# AN1152
# APPLICATION NOTE

# OPTIMIZING THE USAGE OF THE ST92F120 EEPROM

**By Micro Division Applications**

## 1 PROGRAMMING EEPROM

This application note document describes the on-chip emulated EEPROM of the ST92F120 paying special attention to intelligent data management aspects, to optimize the cycling, in order to maximise the endurance.

### 1.1 INTRODUCTION

On this product, up to 1 Kbyte of on-chip EEPROM is available. It consists of a hardware emulation of EEPROM functionality implemented through a physical FLASH memory concept and circuitry.

One of the main differences from the user point of view between EEPROM and FLASH is the minimum erasable set of data. Typically, EEPROM allows the user to modify single byte locations: a data stored at a specific address can be updated individually. In contrast, a FLASH memory can be updated only by sectors (sector size is defined by architectural and technological constraints). A second aspect, which does not impact the user application, but is not less important, is the physical basic memory cell size: a FLASH basic cell is typically three/four times smaller than an EEPROM cell. This can have big impact on the product die size, with obvious effects on its price; consequently a bigger memory cut can be implemented using a FLASH concept in place of EEPROM without impacting die size.

A specific hardware solution has been implemented in order to create an EEPROM using a FLASH: a compromise between real EEPROM characteristics and FLASH specifics has been adopted. The EEPROM available on the ST92F120 can be updated only at page level (each page corresponds to 16 bytes); besides, since the updating of the FLASH content requires a sector erasing phase followed by a writing phase, a redundant physical memory space is used to implement a typical EEPROM data updating cycle (which usually requires a local and faster erasing phase on the single byte: this phase is hidden to the user).

Two FLASH memory sectors of 4 kbytes each are used to implement 1 kbytes of emulated EEPROM. The emulated EEPROM can be directly addressed by the user as a contiguous 1 kbyte area from 220000h to 2203FFh, without considering the internal mechanism which uses the two 4 kbyte FLASH sectors.

An important parameter to be aware of when using EEPROM emulation through a FLASH, is the endurance, defined as maximum number of updating cycles. A real EEPROM guarantees typically 100 kcycles for each byte; since the FLASH is erased and written by sectors, it is not possible to guarantee the number of cycle for each byte. A FLASH endurance parameter is also provided, but it refers to sector based updating.

A brief explanation of the emulation mechanism is given here in order to provide the user with some guidelines for efficient software application development. By smartly managing the storage addresses of variables in EEPROM according to their updating frequency, it is possible to obtain a higher number for cycling endurance at byte level, with the same number of erase cycles for the corresponding FLASH sectors.

### 1.2 HARDWARE EEPROM EMULATION

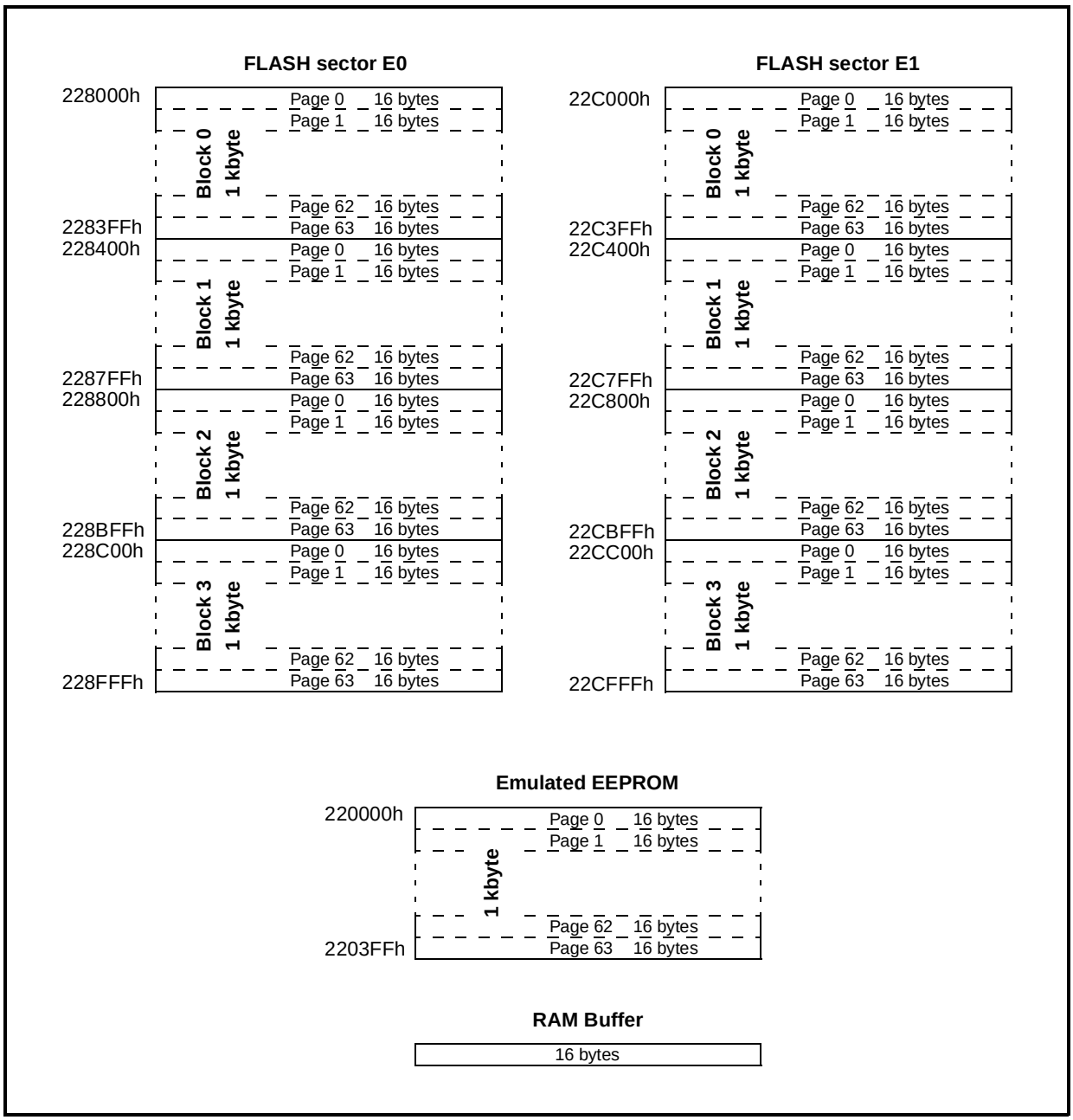
Both the emulated EEPROM (1 kbytes) and the two physical FLASH sectors E0 and E1 (4 kbytes each) are mapped on page 22h.

Each sector is divided in 4 blocks of 1 kbytes each (size of EEPROM to emulate). Again, each block is composed of 64 pages of 16 bytes each: the page is the minimum set of data that can be updated in the emulated EEPROM. Of course, the user can also update less than 16 bytes, but internally the algorithm always performs the full page update (copying the unchanged data).

Figure 1. shows a schematic representation of the physical FLASH organization for the EEPROM emulation sectors E0 and E1. A 16-byte RAM buffer (page size) is used for page programming operations.

The emulated 1 kbyte EEPROM is composed of 64 pages: Page 0 physically corresponds to one of the eight Page 0s inside the two sectors (one for each block); the same for Page 1, Page 2 and so on. The correspondence is defined through a set of block pointers (one for each 16-byte page); another dedicated pointer selects the active sector (one common pointer for all the pages). A simplified diagram showing an example of the user EEPROM space definition is given in Figure 2., at any time, the single page of user EEPROM is determined through the block selector (one independent block selector for each page, correspondent to a block pointer); besides, a multipolar switch is used to define the active sector (a common switch for all the pages, corresponding to the sector pointer). The pointer values are defined by the emulation mechanism described in the next paragraph.

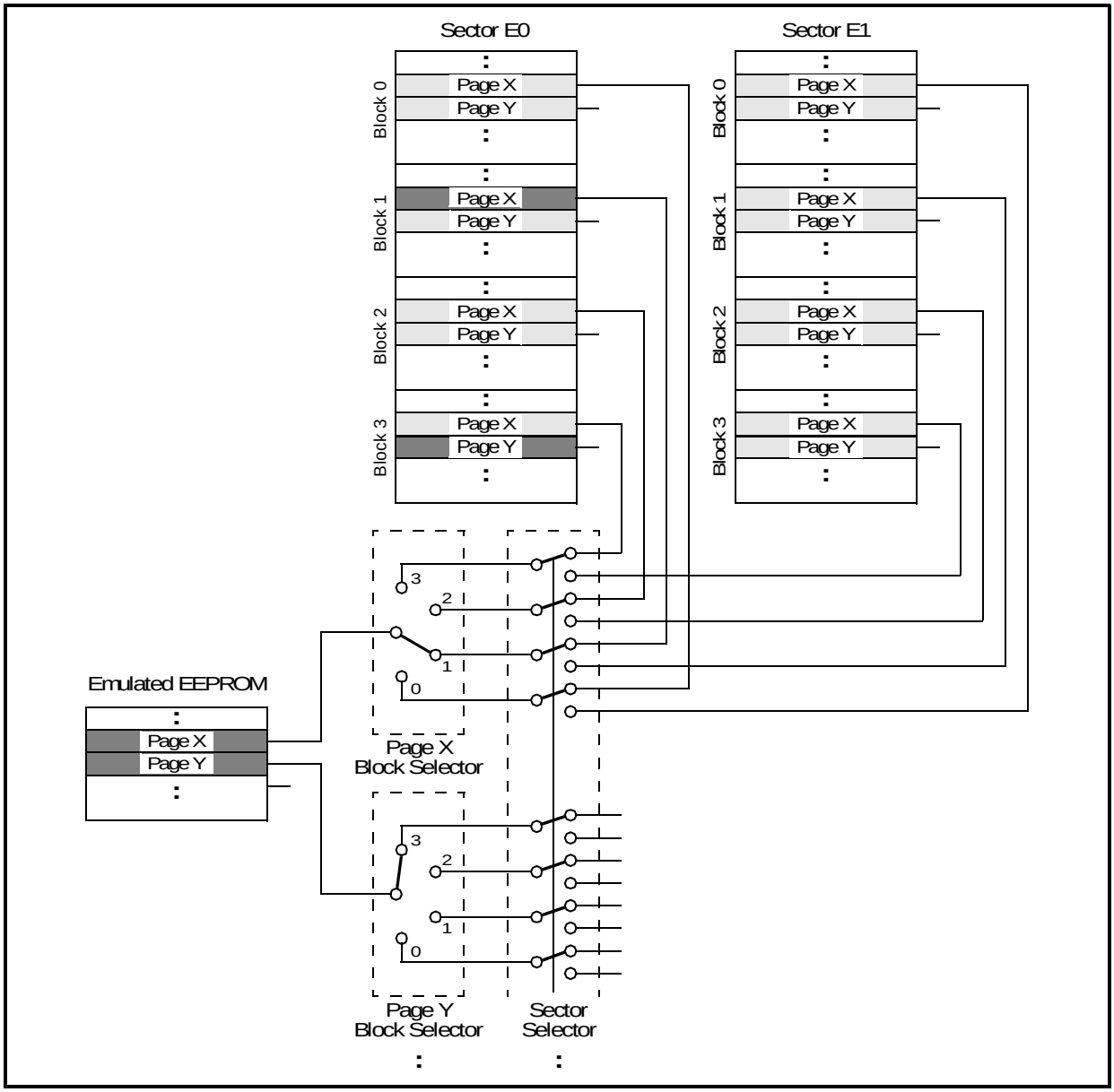
**Figure 1. Mapping of Emulated EEPROM and Physical Flash Sectors E0 and E1**



### 1.3 EMULATION ALGORITHM

When no data has been stored in user EEPROM yet, the pointers are set to address all the pages of the first block (Block 0) of Sector E0. In this way, the first data programming for each page is physically done on Block 0 of Sector E0. First data programming includes also initialization and test routines run before the product shipment: this means that when the final user starts working with the device, the pointer configuration can be different; anyway it does not have any impact on the software application in addressing the user EEPROM (it is a transparent mechanism), and it is assumed as hypothesis for the treatment.

**Figure 2. User EEPROM definition**



Each time the user updates the same page content, the Block Selector pointer is incremented (automatic internal operation); since the data are physically stored in a FLASH sector, when a page update occurs, it is not possible to erase the whole sector: so the updated page is written on the following block of the same sector (on the corresponding page). This operation is repeated each time the same page is updated (this is not visible to the user, who always addresses the same data location).

Assuming (for simplicity) a starting configuration with all pages physically mapped on Block 0 of Sector E0, each page can be updated up to four times without switching the sector (moving from Block 0 to Block 3).

When a new page update operation is launched (the fifth update of the same page), a sector switch also occurs: Sector E1 and Block 0 are selected, and the new data are stored in these locations.

Simultaneously, a Sector Erase operation is started on sector E0, and it is divided in four phases: during the next four page update instructions (whatever the involved pages are, either four different pages or four times the same one), Flash Sector E0 is automatically and completely erased, allowing a new sector switching from E1 to E0 again.

Figure 3., Figure 4., Figure 5. and Figure 6. show the evolution of the internal situation when the same page is continuously updated. From this sequence of drawings, it is clear that a sector erasing is carried out in a minimum of 4 page update operations on the user EEPROM: erasing of both the sectors requires 8 update instructions. It is a minimum since a cycle of 8 consecutive update operations of the same user page is considered. Normally, the user updates randomly the data in the pages, increasing the number of average page update operations for each sector erasing.

## 1.4 ENDURANCE COMPUTATION

One of the most important characteristics of a non-volatile memory is the number of times each byte can be updated without degrading data retention capability. As already explained, the emulated EEPROM implemented through a FLASH memory has some constraints which can have an impact on the maximum cycling. The following gives some examples on how to compute the maximum number of updates, referring to different data storage characteristics.

This takes into account the endurance specification guaranteed for the ST92F120 EEPROM: 100 kcycles/sector. This means that, if the data are properly managed, the byte endurance can be comparable with that of a standard EEPROM.

It must be clear anyway that 100 kcycles are guaranteed on a sector base (each sector can be erased up to 100000 times): this leads to having a greater number of cycles on each block (up to 8 times, since there are 8 blocks when both sectors are considered). This will be more clear during the examples presentation, where the 100000 erasing phases are considered for the sectors.

### 1.4.1 Regular updating of all the variables

This case represents the ideal way to manage the Emulated EEPROM to get the maximum number of cycles/byte. It is reported purely as an example, since it is really an untypical flow for EEPROM data management.

It consists of the following operative hypotheses:

- Each time that a byte variable is updated, all 16 variables stored in the same page are updated (it corresponds to updating 16 variables in one shot).
- A single page is not updated before all 64 pages are also updated; the order it is not important: it is mandatory that all 64 pages are updated before repeating the operation on the first updated page.
- In this way, a complete block switching is performed with a total of 1024 updated bytes:

  64 pages x 16 bytes/page = 1024 bytes

- Assuming that for one full erase of all blocks, 8 updating operations are required, and that an endurance of 100 kcycles/sector is guaranteed, then the maximum global cycling for each byte is:

  8 x 100 kcycles = 800 kcycles

### 1.4.2 Continuous updating of the same variable

In contrast to the previous one, this is an extreme case, which represents the worst way to manage the data in the EEPROM.

It consists of the following operative hypotheses:

- The same byte variable is continuously updated, repeating the operations the maximum number of allowed cycles.
- The maximum number of cycles would correspond to 800 kcycles, if only one variable were updated (see previous example): practically, all possible cycles are used for the same variable, but nothing else can be cycled.
- Assuming this operation is repeated for each byte and the number of variables is 1024 (1 kbyte of user EEPROM), the maximum number of cycles for each byte is:

800 kcycles: 1024 = 781 cycles

### 1.4.3 Updating a few variables

This is one of the most common situations in EEPROM data management. It is the case in which only a subset of bytes of the available EEPROM space is frequently updated. It is really important to provide the user with guidelines on how to distribute the variables inside the memory space in order to optimize the effective number of cycles for each of the frequently updated variables. In the following, three separate examples are reported in order to give an idea of how to proceed in defining application software constraints.

#### 1.4.3.1 Case A

The first specific case is represented by:

– 64 frequently updated byte variables

– 960 fixed byte variables (never updated)

The following operative hypotheses are considered:

- Each one of the 64 variables is stored in a different page (one per page over all 64 pages).
- Each of the 64 variables is updated with the same probability. Ideally (or statistically), after a variable update operation has been performed, the same variable is not modified again before all the other 63 are also updated.
- The updating of the 64 variables can occur in any order.
- In this way, each internal block is filled up before switching to the next; this means that a block switching takes 64 update cycles (1 for each variable).
- Since a full erase takes 8 block switching operations, this produces 8 updates/byte for each erasing cycle; guaranteeing 100 kcycles/sector, each of the 64 variables can reach:

8 x 100 kcycles = 800 kcycles

This example shows the best way to use memory space for storing variables. This is valid when the number of frequently-updated variables is from 1 to 64, with no change to the final maximum number of cycles for each variable.

In the next example the effect of less frequently updated variables is shown: again, a statistical computation of the impact on the maximum number of cycling for the most frequently updated variables is given.

#### 1.4.3.2 Case B

This second and more generic example is represented by:

– 64 frequently updated byte variables

– 960 rarely updated byte variables

The following operative hypotheses are considered:

- The 64 variables are stored one in each page.
- Each of the 64 frequently updated variables is changed with the same probability. Ideally (or statistically), after a variable updating operation has been performed, the same variable is not modified again before all the other 63 are also updated.
- The rarely updated variables are assumed to be changed with a rate equal to 1/960 of the rate of change of the frequently updated variables (value assumed for simplicity).
- One of the 960 rarely updated variables is changed each time all the 64 frequently updated variables are changed.
- Each of the 960 rarely updated variables is changed with the same probability. Ideally (or statistically), after a variable update operation has been performed, the same variable is not modified again before all the other 959 are updated also.
- To complete at least one change of all variables (rarely and frequently updated) it takes:

$$960 \times (64 + 1) = 62400 \text{ variable updates}$$

- To understand this refer to the flow below as an example, where variables *Fx(t)* and *Ry(t)* represent the “frequent” and “rare” variable update operations respectively (“*x*” varying from 1 to 64, “*y*” from 1 to 960, where “*t*” is the number of update occurrences for the considered variable).

| Update # | Updated variables |
|---|---|
| 1 | F1(1) |
| 2 | F2(1) |
| : | : |
| 64 | F64(1) |
| 65 | R1(1) |
| 66 | F1(2) |
| 67 | F2(2) |
| : | : |
| 129 | F64(2) |
| 130 | R2(1) |
| 131 | F1(3) |
| 132 | F2(3) |
| : | : |
| : | : |
| : | : |
| : | : |
| 62399 | F64(960) |
| 62400 | R960(1) |

- Taking into account now that a block switching occurs every 64 variable update operations (on average, considering this specific hypothesis of the described variable updating flow), and again that a full erase occurs each 8 block switching, the number of erase phases results:

(62400 : 64) : 8 = 122 erase phases

- The relationship with the frequently updated variables is simply obtained; during this number of erasing phases the 64 variables are updated 960 times each, so:

960 : 122 = 7.8 update/byte (for each erase)

- In conclusion, guaranteeing 100 kcycles for each sector, it is simple to obtain the maximum number of cycles for the frequently updated variables:

7.8 x 100 kcycles = 780 kcycles

- Repeating the reasoning for the rarely updated variables it can be derived that one variable is updated every 122 erasing phases; it directly gives the maximum number of cycles:

(1 : 122) x 100 kcycles = 820 cycles

Summarizing the results obtained from cases A and B, the effect on the frequently updated variables of other rare variable updating events is almost negligible.

### 1.4.4 Case C

This third example gives the cycling computation when the few frequently updated variables are randomly changed, without any constraints in the flow (as hypothesized in the previous cases).

The following definition are used:

Nf = number of frequently updated variables
Nr = number of rarely updated variables
Pf = probability of frequently updated variables
Pr = probability of rarely updated variables
Cf = cycles of each frequently updated variable
Cr = cycles of each rarely updated variable

Firstly, it must be said that:

$$Pf + Pr = 1$$

Considering the maximum 100 kcycles on the sectors, corresponding to 800 kcycles at block level, the relationship which gives the maximum cycles for each frequently updated variable is simply:

$$Cf = (800 \text{ kcycles} \times Pf) : Nf$$

Similarly, for the rarely updated variables:

$$Cr = (800 \text{ kcycles} \times Pr) : Nr$$

The variables are supposed to be randomly distributed inside the EEPROM space; nevertheless it is still suggested to place frequently updated variables in different pages: in this way, the maximum number of cycles can be significantly improved, since it is unlikely that a real full random flow is adopted (full random is considered the case in which the same variable is updated consequently for all the maximum number of allowed cycles).

Table 1 presents the computation for some cases. The user can easily deduce different application configurations.

## 1.5 CONCLUSION

This document provides the user with a methodology for using emulated EEPROM.

It highlights that the number of cycles for each byte can be significantly high, and comparable with the normal EEPROM characteristics. This is always true, although the examples shown are simplified and consider ideal hypotheses.

The large size of the emulated EEPROM space (1 kbyte) should give the user enough flexibility for application purposes, in which a set of variables must be updated with high frequency, while for other variables the rate of change is significantly lower.

**Table 1. Randomly updated variables**

| Nf | Nr | Pf | Pr | Pr / Pf | Cf (cycles) | Cr (cycles) |
|---|---|---|---|---|---|---|
| 8 | 1016 | 100/101 | 1/101 | 1/100 | 99.000 | 8 |
| 8 | 1016 | 10/11 | 1/11 | 1/10 | 90.900 | 72 |
| 8 | 200 | 100/101 | 1/101 | 1/100 | 99.000 | 40 |
| 8 | 200 | 10/11 | 1/11 | 1/10 | 90.900 | 364 |
| 64 | 960 | 100/101 | 1/101 | 1/100 | 12.380 | 8 |
| 64 | 960 | 10/11 | 1/11 | 1/10 | 11.360 | 75 |
| 64 | 200 | 100/101 | 1/101 | 1/100 | 12.380 | 40 |
| 64 | 200 | 10/11 | 1/11 | 1/10 | 11.360 | 364 |
| 1024 | - | 1 | - | - | 781 | - |

**Figure 3. EEPROM data updating - Phases 1 and 2**

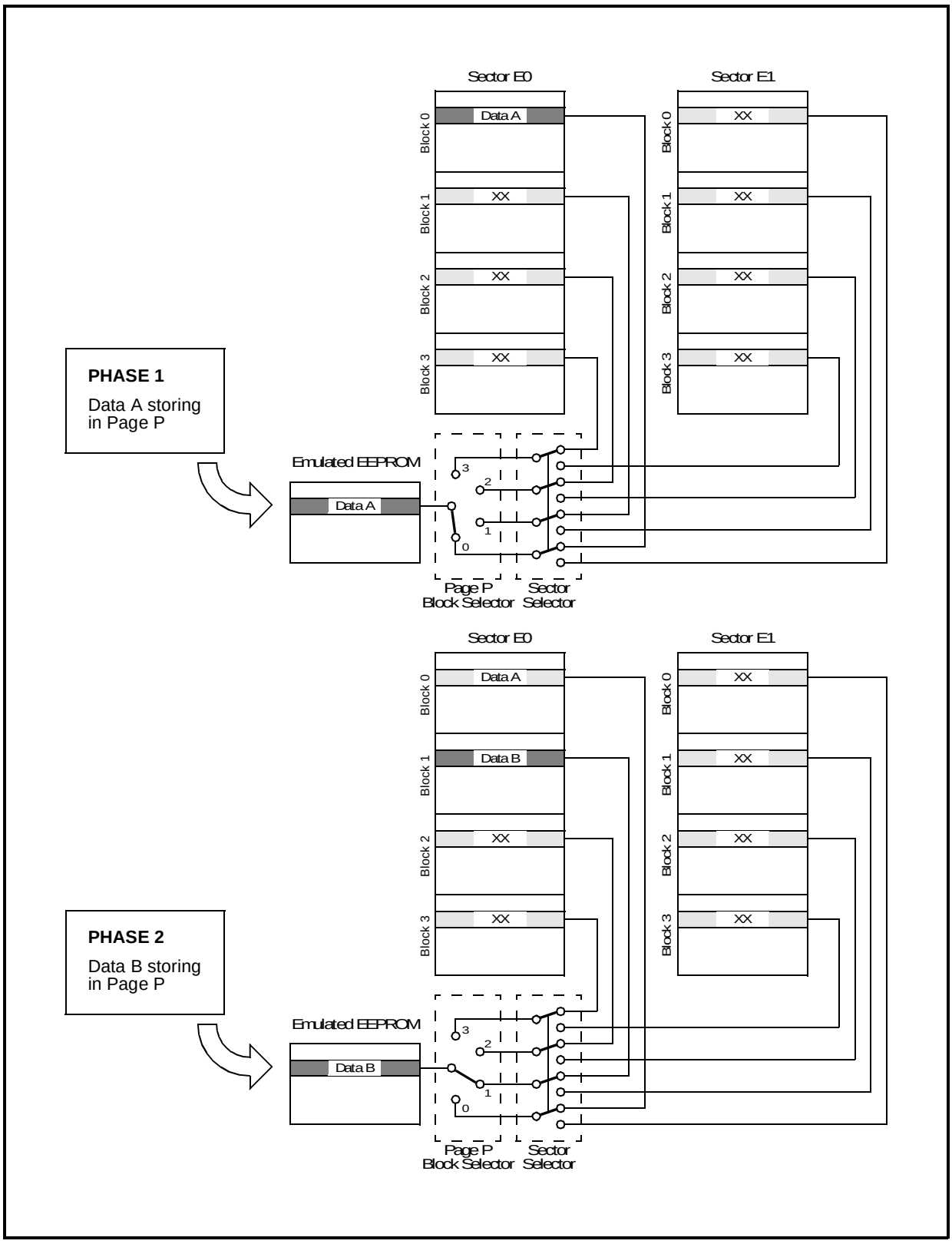


**Figure 4. EEPROM data updating - Phases 3 and 4**

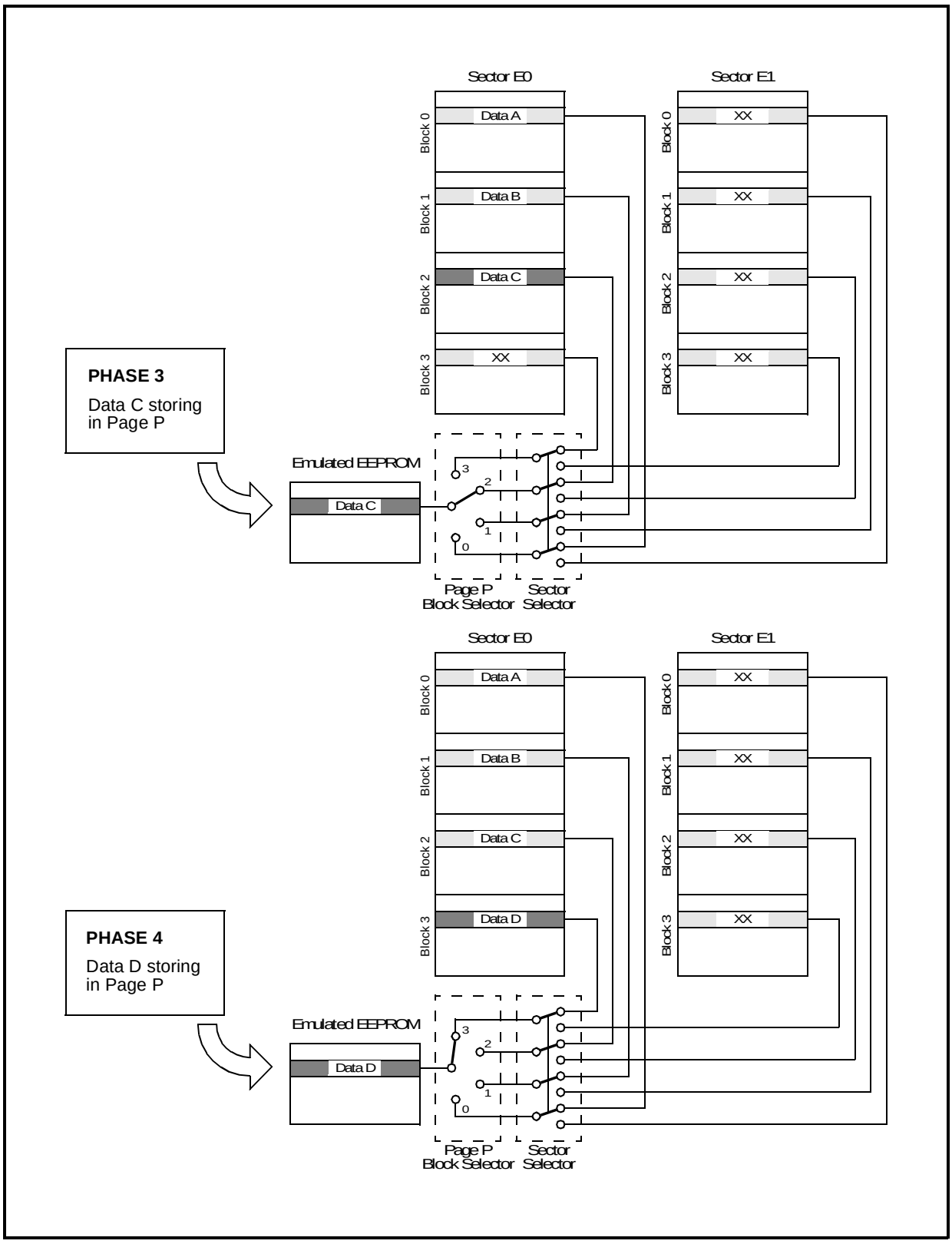


**Figure 5. EEPROM data updating - Phases 5 and 6**

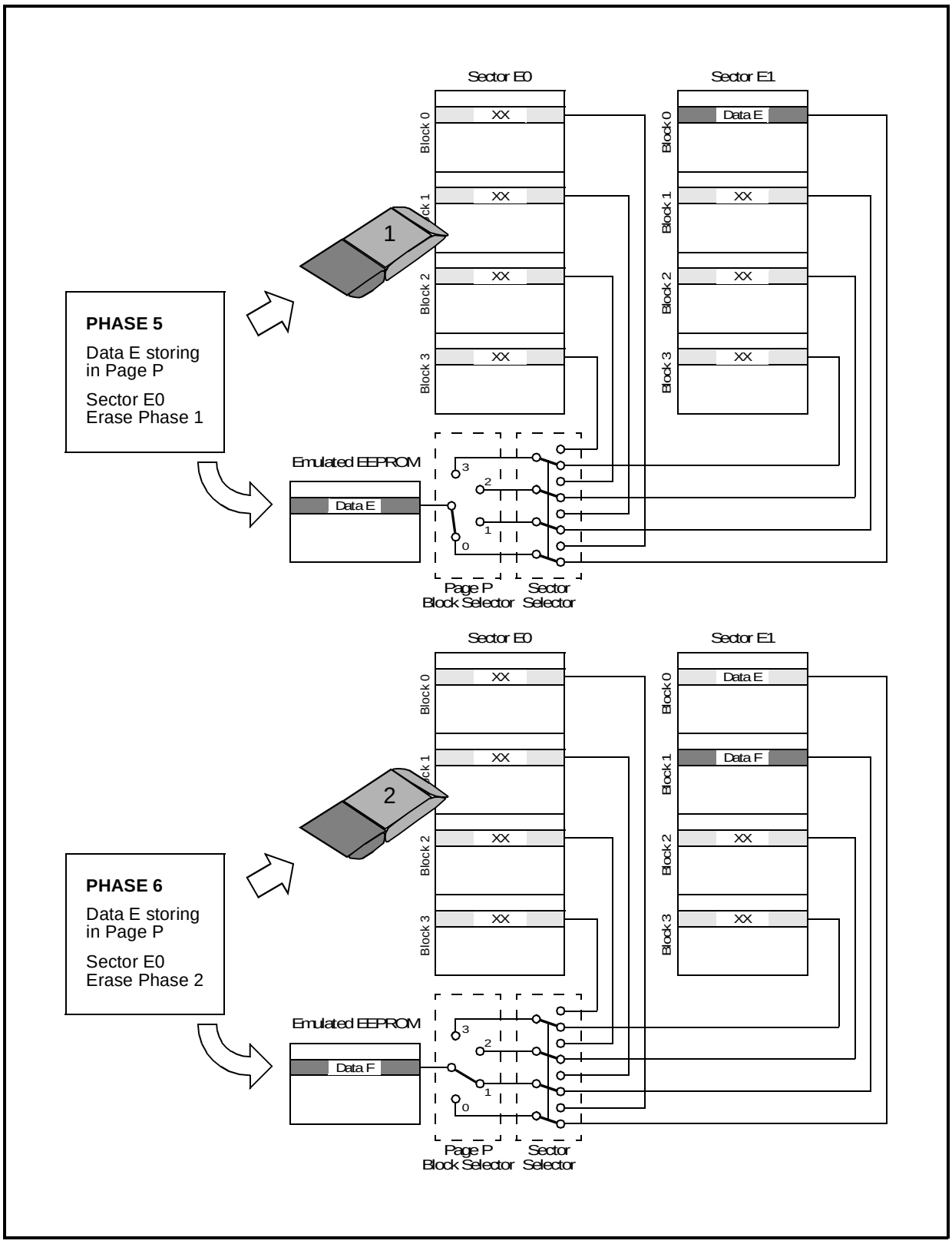
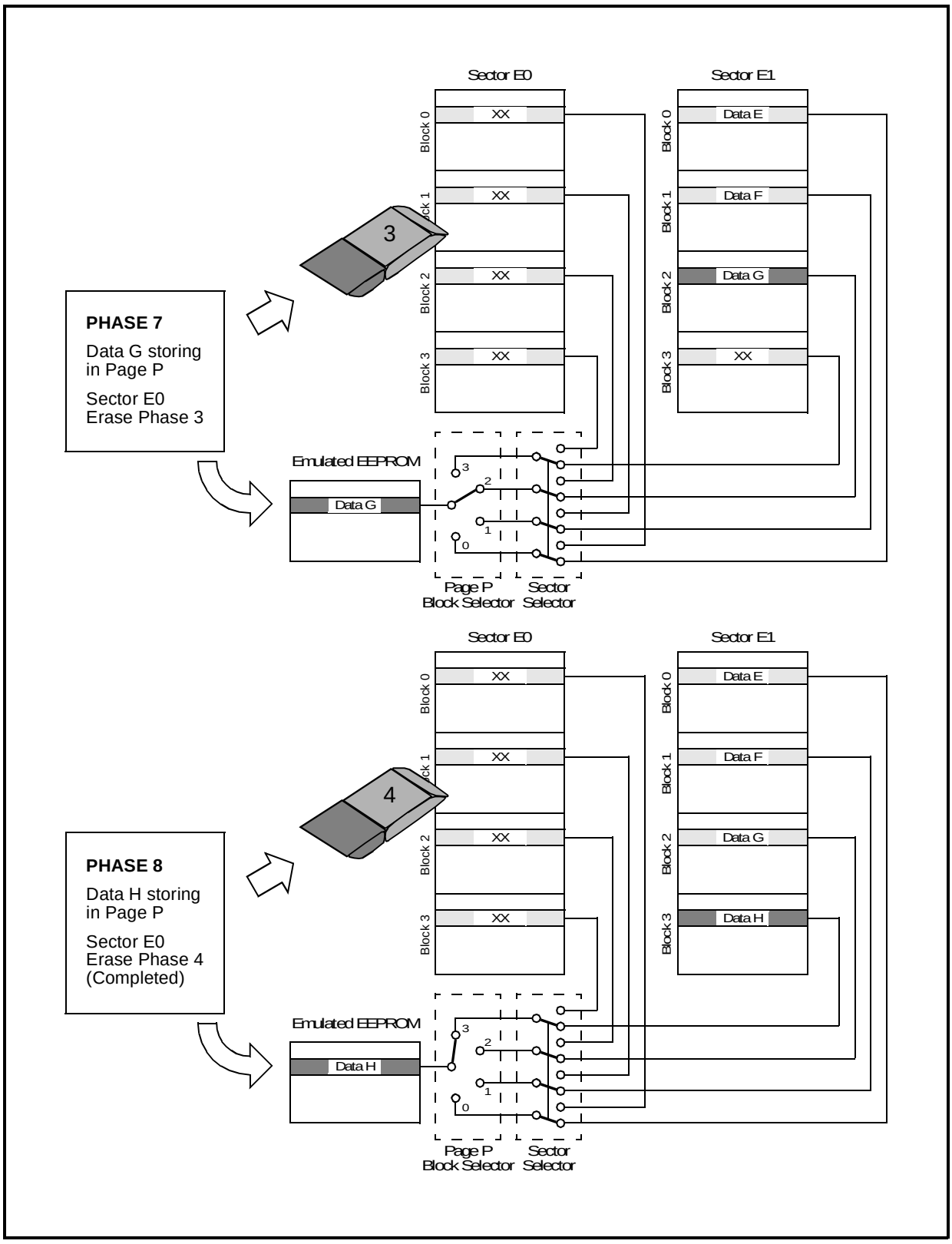


**Figure 6. EEPROM data updating - Phases 7 and 8**



littlediode
COMPONENTS FOR ALL

Littlediode supplies new, hard to find or obsolete electronic components and semiconductors all over the world.

With over two million different components listed you are sure to find the part you need.

Feel free to visit us today at our online store:

**LittleDiode.com**

Looking forward to providing you with the best possible service.