ABB	**Software Design Description** NV-Handler APR/ IIMS Minden				**SDD**
Responsibility: **APR-MIMS**	Date: **See below**	Language: **en**	Filing system : **VSS**	Revision: **2.2**	Page: **1/47**
Issued by: Heiko Kresse	Approved:		Released:		Area of validity: DEAPR/M

Title: NV-Handler APR / IIMS Minden

References:

Distribution Software Archive

Author MIMS Heiko Kresse Date: 2009-04-21

Approved see Review Document

Remarks:

Contents

1 Introduction

This document describes the software design for the Non-Volatile-Storage-Handler developed at IIMS / APR Minden which is driven by the design idea of the “Common Framework”, thus it is compatible to all versions of the framework.

To handle different Hardware-situations the design consist of more then one class, all classes are collected in the nv-handler.

The software is designed in accordance to the idea of reuse – one nv-handler fulfilling most requirements of the different device families. It should be used without intensive modifications, moreover the design and documentation should allow developers to replace or add parts of the nv-handler in an easy way.

Important!

The nv-handler main objectives are

- Cope with a low power environment with less computing-power
- Handle EEPROM-Chips via a serial bus like I²C-Bus or SPI-Bus
- Suppressing error-sources, detecting multiple errors and repair single errors.
- Suppressing burst-problems is not part of this nv-software, is must be done by the physical-layer or the data-link-layer

A proper knowledge of I²C-Bus or SPI-Bus and EEPROM-chips is presumed for reading this document.

Please note also the related Requirement Specifications for Non-Volatile Memory Handling in the Common Framework Package.

This NV-Handler is tested for the data sheet environment shown in chapt. 2 – the module test cases, the test execution and the test results are documented within the source code package. Any changes on the environment conditions and NV Handler implementation shall be carefully validated by the modifier.

2 Data Sheet

This chapter gives an overview about all important facts of the subsystem. It can be used by a developer who would like to reuse this subsystem.

Category	*Item*	*Description*
Development	Version / Status	1.0 / implemented; module tested; lint level 3 free
	Known Bugs	None
	Planned Improvements	None
HW-Platform	Type	Independent
	Clocking	
SW-Development Environment	Compiler	Independent
	Operating System	Independent
	Code-Generation Tool	None
Required Re-sources	Operating System	The execute-method must be scheduled every 100ms. Call from low prior task (like idle task), because the execu-tion-time could be greater than 1s!
	HW	None
	RAM	(implementation related) + 30 * number of segments + 2*(number of sements+7)/8 + 90
	NVRAM	32 * (number of segments+1) * 2
	ROM	implementation related
	Execution Time	some seconds in background

	Special HW	I²C-Bus Multimaster, serial FRAM or EEPROM via I²C-Bus with 16-Bit addressing and pages of >=32byte Alternative SPI-Bus serial FRAM or EEPROM with 16-Bit addressing and pages >=32byte.
	Subsystems	
	Data Objects	
Standards	Safety	For this service-subsystem the following safety issues are required: - PCLint level 3 - ABB Coding Conventions for embedded software, V1.8 - Code Review
	Other	
Documentation	Requirements	Software Requirement Specification – Non-Volatile Memory Handling in the Common Framework
	Public Interface Description	nv_mem.h
	Test Specification	ModuleTest see code

3 Acronym and definitions

MiLe2	Minden & Lenno 2nd joined project
NV	Non Volatile
I²C	Inter Integrated Circuit
SPI	Serial Peripheral Interface
EEPROM	Electrically Erasable and Programmable Read Only Memory
FRAM	Ferroelectric RAM
HART	Highway Addressable Remote Transducer
FF	Fieldbus foundation
PA	Profibus PA
HMI	Human Machine Interface
NOVRAM	Non Volatile RAM
RAM	Random Access Memory
SCL	Serial Clock Line
SDA	Serial Data Line
ACK	Acknowledge
CRC	Cyclic redundancy check
MSB	Most Significant Bit

4 Basic Requirements

Basic requirements for transmitter and actuator/positioner are:

1. In spite of a bus clock less or equal than 2MHz, the application must not be influenced! Communication (HART, FF, PA) and HMI should not be influenced.
2. The application, communication (HART, FF, PA) and HMI should not wait longer than 5ms, regularly 2ms, for access to nv-data.
3. Single faults should be recognized and repaired. (Basic requirement)
4. Double (or even better multiple) errors should be recognized. (Basic requirement)
5. Consider the possibilities of the µCs internal FLASH.
6. The design must use information-hiding and strict separation of responsibilities. (Basic Software requirement)
7. It should be possible to define different areas, which are treated differently e.g. the data are stored in separate nv-chips. (Software Requirement Spec. Non Volatile Memory, also Framework description)
8. It must be possible to replace the EEPROM by FRAM. (Related to BUI)

5 Analysis

This chapter shows the error, timing and resource analysis and conclusions. The Design decisions for the Storage-Handler and for the Chip-Handler based on this analysis, so it is important to now them for design review.

5.1 Error Sources

Communication – single error

The exact error rate is unknown; we suppose it is much better than 1 error bit per 10000 transferred bit. Nevertheless an error could occur in every situation, a proper analysis must be done.

Communication – Burst

Our devices usually work in environments that are particularly noisy; here the worst case is a group of high frequency signals which is called “burst”. For the analysis the burst could be handled as many single-errors in a short time.

Reset

It is the main error source. During commissioning usually the supply is not guaranteed. During connecting the device the supply is active and this leads to repeated power on resets. A lousy connection could also lead to a power lost in the very moment the data are stored.

Defect RAM and NOVRAM

Very seldom, nevertheless it should be detected.

Corrupted Bits (RAM and NOVRAM)

Very seldom, nevertheless it should be detected.

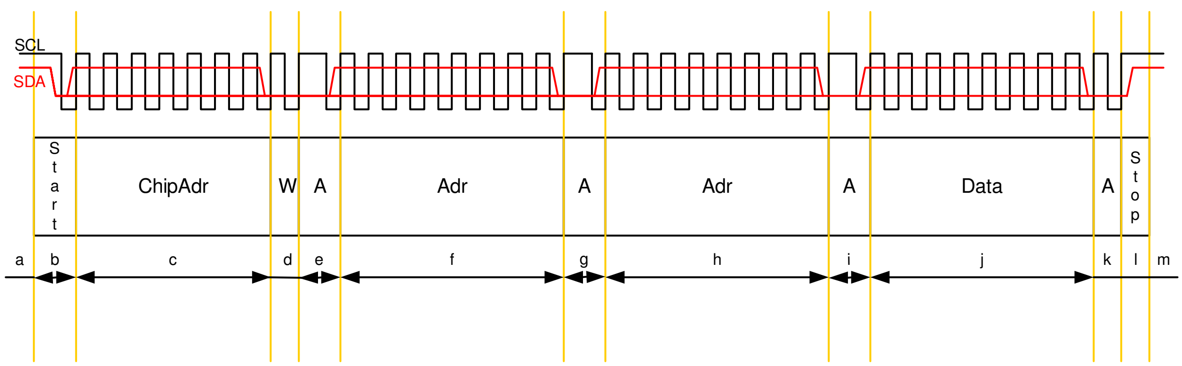
5.2 Protocol I2C-Bus

The analysis is done in context of the two used I²C-Bus block transfers. First the simple write-operation and second the combined-operation.

5.2.1 Write Operation

Next figure shows the bus signals during a write operation. SCL is the clock signal and SDA is the data-signal of the I²C-Bus. (Refer Philips I²C-Bus protocol description) The protocol is fragmented; each fragment has a char for identification and a short description what happens inside.

Figure 4.1



5.2.1.1 Communication Error

A single error on SDA or SCL leads one of three possible events:

- **A start condition is detected inside the fragment**
- **A stop condition is detected inside the fragment**
- **Wrong data**

Table 4.1

Fragment	Effect	Recognition / Repair
a, m	**No effect – each START resets an I²C-Bus-statemachine** **No effect – the following START override the STOP, respectively a STOP after a STOP is only a STOP** **No effect – will be ignored**	**-** **-** **-**
b	**-** **next byte without ACK** **next byte without ACK**	**-** **no ACK / retry** **no ACK / retry**

c	No ACK – restart I²C state machine No ACK – I²C state machine stops 1. no ACK 2. ACK from an other chip	No ACK / retry No ACK / retry 1. no ACK / retry 2. see “chip address error”
d	Like c Like c Read operation instead of write → no ACK; data will not be corrupted	No ACK / retry No ACK / retry No ACK / retry
e,g,i,k	Not possible as single error, because the slave pulls SDA down, a missing clock on SCL leads to an error in f,h,j,l	-
f,h	Like c Like c Writing to wrong address	No ACK / retry No ACK / retry See “address error”
j	Like c In case more than on byte are written, the STOP leads to no more ACK. The STOP starts the burning process, the last byte might be corrupted. Wrong data are written	No ACK / retry NO ACK / retry, consider burning time Verify / retry
l	Data will not be stored, without double error no other effect - Data will not be stored, without double error no other effect	Verify / retry - Verify / retry

Chip Address Error

In multiple nv-chip designs is it possible to write into the wrong chip. This case could not be recognized by the bus protocol. A byte-verify could detect this error, if the data to be written and the EERPOM data are different in one byte at least.

A repair includes two actions. First repeat the write operation, second repair in all other nv-chips the overwritten data. This is necessary, because except for two-chip designs it is impossible to determine the chip written into. For the second step it is necessary to know what must be stored at the corrupted location.

Proposal:

Use separate busses for each nv-chip or use the nv-chip write-protection. The write-protection must protect the whole address range.

Address Error

This case could not be recognized by the bus protocol. A byte-verify could detect this error, if the data to be written and the EERPOM data are different in one byte at least.

A repair includes two actions. First repeat the write operation, second repair in all other nv-chips the overwritten data. The problem is to determine the location of overwritten data, because it is not possible to get the write pointer.

Proposal:

- A segmentation of the nv-memory into segments of 32 bytes (write buffer length=32)
- Each segment has a 16-Bit CRC

  With the CRC it should be possible to find every corrupted segment, except the incorrect address is 32-byte-aligned.

  → The complete segment address (nv-chip +first byte) is included in the CRC calculation. Considering the address guarantee different CRC for same data at different location.
- A backup of every segment is necessary to repair the corrupted segments.

  An address error leads normally to two corrupted segments, which are directly neighbors. If the direct neighbor is the backup segment, the segments could not be repaired.
- Beware of direct neighborhood of a segment and its backup segment.
- After recognition of an address error, the corrupted segments should be found and repaired immediately.

5.2.1.2 Burst

The 16bit CRC is a very good protection against burst-errors but not a perfect one. It is possible that data corruption will not be recognized. A byte-verify will reduce this chance to nearly zero%.

5.2.1.3 Reset Error

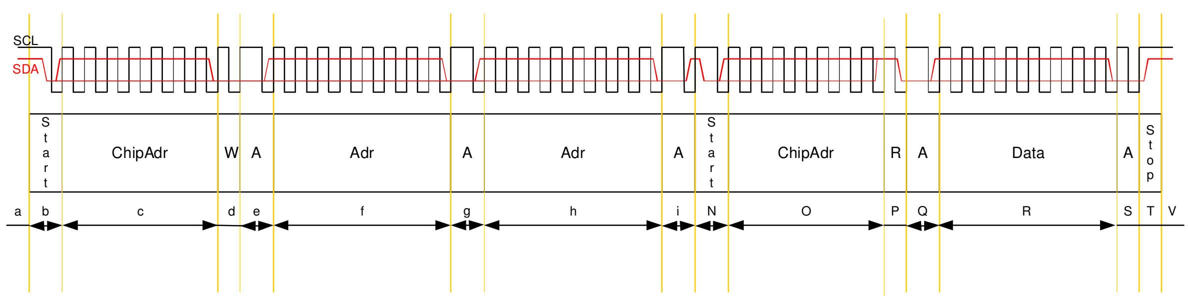
Table 4.2

Fragment	Effect	Recognition / Repair
a,b,c,d,f,h,j,l,m	The START condition of the first frame after a reset stops the write operation. An EEPROM will not burn the received data. A FRAM will have incomplete written data	Incomplete data will be found by segmentation + CRC / repair with backup Without the knowledge of the last state before reset, it is necessary to check all segments.
E,g,i,k	The START condition of the first frame after a reset will be overwritten by the ACK of the SLAVE, thus at least one byte will be overwritten (i, k only).	Same as before. Better solution: Send two start conditions after Reset. The additional start condition generates a CLK that end the ACK of the slave.

5.2.2 Read Operation

Next figure shows the bus signals during a read operation. SCL is the clock signal and SDA is the data-signal of the I²C-Bus. (Refer Philips I²C-Bus protocol description) The protocol is fragmented; each fragment has a char for identification and a short description what happens inside.

Figure 4.2



5.2.2.1 Communication Error

A single error on SDA or SCL leads one of three possible events:

- **A start condition is detected inside the fragment**
- **A stop condition is detected inside the fragment**
- **Wrong data**

Table 4.3

Fragment	Effect	Recognition / Repair
a-i	The read operation is exactly the same as write operation until fragment N.	
N	- **Without a new START condition the following data will be ignored.** **Missing the START condition overwrite two segments, because 33 byte (0xFF) will be written. The chip address is the first byte, the master generate the CLK for the next 32 Byte.**	- **CRC-error / retry** **CRC-error, all received bytes are 0xFF/ repair the active segment with backup. (And the next one; for Chips with only 32-byte write buffer only one segment is damaged.)**
O	**No ACK** **No ACK** **No ACK or data from wrong nv-chip**	**No ACK / retry** **No ACK / retry** **No ACK or CRC / retry**
P	**No ACK** **No ACK** **The actual segment will be overwritten with FF**	**No ACK / retry** **No ACK / retry** **CRC-error, all received bytes are 0xFF / repair the active segment with backup. (**
Q,S	**Reset all I²C-slaves, Master generates CLK, FF is not a possible nv-chip address, so nothing happens** **Without new START nothing happens.** **NO ACK**	**CRC-error / retry** **CRC-error / retry** **No ACK / retry**
R	**Reset all I²C-slaves, Master generates CLK, FF is not a possible nv-chip address, so nothing happens** **Without new START nothing happens.** **CRC error**	**CRC-error / retry** **CRC-error / retry** **CRC-error / retry**
T, V	**Nothing happens** **Nothing happens** **START of the next frame stops slaves reading mode.**	**-** **-** **-**

5.2.2.2 Burst

The 16bit CRC is a very good protection against burst-errors but not a perfect one. It is possible that data corruption will not be recognized.
An error during “N” followed by a errors on the data-line will damage one or two segments without the chance to recognize it, because the data are not completely 0xFF!
For single-bit errors it is a proper design to repeat the read-operation. But not for burst-errors, with every repeat the chance to damage the data in the nonvolatile storage increases. Unfortunately there is not a real chance to distinguish between a single-bit and a burst-error only by CRC.

Protection against burst is the responsibility of the hardware in case of I²C-bus. The software is not able to protect the nv-storage!

5.2.2.3 Reset Error

Table 4.4

Fragment	Effect	Recognition / Repair
a,b,c,d,f,h	The START condition of the first frame after a reset stops the write operation.	No data was written → no error happens.
E,g,i	The START condition of the first frame after a reset will be overwritten by the ACK of the SLAVE, thus at least one byte will be overwritten (i only).	Send two start conditions after Reset. The additional start condition generates a CLK that end the ACK of the slave.
S,T,V	No effect for single error	-
N,O,P	No effect because START resets the I²C-state machine	-
Q	The START condition of the first frame after a reset will be overwritten by the ACK of the SLAVE, thus at least one byte will be overwritten.	Send two start conditions after Reset. The additional start condition generates a CLK that end the ACK of the slave.
R	The slave is able to pull down SDA for 8 cycles START conditions will be overwritten.	Send 9 start conditions after Reset.

5.2.3 Conclusion

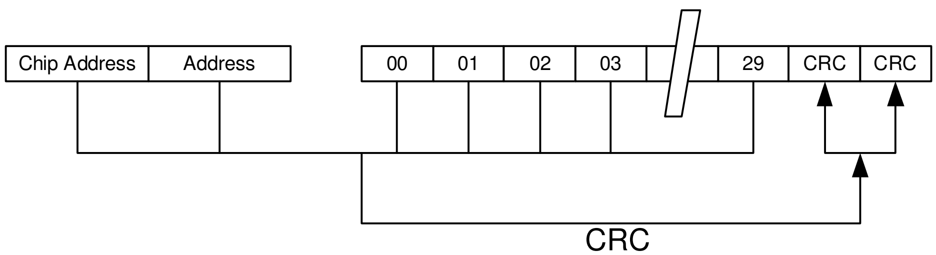
1. Retry read or write operation if the slave acknowledge is missing
2. The nv-memory is fragmented into 32-byte segments. A read or write operation access always a complete segment. (For EEPROM with 64-byte write buffers or FRAM, 64-byte segments are recommended.)
3. Each segment has a 2-byte CRC and 30-byte data
4. The CRC goes over the complete segment address + the 30 data bytes.
5. Read operations are validated by CRC
6. Write operations are validated by a complete (data + CRC) byte verify.
7. Write a segment only if at least one byte is different to the data inside the nv-memory. Only this guarantees that an address error will be recognized by byte verify.
8. In case of an address error all segments must be validated by their CRC
9. Each segment has a backup segment, so it could be repaired in any case.
10. Beware of direct neighborhood of segment and its backup; because address errors usually overwrite two neighboring segments.
11. After recognition an address error, the corrupted segments should be found and repaired immediately.
12. Protection against burst is the responsibility of the hardware.

5.3 Segmentation

Chapter 4.2 has as one conclusion the requirement to organize the nv-storage into segments. This chapter analyzes if the segmentation solve all problems or if new problems pop up.

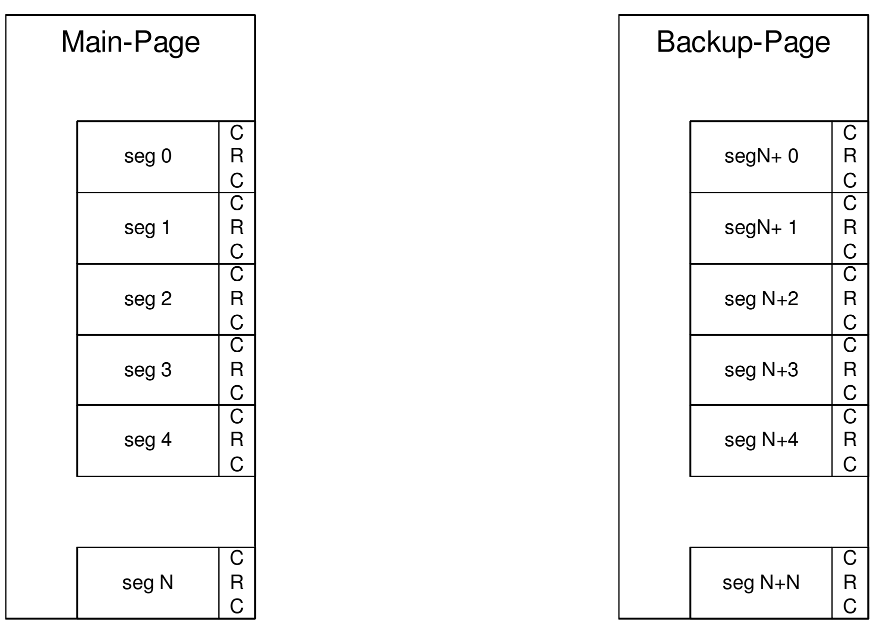
The nv-memory is fragmented into 32-byte segments. A read or write operation access always a complete segment. And each segment has a backup segment, so it could be repaired in any case.

Figure 4.3



The sum of all segments is called the Main-Page; the sum of their backup segments is called the Backup-Page.

Figure 4.4



5.3.1 Communication Errors

Data Error

Recognized by verify. It will be repaired be rewriting the segment.

Address Error

In most address-error situations it is not possible to determine which segments are corrupted before the whole storage is analyzed. Thus it is possible that both pages are corrupted (e.g. single-chip-design). This situation is very dangerous because a power down could damage the device. Obviously the time both pages are potentially damaged must be as short as possible.

The fastest action is to stop the actual operation and to store the complete main-page. That is possible, because this error could only happen after the nv-data are loaded into ram.

5.3.2 Reset Error

A reset during one segment write will be recognized by a CRC-error. The page the corrupted segment belongs to must completely repaired. That means copy the faultless page into the damaged page.

Comparing both pages will recognize a reset between writing main-page and writing backup-page. If both OK but different, the main-page must be copied into the backup-page.

A reset between two segments of the main-page is a problem. After the reset both pages are OK but different, and the not consistent main-page will be copied into backup. To solve this problem each page must have a non-volatile valid-flag. Writing a page starts with clearing the valid flag. If the page is stored without an error the valid flag will be set. After a reset the valid-flags will be loaded. If one page is not valid it will be overwritten with the page valid.

5.3.3 Data Consistence

Information hiding and separation of responsibility, here the separation of data handling and nv-memory-handling, have many advantages but there is at least one problem:

The nv-handler does not know anything about the data structures to be stored.

That means single data or structures are spread over more than one segment. For some circumstances it is not good or forbidden, to repair only single segments. The nv-handler is not able to decide if it is allowed to repair single segments or not. Thus the nv-handler repairs always a complete page.

→ It is not allowed to repair two damaged pages by merging not damaged segments!!!!

It is important to have at least one valid page in every situation.

- It is not allowed to store the segments alternating between main-page and backup-page.
- It is necessary to store first the complete main-page. If no error occurs the complete backup-page will be stored next. Complete means all segments different to the stored data.

→ Save or repair one page in a time

The chance that an address-error during an access to the main-page will overwrite a segment in the backup-page must be less as possible. That means main and backup segments must not be mixed inside the address-range. Because there are two pages, the address of a main-segment and the corresponding backup-segment differs at least in the MSB. It might be good if the address is also different in the lower significant part. But there is no real advantage, because pages will be repaired completely.

5.4 Multiple Errors

Two or more errors could overcome the error-protection. Nevertheless the damaged data inside the nonvolatile memory must be recognized, to prevent the device from undefined action.

The byte-compare after writing is able to detect every error-combination. But the read operation could only protected by a checksum. Reading and comparing the checksum, too, will detect detection of errors after a reset. So it is possible that some error-combinations could not be detected, because they have a legal checksum. To reduce this risk to a minimum a 16-Bit CRC will be used to protect each segment.

The worst error-combination for reading is an error in the address, missing the second start-condition, writing FF into two segments, getting the stop condition and after all a reset occurs. In this case parts of two segments are overwritten with FF. The CRC should be able to detect this in nearly every situation.

The worst error-combination for writing is an error in the address and a reset after the writing has finished. In worst-case two segments are overwritten. In this case every data-combination is possible. That is a problem for the first segment, because the checksum will be overwritten to. So the chance for a legal checksum is 1/65536.

As a conclusion there is no absolute protection against combined errors. But the combinations that overcome the protection of NV-MEM are very strange. Thus the error protection needs no enhancements for multiple errors.

5.5 Timing

5.5.1 I²C-Bus via interrupt or polling

The communication-speed depends on the bus capacity and the pull-up resistance. The capacity is given by the typical input-capacity of CMOS so we have 10-20pf. Assuming that the speed will be set to the possible maximum for the given resistor, the power-consumption (energy) depends only on the amount of data that is transferred over the bus. But the current that must be provided increases linear to the communication speed. If we choose 100Kohm for pull-up 30µA for SDA and also for SCL must be provided. For a proper function the SCL-signal must be about 80% of the supply-voltage for the half period time of the maximum I²C-bus frequency. Together with the given delay-time to reach 80% the maximum speed for UHTE are

$$f = \frac{1}{2\left(-R\,C\,\ln 0{,}2 + \frac{1}{200KHz}\right)} = 60{,}8KHz$$

The nv-chips are usually able to work with 400KHz I²C-Bus speed so 100KHz should be possible without problems. Actually 75KHz are used, thus transfer of one byte (8Bits + Acknoledge) needs 120µs. For 1,8MHz it is not possible to write an interrupt function with less than 30µs. That means the transfer via interrupt cost 25% of the available computing time. The disadvantage is, that interrupts interrupt all tasks, also the process calculation or the sensor-interface and influence their real time behaviour. In fact they do not reach their real-time requirements anymore.

Conclusion:

Polling waste 75% computing-time but if it is done in the background-task, it doesn't influence higher prior tasks. So the UHTE-project I²C-Bus-handler uses polling.

5.5.2 Background-Saving

Writing one segment means reading segment, writing and reading again. So 96byte must be transferred plus 12Byte overhead plus some computing (e.g. checksum) → 15ms.
It could happen, that an object lies over the boarder of two segments, then two segments must be stored in the main-page and in the backup-page. All together its possible that 4 segments must be written to store one object. The minimum time for this is 60ms. Considering that the background task could run for 30ms in a 100ms-slot, it cost between 130ms and 215ms. The Layer 7 of HART is scheduled every 100ms together with 215ms saving time in worst case, the maximum response time of HART could not be guaranteed, and so we have to save in background.

5.6 Ram-Page

As described in “4.3.2” the EEPROM – access must be done in background. The usual solution for this problem is a shadow of the EEPROM inside RAM. Every access from the application to the NOVRAM will be done to the RAM.
Requirements:

1. Resources like RAM are limited in our domain, non-volatile data must be stored only once in RAM.
   → The non-volatile data are stored directly into the Ram-Page (Ram-Shadow). That means no double buffering for non-volatile data!

2. To save Time only changed data should be stored.
   → The Ram-Page is fragmented into segments like the Main-Page. NOVRAM-operations handle always a whole segment, so a dirty-flag for each segment is enough to save maximum time.

3. The NV-handler is responsible for consistence of NOVRAM and RAM-Shadow.
   → Load MAIN-Page into the RAM-Shadow called Ram-Page
   → Save Ram-page segments marked as “dirty” 5 seconds after the safe-flag is set
   → Check Ram-page, Main-page and Backup-page from time to time

4. Fast read access for the application must be guaranteed.
   → A direct read access to the application non-volatile data must be possible.
   → The application defines const-variables sharing the same address-range as the Ram-Page.

5. Information hiding → no knowledge of the structures to be stored
   → The linker will do the sharing of the address-range. (Overlaying linker segments)

6. The non-volatile data and the NOVRAM-shadow will be accessed from different tasks at the "same time".
   → The application uses resource semaphores to cope concurrency problems, e.g. each sub-system uses a resource semaphore for its nv-data. → The ram-page does not need an own semaphore.
   → The nv-handler stores all dirty marked segments until no more segments are dirty.
   → The dirty-flag handling guarantee that not dirty-flag get lost.

   Writing into the ram-page: first write data, than set dirty

   Copy from ram-page into nv-chip: clear dirty, write data

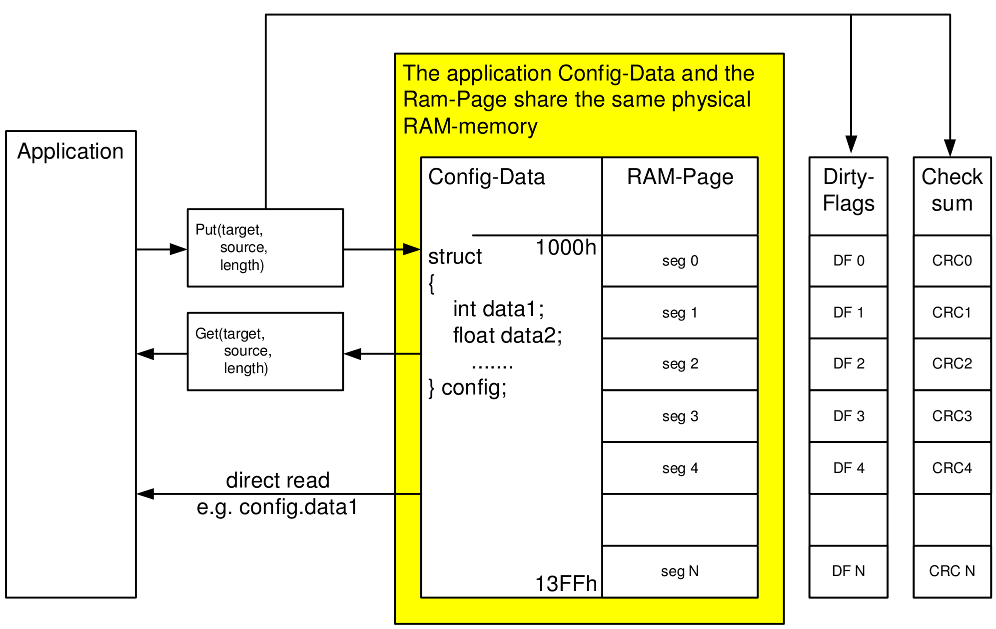
   As long as the page will be written, the page is marked nonvolatile as invalid!

   The page will be written until all dirty-flags are cleared for one complete check of all dirty flags!

   Warning: Cyclic writing into non volatile data faster then they could be stored leads to endless action!

→ The application must not work during the load operation! Or the application is stable against temporally inconsistent data.

Figure 4.5



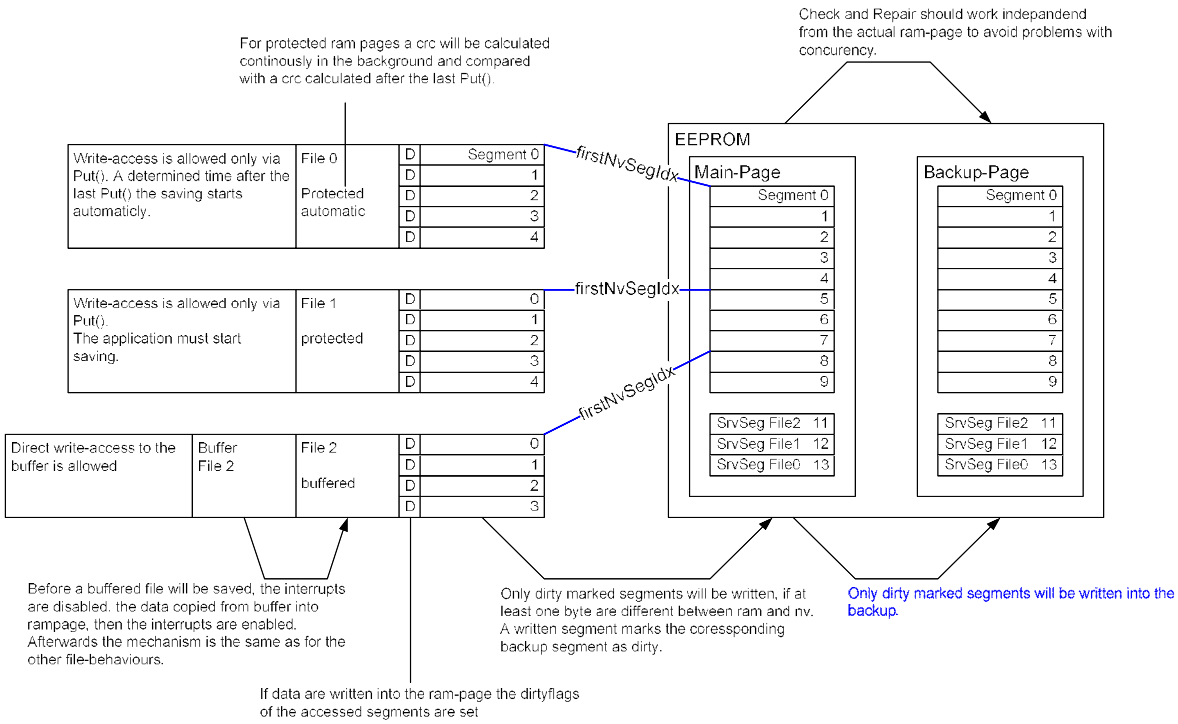
6 Design

6.1 Files

The nonvolatile behavior is usually not the same for all objects in a device. Thus different handling should be offered by the nv-handler. Data with the same handling are collected in one area that will be called “file”. The class “file” defines the structure, attributes and methods for such files.
The file-design based on the ram-page idea described in chapter 4.7. A file defines an area in the ram storage in which the subsystems could store their data-classes. The subsystems are allowed to read directly but for write-access they must use PutData(), because the nv-handler has to handle dirty-flags and must start a copy action to make the data nonvolatile.

Supported Non volatile behaviors:

1. Save AUTOMATIC
   Automatic is the default-behavior of a file. It means that the file will be stored without further action of the application than to write data via PutData(). Please have in mind that all changed data in the file will be stored.
   → Data in an AUTOMATIC-file are nonvolatile a determined time after the last PutData().
2. Save ON_DEMAND
   When the application does not want an immediately save, or it saving needs special permission, then the file need to be stored on demand.
   → Data in an ON_DEMAND-file are nonvolatile after the application calls Save()
3. PROTECTED ram-shadow
   Sensitive data may need protection against wild running pointers, at least it should be recognized that the data has changed without intention to change them. The recognition will be done by a CRC for the ram-shadow.
   → The data inside the RAM-Shadow are protected by a CRC-checksum.
4. BUFFERED
   Some non-volatile data changes faster as a complete save of a file last, or the overhead of the PutData()-method is not tolerable. In these cases a second buffer between application and ram-shadow is necessary. The application is allowed to write directly into this ram. When it comes to saving, the nv-handler disables the context-switching and copies to the ram-shadow, enables the context-switching and triggers background-saving immediately.
   → For data inside a BUFFERED-file the direct write-access is allowed.

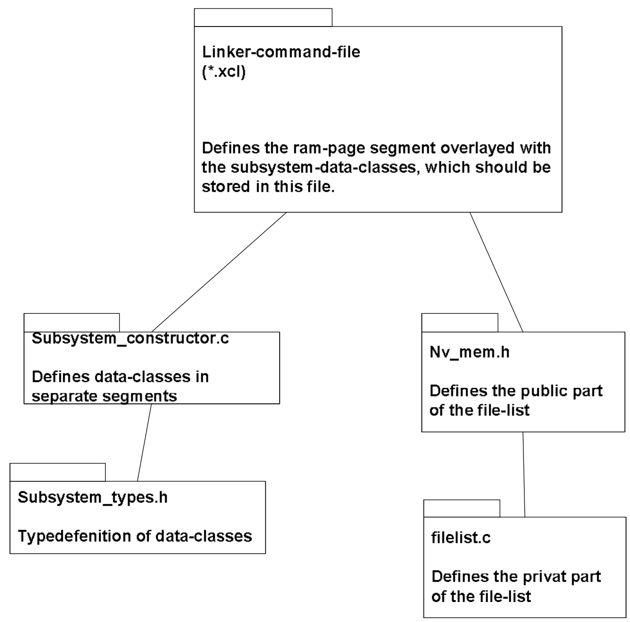


For each file there is a service-segment in which the valid-flag and the write-counter of the file are stored. The remaining bytes of a service-segment are free to use by the application.

6.2 Subsystem data-classes and ram-pages

Data-classes must be defined inside the ram-pages without becoming part of the nv-handler namespace, because the subsystem implementation shall be independent from the nv-handler used in the target system.

The first possible way is to define each subsystem-data-class inside its own memory-segment. Also the ram-page of each file will be defined in its own memory-segment. In the linker-command-file the data-class segments are defined inside a ram-page.



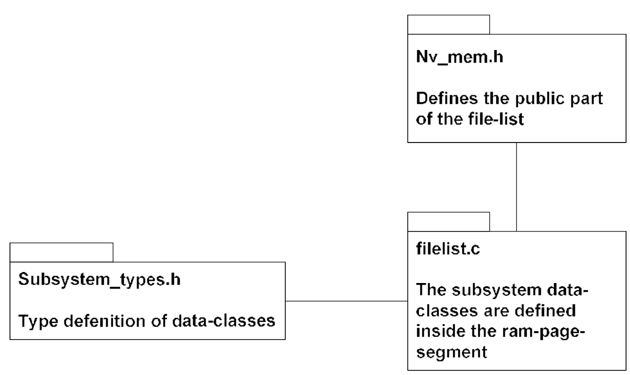
Advantages:

- best information hiding

Disadvantages:

- changes to the file-list must be done in three places: in the linker command file, in nv_mem.h and in fileList.c.
- the address-information of the ram-page-segments must be handled by hand.

It is also possible to define the subsystem-data-classes inside the memory-segment of a ram-page, which could be done together with the filelist-definition.



Advantages:

- all address and length information will be found by the compiler
- the complete definition will be done in one place

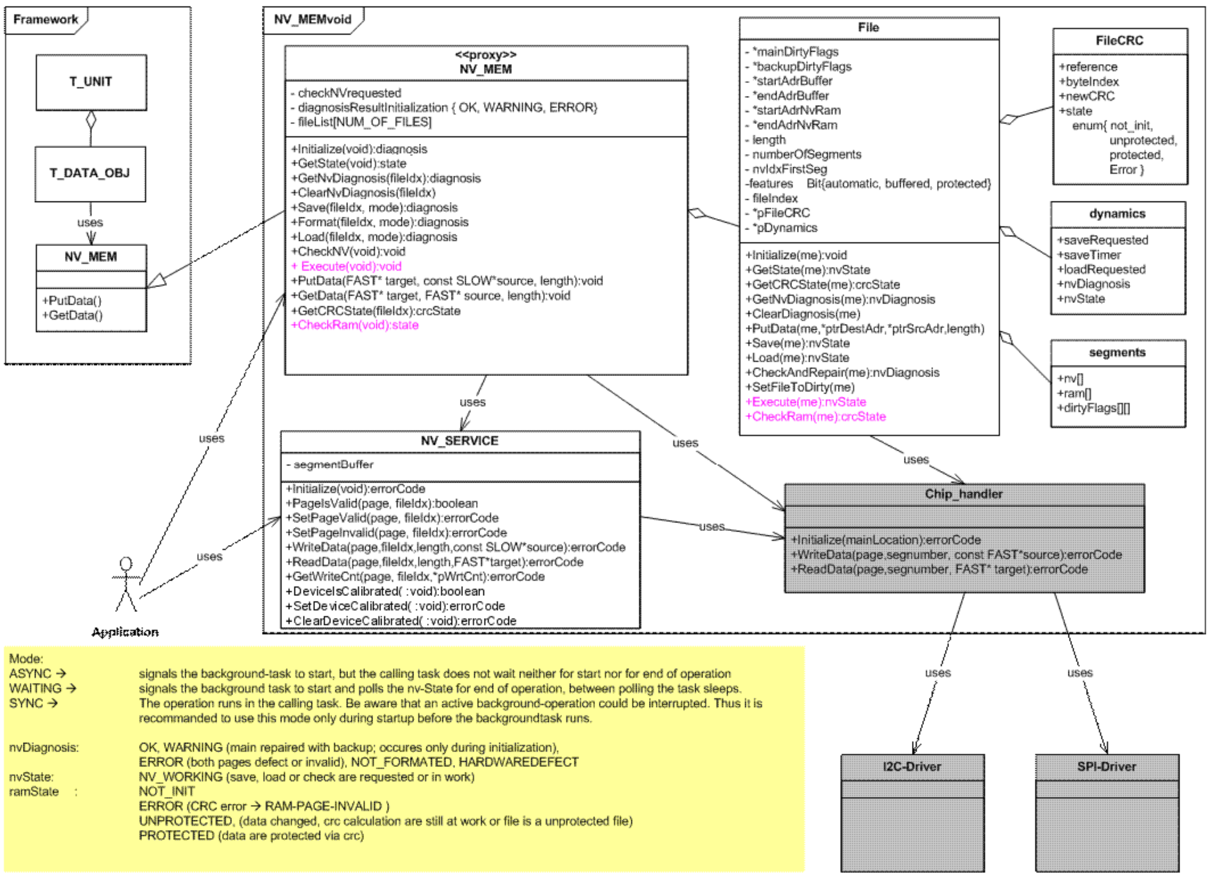
Disadvantages:

- filelist.c must include all subsystem type-definitions

6.3 Static Modeling

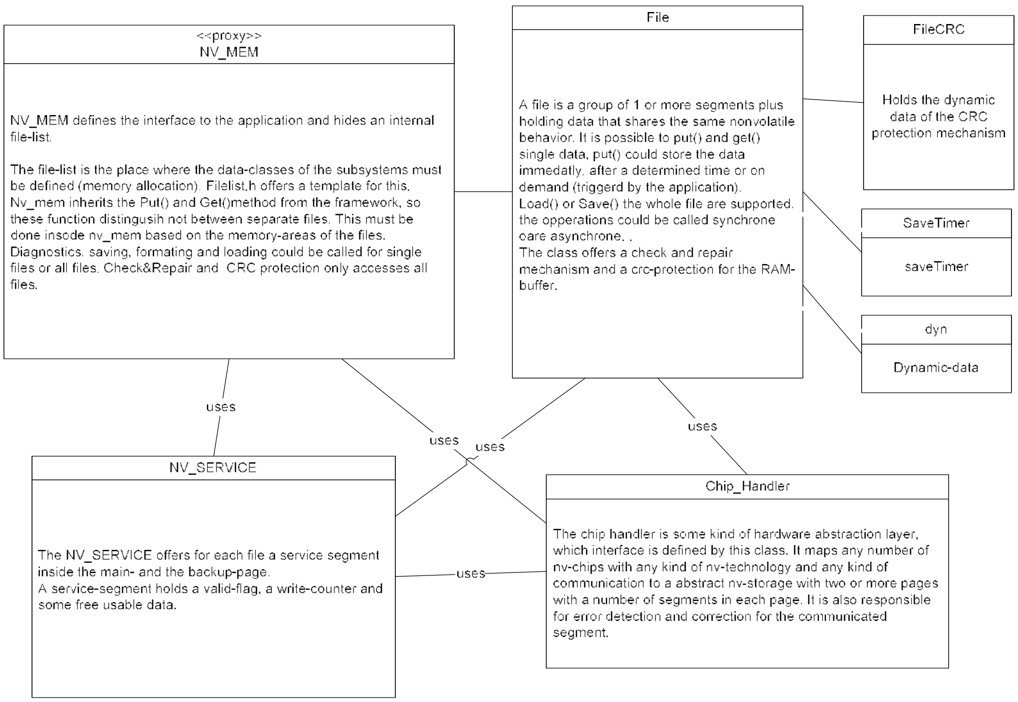
First the global structure of the nv_mem-component is shown in a class diagram. Due to the fact, that to many information in one diagram reduces the readability, attributes and methods are depict in one diagram and the responsibilities in a second one.

Class-Diagram:



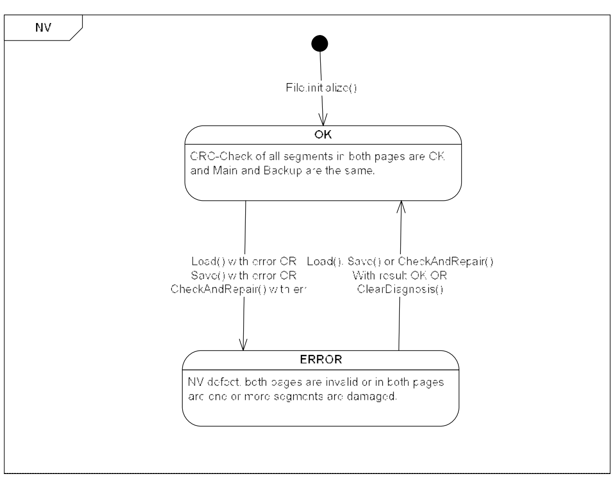
The service component "nv_mem" defines the interface of a chip-handler but the implementation is not part of the component. Only a chip-handler for module-testing is part of the component.

Responsibilities:



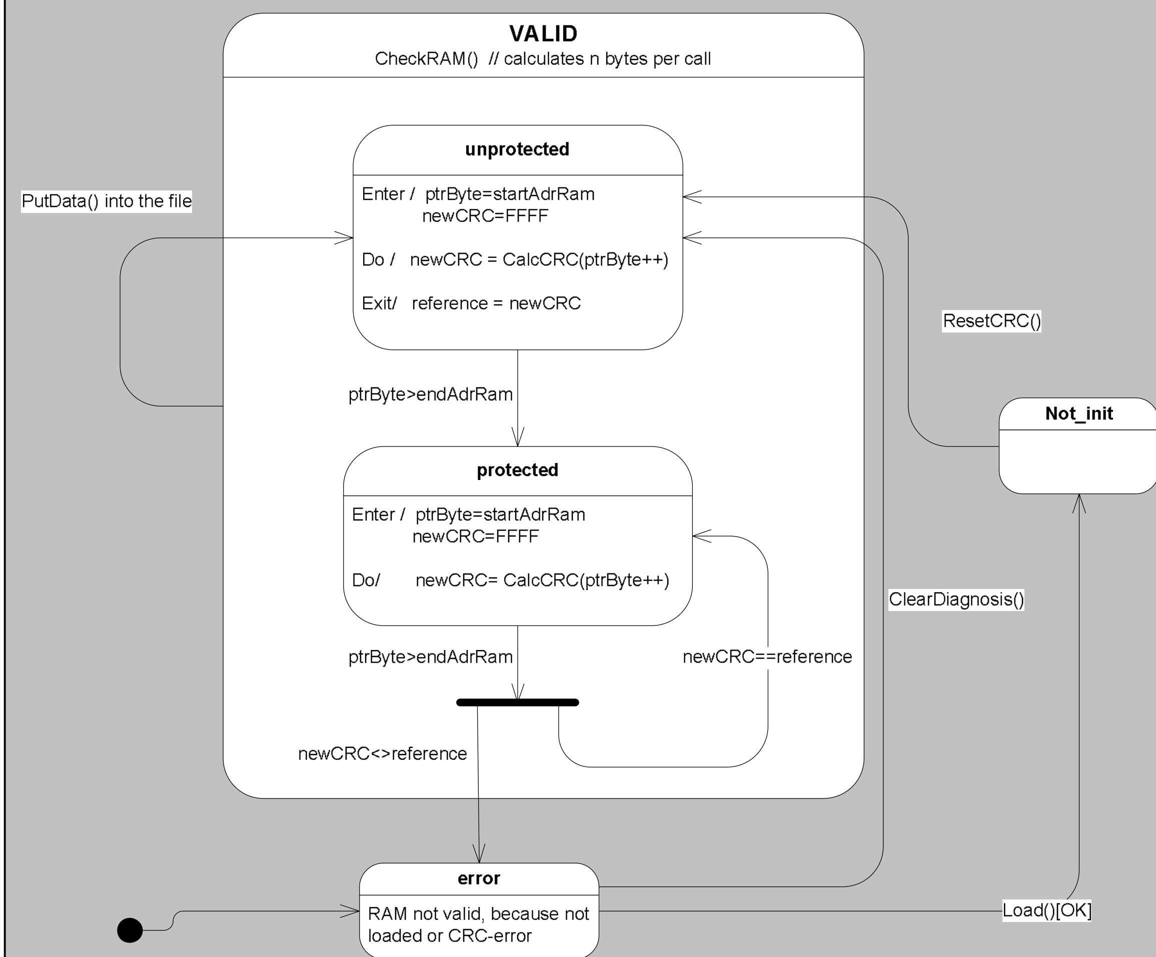
6.4 Dynamic Modeling

The behaviour of methods is described in the following structural design. But for the states of the nv-storage and of the ram-page, which are not handled be only one method, the behaviour are defined by the following state-charts.



The nv-storage, that means the data stored in the nv-chip or the communication to the chip, could be OK or it is erroneous (ERROR).

The next state-chart describes the complete design for the ram-page crc protection.



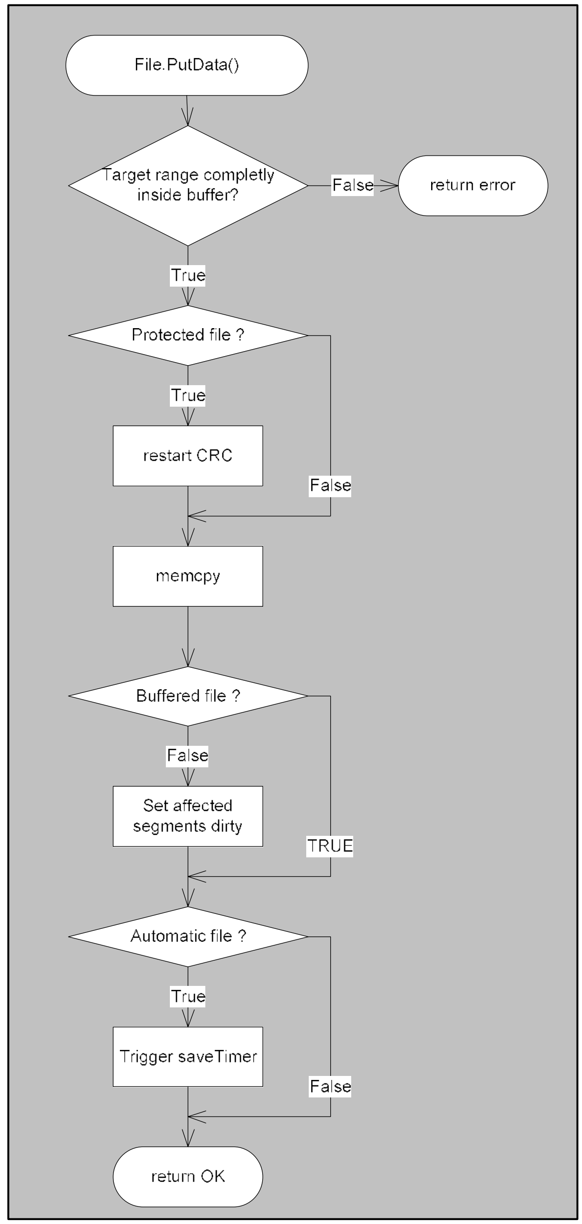
It is not allowed to call PutData() from a task with lower priority than the task in which the nv-execute runs. Otherwise data could be changed before the state chganged by PutData().

It is strongly recommanded that the nv-execute runs in the task with the lowest priority.

6.5 Class File

The complete and correct definition and implementation are found in the source code. In this paper the more complex functions will be explained with flow-charts. They are not for showing the code they are good to explain the design idea. Very simple functions are not explained here.

6.5.1 PutData



PutData(*me, *dest, *src, len):result

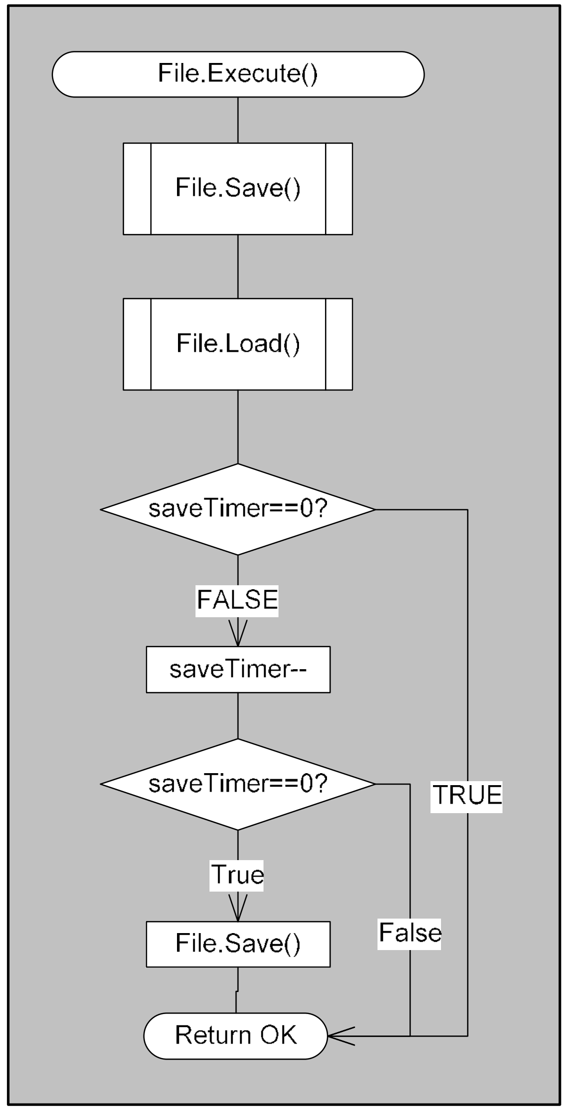
When the destination lays completely inside the file ram-page, then copy the data and handle crc-state machine, the dirty flags and the save timer, if the corresponding file options are enabled.

6.5.2 Execute

ExecuteFILE(*me):result

This method must be called from the lowest task in the system. For the timer-functions it should be schedule every 100ms.

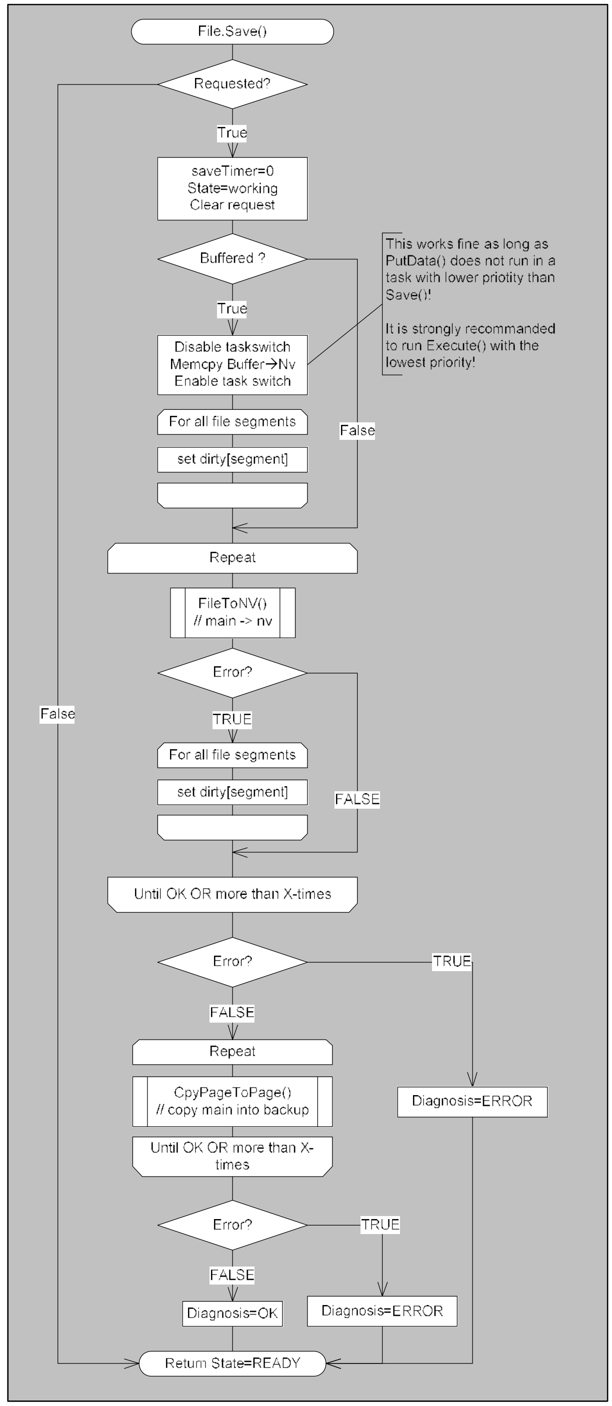
Saving or loading are requested from higher prior tasks by calling SaveFILE() or LoadFILE(), which set the save ore load request flag.



6.5.3 Save

SaveFile(*me):void

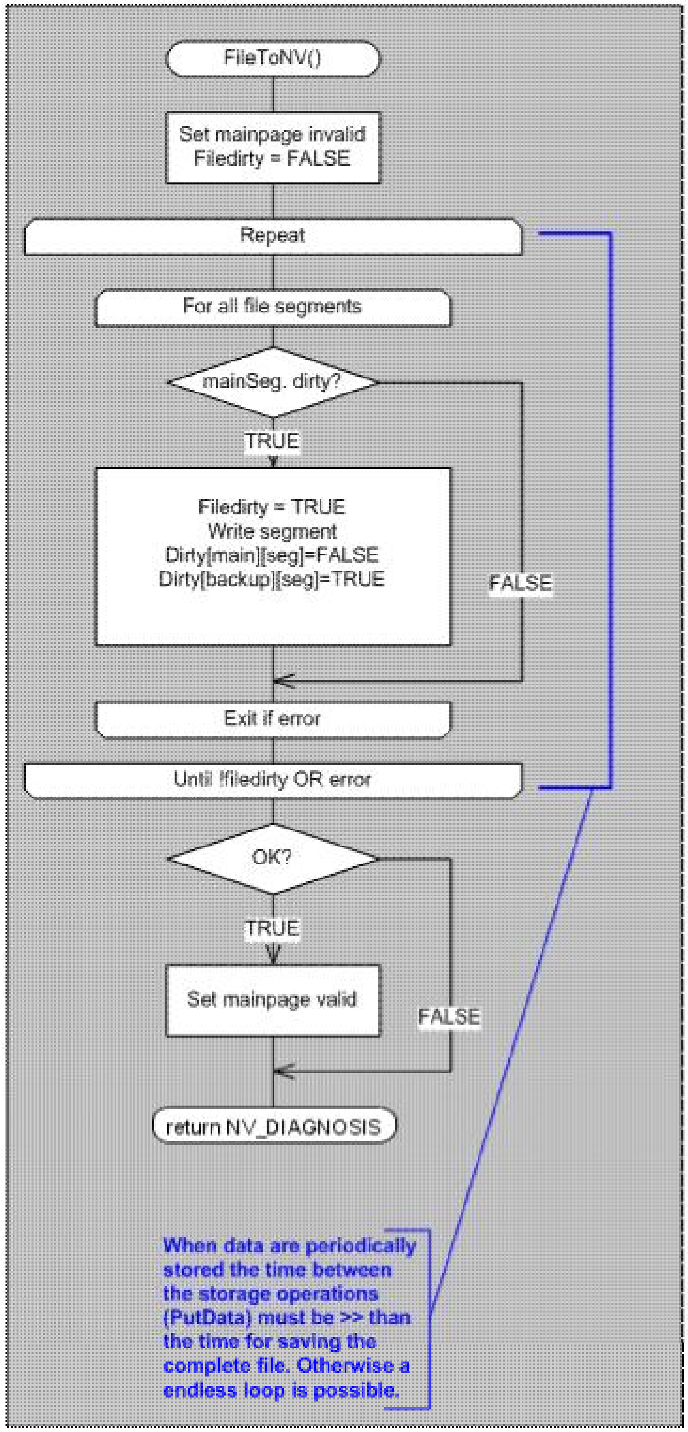
Stores the ram-page into the nv-storage.



6.5.4 FileToNv

FileToNV(*me, page):result

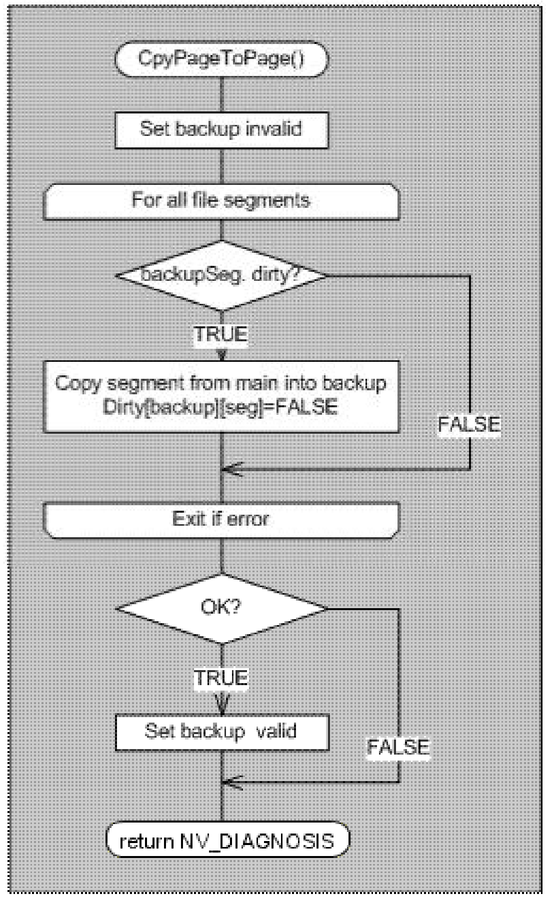
Stores the ram-page into the main or the backup-page. The flow chart shows the action for the main-page. Backup is the same action only for backup storage and the dirty-flag arrays first index are exchanged.



6.5.5 CpyPageToPage

CpyPageToPage(*me,dstPage,srcPage):result

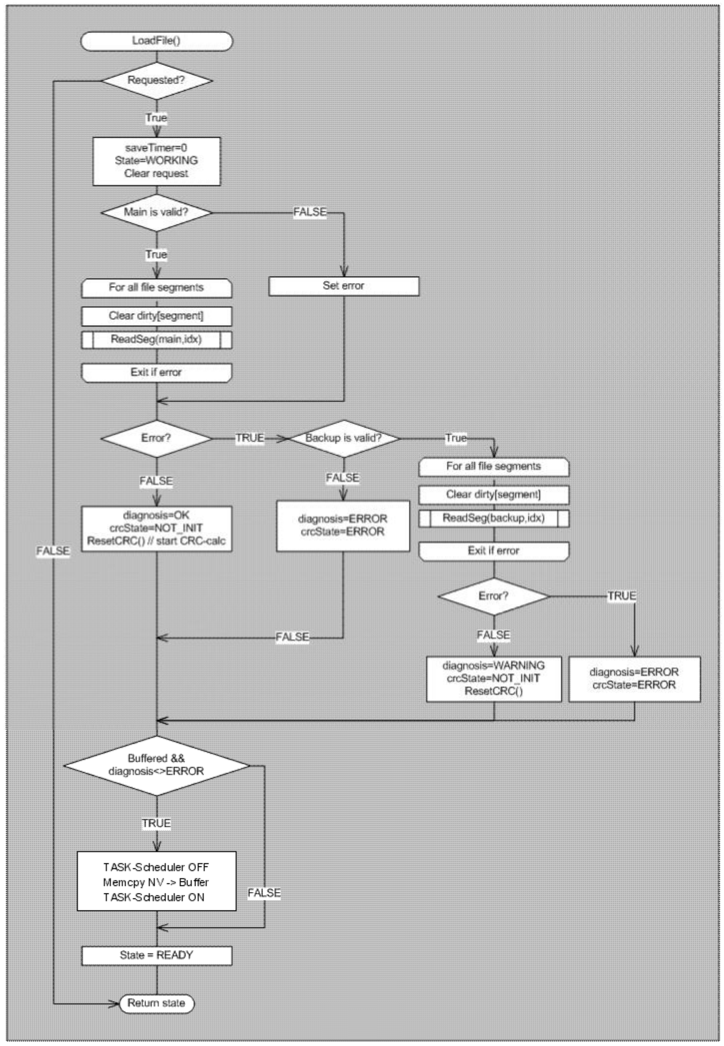
Copy one nv-page to another. The flowchart shows the action for copy main to backup.



6.5.6 Load

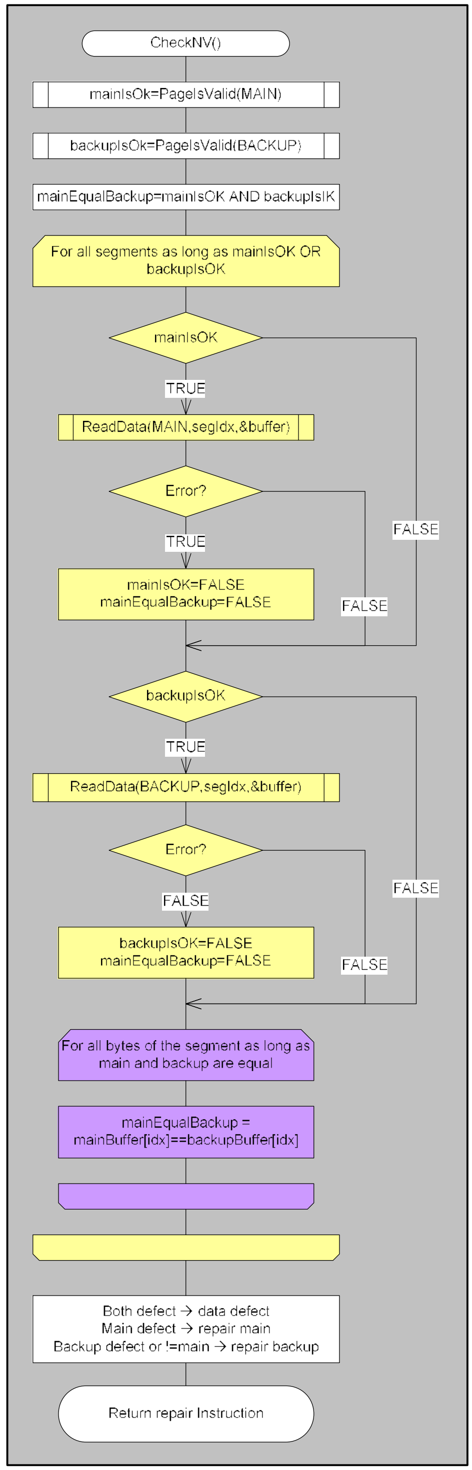
LoadFile(*me):result

Load the ram-page from main-page. If main-page is erroneous then load from backup instead. If backup is erroneous, too, then the dataset inside the nv-chip is defect or the communication is not possible due to a noisy environment. In both cases the ram-page is in error-state and also the nv-storage is in error-state.



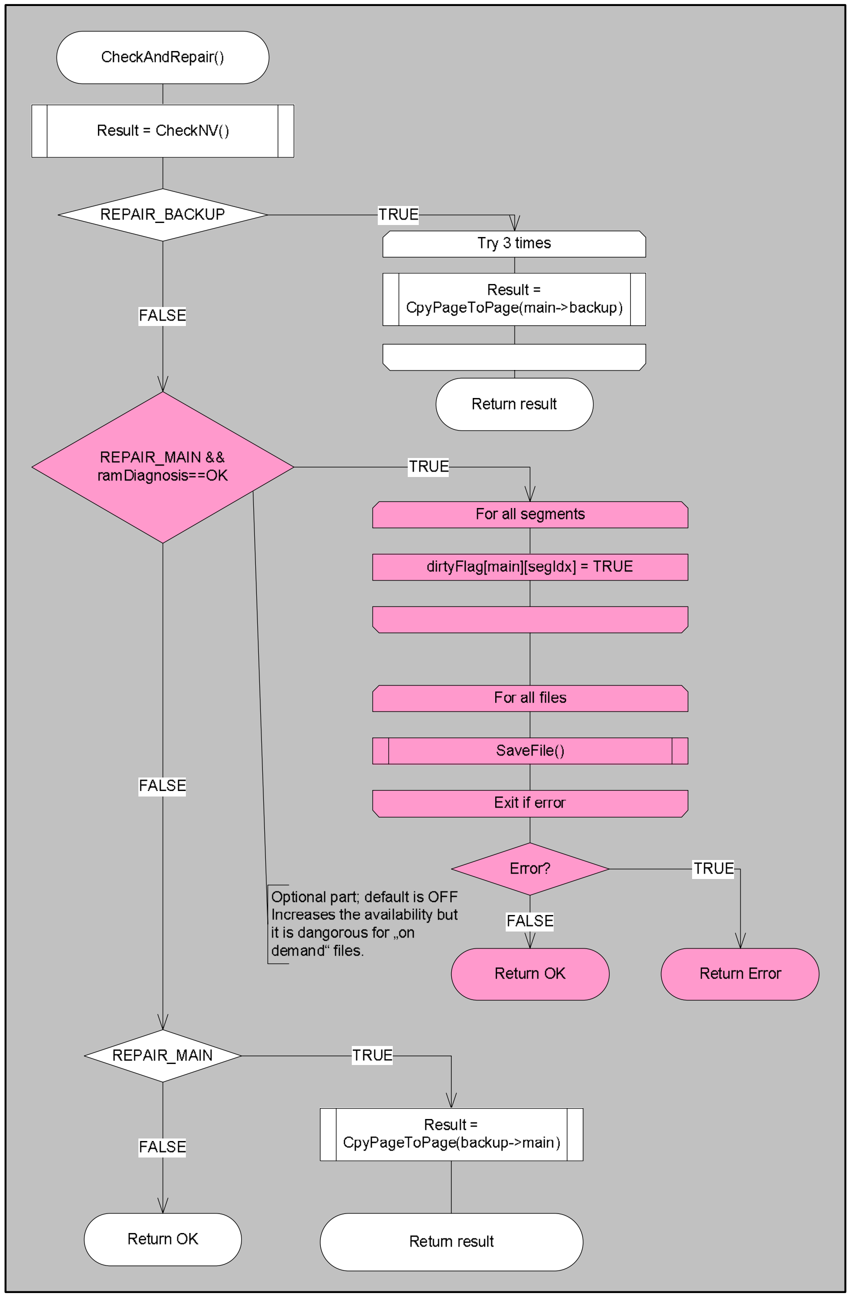
6.5.7 Check

CheckNV(*me):repairInstruction



6.5.8 CheckAndRepair

CheckAndRepairFILE(*me):result



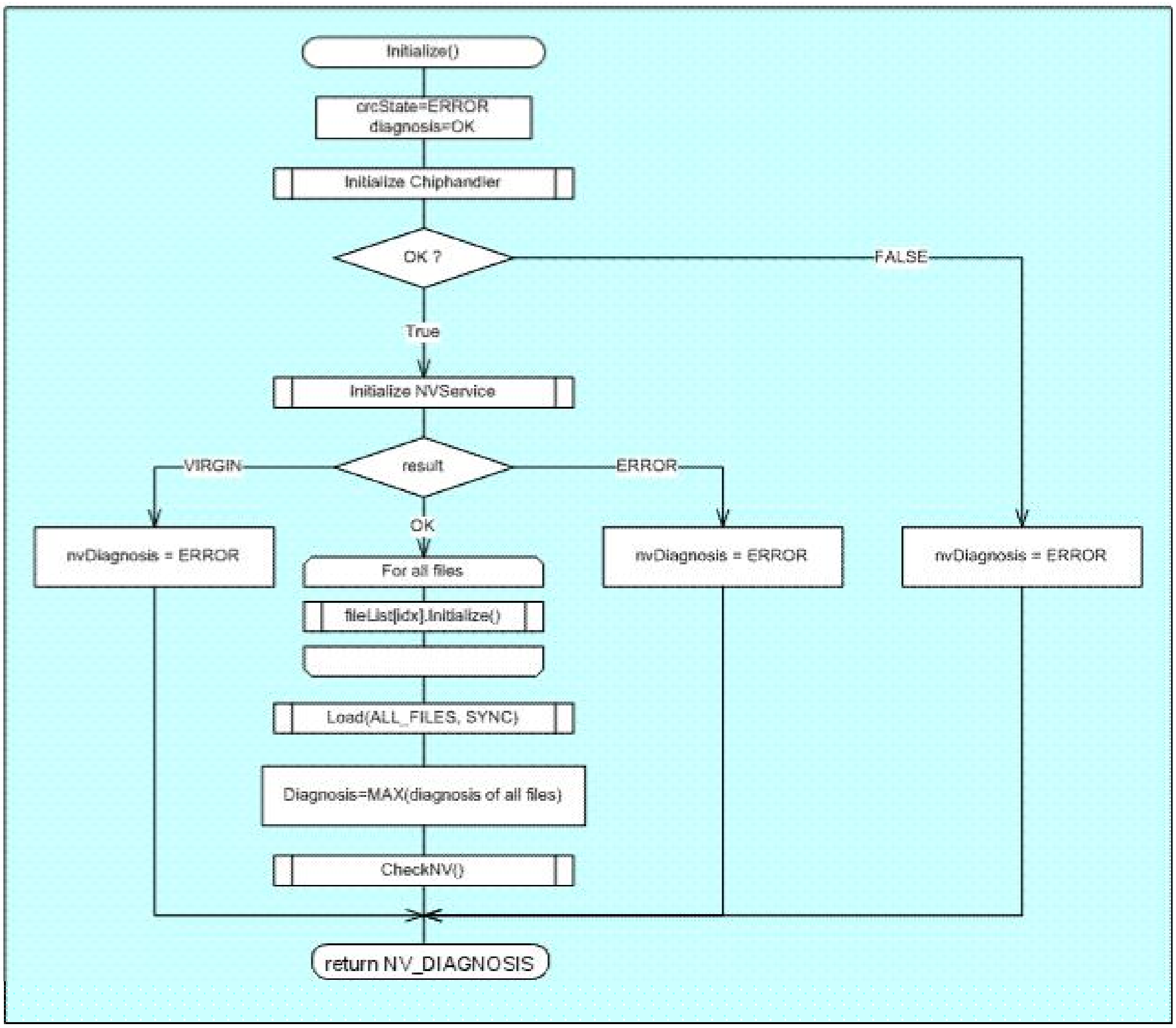
6.6 Class NV_MEM

The complete and correct definition and implementation are found in the source code. In this paper the more complex functions will be explained with flow-charts. They are not for showing the code they are good to explain the design idea. Very simple functions are not explained here.

6.6.1 Initialize

InitializeNV(void):diagnosis

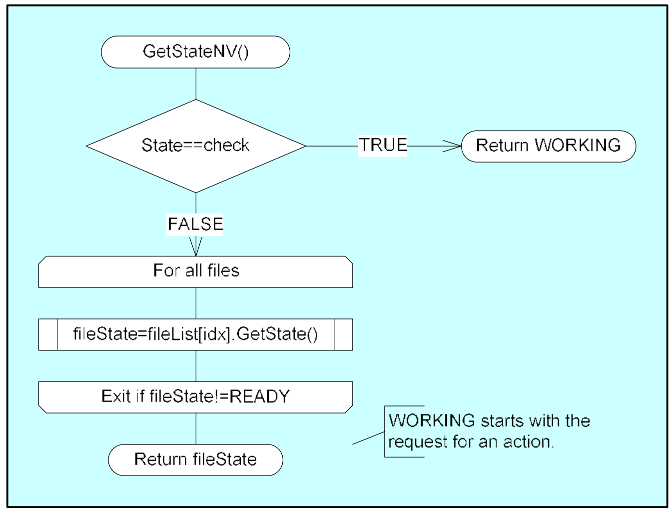
Initializes states, data and used services.



6.6.2 GetStateNV

GetStateNV(void):state

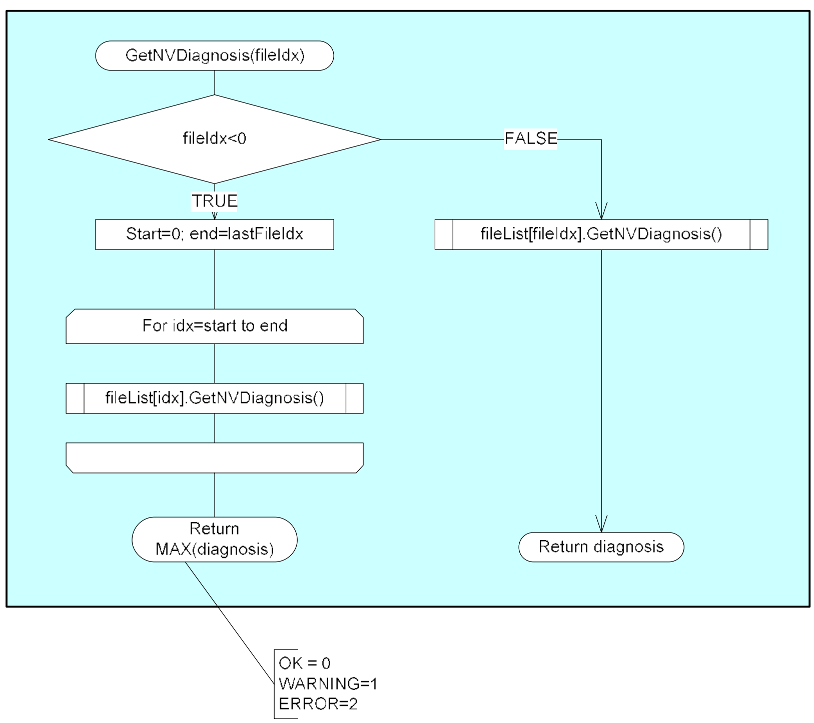
Until no check is requested and no file is working, the nv-state is working.



6.6.3 GetNvDiagnosis

GetNvDiagnosis(fileIndex):diagnosis

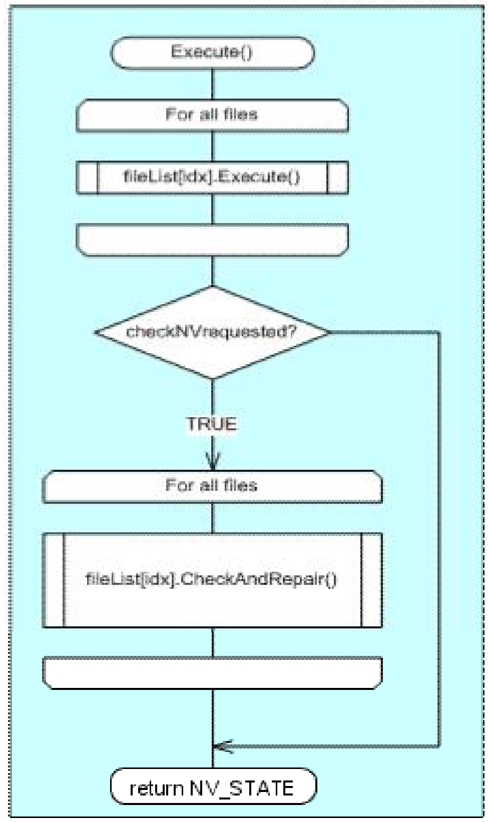
Returns a single file diagnosis or the maximum of all file diagnosis's.



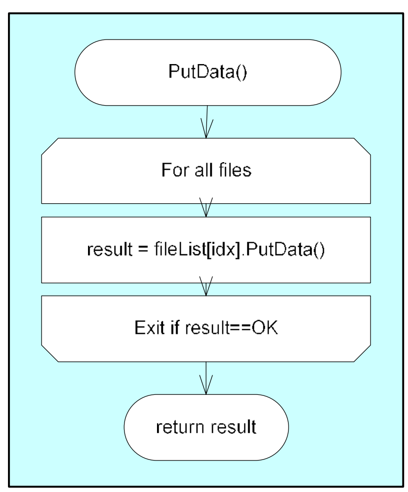
6.6.4 Execute

ExecuteNV(void):void

It is strongly recommanded that this function runs in the task with the lowest priority



6.6.5 PutData



6.6.6 Save, Load, Format

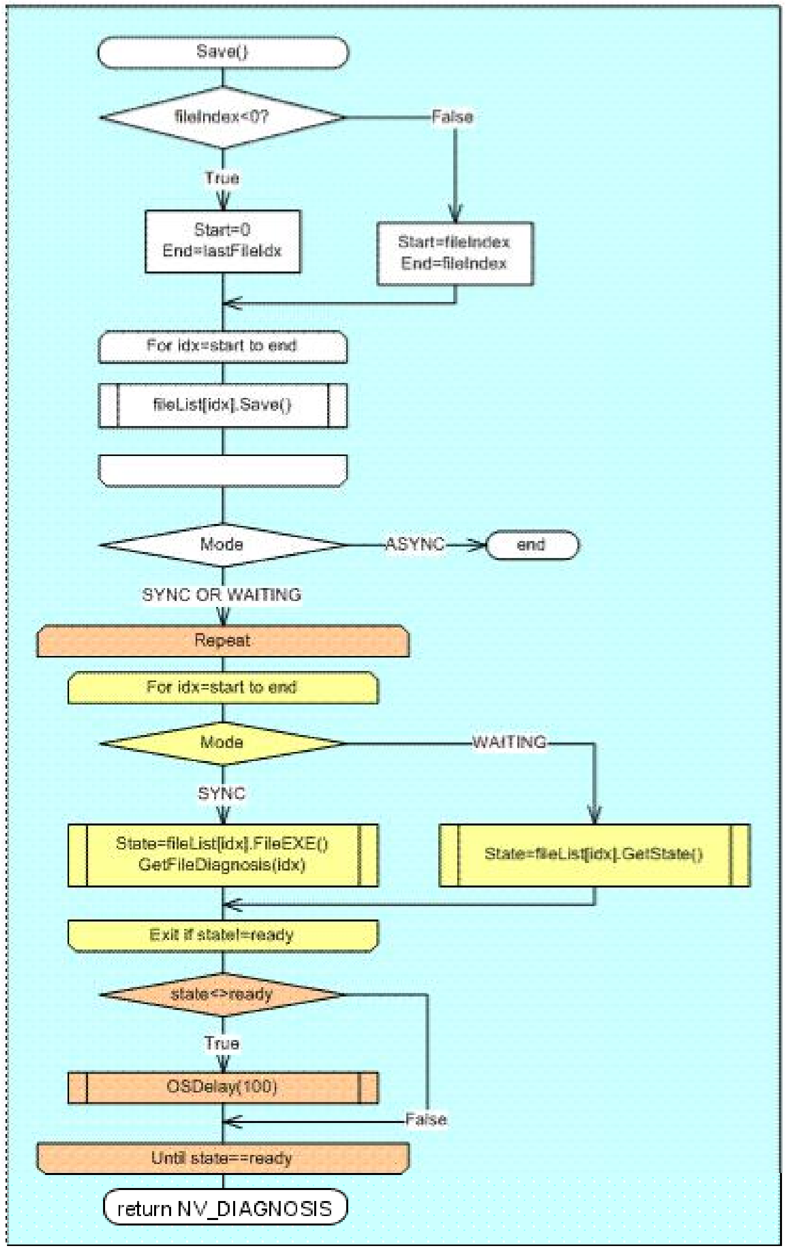
Save(fileIdx,mode):diagnosis

Load(fileIdx,mode):diagnosis

Format(fileIdx,mode):diagnosis

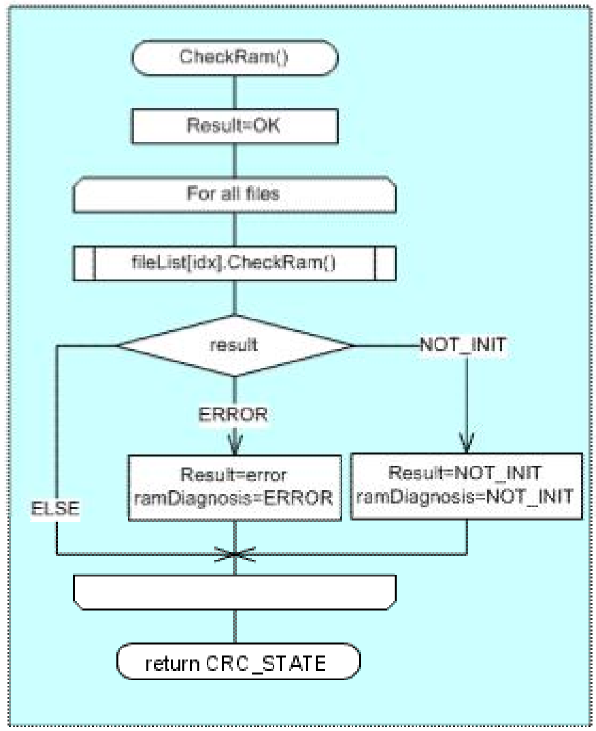
The flowchart for Load is the same, except file.Save() file.Load() will be called.

Format set all segments of the file to dirty and jumps then to Save().



6.6.7 CheckRam

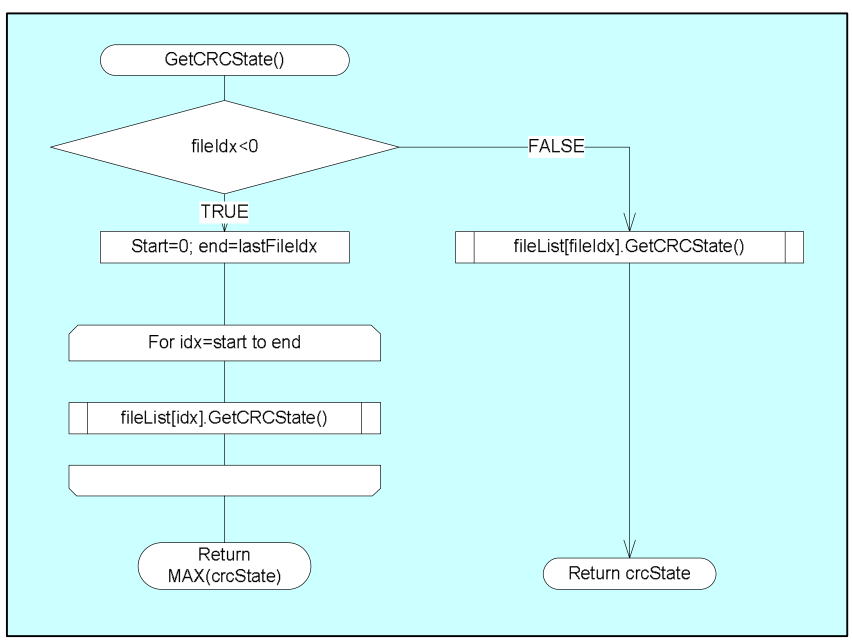
CheckRamNV(void):crcState



6.6.8 GetCRCState

GetCRCState(fileIdx):crcState

Returns a single file crc-state or the maximum of all file crc states.



6.7 NV-Service

As described in the analysis of Reset-Errors and their influence to the segmentation, it is necessary to store a Flag non-volatile for each page that shows if write access is actually active for one page. From beginning of the write-access until the end the page is not valid. The valid-flag will be stored in a special-segment, the service-segment; each page owns one service-segment. If a reset occurs between the writing of two segments, all segments are OK but the application-data could be a mismatch of old and new data. The valid flag indicate this possible mismatch. If after reset one page is not valid, it will be overwritten with the page valid. Only 4 byte of 30byte in one segment are used for the valid-flag. The UHTE-Implementation uses these bytes for a write-access-counter, a calibrated-flag and free accessible data. Free accessible data could be used by the application e.g. for a version numbering of the nv-data-structures.
Both service-segments (main-page and backup-page) have a complete shadow inside the RAM. So the nv-service-subsystem is independent from rampage and storage-handler.

6.7.1 Initialize()

```
TUSIGN16 Initialize_NVSERVICE(void);
```

The Initialize()-method load the service-segments out of the nv-storage into the service-shadows (RAM). If both are damaged or not readable than the nv-storage is virgin; that means it isn’t formatted. A damaged service segment will be overwritten and the valid-flag will be set to “invalid”. If writing fails, the nv-storage is defect.
As long as the calibrated-flag is set in one service-segment the device is calibrated and the calibrated-flag will be set in both service segments.
At the end of the method the system knows if the nv-storage is defect, not formatted (virgin) or formatted and not calibrated or calibrated.

6.7.2 PageIsValid(), SetPageValid(), SetPageInvalid()

```
TBOOL PageIsValid_NVSERVICE(TUSIGN8 page);
```

Test if the valid-flag of the page “page” is set. Returns TRUE if the page is valid.

```
TUSIGN16 SetPageValid_NVSERVICE(TUSIGN8 page);
```

Set the valid-flag of the page “page” to valid.

```
TUSIGN16 SetPageInvalid_NVSERVICE(TUSIGN8 page);
```

Set the valid-flag of the page “page” to invalid and increment the write-counter.

6.7.3 DeviceISCalibrated() / SetDeviceCalibrated() / ClearDeviceCalibrated()

```
TBOOL DeviceISDAlibrated_NVSERVICE(void);
```

Returns TRUE if the calibrated-flag is set in both service-segments.

```
TUSIGN16 SetDeviceCalibrated_NVSERVICE(void);
```

Set the calibrated-flag in both service-segments.

```
TUSIGN16 ClearDeviceCalibrated_NVSERVICE(void);
```

Clear the calibrated-flag in both service-segments.

6.7.4 GetWriteCnt()

```
TUSIGN32 GetWriteCnt_NVSERVICE(TUSIGN8 page);
```

Returns the 32-bit write-counter of the page “page”. The counter will be incremented each time the valid-flag will be set to invalid.

6.7.5 GetData() / WriteData()

The application could access the unused data of the service segment via these methods. The macro-const “NVS_DATALENGTH” defines the number of bytes that are free to use.

```
TUSIGN16 GetData_NVSERVICE(TUSIGN8 page, TUSIGN8 length, void __data16* ptrData);
```

Copy “length” byte from the service-segment of page “page” to the address “ptrData”.

```
TUSIGN16 WriteData_NVSERVICE(TUSIGN8 page, TUSIGN8 length, const void __far*ptrData);
```

Copy “length” byte from the service-segment of page “page” to the address “ptrData”.

7 Revision Chart

Rev.	Description of Version/Changes	Primary Author)s)	Date
0.0.0		Heiko Kresse	2004-09-09
0.0.1	adapted to changed NV-MEM definition	Heiko Kresse	2004-09-22
1.0	Release	Heiko Kresse	2005-09-05
2.0	Extended nv_manager	Heiko Kresse	2009-04-03
2.1	Reviewed by A. Stelter / IIMS - minor corrections - - none design issues changed	Heiko Kresse	2009-04-20
2.2	Modified after document review for MiLe2 Project	Giovanni Invernizzi	2009-04-21

ABB	Software Design Review				SDR
Responsibility: Instruments	Date:	Language: **en**	Filing system : **-**	Revision: **-**	Page: **1/2**
Template Issued by: A. Stelter	Template Approved: PSM AP2.2		Template Released: 10/2004		Area of validity: APR/I

Project:	**NV_mem**
Document under Review:	**Software Design Description NV-Handler IIMS_V2_1**
Revision:	**2.1**
Review Date:	**29-06-2009**

Review-Participant:

Place	*Dept.*	*Name*	
Bangalore	INCRC	Ashwin Herur R	
Bangalore	INCRC	Ganapathi R	

Decision of the Review:

Decision	*next steps*
❑ Inspection passed ***without restrictions***	Phase finished
■ Inspection passed ***with restrictions***	some changes must be done
❑ Inspection ***not*** passed	Inspection must be repeated

Changes are proved: The Reviewer confirms that all changes are done:

proved Rev:	Date:	Reviewer:
2.2	10-07-2009	Ashwin Herur R

Check list:

		yes	no
1.	Is the software architecture distinct and documented?	**Y**	
2.	Fit the modules together?	**Y**	
3.	Are complex algorithms/procedures explained?	**Y**	
4.	Is a strategy for error handling designated?	**Y**	
5.	Is the configuration management system well prepared?	**Y**	
6.	Are all open issues transferred to the defects table?	**Y**	

Remarks:

Defects

No.	Check point	Description	Major defect	done Date
1		Document name has no numbering **Answer: the document is an external document (Minden), so it a part of an external package of documents called 266_FE_NV_MEM**	N	No Action
2		Area of Validity need to be checked **Answer: the document is an external document (Minden), area of validity is referred to Germany**	N	No Action
3		Acronyms need to be added.	Y	
4		4.2.3 Conclusion: starts with Sl. No. 2, Number 1 is missing.	Y	09/07/2009
5		4.3 Protocol SPI-Bus is not applicable for MiLe2. So it can be deleted or the design should be customized for different project.	N	09/07/2009
6		4.6.1 I²C-Bus via interrupt or polling: 1 Byte is given as 9 bits. Please check **Answer: modified in 8 bits + Acknowledge**	N	09/07/2009
7		Page 23 is left blank.	N	09/07/2009
		5.3 Static Modeling: Mismatch in the methods in code and Class Diagram.	Y	09/07/2009
8		5.5.6 Load: in the flow chart what is Task-Wechsel?	N	09/07/2009
9		For the Functions CpyPageToPage, FileToNV,InitializeNV, CheckRam, Save, Load, Format etc use return instead of end in the flow charts.	N	09/07/2009
10		Class design missing for NV_Service.	Y	09/07/2009
11		Sequence diagrams can be added to show the Read and Write operations to NV. **Answer: too complex to be in only one diagram**	N	No Action