

AP08057

XC866/XC886/XC888

EEPROM EMULATION

Microcontrollers



Never stop thinking

Edition 2007-11-20

**Published by
Infineon Technologies AG
81726 München, Germany**

**© Infineon Technologies AG 2007.
All Rights Reserved.**

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (www.infineon.com).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

AP08057	
Revision History:	2007-10 V1.0
Previous Version:	none
Page	Subjects (major changes since last revision)


<p>We Listen to Your Comments</p> <p>Any information within this document that you feel is wrong, unclear or missing at all? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:</p> <p>mcdocu.comments@infineon.com</p>	
---	---

Table of Contents	Page
1 INTRODUCTION	5
2 IMPORTANT FLASH PARAMETERS	5
2.1 DATA SHEET INTERPRETATION	5
2.1.1 Data retention versus Endurance versus Size	5
2.1.2 Specification of the Data Flash	6
2.1.3 Hardware Error Correction Coding (ECC)	8
3 APPLICATION SPECIFIC REQUIREMENTS	10
4 IMPLEMENTATION OF A REAL-TIME EXAMPLE	11
4.1 EEPROM EMULATION CONCEPT	11
4.1.1 The Basic Problem	11
4.1.2 The Size and Update Rate of the Data-set	12
4.1.3 Emulation Algorithms	12
4.1.4 Running the Application	14
4.2 EMULATION SCENARIO – EXAMPLE A	16
4.3 EMULATION SCENARIO – EXAMPLE B	19
4.4 EMULATION SCENARIO – EXAMPLE C	22
4.5 APPLICATION STATE MACHINE	26
4.6 COMMON FLOWCHARTS OF THE EXAMPLES	27
4.6.1 Main Routine	27
4.6.2 Interrupt Service Routine of Timer 0	28
4.6.2.1 Application Relevant States	28
4.6.2.2 States for Programming	29
4.6.2.3 States for Erasing	30
4.6.3 Non Maskable Interrupt Service Routine	31
4.6.4 External Interrupt	31
4.7 INDIVIDUAL FLOWCHARTS FOR EXAMPLE A	32
4.7.1 PROGRAMMING_DATA()	32
4.7.2 IdentSector()	33
4.7.3 IdentWL()	34
4.8 INDIVIDUAL FLOWCHARTS FOR EXAMPLE B	35
4.8.1 PROGRAMMING_DATA()	35
4.8.2 IdentWL()	36
4.8.3 IdentSector()	37
4.9 INDIVIDUAL FLOWCHARTS FOR EXAMPLE C	38
4.9.1 PROGRAMMING_DATA()	38
4.9.2 IdentWL()	38
4.9.3 IdentSector()	39
5 TESTING AND DEBUGGING	40
6 GLOSSARY	41
7 SOURCES AND LINKS	41

1 Introduction

Many MCUs do not contain on-chip EEPROM. To gain EEPROM functionality without the cost of an external EEPROM, EEPROM emulation, sometimes called Flash Emulated EEPROM (FEE) is often used. This application note gives an example for an EEPROM emulation algorithm for the XC866 and XC88x's flash modules. The XC866/XC88x devices each have two different types of flash modules. The PFLASH or Program Flash is intended to be used to store code and constants with typical flash write/erase cycles and retention times. The DFLASH or Data Flash can also be used for code and constants but has some special features which make it more suitable for EEPROM emulation.

2 Important Flash Parameters

From XC866/XC88x data sheet some important flash parameters have to be considered. Values like data retention, endurance and failure rate have to be discussed. Flash-module specific values like minimum size for an erase, minimum programming width or gate disturb are important for the algorithm. Last but not least the specific timing for erase and programming have impact on hardware resources like external capacitors and timers. The factors mentioned above are not isolated, they influence each other. Together all of the factors must match the application specific requirements.

2.1 Data Sheet Interpretation

2.1.1 Data retention versus Endurance versus Size

In general dealing with flash-memories involves statistics and likelihood of failures in flash contents. This includes all kind of failures where the read out value is different from the originally programmed value. Practically the failure-rate cannot be zero and is influenced by several effects. The data sheet shows some cornerstones of a multi-dimensional matrix. The values stated are characterized in the product qualification process according to the automotive AEC_Q100 standard. "Error-free" or "Zero-Defect" in this context means that the error rate for data retention is lower than 1 ppm (parts per million) for the stated maximum cycle figures over time and size for the data flash bank.

Generally it can be said that the lower the endurance (number of program and erase cycles) the longer the "error-free" data retention (guaranteed time where the flash cell keeps its memory after cycling). Under the assumption that the programming and erasing sequence is correct, the retention related failure rate for a single flash cell is dependent on temperature over time (programming, erasing and storage) and the number of cycles. The temperature influence is non-linear and can hardly be adjusted within an application. Therefore the values for the data flash EEPROM emulation are specified over the whole temperature range. To get the total failure rate of a cycled flash block, the cell failure rate has to be multiplied by the numbers of flash cells in this block (linear dependency). In other words, if the endurance is increased, the numbers of flash cells have to be reduced to get a "zero-defect" data retention over a certain time.

The values given in the data sheet for "Data Retention" mean Retention After Cycling (RAC). For most applications this is a very unrealistic case as the cycling is usually distributed over the product lifetime. As an example a device is cycled in its first year 10k times, in the second year another 10k times etc. and after 10 years operating time the full number of allowed cycles is exhausted but also the device is at its end of life. This more realistic scenario relaxes the flash specification (retention and failure rate) very much. Nevertheless RAC-specification is the worst case which is required for a robust specification.

Another relaxing factor is that the dataset size is usually much smaller than the totally emulated flash memory. But the failure rate is always calculated with the total size. This means that the application is on the safe side and there is some headroom left.

Use Case	Retention	Endurance	Size
Program Flash (PFLASH)			
	20 years	1,000 cycles	up to max. program memory
Data Flash (DFLASH)			
I	20 years	1,000 cycles	4096 bytes
II	5 years	10,000 cycles	1024 bytes
III	2 years	70,000 cycles	512 bytes
IV	2 years	100,000 cycles	128 bytes

Table 1 EEPROM Use Cases I - IV: different retention versus endurance and size

Table 1 shows the specified Retention versus Endurance and Size for the XC866/XC88x flash modules. This table should be interpreted as follows:

For a retention time of 20 years, up to 4096 bytes can be cycled up to 1000 times.

For a retention time of 5 years, up to 1024 bytes can be cycled up to 10000 times.

For a retention time of 2 years, up to 128 bytes can be cycled up to 100000 times or up to 512 bytes can be cycled up to 70000 times.

2.1.2 Specification of the Data Flash

The data flash bank consists of the flash cell array, the read amplifiers, the charge pumps and the digital flash interface to the CPU core. The programming of a wordline (WL) is done by a hardware “assembly buffer” which consists of 32 bytes. The CPU has to write the data to this buffer and the address of the WL to be programmed. After triggering the programming state machine, the charge pumps power up and after reaching the final voltage level, all 32 bytes of a WL are programmed at once. After that the charge pumps power down again. While the charge pumps are active, a read access to the flash bank is not possible. The programming process takes time and stresses the device, i.e. there is physical limit for the number of programming cycles. The erase cycle works in a very similar way with the same restrictions. Reliability cannot be guaranteed when using the device above these limitations. Therefore it is **very important that these limits are understood by the user who implements an EEPROM algorithm.**

a) Erased State

The erased state of a flash cell is ZERO. After programming a ONE to a cell, only an erase can bring it back to ZERO. Writing a second time to the same cell with a ZERO does not change the ONE. But a ZERO can always be programmed to a ONE.

b) Minimum Program Width

The minimum program width is one WL. This is because there is just one 32 byte wide assembly buffer which is copied all at once to the WL. For the DFLASH there exists an additional feature compared to the PFLASH. It is allowed to have two “*gate disturbs*”. This means that the WL can be written two times. Hence this results in a “*virtual minimum program width*” of one byte. Examples:

- *1st write* to lower 16 bytes of a WL (assembly buffer: upper 16bytes containing all ZEROS), *2nd write* to upper 16 bytes of the same WL (assembly buffer: lower 16 bytes either unchanged or all ZEROS).
- *1st write* one byte of a WL followed by the *2nd write* of remaining 31 bytes of the same WL with ZEROS or unchanged value to the previously programmed byte.

c) Maximum Program Width

The maximum program width is one WL (32 bytes).

d) **Minimum Erase Width**

The minimum erase width is one sector, which is a multiple number of WLs. The DFLASH consists of 10 sectors with different sizes.

- Sec 9-6 have 4 WLs each ($4 \times 32 = 128$ bytes)
- Sec 5-4 have 8 WLs each ($8 \times 32 = 256$ bytes)
- Sec 2-3 have 16 WLs each ($16 \times 32 = 512$ bytes)
- Sec 1-0 have 32 WLs each ($32 \times 32 = 1024$ bytes)

e) **Maximum Erase Width**

The complete flash module can be erased all at once. Multiple sectors can be erased all at once.

f) **Maximum Endurance for a Single Flash Cell**

The maximum allowed endurance for a single flash cell is 100k cycles. This value is the basis for all further considerations regarding endurance. A single flash cell cannot be programmed or erased due to the minimum program and erase widths, e.g. one wordline for programming and one complete sector for erase (see above). Therefore the architecture of the flash array has to be regarded.

g) **One Flash Cycle**

One Flash Cycle is defined as the programming of a sector followed by an erase of the sector. Following example:

- One cycle for SEC9 can be
 - one to eight write accesses as the SEC9 has four WLs
 - followed by one erase

h) **Number of Flash Sector Erases**

The maximum number of erase cycles per sector is defined to 100k times (same as f)

i) **Number of Flash Bank Erases**

The maximum number of erase cycles per flash bank is defined to 300k times.

j) **Number of Flash Bank Programmings**

The maximum number of programming cycles per flash bank is defined to 2500k times.

k) **Programming Time**

The programming time is ~2.6ms.

l) **Erase Time**

The erase time is ~102ms.

m) **Aborted Erase**

An ongoing Erase can be aborted under defined circumstances. An Aborted Erase counts as one Cycle.

n) **Combined Use Cases**

It is allowed to combine the Use Cases of Table 1 with certain restrictions. To reach the "Zero-Defect"-failure rate for the whole data flash, each Use Case can be initiated only one time. The combination of the Use Cases must not violate the boundary conditions given in a) – m). This means e.g. that Use Cases II + III + IV can be used at the same time as long as they all use different sectors. Whereas cycling two times Use Case IV even in two different sectors would violate the "Zero-Defect" definition. Here the overall failure rate would be doubled.

Notice that the specification of Table 1 limits the number of bytes that can be cycled, not the number of sectors. For example the 5 year / 10k cycles / 1024 byte spec could be applied many different memory areas including:

Sectors 4-9 (total of 1k bytes) – 10k cycles could include 60k sector erases

Sector 2-3 (total of 1k bytes) – 10k cycles include 20k sector erases

Sector 1 (1k bytes) – 10k cycles could include 10k sector erases

Sector 0 (1k bytes) – 10k cycles could include 10k sector erases

Examples:

- If a retention of 20 years is required, a flash size of 4 Kbytes can be cycled up to 1000 times (from Table 1).
- If an endurance of 100k cycles is needed, the maximum number of bytes is 128; for this a data retention of 2 years is guaranteed. If more than 128 bytes have to be cycled the endurance has to be reduced; alternatively a new sector can be used with another 100k cycles – but this would double the failure rate.
- A retention of 3 years with an endurance of 70k is outside the specified limits, i.e. the failure rate is too high according to “Zero-Defect” definition. To achieve the desired retention time and endurance we can use the 5 years / 10k cycles / 1 Kbyte specification. This specification says that up to 1k Byte of flash can be cycled up to 10k times with a retention time of 5 years. We can cycle 7 sectors 10k times each we could emulate 70k cycles and still meet the specification as long as only 1k bytes are used. The following is an example of how this can be done:
 - cycle Sec9 10k times, then Sec8 another 10k times, then Sec7, Sec6 etc... until Sec3 is cycled (for a total of 70k cycles). Take care that the sum of all bytes is not higher than the specified limit of 1 Kbyte, i.e. 4*128 (sectors 9-6)+ 2*256 (sectors 5-4) + 1*512 (sector 3) is bigger than 1 Kbyte. Therefore reduce the number of used bytes from Sec 4/5 to 128 and from Sec3 to 256.
- Example for multiple sets of data with different requirements: first set of NVM data needs a retention of 2 years / 70k cycles / 512 bytes (dataset size is 16 bytes) and second set of NVM data needs 5 years retention / 5k cycles / 1024 bytes (dataset size is 128 bytes alternating in two sectors for redundancy).
 - 2 years / 70k cycles / 512 bytes
 This can be implemented with Sec 6, 7, 8, 9 each with 4WLs (128 bytes). Each sector can be written up to 8 times, each sector can be erased up to 70k times. This results in
 $4 \text{ sectors} * 70k \text{ erases} = 280k$ sector erases for the DFLASH bank (this is below the 300k limit per bank, OK) and
 $2 \text{ programs per WL} * 4 \text{ WL per sector} * 4 \text{ sectors} * 70k \text{ cycles} = 2240k$ programming cycles for the DFLASH bank (< the 2500k limit per bank, OK).
 - 5 years / 10k cycles / 2*512 bytes
 In addition to the above data set it is allowed to have two other sectors, e.g. Sec 2, 3, each with 16WLs (512 bytes) with the following cycling:
 $2 \text{ sectors} * 5k \text{ erases} = 10k$ sector erases ($280k + 10k = 290k$ total erases per bank, still OK)
 and
 $1 \text{ write per WL} * 16 \text{ WL per sector} * 5k \text{ cycles} * 2 \text{ sectors} = 160k$. which in totals $2240k + 160k = 2400k$ programming cycles per DFLASH bank which is in spec.

2.1.3 Hardware Error Correction Coding (ECC)

The XC866/88x flash modules have an on-chip hardware error correction. This is a digital failure detection and correction mechanism based on the Hamming-Code principle. The implementation is an 8bit + 4bit Hamming Code, i.e. every data byte consists of 12 bits, 8-bits of user data + 4 hidden bits with the ECC Hamming Code. Note that the hamming code is hidden and doesn't count toward the advertised size of the flash module. So if a part is described as having 8k bytes of flash, there is actually 8k bytes of user flash + 4k bytes of ECC Hamming Code. Physically the 4 ECC bits are distributed within the 12 bit word. If a byte is read or written 12

bits are always handled internally. The Hamming Code is calculated by the flash interface before it is written to the flash cell array and is decoded upon a read. If a single bit error happens within any 12-bit read (8-bit user data or even the 4-bit Hamming Code itself) the scheme can always detect and even correct this error. In this case the ECC-error will be signaled (together with the failing address) and an NMI interrupt can be triggered if enabled. This feature gives an additional safety margin and can be used for the EEPROM algorithm.

The effectiveness of the ECC can be easily shown with following example for 16k bytes of code memory:

- Without ECC the likelihood for a corrupted code is **1 Bit out of 131072 Bits** (16k*8bits).
- With ECC the likelihood for a corrupted code is **2 Bits out of 12 Bits**, as all other failures can be corrected.

3 Application Specific Requirements

Every application has its specific requirement and therefore an EEPROM algorithm has to be developed individually. No single algorithm can fit all applications. The following guideline describes how the different requirements can be implemented with the XC800 flash.

A few questions regarding the applications boundary conditions have to be answered upfront. Below is a typical questionnaire that can help to make the requirements clearer. Here the example “**Operation Hour Counter**” is given. This example program keeps track of how long the device has been operating. Most of the time the MCU is idle (either in power down mode or off). The MCU periodically wakes up to update the operation time variable in emulated EEPROM and then goes back to sleep. The active time is very short (a few days) compared to the product lifetime (many years). If the operating hours exceed a limit the application signals that it needs service / maintenance. Examples for such applications can be found in automotive body control modules (intelligent actuators) or in industrial tooling equipment.

EEPROM Emulation Questionnaire		
	Question	Answer
1	When is the emulation needed?	
a	Only after powering up/down?	No, always when application is running.
b	Application can wait (is stalled) while programming or erasing (polling method)?	No, application cannot wait.
c	Is real-time operation critical?	Yes, application has to run. Programming and Erasing should run in background. Interrupts have to be processed in real-time.
2	What happens during power failure?	
a	Can a power failure be excluded while programming or erasing?	No, power failure is possible.
b	Can a power failure be detected while programming or erasing?	Yes, by using the Early Warning Feature for VDDP or other external hardware together with ADC
c	Should the algorithm be tearing-safe, i.e. no data loss from a power loss?	Yes, if supply voltage drops, the actual value has to be programmed resp. the old value must not be lost.
3	Which safety level is needed?	
a	Keep the last dataset in another sector?	Yes, the last valid count value must be available as backup dataset.
b	Keep one basic data-set in PFLASH for having a fallback-setup?	No, not possible as application is time-keeping.
c	Double buffering of every new data-set?	No.
4	Dataset size and endurance/retention?	
a	Minimum dataset of non-volatile bytes?	< One WL.
b	Number of programming cycles and required retention.	Maximum operation time is about 250 (2500/25000) hours with granularity of 1 (10/100) second and a retention of 2 years.

Table 2 EEPROM Emulation Questionnaire for example “Operation Hour Counter”

From this questionnaire the emulation scenario can be developed. Several solutions are possible.

- The dataset size fits into one WL.
- The update rate is 1 sec for a max. time of 250 hours which results in $250 \times 3600 = 900k$ programming cycles.
- Using four sectors (SEC9 – SEC6) each with 4 WLs and writing only one time per WL results in 900k program cycles / 4 sectors = 225k programming cycles per sector and in 225k program cycles / 4 WL = 56250 erase cycles per sector (one sector erase after all 4 WL are written). The desired retention time is 2 years.
- The total emulated flash size is $4 \times 128 \text{ bytes} = 512 \text{ bytes}$.
- Specification Check: 2 years / 70k cycles / 512 bytes
 - ✓ max 300k programming cycles per sector (here 225k)
 - ✓ max 70k erase cycles per sector (here 56250)
 - ✓ max 300k erase cycles per flash bank (here $4 \text{ sectors} \times 56250 = 225k$)
 - ✓ max 2500k programming cycles per flash bank (here 900k)
 - ✓ max size is $4 \times 128 \text{ bytes} = 512$

4 Implementation of a Real-time Example

4.1 EEPROM Emulation Concept

4.1.1 The Basic Problem

Every EEPROM emulation technique has the problem that data has to be stored in a non-volatile data memory. Usually the data memory is a bitwise organized RAM and data can be written or overwritten with one write-access. If the non-volatile memory is flash-based the access mechanism for reading, writing or changing is different than for RAM. The device specific flash-architecture usually allows bitwise reading, while writing is possible in units of several bytes (here one WL) and erasing in even bigger portions (here one sector). In any case the software has to handle this particular flash-architecture. The first problem is that old data cannot be overwritten like in a RAM. The new data has to be stored

- a) either in another memory location
- b) or the old data has to be erased before it can be updated.

Both will generate a second problem:

- a) if the address of the data is changing, an algorithm is needed to find the actual valid data (even after a power loss);
- b) in the other case, an upfront erase will erase more data than it should (an entire sector); this means that the active data-set is usually much smaller than the size of the memory that it occupies.

The starting point for any EEPROM emulation algorithm is to decide how many bytes of flash are required to store the data set(s). The next question is the cycling rate, i.e. endurance/retention of this section of flash. If the numbers for such a cycling data-set are clear, an emulation algorithm can be developed according to the specific flash architecture. If there is more than one data-set the whole algorithm can be very complex. Practically software will face a combination of both problems.

4.1.2 The Size and Update Rate of the Data-set

A few examples for different data-set sizes are given below. By varying the data retention different update rates are possible. In Table 3, Example A shows 3 ways to emulate 31 bytes of data. The first way gives 1120k writes with 2 years retention, the second method is for 160k writes with 5 years retention and the third method gives 16k writes with 20 years retention. All methods for example A use a total of 512 bytes of flash and 4 sectors. The last column gives the maximum possible flash size for a failure rate below 1 ppm. The other examples can be interpreted the same way.

Example	Data-set Size (up to)	Number of Writes (update rate)	Retention	Total Flash Size Required for Emulation	Number of Erases	Remarks
A	31 bytes	1,120k	2 years	512 bytes in 4 sectors	4x70k	
		160k	5 years		4x10k	
		16k	20 years		4x1k	
B	95 bytes	700k	2 years	1024 bytes in 2 sectors	2x70k	¹⁾
		100k	5 years		2x10k	
		10k	20 years		2x1k	
C	127 bytes	280k	2 years	512 bytes in 4 sectors	4x70k	
		40k	5 years		4x10k	
		4k	20 years		4x1k	

Table 3 Different size of data-sets results in different update rates

¹⁾ Here the failure rate would be bigger than 1 ppm. According to Table 1 70k cycles for 2 years are limited to 512 bytes, see also paragraph 2.1.2 n).

4.1.3 Emulation Algorithms

Different solutions are possible for the same problem. Here a very simple algorithm is demonstrated based on the **round robin** principle. This is easy to understand and the endurance calculation is simple. The algorithm does not need much computing performance but it requires a large amount of flash memory.

Real-time Compliance

The algorithm is embedded in a real-time system, i.e. there is an operating system timer which gives the tick for the different application states. The states for the EEPROM emulation are changed with this tick. This makes the emulation algorithm easy to adapt to any interrupt driven application.

The trigger event for a data-set update is usually a command or a signal. To simplify the application there are two possibilities here:

- external signal (e.g. pushbutton on pin 1.6)
- internally generated signal (set/reset port 1.6 periodically by timer interrupt)

This event triggers the emulation software state machine and the new data-set values are stored in the flash memory.

The Buffer Capacitor

It is assumed that the long erase time cannot be buffered by an external capacitor in case of sudden power drop. But it has to be assured that there is enough time to finish the programming step (~2.6ms). The XC800 family has a internal comparator that can trigger a NMI when VDDP is below 4V. When VDDC (the output of the internal voltage regulator that supplies the core) is below 2.3V a second PreWarning (VDDCPW) NMI can be generated, and when VDDC is less than 2.1V a brown-out reset occurs. The minimum voltage drop across the internal regulator (VDDP-VDDC) is approx. 0.5V. Therefore following calculation is done:

- (1) $Q = I * t = C * U$,
- where C = 47uF buffer capacitor on VDDP,
 - where I = 20mA total current consumption of the device on VDDP,
 - where U = 1.2V voltage drop of VDDP from the point where the device first recognizes the drop (VDDP = 4.0V) until the programming is no longer possible due to a quickly approaching brown-out (at VDDP = 4.0V-1.2V = 2.8V → VDDC ~ = 2.8V-~0.5V = ~2.3V).
 - note: here typical values are taken. In case of power fail detection while programming is in progress, it is recommended that some power saving modes are entered immediately. This will lengthen the critical time and might reduce the required buffer capacitor size.
- (2) $t = C * U / I = 47 \text{ uF} * (4V - 2.8V) / 20\text{mA} = 2.82 \text{ ms}$

Indication Bytes

An incompletely erased sector (due to power fail) might appear fully erased upon the next power up. But the retention time of the erased state cannot be guaranteed. There must be a method to detect a completely erased sector. Here the algorithm confirms an erased sector by writing a special "*confirm erase indication*" byte to the highest address of the sector. If this information is not found in the sector, it has to be erased before it can be written.

The algorithm also needs a mechanism to distinguish the most recent actual data from old data. This mechanism has to work after a power up as well, where information from volatile memories (pointers, variables in RAM) cannot be used. Therefore the highest byte of each WL is used to indicate if a WL is already programmed - "*programmed indication*".

By using a special trick, the information "*confirm erase indication*" and "*programmed indication*" can be combined in one byte. This trick is using the fact that DFLASH allows a second gate disturb on a WL. This means that two write accesses are allowed on a single WL. The first write access will write the "confirm erase indication" byte (value 0x80) to the highest sector address after a successful erase of the sector. If data is also written to that WL (second write access → gate disturb), the highest address is written with the "programmed indication" byte (value 0x81). This gives the software the necessary information about:

- the **free sectors** which can be programmed
- the **actual dataset** which is always the one which has the value 0x81 with the lowest sector address
- a full **sector to be erased**

The values 0x80 and 0x81 are selected especially by using the Hamming-Code of the hardware error correction (ECC) in a proper way. Otherwise two bytes would be needed to store the same information. By knowing the Hamming-Coding it is possible to overwrite a value 0x80 with 0x81. Other values might lead to an ECC-event.

Round Robin Principle

The data-sets are written consecutively in a descending address order. If the last WL of the last sector is programmed the algorithm wraps around and starts again with the first sector (highest address). Once a sector is "full" and the next sector contains new data, the full sector(s) will be erased and confirmed. It is

possible that the erase step is aborted because of immediate read/write request of new data or due to sudden power loss. In this case the erase is aborted and the partially erased sector will not be confirmed and has to be erased again.

The Search Functions

a) IdentSector()

This function searches for the sector with the actual most recent data and for full sectors that have to be erased. It also recognizes if a sector erase still has to be confirmed or if there is no valid data in a sector.

b) IdentWL()

This function gets the “actual sector” information from IdentSector() and searches for the actual WL within this sector. It checks if the sector is full and calculates the address of the WL that has to be programmed next.

Both functions have to be adapted to the requirements of the specific model.

The Flash Timer Interrupt

The flash module is supported by a hardware timer which ensures the proper timing for “programming”, “erase” and “aborted erase”. The overflow-bit can generate an NMI interrupt. Inside the function INT_vinmilsr() several global bits have to be handled which interact with the emulation software state machine.

The Emulated Data-set

Within the application “Operation Hour Counter” there exists only one data-set which is cycled. Example A uses just one WL where in Example B and C multiple WLs are used (see Table 3). The non-volatile variables in the dataset can be used for debugging purpose and plausibility checks like the following:

- count power loss events to prove the tear safe feature
- verify if every update trigger leads to successful programming
- check if relation between program and erase counts are as expected

Variable Name	Meaning	Length	Position
ulCountVal	Measures low time of P1.6 in T0 periods (pushbutton)	4	0 – 3
ulEraseCount	Counts erase events	4	4 – 7
ucAbortEraseCount	Counts aborted erase events	1	8
ulProgCount	Counts programming events (every WL)	4	9 – 12
ucAA	not used	1	13
uiVDDPreWarnCnt	Can be used to count power loss events	2	14 – 15
ulFallEdgeCnt	Counts falling edges on P1.6	4	16 – 19
ulRiseEdgeCnt	Counts rising edges on P1.6	4	20 – 23
ulWritingCycles	Counts programming events (every data-set / Operating Seconds)	4	24 – 27
uiECCCnt	Can be used to count ECC_NMI events	2	28 – 29
ucIndProg	Not used	1	30
ucIndErase	Indicator Byte	1	31

Table 4 Contents of the emulated data-set; the position is the byte address within the WL

4.1.4 Running the Application

The examples can be downloaded to a target application such as a starterkit. For each example there is two versions, one for the XC866 and one for XC886/888 family. The count state of the operation seconds is

displayed on P3. Alternatively interesting values can be printed to the COM-port (only coded for Example A). With this it is easy to do some experiments like pressing RESET or cycling power.

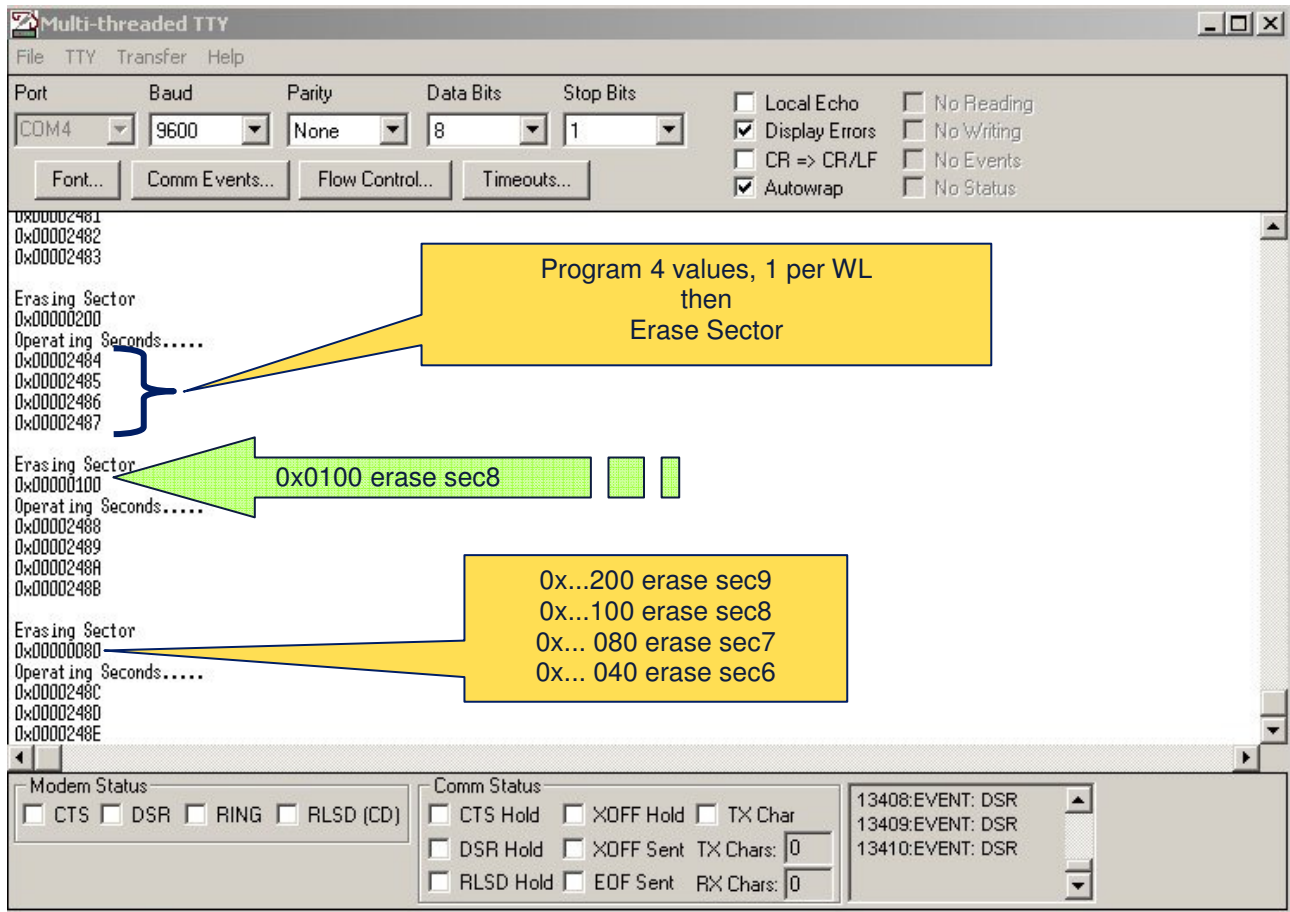


Figure 1 Screenshot of the MTTY window for Example A

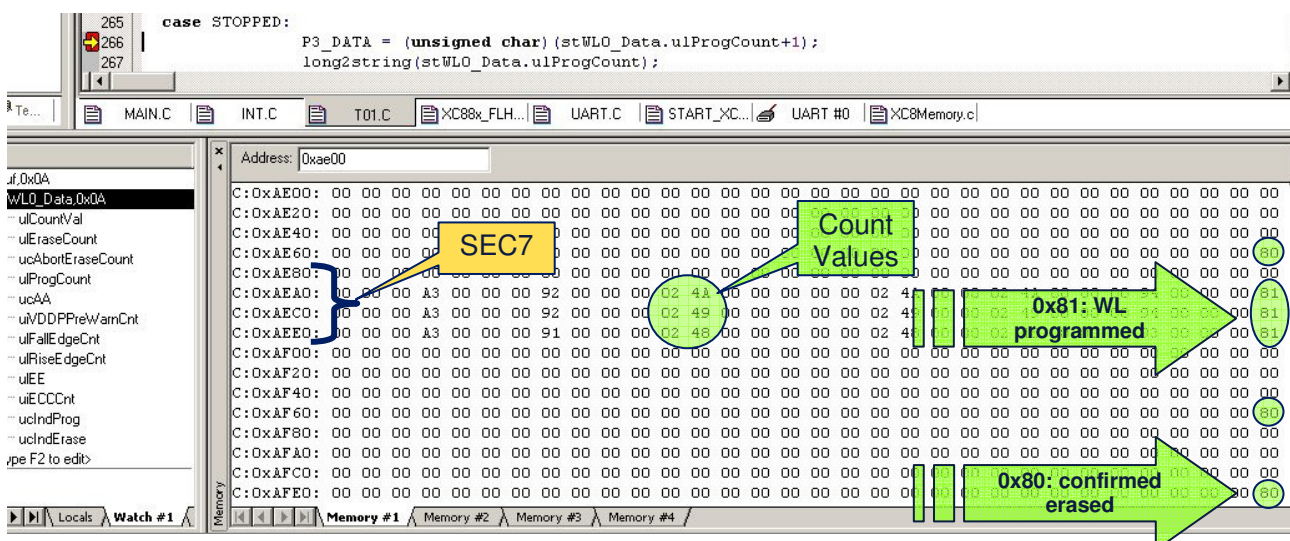


Figure 2 Screenshot of a debug session with uVision

4.2 Emulation Scenario – Example A

Idea	Cycle one WL through four sectors in a round robin			
Used Sectors	Sec 9 – 6 (128 bytes (4 WL) each, total size 512 bytes)			
Data-set	1 WL → max. 31 Data bytes (+ Indicator Byte)			
IndicatorByte	shows the status of a WL, the upper most IndicatorByte in a sector also shows the erase status of the sector			
One Flash Cycle (for endurance calculation)	4x Program, 1x Erase			
Data-set Update	Program	Erase	Aborted Erase	Note
Min. Time	1x	1	-	+ algorithm
Max. Time w/o Aborted Erase	2x	1x	-	+ algorithm
Max. Time w/ Aborted Erase	2x	1x	1x	+ algorithm

Table 5 Overview Example A

Program flow: assumption: used sectors are successfully erased & confirmed

- (1) program first WL of the first sector (starting with highest address WL in Sec 9)
- (2) next data update: program 2nd WL (in descending order)
- (3) next data update: find the WL with the actual data and program the next WL
- (4) if sector is full (all 4 WL are programmed) → program first WL of next sector & erase old sector(s) & confirm erase
- (5) if last sector is full (Sec 6) → repeat step (1) to (4)

Meaning of Indicator Byte	Position	Value	Interpretation
Conf & Prog	highest byte of a sector	0x00	sector is erased but not yet confirmed
		0x80	sector is erased & confirmed
		0x81	WL is programmed
		other	corrupted data → sector to be erased
Prog	highest byte of a WL	0x00	WL is not programmed
		0x81	WL is programmed
		other	corrupted data → sector to be erased

Table 6 Meaning of the Indicator Byte for the Search Functions

The following gives a graphical overview of the emulation scenario for “Example A”. For simplification only two sectors are shown. The details can be taken from an EXCEL spreadsheet “**emulation_exA.xls**”.

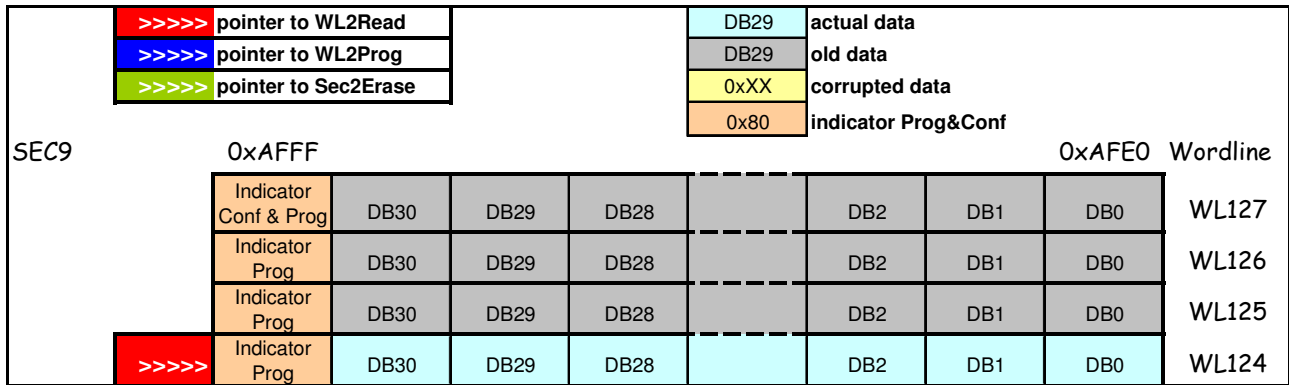


Figure 3 Fully programmed Sector 9 with Indicator Bytes, WL124 contains the valid data

Emulation Steps for Example A

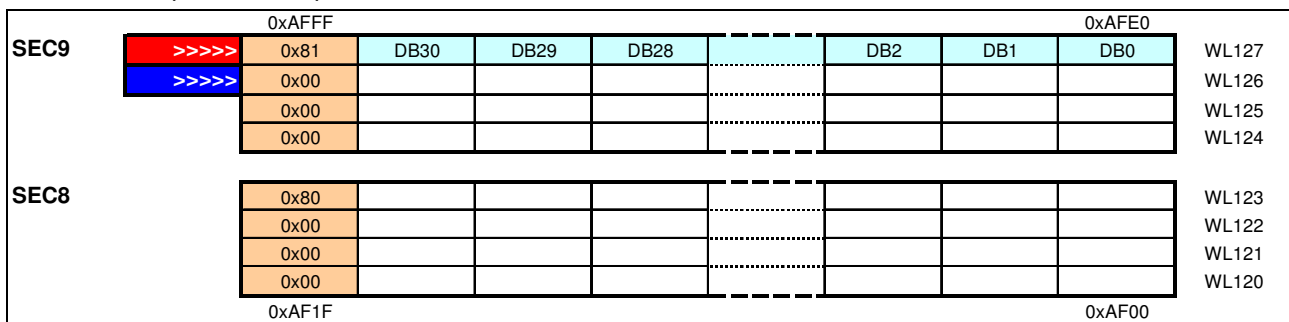


Figure 4 First time write any valid data to the upper most WL of SEC9, after SEC9 and SEC8 are successfully erased and confirmed

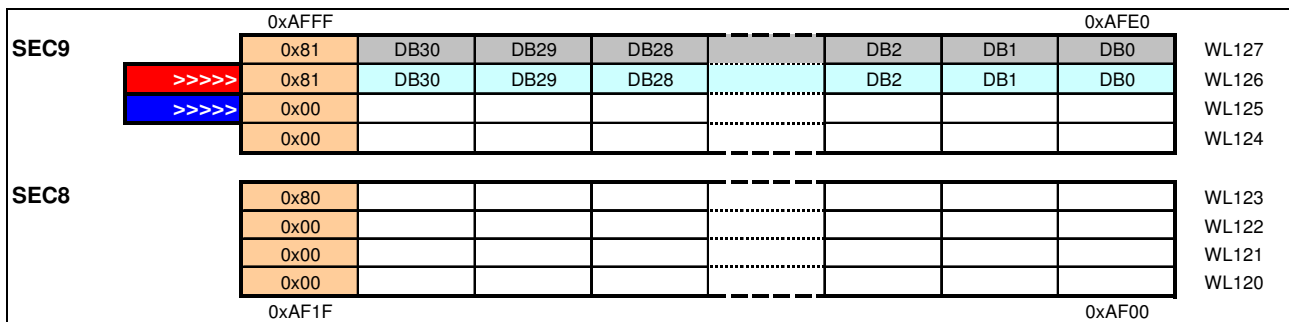


Figure 5 First data-set update

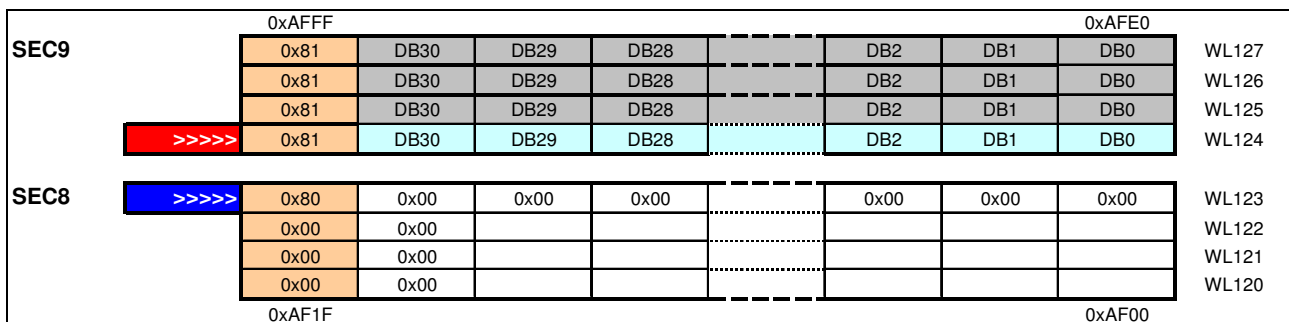


Figure 6 Data-set update; SEC9 is now fully programmed and SEC8 has to be used for next data

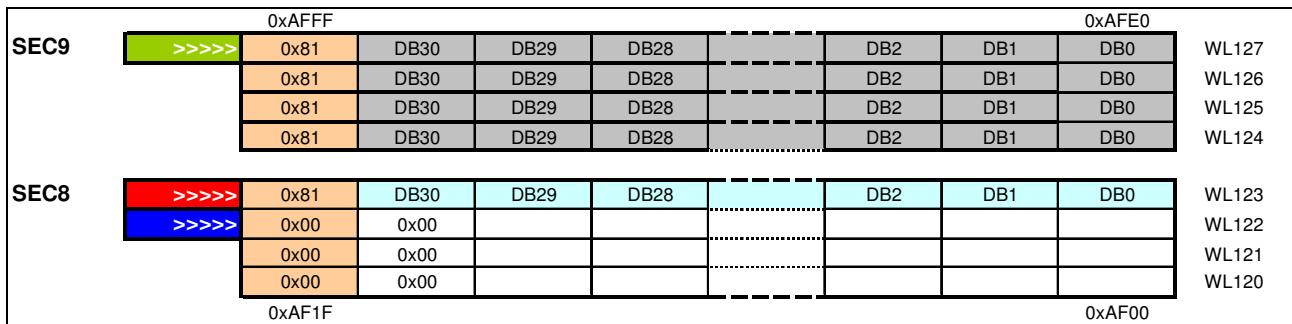


Figure 7 SEC9 can be erased as it contains old data only

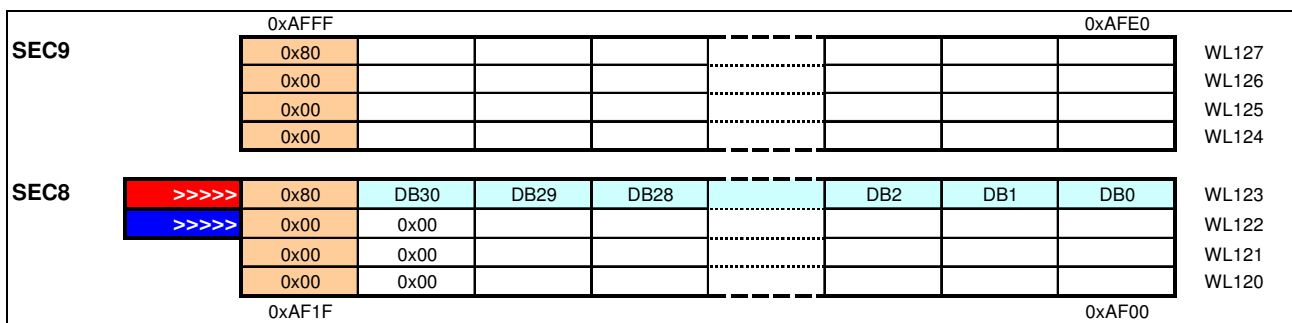


Figure 8 SEC9 is now confirmed erased and ready for new data

4.3 Emulation Scenario – Example B

Idea	Cycle three WLs through two sectors in a round robin			
Used Sectors	Sec 3 – 2 (512 bytes (16 WL) each, total size 1024 bytes)			
Data-set	3 WL → max. 95 Data bytes (+ Indicator Byte)			
IndicatorByte	shows the status of a WL, the upper most IndicatorByte in a sector also shows the erase status of the sector			
One Flash Cycle (for endurance calculation)	15x Program, 1x Erase (for 5 data-sets)			
Data-set Update	Program	Erase	Aborted Erase	Note
Min. Time	3x	-	-	+ algorithm
Max. Time w/o Aborted Erase	4x (first data-set)	1x	-	+ algorithm
Max. Time w/ Aborted Erase	5x	1x	1x	+ algorithm

Table 7 Overview Example B

Program flow: assumption: used sectors are successfully erased & confirmed

- (1) first program three WLs of the first sector in descending order (starting with highest address WL-1, here Sec3 WL94-92), use the last byte as “Prog Indicator” (0x81) which indicates that data-set is successfully programmed
- (2) program “Conf Indicator”(0x81) to the highest address WL of the sector in a fourth programming access (only necessary for first dataset per sector); this indicates that the sector is not empty
- (3) next data update: program the next 3 WLs in descending order (here WL 91-89)
- (4) repeat until sector is fully programmed (5x dataset can be programmed)
- (5) if sector is full continue with next sector and erase if first dataset after successful programming

Meaning of Indicator Byte	Position	Value	Interpretation
Conf & Prog	highest byte of a sector	0x00	sector is erased but not yet confirmed
		0x80	sector is erased & confirmed
		0x81	sector is programmed
		other	corrupted data → sector to be erased
Prog	highest byte of a WL	0x00	WL/dataset is not programmed
		0x81	dataset is programmed
		other	corrupted data → sector to be erased

Table 8 Meaning of the Indicator Byte for the Search Functions

The following gives a graphical overview of the emulation scenario for “Example B”. For simplification only one sector is shown. The details can be taken from an EXCEL spread-sheet “*emulation_exB.xls*”.

SEC3

0x80	indicator
DB29	old data
0x00	erased
DB29	actual data
0x00	not used

0xABFF				0xAB00				Wordline	
Sector	not used	not used	not used		not used	not used	not used	WL95	--> Info WL
DB31	DB30	DB29	DB28		DB2	DB1	DB0	WL94	Data Block 1
DB63	DB62	DB61	DB60		DB34	DB33	DB32	WL93	
Block	DB94	DB93	DB92		DB65	DB64	DB63	WL92	
DB31	DB30	DB29	DB28		DB2	DB1	DB0	WL91	Data Block 2
DB63	DB62	DB61	DB60		DB34	DB33	DB32	WL90	
Block	DB94	DB93	DB92		DB65	DB64	DB63	WL89	
DB31	DB30	DB29	DB28		DB2	DB1	DB0	WL88	Data Block 3
DB63	DB62	DB61	DB60		DB34	DB33	DB32	WL87	
Block	DB94	DB93	DB92		DB65	DB64	DB63	WL86	
DB31	DB30	DB29	DB28		DB2	DB1	DB0	WL85	Data Block 4
DB63	DB62	DB61	DB60		DB34	DB33	DB32	WL84	
Block	DB94	DB93	DB92		DB65	DB64	DB63	WL83	
DB31	DB30	DB29	DB28		DB2	DB1	DB0	WL82	Data Block 5
DB63	DB62	DB61	DB60		DB34	DB33	DB32	WL81	
Block	DB94	DB93	DB92		DB65	DB64	DB63	WL80	
0xAAFF				0xAA00					

Figure 9 Fully programmed sector

[illegible]

Figure 10 Erased and confirmed sector

[illegible]

Figure 11 First data-set update and writing sector indicator

0x81	0x00	0x00	0x00		0x00	0x00	0x00	WL95	--> Info WL
DB31	DB30	DB29	DB28		DB2	DB1	DB0	WL94	
DB63	DB62	DB61	DB60		DB34	DB33	DB32	WL93	Data Block 1
0x81	DB94	DB93	DB92		DB65	DB64	DB63	WL92	
DB31	DB30	DB29	DB28		DB2	DB1	DB0	1st WL	
DB63	DB62	DB61	DB60		DB34	DB33	DB32	2nd WL	Data Block 2
0x81	DB94	DB93	DB92		DB65	DB64	DB63	3rd WL	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL88	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL87	Data Block 3
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL86	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL85	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL84	Data Block 4
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL83	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL82	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL81	Data Block 5
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL80	

Figure 12 Second data-set update

0x81	0x00	0x00	0x00		0x00	0x00	0x00	WL95	--> Info WL
DB31	DB30	DB29	DB28		DB2	DB1	DB0	WL94	
DB63	DB62	DB61	DB60		DB34	DB33	DB32	WL93	Data Block 1
0x81	DB94	DB93	DB92		DB65	DB64	DB63	WL92	
DB31	DB30	DB29	DB28		DB2	DB1	DB0	1st WL	
DB63	DB62	DB61	DB60		DB34	DB33	DB32	2nd WL	Data Block 2
0x81	DB94	DB93	DB92		DB65	DB64	DB63	3rd WL	
DB31	DB30	DB29	DB28		DB2	DB1	DB0	1st WL	
DB63	DB62	DB61	DB60		DB34	DB33	DB32	2nd WL	Data Block 3
0x81	DB94	DB93	DB92		DB65	DB64	DB63	3rd WL	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL88	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL87	Data Block 4
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL86	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL85	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL84	Data Block 5
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL83	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL82	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL81	
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	WL80	

Figure 13 Third data-set update

4.4 Emulation Scenario – Example C

Idea	Cycle four WLs through four sectors in a round robin			
Used Sectors	Sec 9 – 6 (128 bytes (4 WL) each, total size 512 bytes)			
Data-set	4 WL → max. 127 Data bytes (+ Indicator Byte)			
IndicatorByte	shows the status of a Sector			
One Flash Cycle (for endurance calculation)	4x write (complete sector in 4 steps), 1x erase			
Data-set Update	Program	Erase	Aborted Erase	Note
Min. Time	4x	-	-	+ algorithm
Max. Time w/o Aborted Erase	5x (first data-set)	1x	-	+ algorithm
Max. Time w/ Aborted Erase	5x	1x	1x	+ algorithm

Table 9 Overview Example C

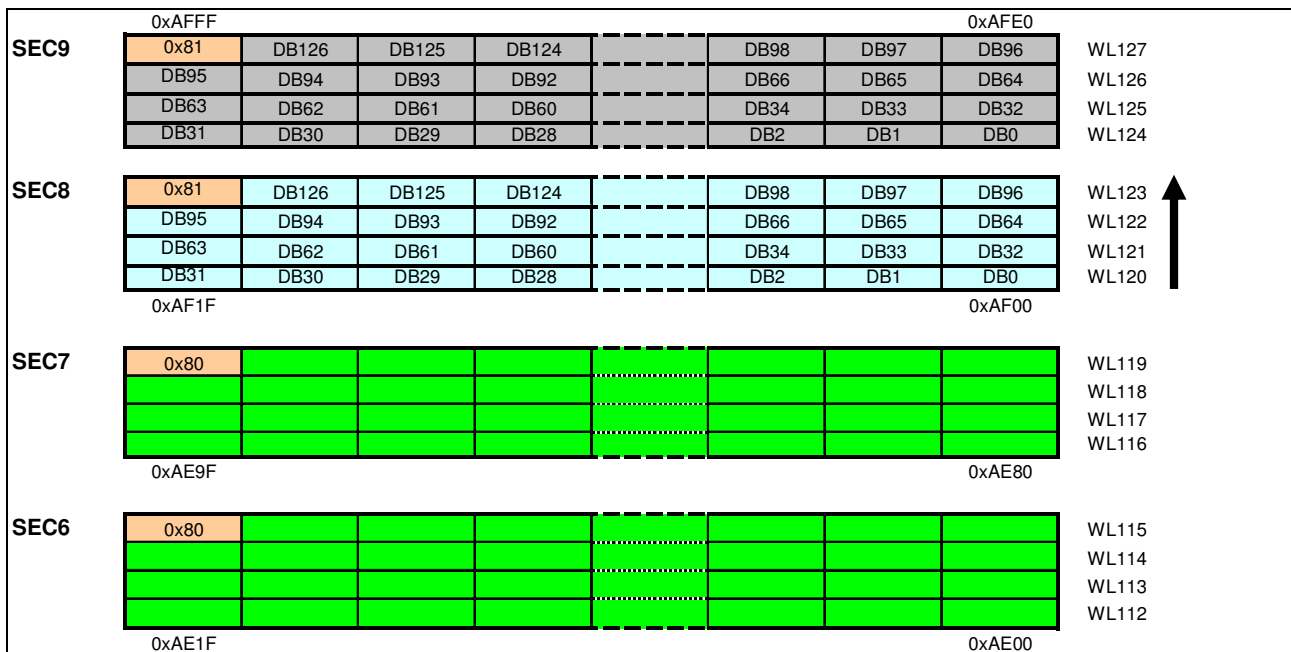
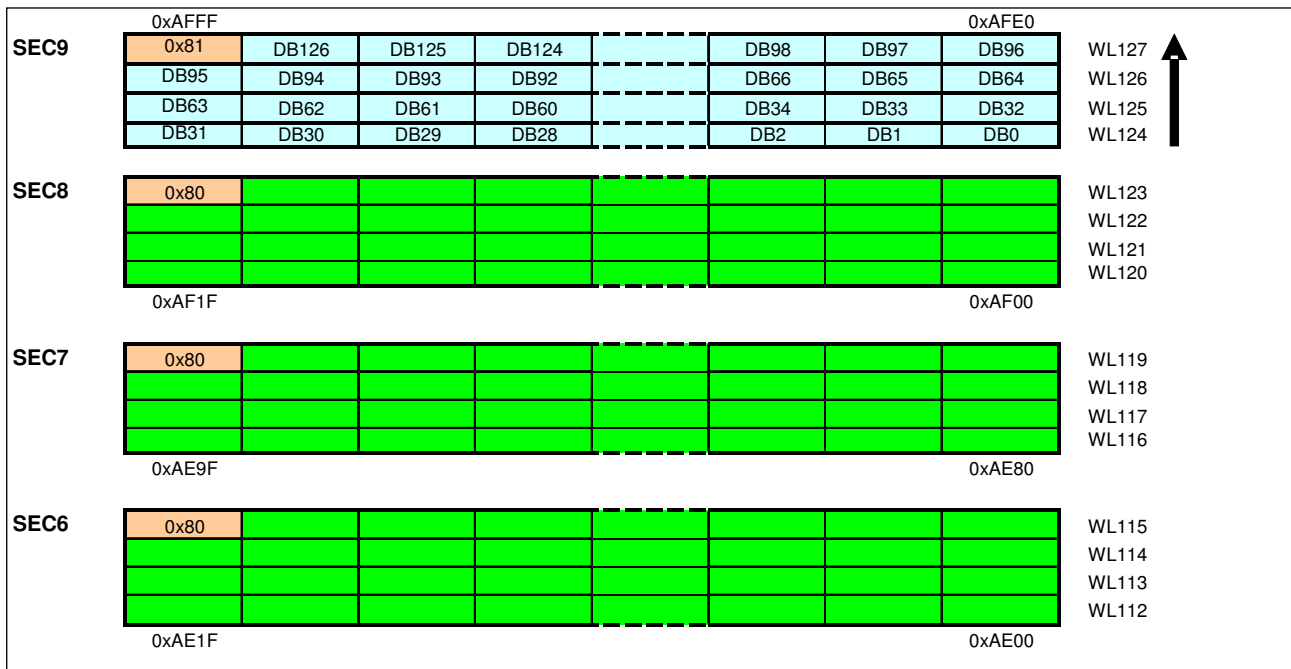
Program flow: assumption: used sectors are successfully erased & confirmed

- (1) program the four WLs of the first sector in ascending order (starting with lowest address) and write 0x81 to the highest sector address
- (2) second data update: program the next sector
- (3) third data update: program the next sector and erase & confirm the previous two sectors
- (4) repeat with (2)

Meaning of Indicator Byte	Position	Value	Interpretation
Conf & Prog	highest byte of a sector	0x00	sector is erased but not yet confirmed
		0x80	sector is erased & confirmed
		0x81	WL is programmed
		other	corrupted data → sector to be erased

Table 10 Meaning of the Indicator Byte for the Search Functions

The following gives a graphical overview of the emulation scenario for “Example C”. The details can be taken from an EXCEL spread-sheet “***emulation_exC.xls***”.



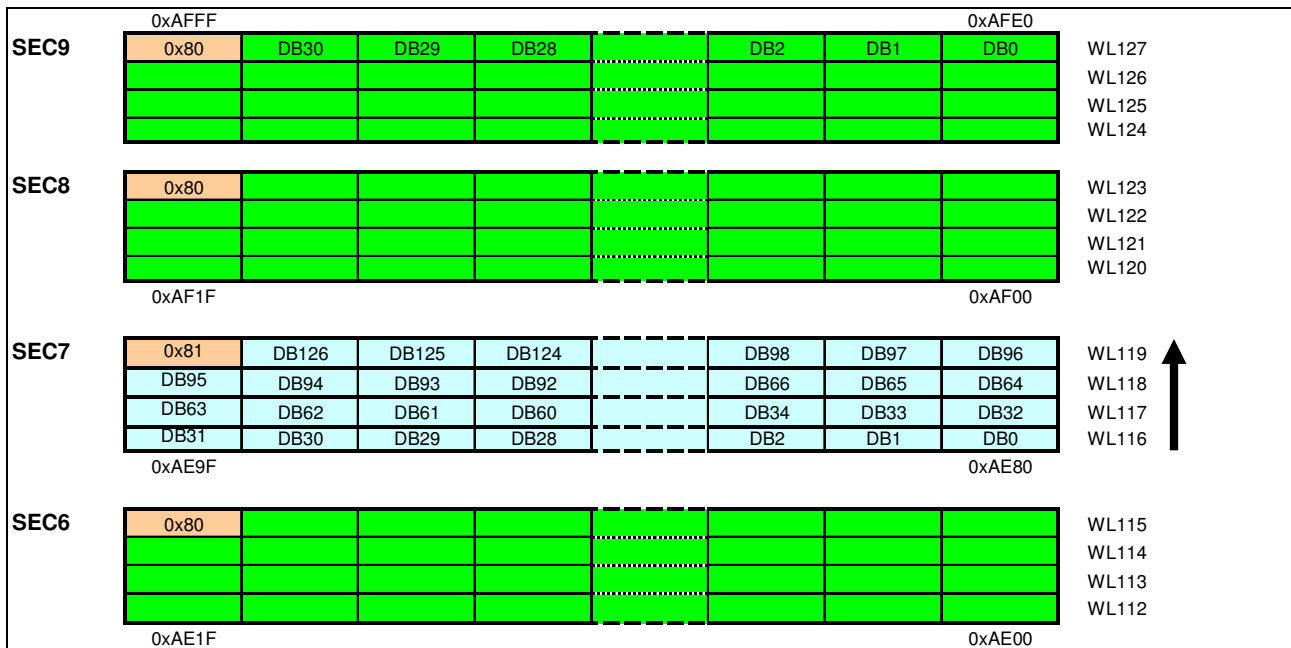


Figure 16 Third data-set update followed by an erase of the old two sectors

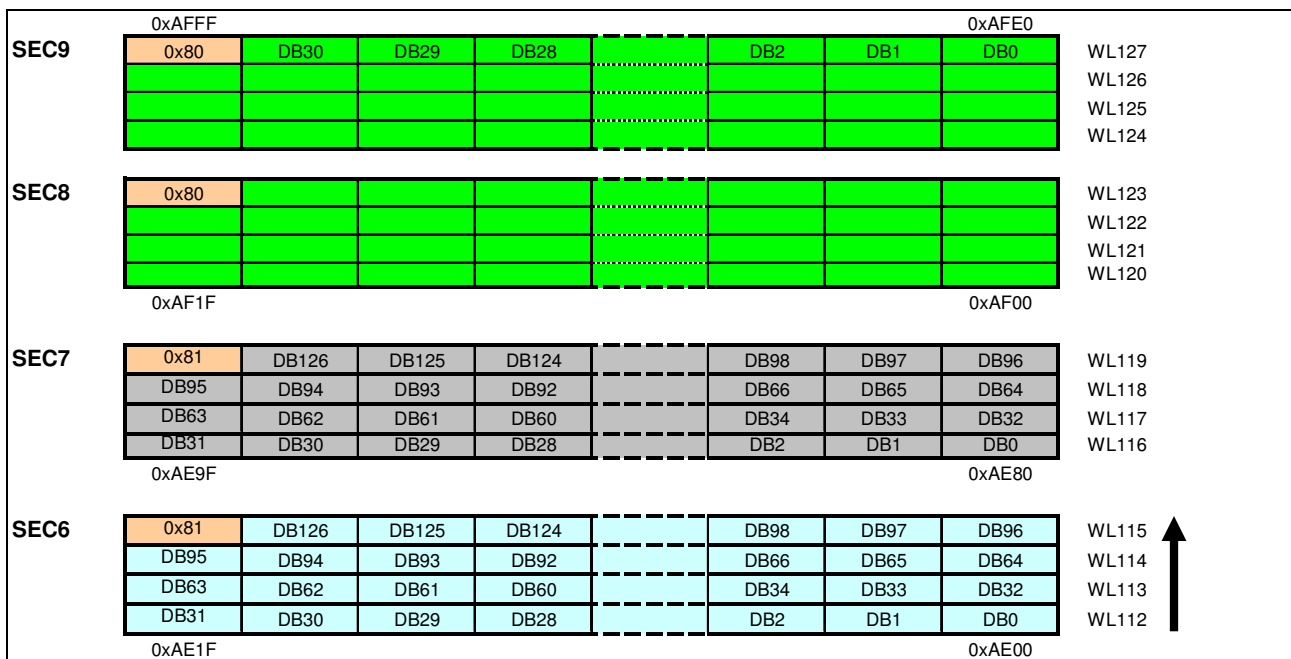


Figure 17 Fourth data-set update

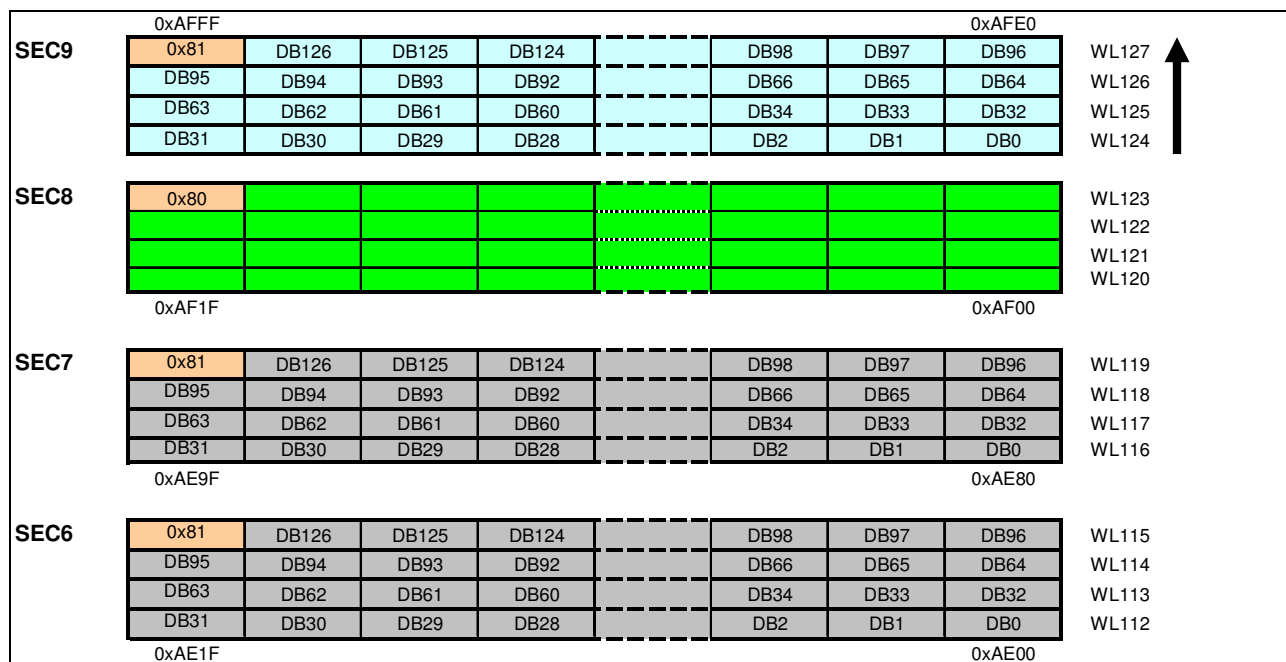


Figure 18 Fifth data-set update with round robin, SEC6 / 7 can be erased, SEC8 ready for programming

4.5 Application State Machine

The whole process is divided into a finite number of states. A diagram of this finite state machine (FSM) is shown below. The general procedure is as follows: after initialization of the device a search for the most recent actual data is performed and the corresponding variables are set accordingly, then the FSM is entered. As soon as a new programming request comes in ('Signal' 1→0→1) the FSM steps through the “**EEPROM Emulation States**”. The states IDLE, COUNTING, STOPPED, ERROR, POWERDOWN are standard “**User Application States**” and do not belong to the EEPROM emulation algorithm itself. Additional states can be easily added.

The FSM is implemented in the interrupt service routine of Timer 0. For all three examples the FSM is designed identically **only the state PROGRAMMING_DATA varies slightly**. In the following sections the common routines are described first then each example has its own chapter for its specific PROGRAMMING_DATA implementation. The states POWERDOWN and ERROR are not implemented.

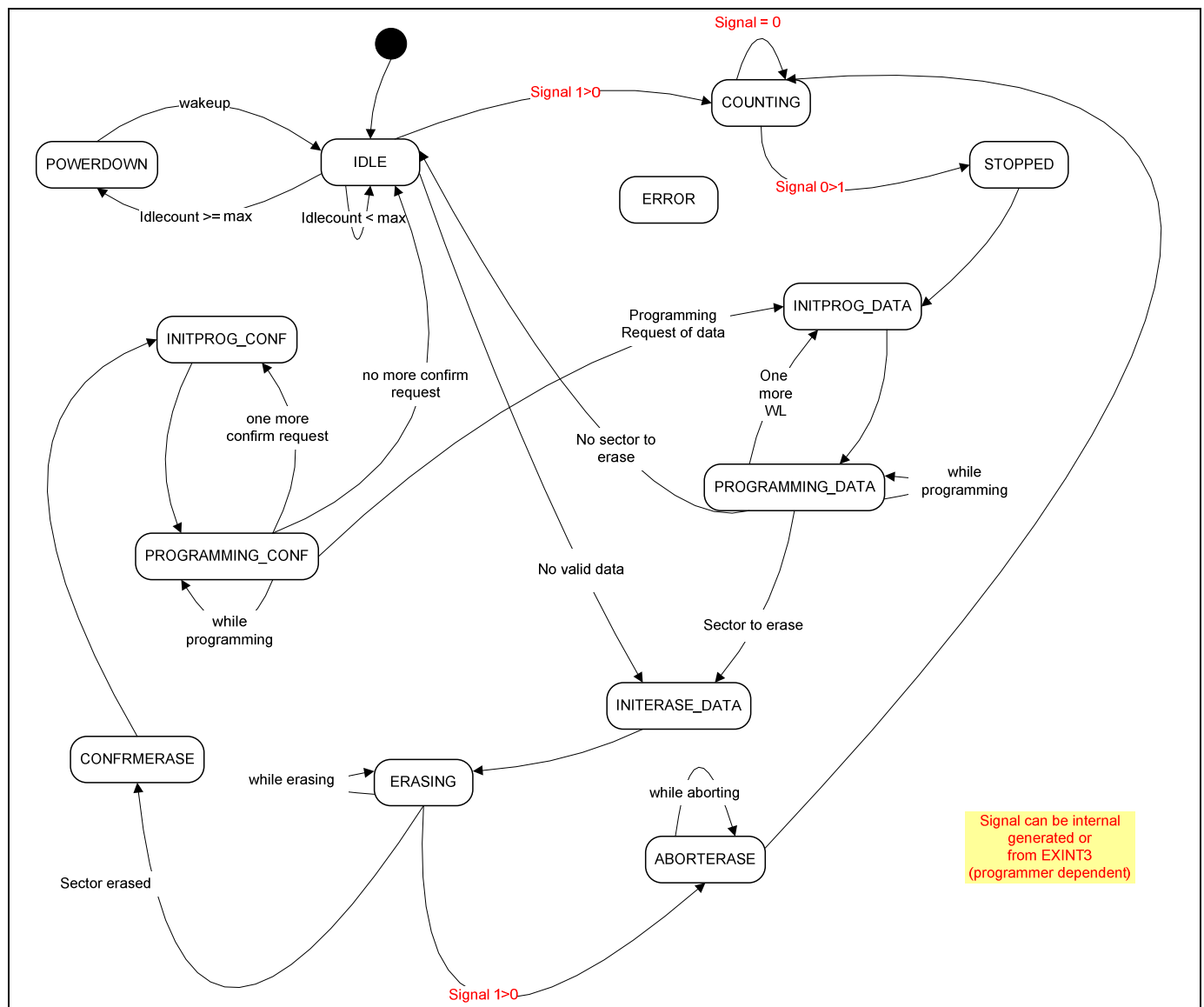


Figure 19 State Diagram of the “Operation Hour Counter” with EEPROM Emulation

4.6 Common Flowcharts of the Examples

4.6.1 Main Routine

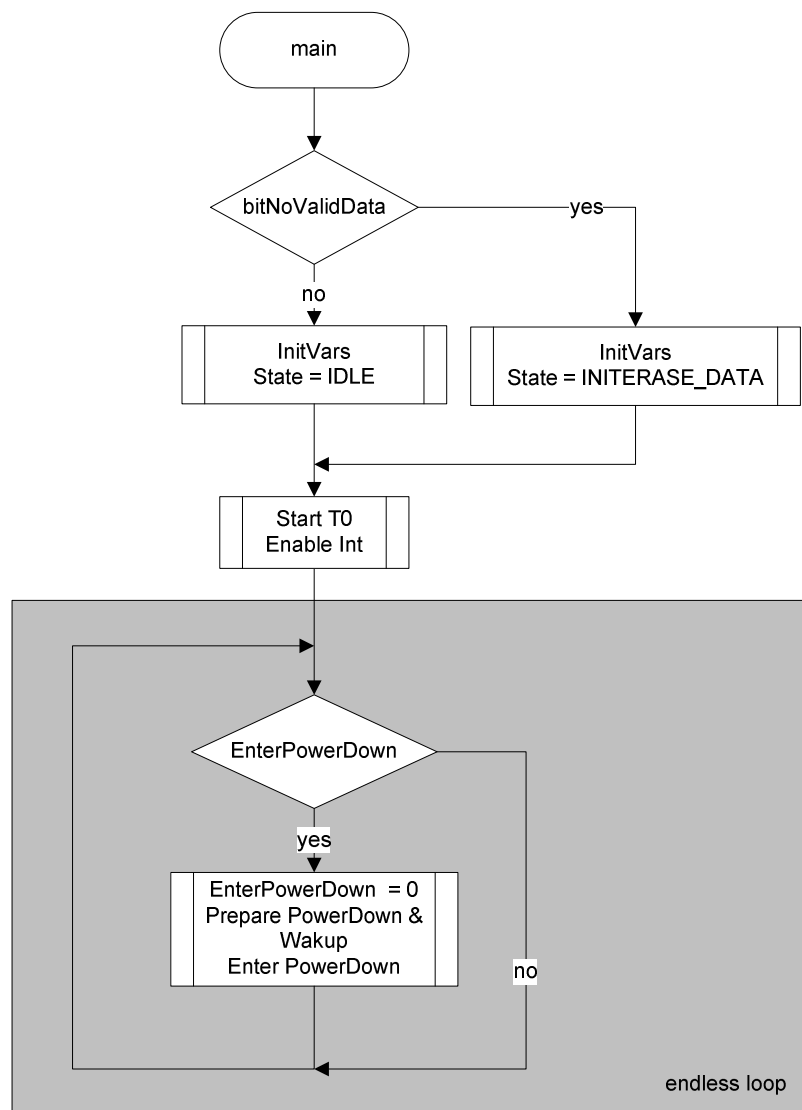


Figure 20 Endless loop for handling powerdown and wakeup

4.6.2 Interrupt Service Routine of Timer 0

4.6.2.1 Application Relevant States

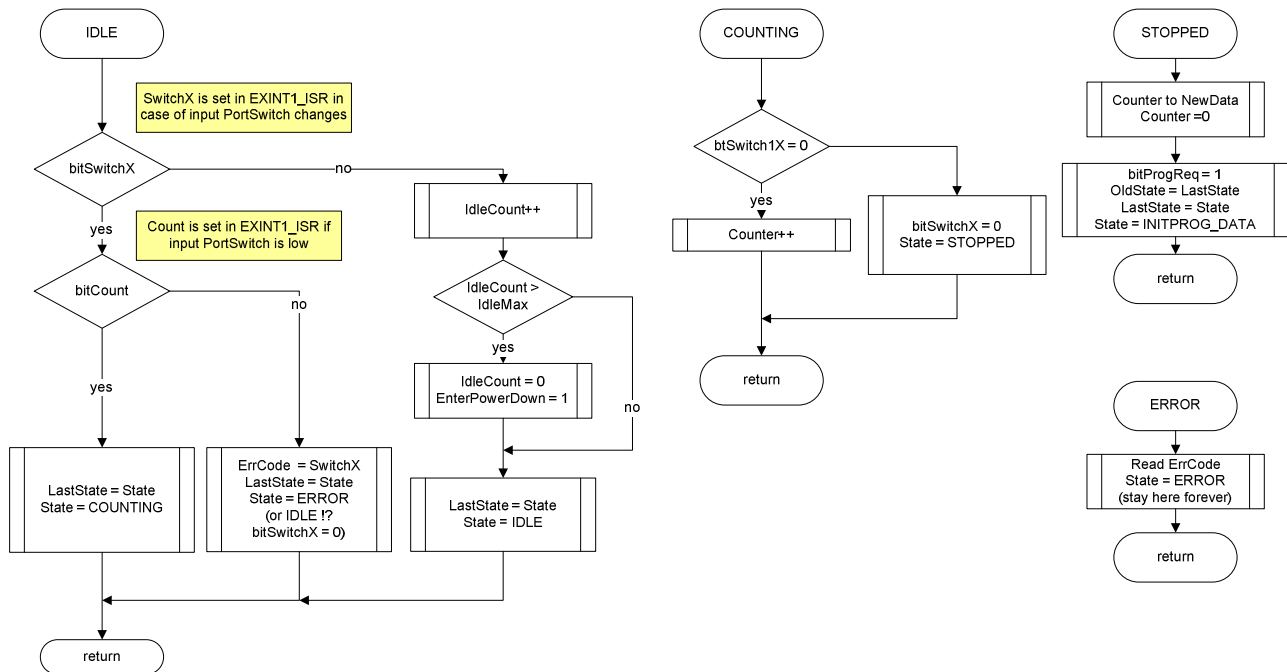


Figure 21 User Application States within T0 Interrupt Service Routine

4.6.2.2 States for Programming

Note: PROGRAMMING_DATA differs for each example

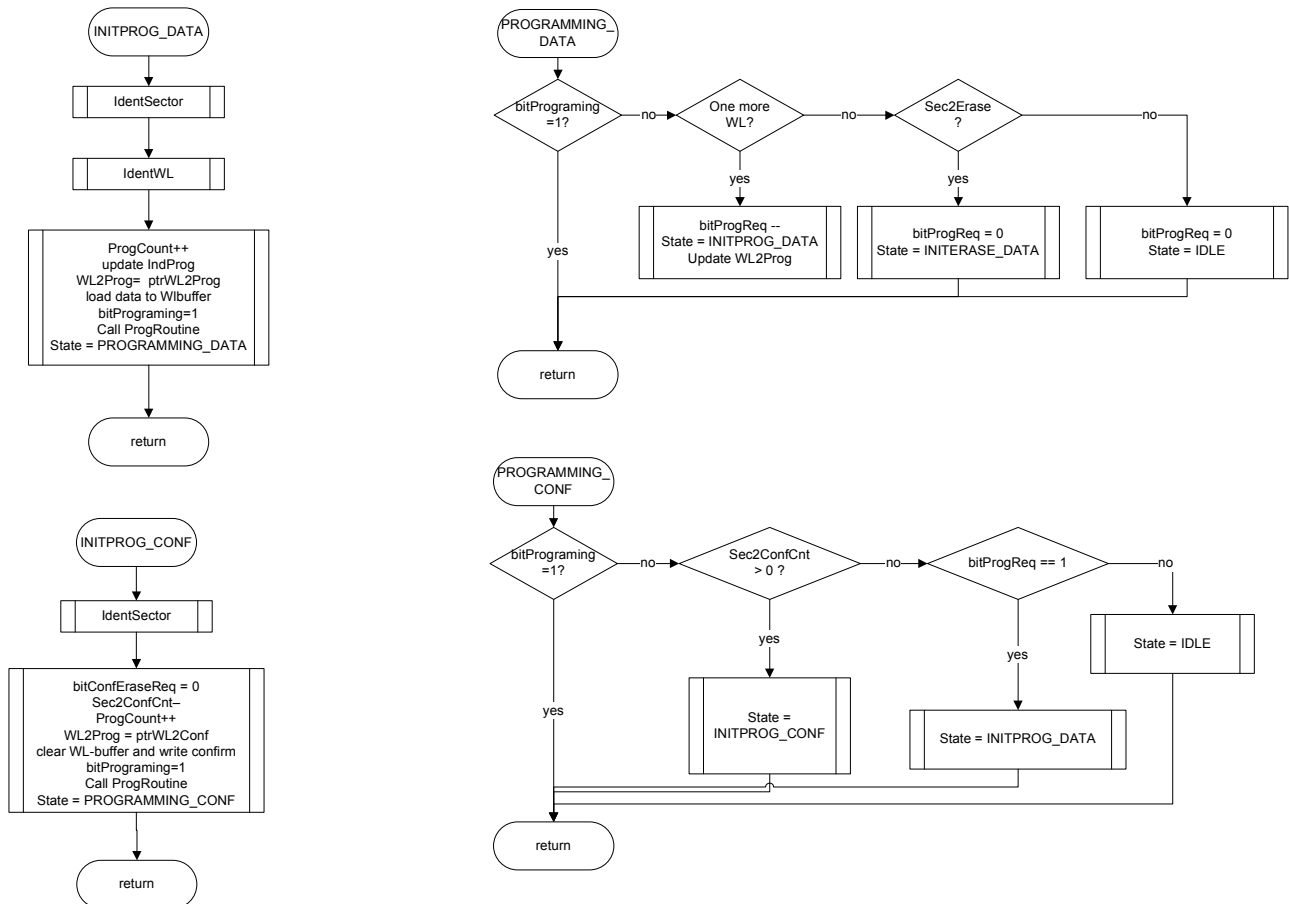


Figure 22 Programming States within T0 Interrupt Service Routine for Example C

4.6.2.3 States for Erasing

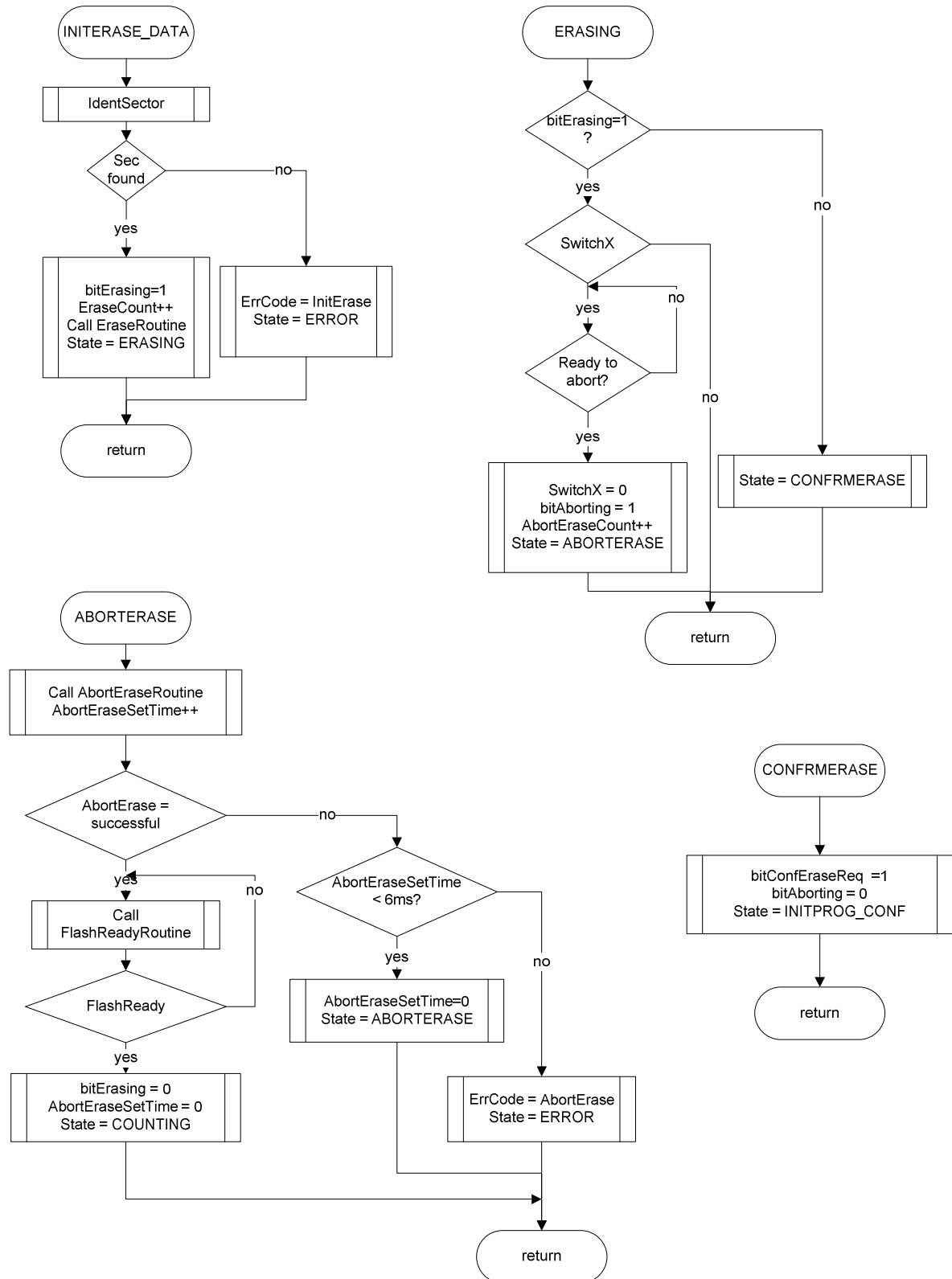


Figure 23 Erasing States within T0 Interrupt Service Routine

4.6.3 Non Maskable Interrupt Service Routine

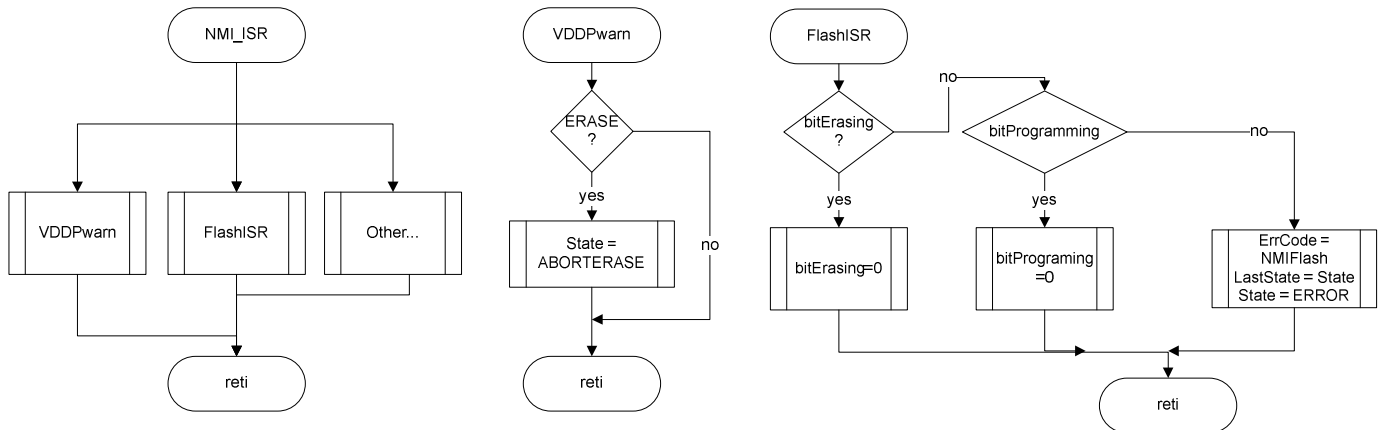


Figure 24 NMI Service Routine for Flash Timer and VDDP Prewarning

4.6.4 External Interrupt

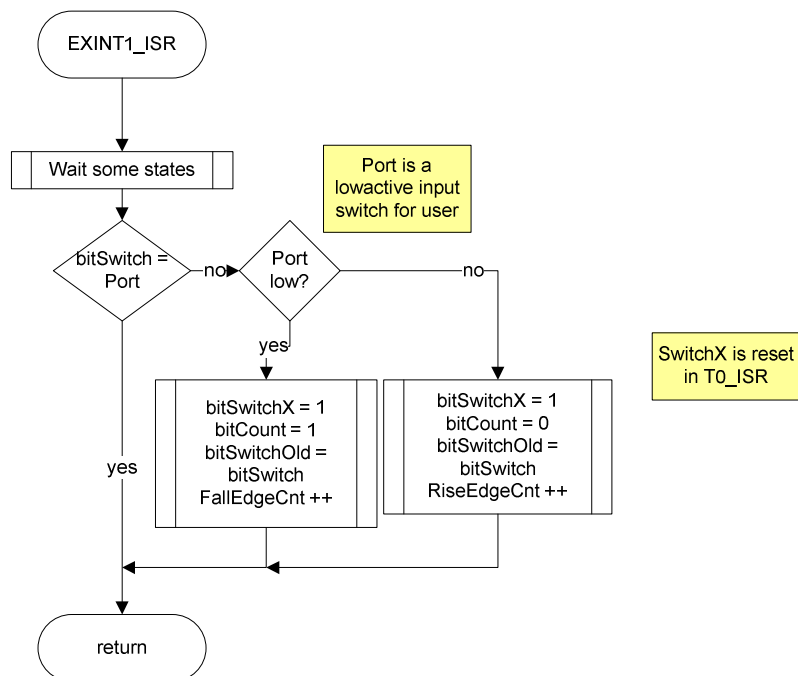
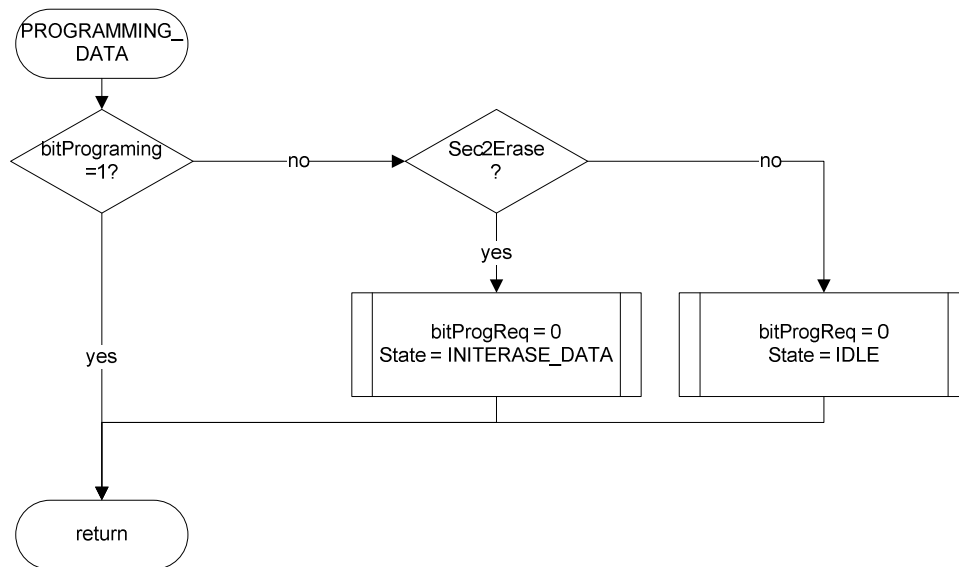


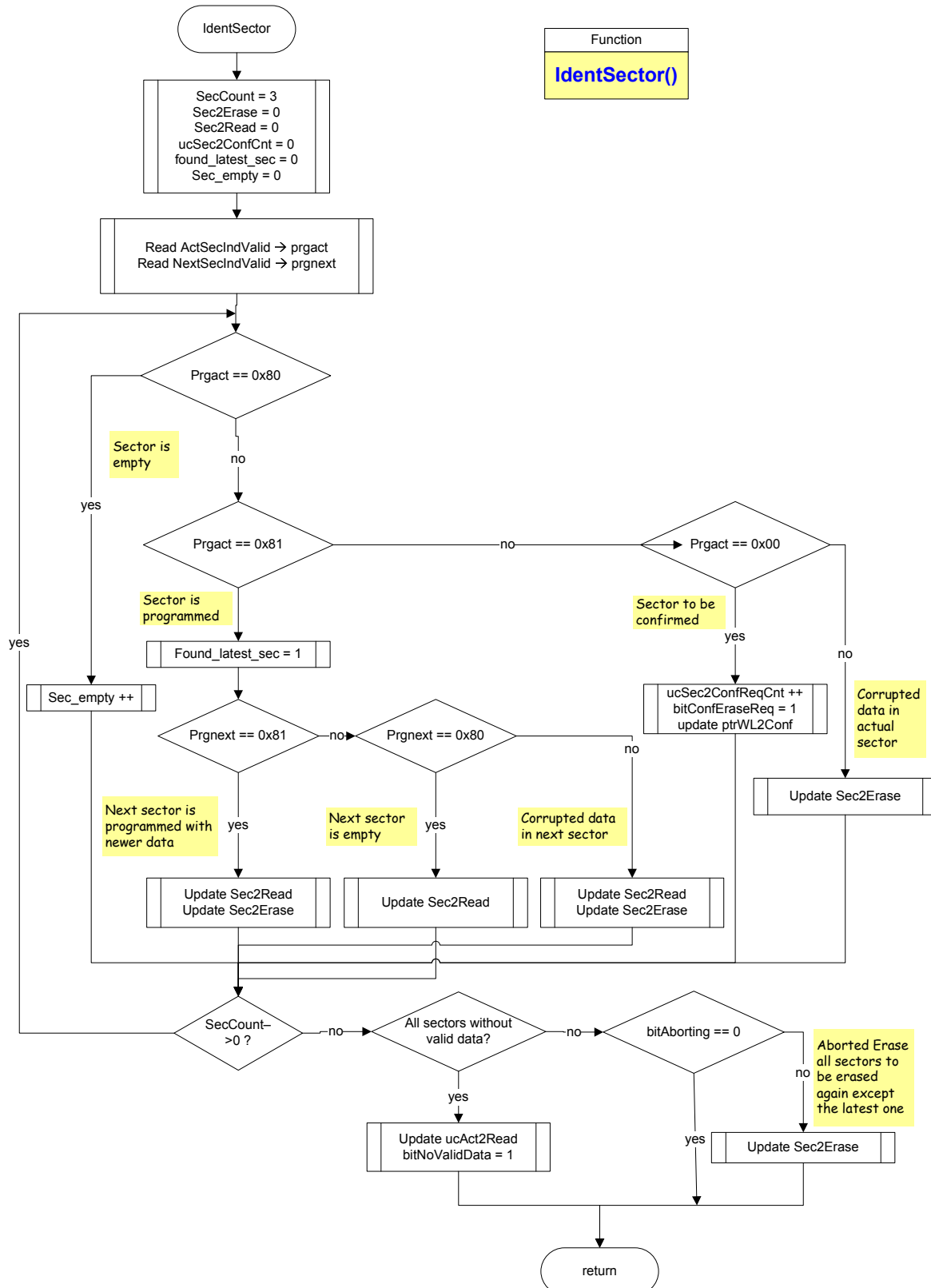
Figure 25 External signal as trigger event for dataset update

4.7 Individual Flowcharts for Example A

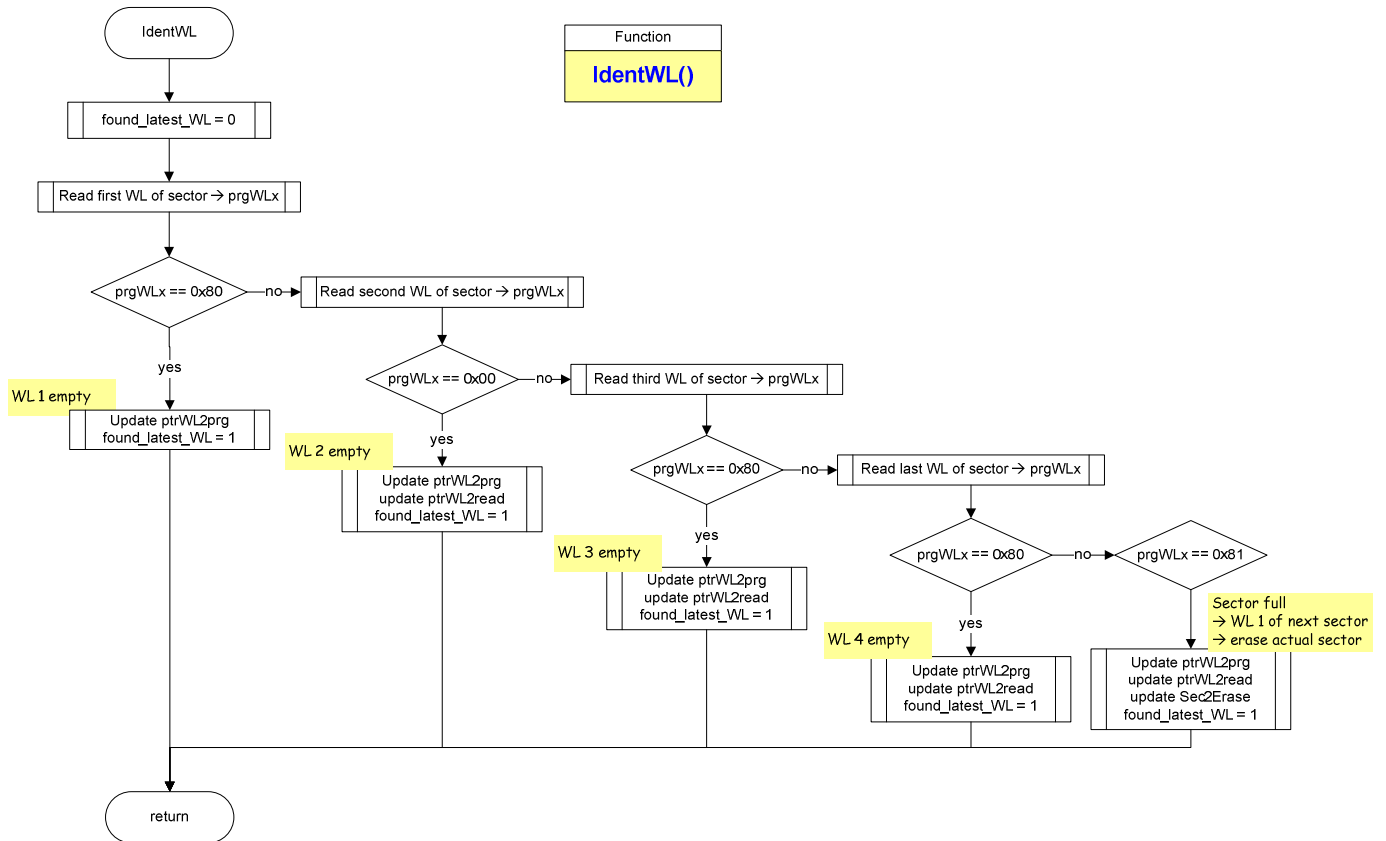
4.7.1 PROGRAMMING_DATA()



4.7.2 IdentSector()

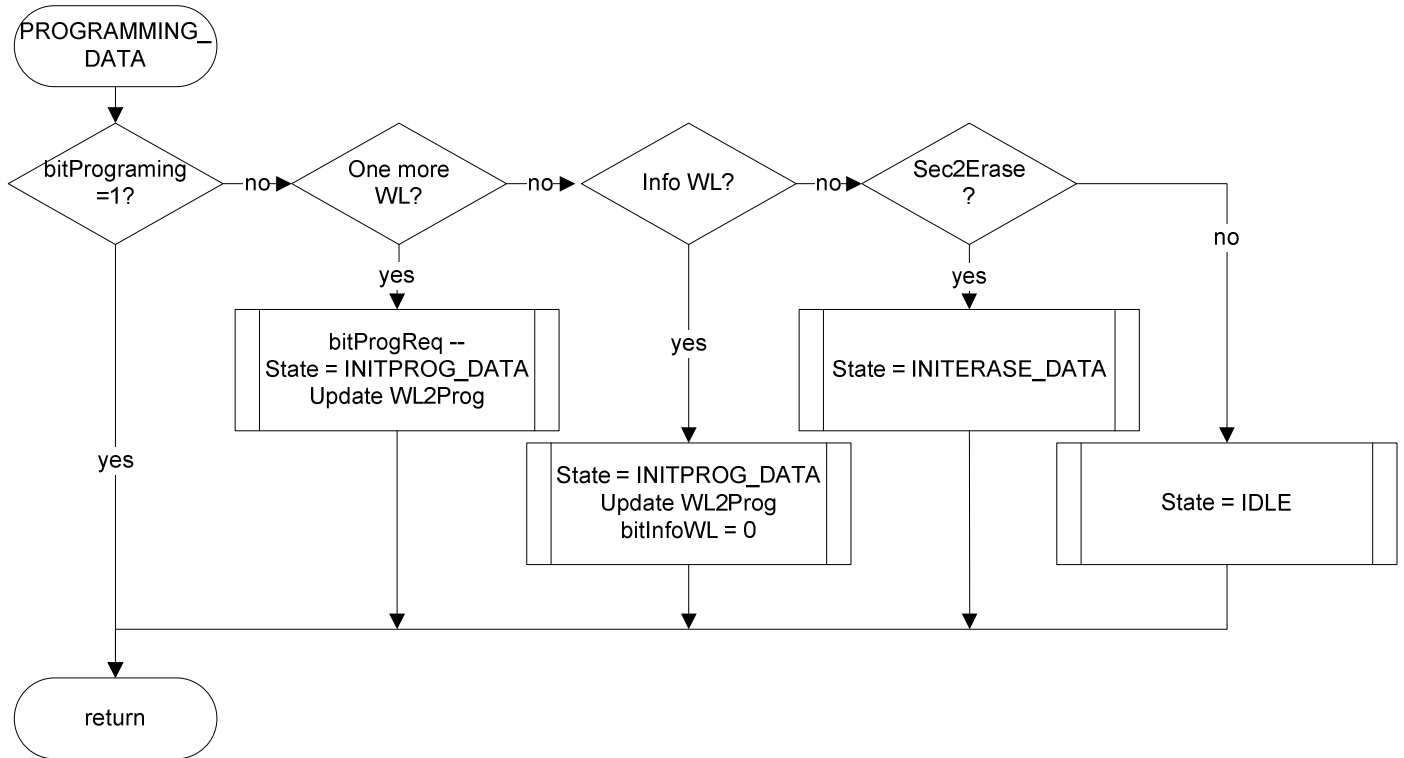


4.7.3 IdentWL()

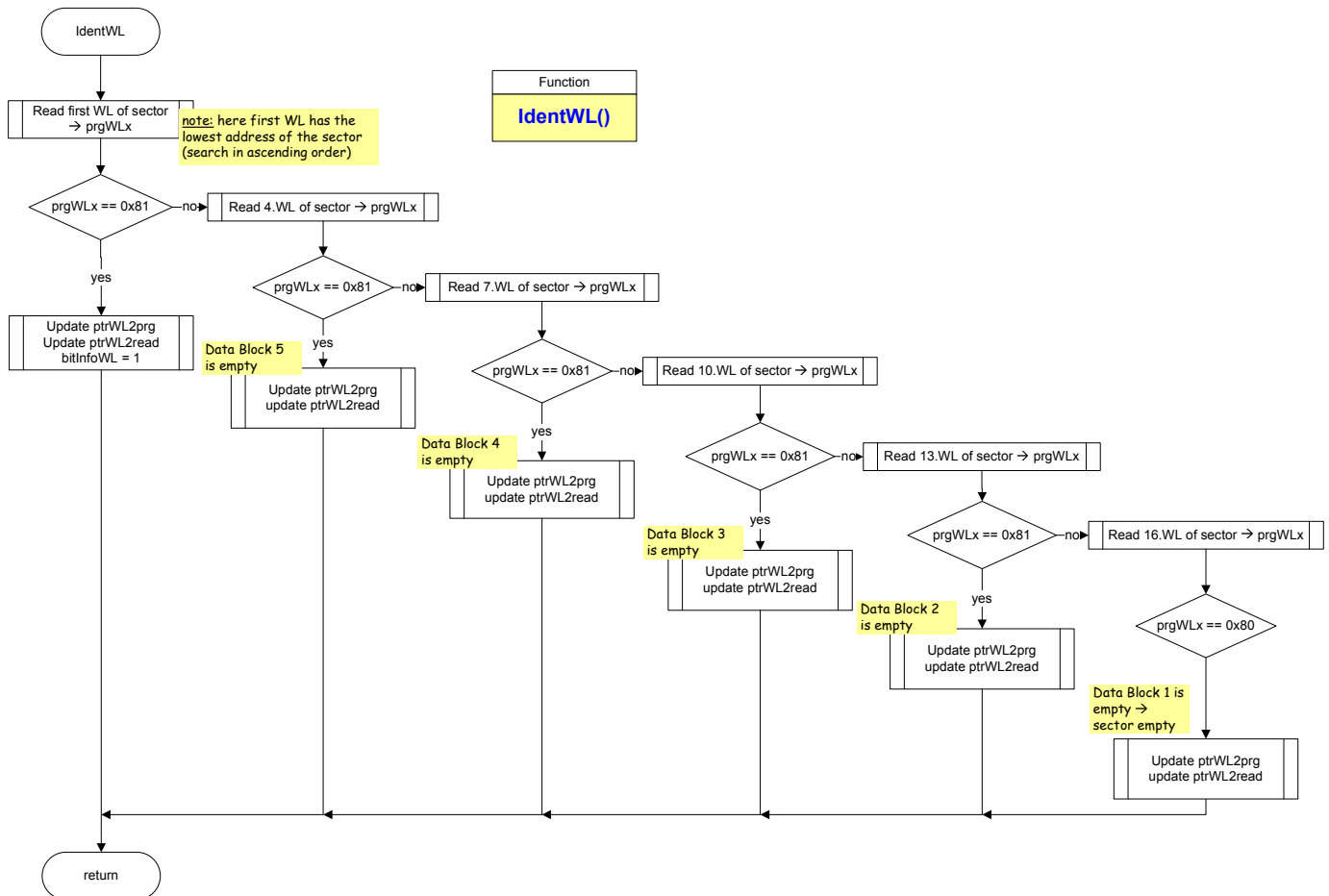


4.8 Individual Flowcharts for Example B

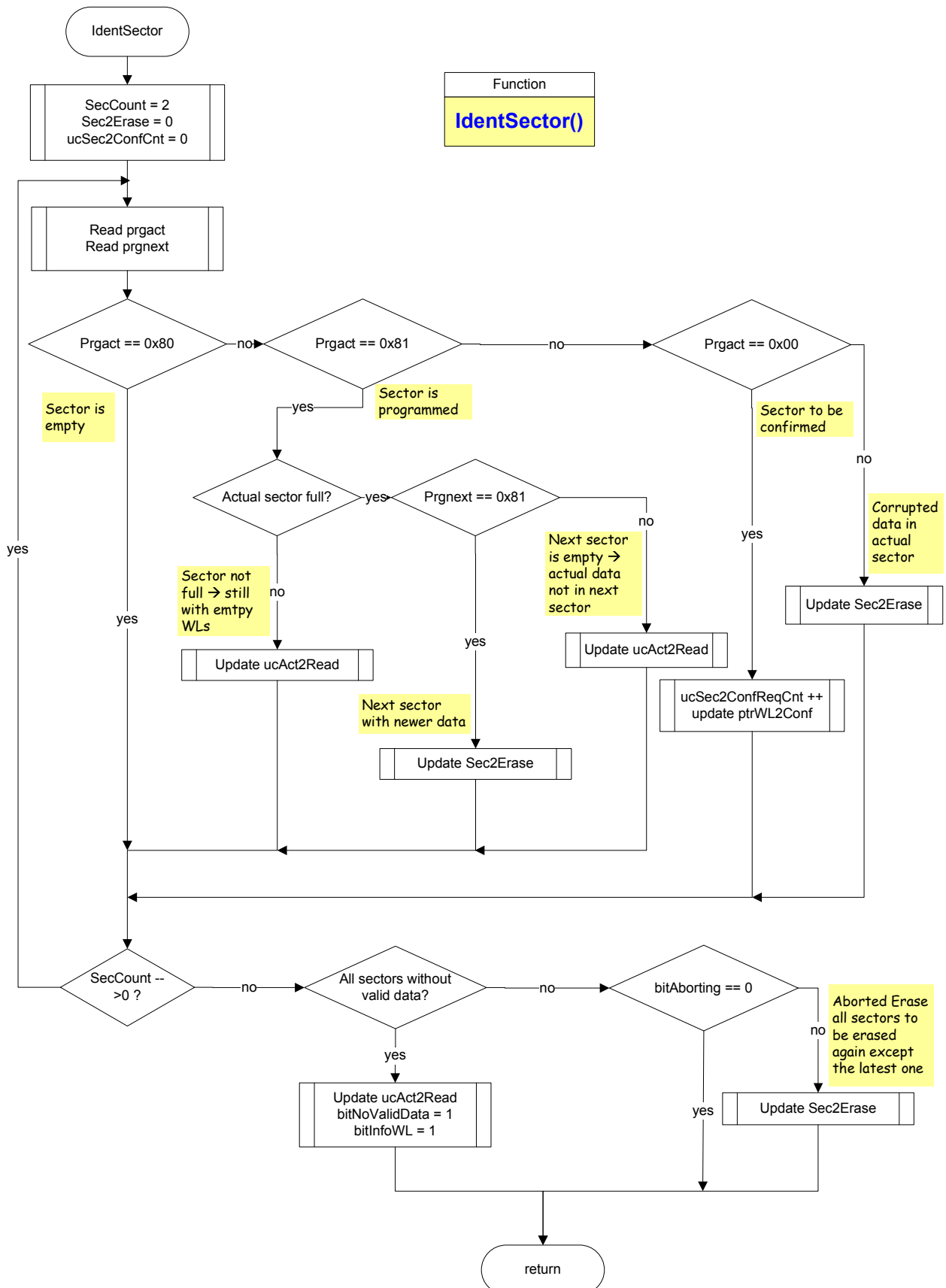
4.8.1 PROGRAMMING_DATA()



4.8.2 IdentWL()

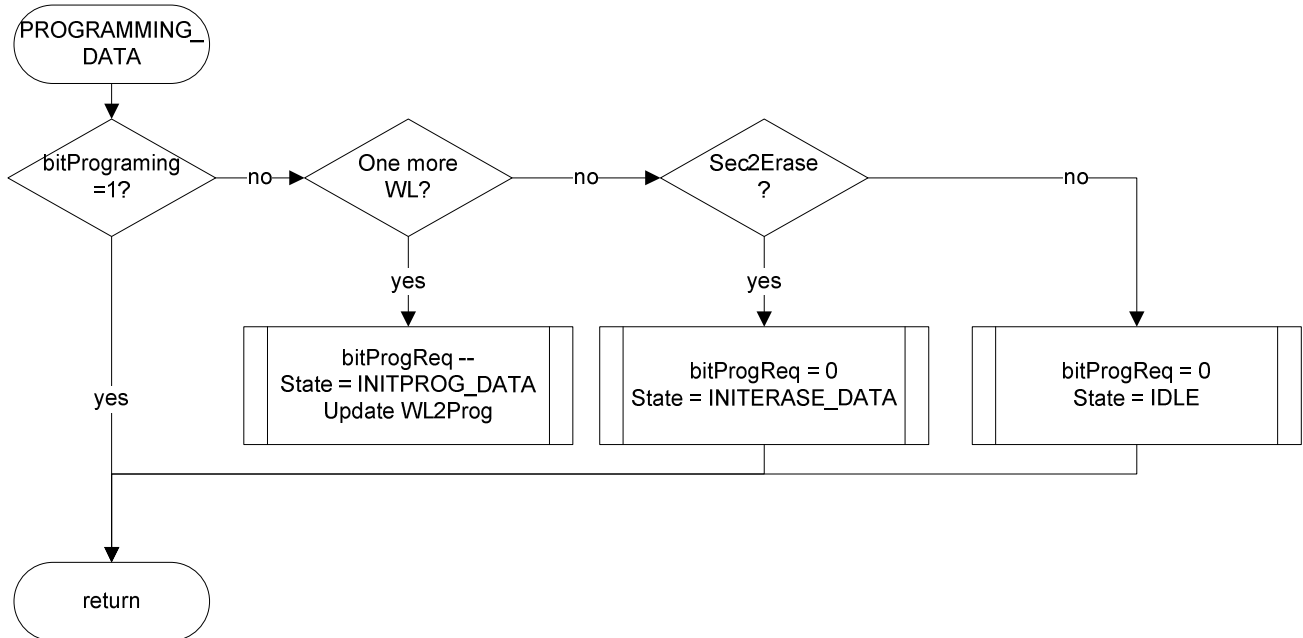


4.8.3 IdentSector()

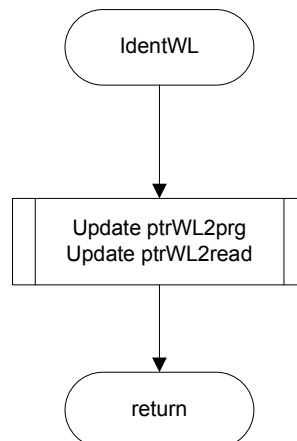


4.9 Individual Flowcharts for Example C

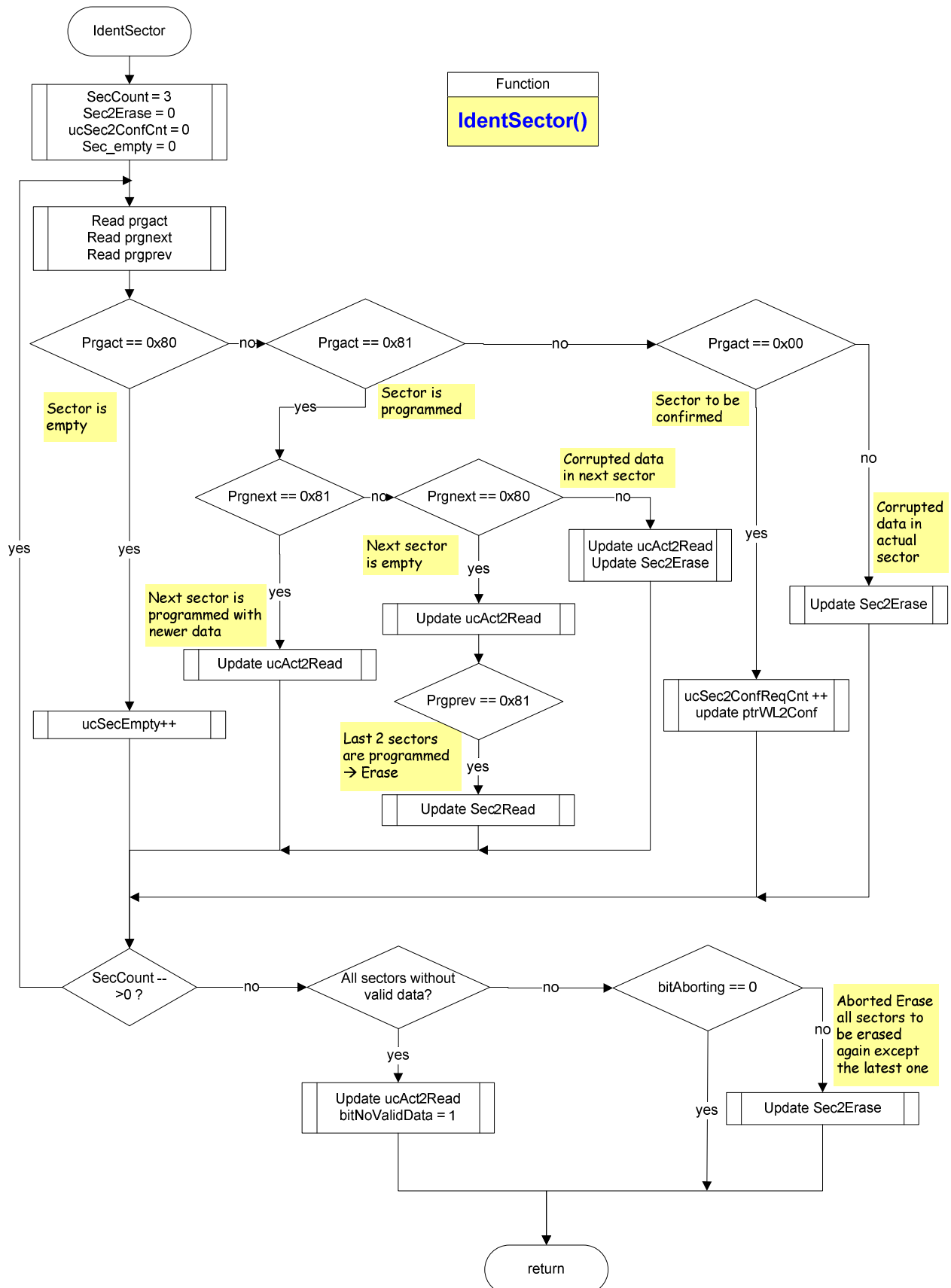
4.9.1 PROGRAMMING_DATA()



4.9.2 IdentWL()



4.9.3 IdentSector()



5 Testing and Debugging

Real-time Debugging is quite tricky especially if timers cannot be suspended (see chapter On-Chip Debug Support in the corresponding Users Manual, note: XC886/888 is more advanced than XC866). The on-chip debug support (OCDS) module is monitor based, i.e. a debug request forces the program counter (PC) to point to the BootROM and triggers an NMI interrupt. The BootROM routines handle the request and communicates via JTAG/UART to the debugger. All interrupts are disabled while executing the monitor routines. The CPU executes the debug monitor functions and debug data is handled in an extra monitor RAM.

The flash timer triggers an NMI interrupt as well and interferes with the OCDS. Therefore breaking (breakpoint or single step) an ongoing programming/erase steps causes problems. Usually the NMI-break is reached but the debug session crashes with the next session (see erratum OCDS_XC8.009 for XC866). For XC866 family a tricky workaround is that the NMI request flags have to be handled manually when debugger hits the break inside the NMI-ISR.

No problems occur when breaking inside the T0 interrupt states.

6 Glossary

AEC	Automotive Excellence Council
DFLASH	Data Flash
ppm	parts per million
EEPROM	Electrical Erasable and Programmable Read Only Memory
ECC	Error Correction Code
FSM	Finite State Machine
ISR	Interrupt Service Routine
NMI	Non Maskable Interrupt
NVM	Non-Volatile Memory
OCDS	On-chip Debug Support
PC	Program Counter
PFLASH	Program Flash
ppm	parts per million
RAC	Retention After Cycling
Sec / SEC	Sector
WL	Wordline

7 Sources and Links

Users Manual, Data Sheet & Errata Sheets XC866

Users Manual, Data Sheet & Errata Sheets XC886/888

ApHint_XC8_05_0001-0006 – FlashApHints on StarterKit CD XC866 and XC888, CD contents on www.infineon.com/cms/en/product/channel.html?channel=db3a304312dc768d0112e23122300536

www.infineon.com/XC866 → Services for Engineers

www.infineon.com/XC888 → Services for Engineers

<http://www.infineon.com>