap to manually dynamicly allocate memory on memory leaks are not possible.

Programmatic Usage

There are a number of hand designed schemes in practice in the code base for handling the placement and movement of memory in the device during its normal operation. All of them aim for two things : Speed and Correctness.

The first set of memory operations that occur are to copy all of the live tunable data up to RAM from their locations in Flash. This is done by a series of functions that reside in the same paged flash space as the data to be copied. Placing these items close together ensures fast access and no linker warnings. In order to have the address dictionary compile cleanly a set of pointer variables are also initialised in these same functions. Before each set of four 1k blocks is copied up to RAM from its Flash page the RPAGE value is set to the appropriate value for the data being copied. Once these functions have all run a function is called that samples a specified pin to determine which set of tables to use. This function is checked periodically such that if the pin changes state (the switch is thrown) then so does the behaviour of the firmware. This should happen seamlessly as all operations will just start using the new data set.

Because only one page of paged data can be accessible at a time, we needed a scheme for RAM tuning data access that would enable us to see all 24k of paged data when we needed it and yet still be simple, maintainable and largely foolproof. This was found by forcing the caller to specify the ram page of the data to be accessed in the table lookup and interpolation function call. The table lookup and interpolation functions take a value for RPAGE, store the current value, set RPAGE to the new value, read the data out and finally restore the previous RPAGE before returning. This scheme allows us to leave the finer grained data permanently exposed in the page window for simple and easy access to it it with the only restriction being no access from the ISR code without explicitly swapping pages in and out as the main table lookup and interpolation functions do.

Functions are placed into flash pages in related groups such that minimal page swapping needs to occur for the most frequent calls to be made. ISR functions are all placed together in text1 such that it is easy to keep track of their total size and because they must be located in linear space. The code that burns to and reads from flash is located in text such that it can freely swap pages without removing itself from visibility. Very little else needs to be present in the linear space and as such, most other things are placed in paged space so that we always know how much linear space we have left for various things that must be stored there.

When a serial packet requesting or supplying a data block arrives we must find out where to get it from or put it. This is accomplished with a dictionary approach whereby a single ID number is passed in with a pointer to a struct that describes a memory blocks possible parameters. The function looks up the ID using a large case statement and then populates the struct with the corresponding data. Finally the function returns and the caller can go ahead and use the parameters that have been retrieved. If the request is for a block of RAM the RPAGE value is saved and then set to the value retrieved from the dictionary before reading the RAM data into the transmit buffer. Likewise, when the request is for a block of flash a similar sequence of save, set, read, restore is performed. Again, when data is supplied to be written to RAM, the same thing happens. For flash it is slightly different as the flash function takes care of the swapping around and is passed the page values, RPAGE as a read from RAM value and PPAGE as a write to flash value. The same goes for just writing to flash, however this is followed up by copying the fresh flash data up to ram to ensure syncronisation.