SGS-THOMSON MICROELECTRONICS

# AN931 APPLICATION NOTE

# ON-CHIP HARDWARE EEPROM EMULATION versus FLASH MEMORY SOFTWARE SOLUTIONS

**by Y. BAHOUT**

## INTRODUCTION

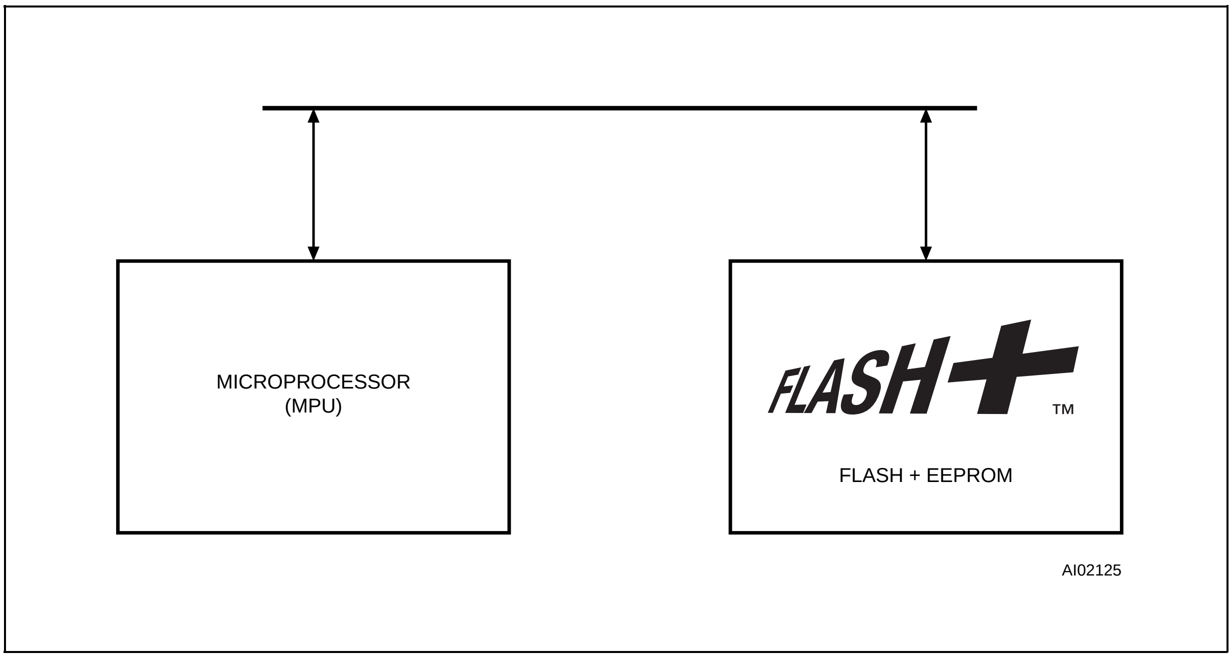
The new FLASH+™ products from SGS-THOMSON offer single chip solutions which combine Flash and EEPROM memory functionality. Based on Flash memory technology, the devices integrate all the control logic to provide a large block of Flash memory and a smaller block which is able to emulate in hardware a full flexibility EEPROM. The first product in this FLASH+ family is the M39432 which features a single supply 4 Mb Flash memory combined with a 256 Kb EEPROM in a TSOP40 package. The Flash and EEPROM blocks can be addressed in a concurrent mode, so that the EEPROM block can be written using an internal algorithm, while the Flash block is being read. This enables parameter data to be updated in the EEPROM concurrently with program code execution in the Flash memory, thus offering the same functionality as separate Flash and EEPROM devices.

The FLASH+ family, of which the M39432 is the first member, is modular and will offer Flash blocks from 1 Mb to 8 Mb and EEPROM blocks from 64 Kb to 256 Kb. The M39432 is the first family member to be available and offers two independant blocks of 4 Mb Flash (512K x 8) and 256 Kb EEPROM (32K x 8).

Using software algorithms and additional external SRAM memory, it is possible to emulate an EEPROM in Flash memory, but this solution has considerable disadvantages compared to a full hardware emulation such as the FLASH+ devices. FLASH+ EEPROM block offers the cell density of Flash memory with the functionality of EEPROM, that is the ability to re-write the memory at the byte level, a write (Erase + Write) operation taking under 10ms.

Moreover the FLASH+ EEPROM write operation can procede concurrently with reading of the Flash block.

**Figure 1. The FLASH+ Solution**



## FLASH MEMORY WITH EXTERNAL SOFTWARE EMULATION OF EEPROM

This solution uses one large Flash memory, split logically into two areas: the executable code is stored in one area - typically all the memory space except two sectors - and the remaining memory is dedicated to EEPROM emulation.

Flash memory may be written byte-by-byte, but cannot be erased, and so cannot be re-written, at the byte level. Flash memory can only be erased by sector, sector sizes in general ranging from 8K byte up to the whole chip size. The inflexibility of the sector erase limitation can be overcome by writing data sequentially in one sector and, when this is filled, copying the latest data written into a second sector. This way of providing EEPROM emulation is based on two Flash memory sectors, the first being the one in use and the second a spare one ready to receive a copy of the latest data and take over the emulation while the first sector is erased. The byte program and sector erase operation in a Flash memory cannot run concurrently with other operations and so the Flash memory code blocks are not available to be read while these operations are proceding.

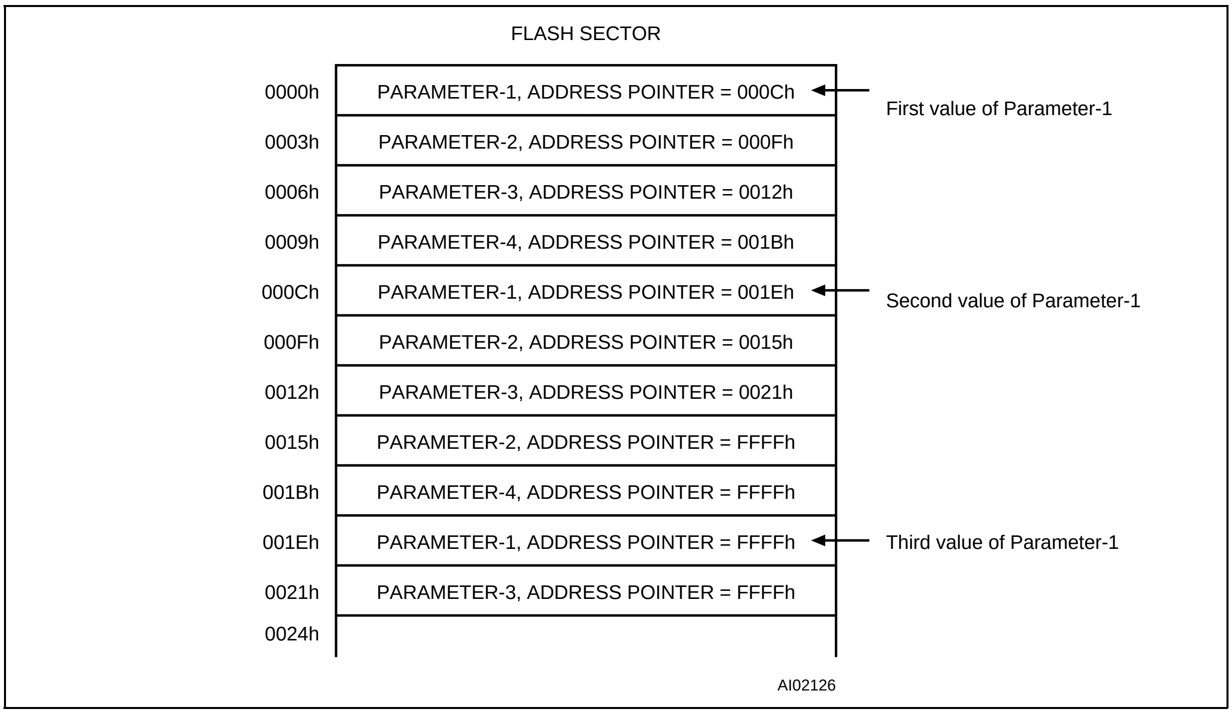
In more detail, at first power-up the two Flash memory sectors are erased (to FFh for a Flash memory). The first write to the emulated EEPROM programs the Parameter-1 at address 0000h of Sector-1. When the application needs to update this parameter with a new value, the emulation software will program three bytes: a 16 bit pointer to the new value and the new Parameter-1 value itself. This is shown graphically in Figure 2.

Parameter-1 was first programmed at location 0000h, it was then modified and re-programmed at location 000Ch and finally its third value was programmed at 001Eh. The value at location 001Eh is the last update which is indicated by the fact that the following 16 bit address pointer value is still blank (FFFFh at locations 001Fh & 0020h). Parameter-2 was programmed at 0003h, then 000Fh and 0015h; Parameter-3 was programmed at 0006h, 0012h and 0021h; Paramter-4 was programmed at 0009h and 001Bh.

The content of the Flash memory sector is a stack of three byte values or parameter/address pointer pairs, forming a linked list whose last value is the latest update of the parameter.

During the application's life time, the parameters are updated many times and the Flash memory sector-1 will be filled. When it is full the application software must copy the latest parameter values to Sector-2 and erase Sector-1 in order to be ready for the reverse order next copy.

**Figure 2. Example of a Flash Memory Sector Emulating EEPROM**


FLASH SECTOR
0000h
PARAMETER-1, ADDRESS POINTER = 000Ch
First value of Parameter-1
0003h
PARAMETER-2, ADDRESS POINTER = 000Fh
0006h
PARAMETER-3, ADDRESS POINTER = 0012h
0009h
PARAMETER-4, ADDRESS POINTER = 001Bh
000Ch
PARAMETER-1, ADDRESS POINTER = 001Eh
Second value of Parameter-1
000Fh
PARAMETER-2, ADDRESS POINTER = 0015h
0012h
PARAMETER-3, ADDRESS POINTER = 0021h
0015h
PARAMETER-2, ADDRESS POINTER = FFFFh
001Bh
PARAMETER-4, ADDRESS POINTER = FFFFh
001Eh
PARAMETER-1, ADDRESS POINTER = FFFFh
Third value of Parameter-1
0021h
PARAMETER-3, ADDRESS POINTER = FFFFh
0024h
AI02126

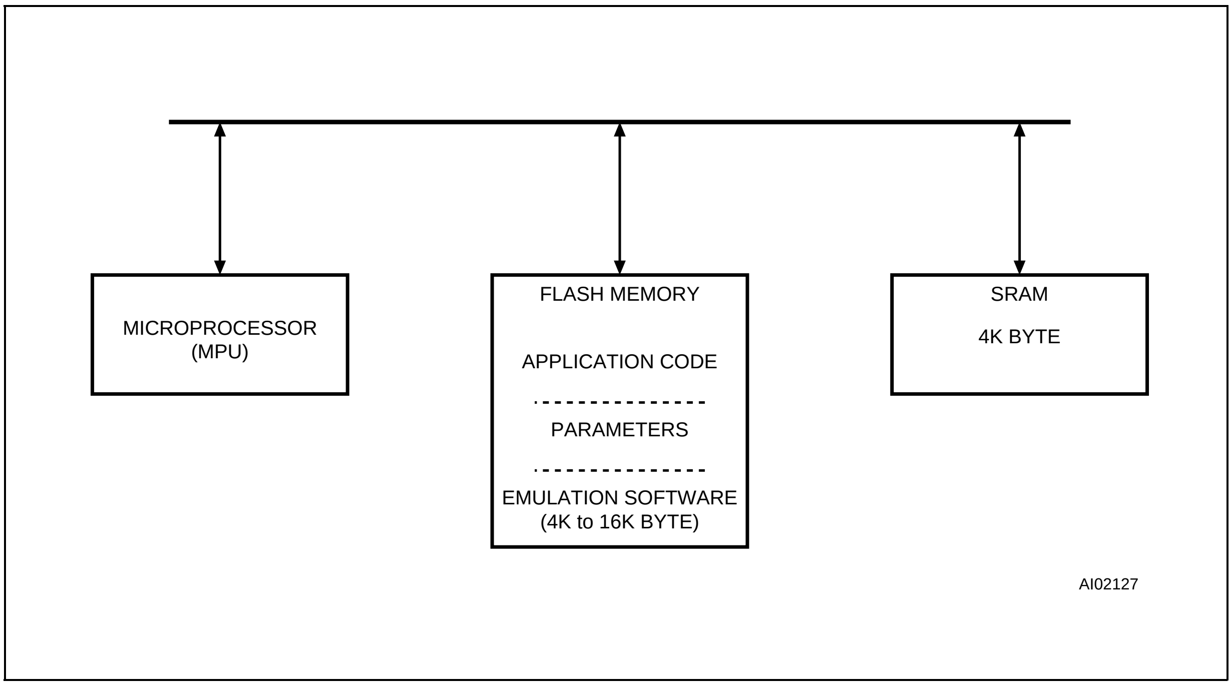
**System Requirements.** Flash memory cannot be read while programming another location or while being erased, even if the program/erase is in different sectors. This is because the internal algorithms of the Flash memory are busy controlling the execution of the program or erase. This means that the software code controlling the programming of bytes in one of the sectors emulating the EEPROM has to be stored OUTSIDE the Flash memory itself.

This means the application hardware design must include an additional memory, often a small SRAM, to store the device drivers for this operation.

The executable code for the EEPROM software emulation is stored in the Flash memory. This code is the software driver for sequencing all the tasks of emulation including the specific "program a byte in Flash memory" driver which has to be copied to the SRAM before execution.

Typical the EEPROM emulation software code is 16K bytes and the part needed to be copied to the SRAM is 4K bytes.

**Figure 3. Memories Required for the External Software Emulation of EEPROM in Flash Memory**



## HARDWARE VERSUS SOFTWARE EMULATION

The emulation of EEPROM can be done in two ways:

– Using FLASH+ devices which provide hardware EEPROM emulation, based on Flash memory technology
– Using standard Flash memory with a small external SRAM, with software emulation of the EEPROM in two Flash memory sectors.

In the comparison which follows hardware emulation will be referred to as OHE (On-chip Hardware Emulation) and the software emulation as ESE (External Software Emulation).

### Access Time

The access times for the two different emulations are not the same, and that for ESE varies with the number of times a parameter is updated.

**OHE Access Times.** Read and write times for On-chip Hardware Emulation are identical to the access times for a standard parallel access EEPROM, that is $t_{READ}$ is one MPU cycle and $t_{WRITE}$ is is one MPU cycle plus a write latency of 10ms during which the EEPROM performs its internal write cycle (and during which the Flash memory may continue to be read).

So the OHE access time is constant and is not weighted by the number of updates performed for each parameter. The 10ms latency between byte writes can be significantly reduced using the page mode write which allows up to 64 bytes to be written during a single 10ms write cycle.

**ESE Access Times.** For ESE, as shown in Figure 2, each parameter byte is associated with a two byte address pointer in a linked list. Reading or writing a new value implies a sequence of MPU read cycles to find the end of the relevant parameter list. Thus, for example, the time to read the last value of the Parameter-1 means performing the following operations:

```
– read the first address pointer at 0001h, 0002h
– if this pointer is not FFFFh, then
      – read the 2nd pointer at 000Ch, 000Eh
      – if this pointer is not FFFFh, then
            – read the 3rd pointer at 001Fh, 0020h
            – if this 3rd pointer is not FFFFh, then
            – ...and so on...
– else read the latest parameter-1 value
```

From this example the general rule for the access time can be derived:

*Time to read last value in list = (N + 1) * ( 2 MPU read cycles + MPU conditional test)*

where N is the number of parameter updates. Table 1 summarises the access times for a few extreme cases of sector configurations.

**Table 1. ESE Access Times**

| | **1 EEPROM byte** | **100 EEPROM bytes** | **2730 EEPROM bytes** (1) |
|---|---|---|---|
| Maximum number of updates possible, N (2) | 2730 | 27 | 1 |
| $t_{READ}$ (N): read time for one byte | (N+1) * (2 read cycles + conditional test cycle) | (N+1) * (2 read cycles + conditional test cycle) | (N+1) * (2 read cycles + conditional test cycle) |
| $t_{WRITE}$ (N): write time for one byte | $t_{READ}$(N) + 3 write cycles | $t_{READ}$(N) + 3 write cycles | $t_{READ}$(N) + 3 write cycles |
| Time to swap last update to new Flash memory sector (3) | $t_{READ}$(2730) + 3 write cycles | $t_{READ}$(27) + 3 write cycles | $t_{READ}$(1) + 3 write cycles |

**Notes:** 1. 2730 is the maximum number of bytes that can be emulated with a Flash memory sector of 8K byes.
2. Assuming each byte is updated the same number of times.
3. When swapping data additional time is required to download the executable drivers to the SRAM and to do the sector erase.

Table 1 shows the cases for an application with only one parameter, for 100 parameters and for the maximum that can be stored in a Flash memory sector of 8K bytes (8K bytes/3 bytes per parameter = 2730 parameters without copying to the alternate sector). If the application MPU can perform a read/write and conditional test in 200ns then we get the following limits for the access times:

For one parameter:

First parameter update takes

– $t_{READ}$ = (N+1) * (2 * read + conditional test) = 1.2µs

– $t_{WRITE}$ = (N+1) * (2 * read + conditional test) + 3 * write = 31.8µs

(assuming the Flash memory byte write time is 10µs).

The 2730th parameter update takes

– $t_{READ}$ = 1.638ms

– $t_{WRITE}$ = 1.669ms

Each 2730 updates the application software has also to copy the last updated values to the alternate Flash memory sector, the Flash memory sector erase takes about 1 second at the beginnig of life but becomes longer with more erase/write cycles, and so the application will freeze for some seconds every 2730 updates.

For 100 paramters:

First parameter update takes

– $t_{READ}$ = 1.2µs

– $t_{WRITE}$ = 31.8µs

The 27th parameter update takes

– $t_{READ}$ = 16.8µs

– $t_{WRITE}$ = 47.4µs

And in this case the application software has to copy the last updates to the alternate Flash memory sector after only 27 updates and the application will freeze for some seconds to erase the Flash memory sector every 27 updates.

For 2730 parameters the read and write times are absurdly high as each update means using a new Flash memory sector and freezing the application for some seconds to erase the alternate.

**Table 2. Read and Write Times after Several Updates**

| Number of Updates | 1 | 27 | 2730 |
|---|---|---|---|
| $t_{READ}$ | 1.2µs | 16.8µs | 1.638ms |
| $t_{WRITE}$ | 31.8µs | 47.4µs | 1.669ms |

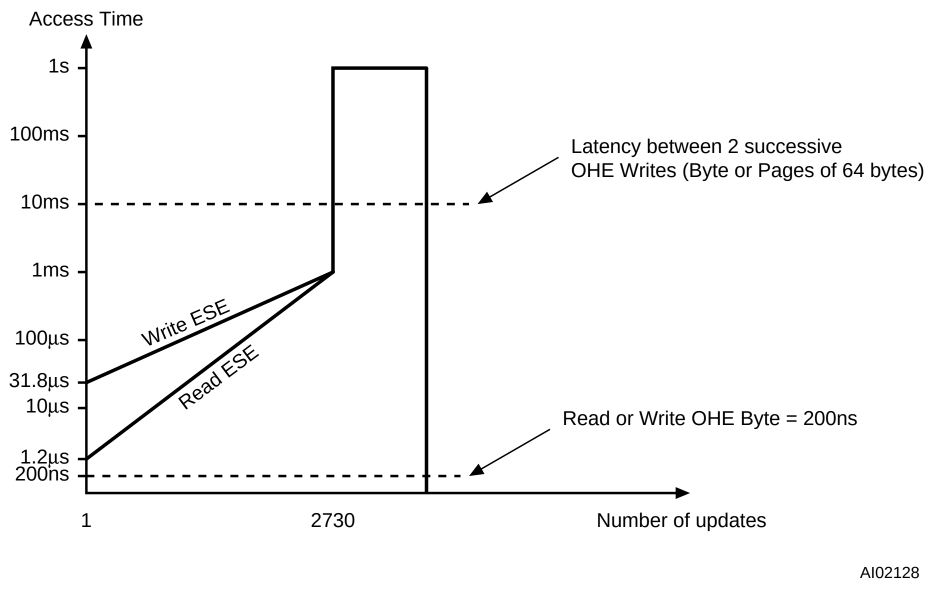
The read and write times for ESE are always larger than for OHE, the worst case being for the last updates within the same Flash memory sector. The long read access time is a weakness of ESE when compared to the 200ns read access time for a parallel access EEPROM offered by OHE.

In addition, the seconds delay for ESE Flash memory sector erase freezes the entire application. The sector erase can be suspended and resumed, but this requires the external drivers in the additional SRAM and prolongs the operation.

This is particularly inappropriate in an application where interrupt requests have to be serviced in a minimum time. If the interrupt request rate is high and parameter refresh is also high, there is a possibility that a catastrophic scenario could develop where the first EEPROM emulation sector is not yet erased before the second sector is full and needs to be erased also.

Figure 4 summarises the access times and the update number.

**Figure 4. Access Times and the Update Number**



## Power Consumption

The power consumption of any single power supply memory is dependent on three parameters: the $I_{CC}$ operating current, the $I_{CC}$ standby current and the time ratio between operating and standby conditions.

A device is active ($I_{CC}$ = operating current) when accessed. The access time of the ESE case can be several times longer than for OHE, and so the active time for ESE is much larger than for OHE. As the standby current value is much less than the operation value, the ratio of power consumption between OHE and ESE concepts depends mostly on the operating current values and active times.

Table 2 shows that the ESE access time is 6 to more than 8000 times longer than the OHE access time. Thus the ESE concept requires considerably more energy than the OHE one.

## Memory Size for Emulation

Two factors are important when comparing the External Software Emulation and the On-chip Hardware Emulation solutions: the number of parameters to be stored and the number of updates.

For OHE the memory size chosen is simply determined by the number of parameters to be stored. The number of updates that can be made is determined by the technology which is able to sustain over 100,000 re-write cycles for the emulated EEPROM.

For ESE, the evalutation is not a simple matter. First is the case where the data is updated a few times, say 1 to 10 times. Most applications that modify data only a few times can accept that they will be frozen during for a few seconds needed for Flash memory sector erase.

During the freeze they will offer limited functionality with most interrupts disabled and no real time operations possible. Table 3 shows the memory size requirements for this case.

**Table 3. Memory Size Required for 1 to 10 Updates**

| Emulated EEPROM Size | | 16Kb | 64Kb | 256Kb |
|---|---|---|---|---|
| ESE solution | ESE Size (1) | 3*2*16Kb = 96Kb | 3*2*64Kb = 384Kb | 3*2*256Kb = 1.536Mb |
| | SRAM Size | 32Kb | 32Kb | 32Kb |
| OHE solution | OHE Size | 16Kb | 64Kb | 256Kb |
| | SRAM Size | none | none | none |

**Note:** 1. One EEPROM byte = 3 ESE bytes, ESE solution requires 2 Flash memory sectors.

If the number of updates is higher, in the range 100 to 1000 times, this is typical of applications that require a more flexible access to the EEPROM. The main concern is to spread the long Flash memory sector erase time into many small Erase Suspend/Resume slots in such a way as this operation does not impact on the applications real time tasks. If we consider an application that cannot afford to be slowed down by the Erase Suspend/Resume except after each 10th parameter update, then this will have to use a Flash memory sector size to emulate the EEPROM which is ten times larger than the data size in order to offer fast access time for the first 10 updates. The values are shown in Table 4.

**Table 4. Memory Size Required for 100 to 1000 Updates, Delay at each 10th Update**

| Emulated EEPROM Size | | 16Kb | 64Kb | 256Kb |
|---|---|---|---|---|
| ESE solution | ESE Size (1) | 3*2*10*16Kb = 960Kb | 3*2*10*64Kb = 3.84Mb | 3*2*10*256Kb = 15.36Mb |
| | SRAM Size | 32Kb | 32Kb | 32Kb |
| OHE solution | OHE Size | 16Kb | 64Kb | 256Kb |
| | SRAM Size | none | none | none |

**Note:** 1. One EEPROM byte = 3 ESE bytes, ESE solution requires 2 Flash memory sectors, each sector is 10 times the EEPROM size.

**Memory Size for Emulation** (cont'd)

In the last case where the parameters are updated thousands of times, say 1000 to 100,000, the application is typically one that requires fast data aquisition of at least 100 parameter updates before it can be slowed to erase the Flash memory sector. In this case to Flash memory sector emulating the EEPROM has to be 100 times larger than the EEPROM memory. This results in huge and impractical Flash memory size requirements (of up to 153.6Mb to emulate an EEPROM of just 256Kb).

In all cases the OHE solution requires just the EEPROM size needed by the application.

## Emulation Software Drivers

No software drivers are needed for the OHE solution as the EEPROM part of the FLASH+ device is accessed directly in the memory address space for read and write. The ESE solution requires software drivers which must be stored in the Flash memory and downloaded to an SRAM of the order of 4K to 16K bytes.

## Hardware Environment and Concurrent Mode

The ESE solution cannot access both Flash memory and ESE blocks concurrently because a write to the ESE has to be controlled by software running outside the Flash memory in the additional SRAM.

The OHE solution however allows reading the Flash memory during the internal EEPROM write cycle. It does not require any addtional external memory.

## CONCLUSION

When developing new applications requiring parameter storage, two important issues have to be taken in to account: the access time and the number of parameter updates forecasted during the application's life.

When these two parameters are known, the designer can evaluate the two competing solutions: External Software Emulation of the EEPROM in Flash memory or the On-Chip Hardware Emulation. The ESE solution will be found to be suitable only for those applications where the the software development costs can be spread over large volume production, where the updates are small (< 100) and freezing of the application for one or more seconds is acceptable. The OHE solution has none of these restrictions and offers all types of application an ideal solution.

Table 5 summarises the two solutions

**Table 5. Summary of the ESE and OHE Solutions**

| | ESE solution | OHE solution |
|---|---|---|
| Read access time | Increases in proportion to the number of updates | Fixed, 200ns typical or one MPU read cycle |
| Write time | 30µs in addition to the read access time | 10ms for 1 up to 64 Bytes |
| Power consumption | $6*I_{OHE} < I_{ESE} < 8000*I_{OHE}$ | $I_{OHE}$ |
| Memory size | Flash memory at least 6 times larger than EEPROM emulated | Equal to EEPROM size |
| Software drivers | Flash memory around 16K bytes + SRAM around 4K bytes | None |
| Hardware requiremetns | SRAM of around 4K bytes | None |