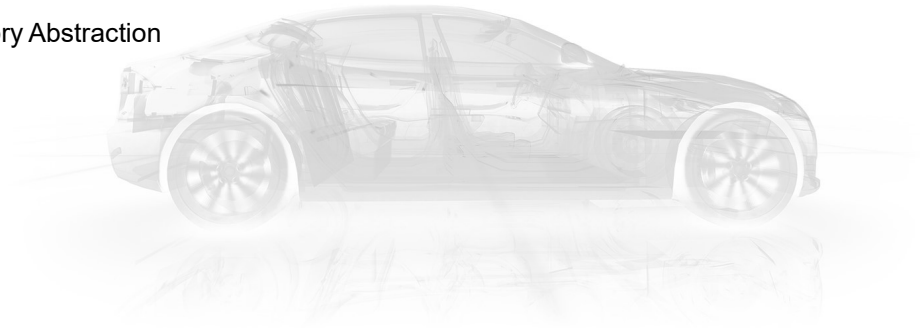
NIT

AUTOMOTIVE SOFTWARE WITH AUTOSAR

AUTOSAR

Agenda

- **Principles of Memory Technologies**
- The MICROSAR.MEM solution
- Memory Handling
- NvBlockSwComponents
- Service Mapping
- Exercise 5 - Memory Abstraction

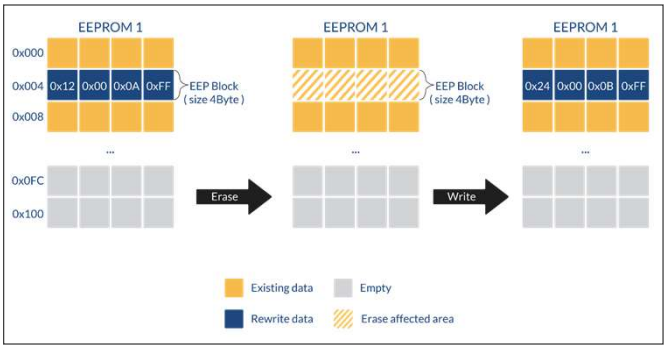


Principles of Memory Technologies
Differences between EEPROM and Flash

EEPROM

› Erasable and rewriteable in small blocks (1-4 bytes)

› Data can be rewritten to same memory cell easily



Principles of Memory Technologies

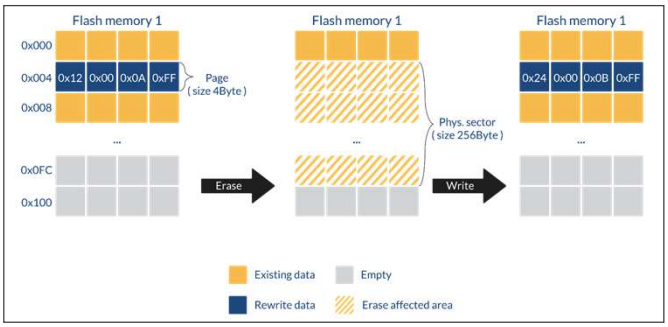
Differences between EEPROM and Flash

Flash EEPROM

› Has larger memory elements

› Smallest addressable memory unit (page) is not erasable on its own

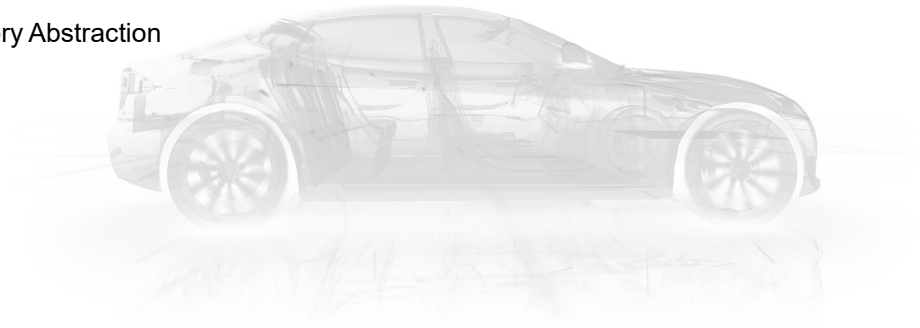
› Multiple pages are combined into a physical sector which is erasable



Having different physical sector sizes also implies different time durations needed for sector erase. To erase 16KB may take hundreds of milliseconds, therefore an erase operation can last many seconds.

Agenda

- Principles of Memory Technologies
- **The MICROSAR.MEM solution**
- Memory Handling
- NvBlockSwComponents
- Service Mapping
- Exercise 5 - Memory Abstraction

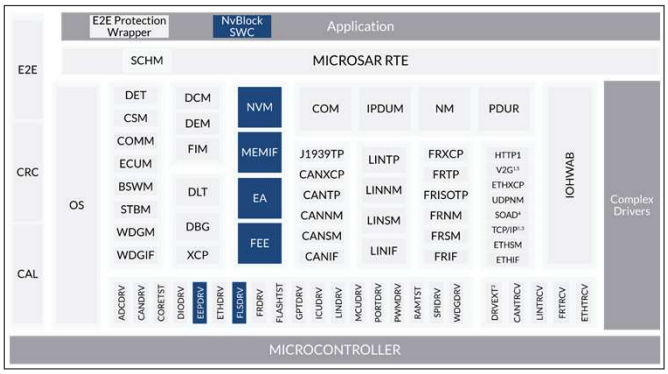


The MICROSAR.MEM solution
AUTOSAR BSW Modules for Memory Abstraction

- NVM
- MEMIF
- EA
- FEE
- EEPDRV
- FLSDRV

Optionally at RTE level:

- NvBlockSwComponentType



The MICROSAR.MEM solution

AUTOSAR BSW Modules for Memory Abstraction

NVM

- Provides abstract data storage
- Startup and shutdown handling for NV data
- Associates NV data with ROM defaults and RAM working copy
  - permanently or temporarily
- Data may be spread across multiple "devices"
- Explicit Synchronization support
- Improved Job handling
- BSWM support*
- Block Status Mode Notification
  - Multi-Block, Single Block

* BSWM Multi Block Job Status or Single Block Status Information
The NVM calls
- `BswM_NvM_CurrentJobMode()` (Multi Block)
- `BswM_NvM_CurrentBlockMode()` (Single Block)

to notify the BSWM upon any job status changes of the Multi Block, no call of the Multi Block or Single Block respectively.

The MICROSAR.MEM solution
AUTOSAR BSW Modules for Memory Abstraction

MEMIF

› Dispatches between memory technologies (FLASH or EEPROM)

FEE/EA

› Abstraction from different NV memory technologies (e.g., segmentation and number of write/erase cycles)

FLS/EEP

› NV memory hardware drivers

› Can access external devices e.g., via SPI

The MICROSAR.MEM solution
Nonvolatile Memory Manager (NVM)

- Queued job processing
  - Selectable between prioritized Queue or simple FIFO
  - Configurable queue depth
  - Separate queue for writing immediate data (e.g., crash data)
    - Guarantees job acceptance and fast start of job
  - Job cancellation API to abort pending requests
- Startup and shutdown handling
  - Read data blocks from NV at startup
  - Write data blocks at shutdown
  - Blocks are user-selectable
- Support of multiple RAM data blocks
  - Via configurable permanently-assigned and temporary RAM blocks
  - Temporary RAM Blocks allow SWCs to implement double-buffering

The MICROSAR.MEM solution
Nonvolatile Memory Manager (NVM)

- Data CRC
  - Configurable per-block: OFF, CRC16, CRC32
  - To protect data in NV memory - detection of errors
  - To protect data in RAM - consistency check after reset
    - E.g., due to watchdog reset or brown-out
  - Block-wise CRC recalculation in background (queued) to reduce CPU load
- Automatic / user-requested load of default data in case of read errors
  - From ROM or via callback (configurable per block)
- Configuration check at start-up - "dynamic configuration handling"
  - Ensures that current configuration is compatible with data layout of NV memory
  - Using NV data or block defaults in case of incompatibility

The MICROSAR.MEM solution
Nonvolatile Memory Manager (NVM)

- Write protection of data
  - Temporary (set/unset): configurable at runtime
  - Write Once: persistent
- Handling of RAM block states
  - Valid/Invalid data→ avoid writing of invalid data
  - Modified/Unchanged data → avoid writing of unchanged data
  - `NvM_SetRamBlockStatus()` API exists to mark RAM data as
    - "Valid and Changed" OR
    - "Invalid and Unchanged"
- Redundancy
  - Increases data availability
    - In case of write error or abort (e.g., through reset), a second instance is available
  - Redundancy shall be used for data which is critical for application
  - Redundant blocks shall always be secured with a CRC

The MICROSAR.MEM solution
Nonvolatile Memory Manager (NVM)

- An NVRAM block consists of:
  - A RAM block (permanent/temporary) to transfer data between application and NVM.
    - Must be provided by application
  - RAM block management information (state, last request result)
  - One or more NV memory blocks of equal size
    - Payload size equal to RAM block size
    - Located in the same NV memory "device"
  - An optional default block which is either
    - A user-provided block of data in ROM (no CRC)
    - A user-provided callback

The MICROSAR.MEM solution
NVM Explicit Synchronization feature

- In the time between `Nvm_WriteBlock/NvM_ReadBlock` and `NvM_JobEndNotification`, the RAM working copy is not accessible or valid for the application.
- Solution alternatives

1. Double buffering using temporary RAM blocks (RAM mirror and double buffer in RTE/SWC)
2. Explicit synchronization (double buffer as extra RAM mirror implemented by the NVM)
   - Can be enabled for single NvBlocks
     - Parameter: `NyMBlockUseSyncMechanism=TRUE|FALSE`
   - When the NVM reads or writes the RAM mirror, it calls a callback function configured in Configurator to collect data from or deliver data to the application RAM mirror
     - `NvMReadRamBlockFromNvM`
     - `NvMWriteRamBlockToNvM`

13 - CONFIDENTIAL MATERIAL -

NVM supports an optional Explicit Synchronization mechanism between application and NVM. It is implemented by an additional RAM mirror in the NVM module itself. The data is transferred by callbacks to the application which are called by the NVM module when needed. The synchronization mechanism can be configured for every NVRAM block separately. The NVM will create its RAM mirror based on the largest NVM blocks which uses explicit synchronization.

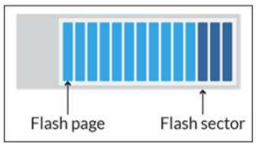
You can configure the names of the callback functions in Configurator:

- `Std_ReturnType <NvM_WriteRamBlockToNvm> ( void* NvMBuffer )`
- `Std_ReturnType <NvM_ReadRamBlockFromNvm> ( const void* NvMBuffer )`

The MICROSAR.MEM solution
Flash EEPROM Emulation (FEE)

FEE is implemented according to the properties of Flash Memory

- Access to the memory using Flash pages only
  - These consist of several bytes
- A Flash page
  - Can be written once, typically*
  - Cannot be erased individually
- Sectors comprise several pages
  - Only sectors can be erased individually
    - All constituent pages are erased together
  - Limited number of write/erase cycles per cell
    - High utilization of a sector is advisable!



→ Therefore a dynamic data layout is used to access the Flash memory and exploit one sector as effectively as possible

*The flash technology only allows to modify bits from 1 → 0, but not from 0 → 1 Therefore a flash page already written has to be erased prior to further usage.

The MICROSAR.MEM solution
Flash EEPROM Emulation (FEE)

- Uses Flash drivers (FLS) to access flash hardware
- Dedicated Data Flash or Read-While-Write feature required in hardware
- Manages data through "linked lists" within a FEE sector
- Memory is always allocated for the configured "chunk" size
- Automatic switching between two FEE sectors
  - Copies most recent data block instances
  - Erases old FEE sector
  - At least two Flash sectors are necessary

The MICROSAR.MEM solution

Flash EEPROM Emulation (FEE) – key features

- Detection of incomplete writes (e.g., due to reset or due to cancellation)
- Combines multiple physical Flash sectors to logical FEE sectors
  → FEE always handles two logical sectors
- Logical sectors may differ in size
- Multiple partitions (in pairs of 2 FEE sectors) are supported for independent operation
- Redundant internal management information for increased robustness
- Configuration update supported
- Shared configuration usage between Flash Bootloader and application possible

The MICROSAR.MEM solution
Flash Driver (FLS)

- Provides access to Flash hardware
- Depending on the platform, Flash area(s) to be used can be configured

- Sometimes limited to address either Data Flash or Program Flash
  - Some types of Program Flash cannot be written at the same time you execute the code from them
  - Normally the application software code resides in ROM (Program Flash)
  - Data Flash can typically be accessed at any time

The MICROSAR.MEM solution
EEPROM Abstraction (EA)

EEPROM Memory

- Strategy to increase write cycles
  - Write one block to several memory locations in an alternating way

The MICROSAR.MEM solution

EEPROM Abstraction (EA) – Key features

- Detection of incomplete write operations by using header and footer
  - Caused by reset or cancellation
- ECC used for internally-added management information
  - Increased robustness and reliability
- “Walking concept" to increase number of write cycles
  - Each NV memory block consists of several instances
  - Written in a round-robin manner
- Automatic alignment of data supported in tool
  - Can speed up write operations
  - Avoids influences on other (adjacent) data blocks/instances

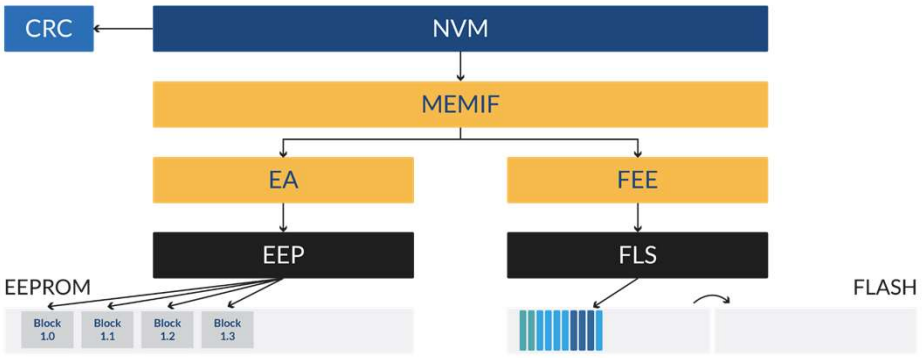
ECC: Error Correction Code - usually not provided by EEPROM hardware
ECC algorithm: combination of parity check and inverse management storage

The MICROSAR.MEM solution
EEPROM Driver (EEP)

- Direct access to the EEPROM HW
- Provides byte-wise writing even if not supported by HW
- Write Cycle reduction (configurable)
  - Write only data if it is different from the current content
- Provides Erase service even if not supported by hardware
  - e.g., SPI EEPROMs don't have an ERASE command
- Publishes device information that is necessary for the EA
  - Size of device, erased value (0x00/0xFF), writeable/erasable unit size

The MICROSAR.MEM solution

BSW Modules Overview – Summary



21 - CONFIDENTIAL MATERIAL -

CRC
CRC Routines

NVM
NVRAM Manager

MEMIF
Memory Abstraction Interface

EA
EEPROM Abstraction

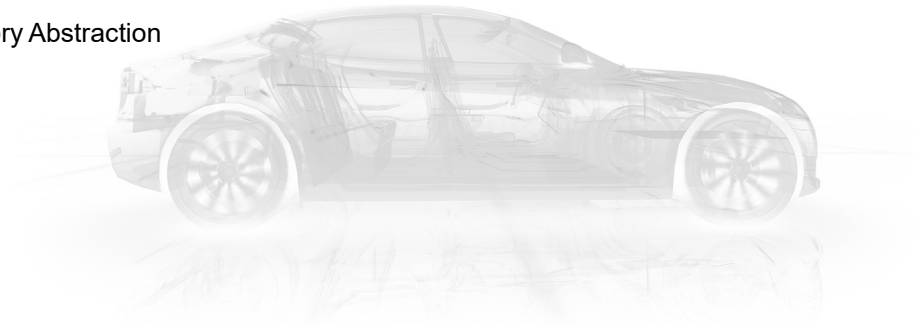
EEP
Internal/External EEPROM Driver

FEE
Flash EEPROM Emulation

FLS
Internal/External Flash Driver

Agenda

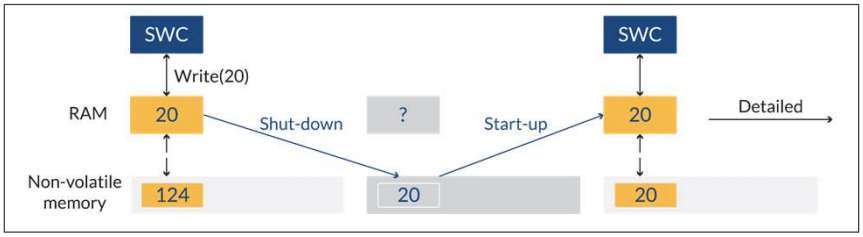
- Principles of Memory Technologies
- The MICROSAR.MEM solution
- **Memory Handling**
- NvBlockSwComponents
- Service Mapping
- Exercise 5 - Memory Abstraction



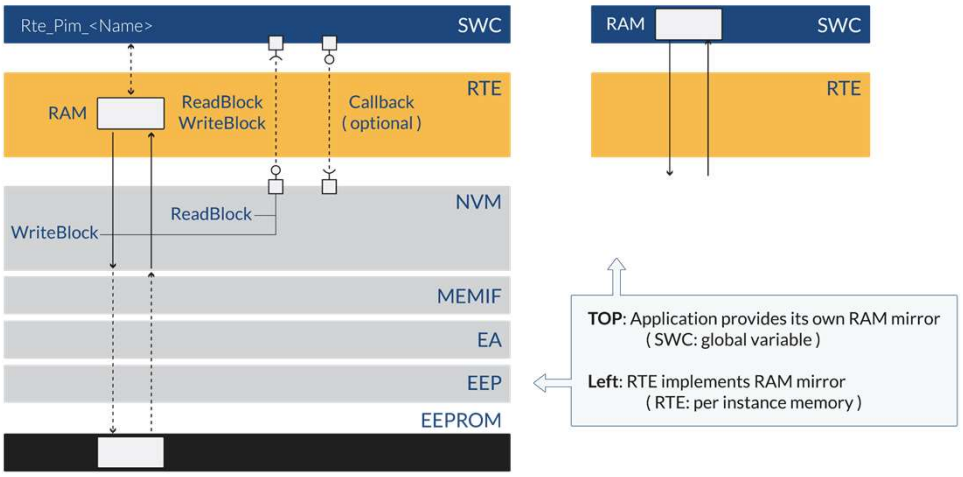
Memory Handling
AUTOSAR Memory Abstraction

Reason for putting data into Non-Volatile (NV) Memory

- Power-off data persistence
- Application works on RAM mirror of the NV data
- At startup data can be copied from NV memory to RAM
- At shutdown data is copied from RAM to NV memory
- Data can be read from or written to NV memory on request



Memory Handling
Interaction with the NVM Module



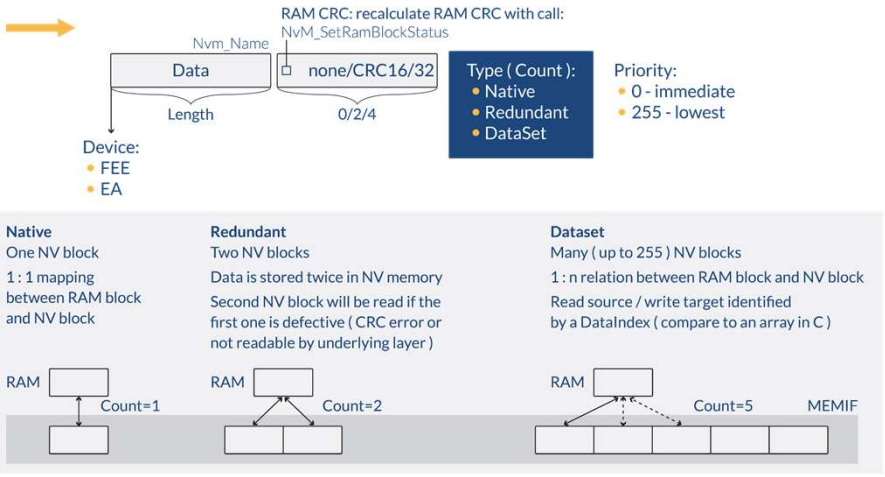
Memory Handling
The NVM Block

Base element is a NV Block

- It represents a memory area consisting of
  - NV user data
  - CRC value (optionally)
- Additional information necessary to define a Block
  - Name, Block ID, Size etc.

The way how the blocks are defined and administered in the NVM memory can be configured →

Memory Handling
NVM Block Parameters



26 - CONFIDENTIAL MATERIAL -

1. `NvM_SetRamBlockStatus()` triggers CRC calculation if configured and marks the RAM block as being changed and valid in order to trigger an NV block update during ECU shut-down
2. The currently used NV block for NVM datasets is configured by `NvM_SetDataIndex()`

The (re-)calculation of a block's CRC is done asynchronously by the `NvM_MainFunction()`. A CRC protected block's CRC value is calculated every time the block shall be written to NV memory. If a block is read from NV memory the CRC value is recalculated and compared to the one that was just read from NV memory.

If `NvM_SetRamBlockStatus(TRUE)` is called, the calculation of the CRC value over the RAM block's data is also initiated, unless the configuration option 'Calculate RAM CRC' was disabled for this block. The purpose of requesting recalculation of the RAM CRC with every call to `NvM_SetRamBlockStatus` is to provide the possibility to reuse the RAM data even if a soft reset (short power loss, watchdog reset) occurs.

Since CRC calculation is quite time consuming, especially if performed frequently and/or over large data blocks, it might be more applicable to disable it using the option mentioned above. Then data can only be restored after a reset if it did not change since last CRC recalculation (caused by a write request or after the last successful read request). In case the CRC in uninitialized RAM still is correct, the NVM will not reload the corresponding blocks but instead use the RAM copy.

Memory Handling
FEE Block Parameters

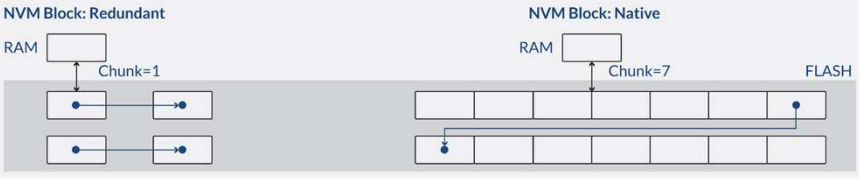
Chunk Block Count:
- 1, 3, 7, 15, 31, 63

Write Cycles:
- No effect on code
- Relevant for **FEE Optimization Assistant**

→ The chunk size should be increased if a higher number of write cycles is expected.

High Priority: this block must be written as fast as possible (e.g. crash data)

Organization of data blocks in "chunks" of configurable size (number of data block instances allocated at one time)
Chunks of one block form a linked list throughout the Flash memory sector
Binary search algorithm used within a chunk to increase performance



27 - CONFIDENTIAL MATERIAL -

In MICROSAR FEE, the chunk size must always be a number equal to $2^n - 1$ because the FEE performs a binary search inside one chunk to retrieve the most recent data instance.

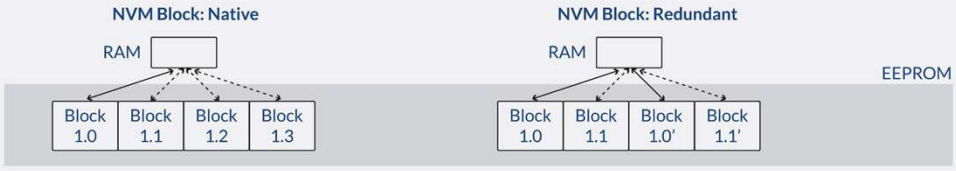
Memory Handling
EA Block Parameters

Write Cycles:
- How often will this block be written during life time of the system

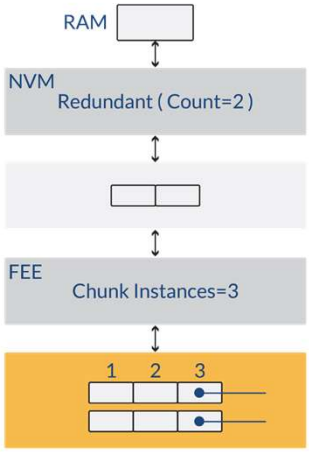
Verify Writing:
- Writing to EEPROM is verified or not
- Recommended for sensitive data

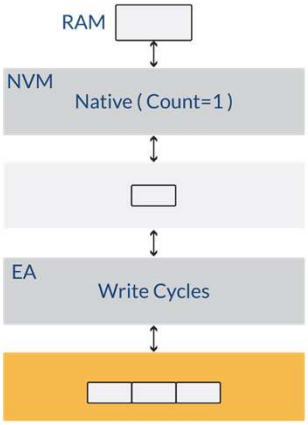
High Priority: this block must be written as fast as possible (e.g. crash data)

Information about how often memory can be rewritten
Physical write cycle value is basis for calculation of necessary instances of the EA block in the EEP
At runtime these blocks are written alternately



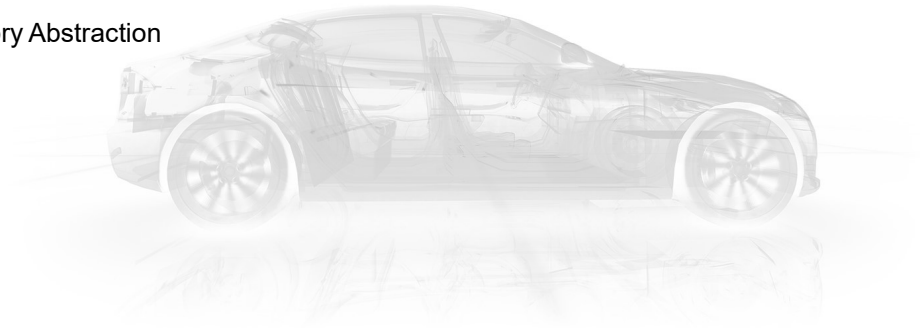
Memory Handling
Summary





Agenda

- Principles of Memory Technologies
- The MICROSAR.MEM solution
- Memory Handling
- **NvBlockSwComponents**
- Service Mapping
- Exercise 5 - Memory Abstraction



NvBlockSwComponents

Introduction to NvBlockSwComponents

- AUTOSAR 3.x: NVRAM handling is block oriented
  - Management at the NV block level is not sufficient especially if small data has to be stored individually
    - Inefficient usage of paged flash memories
  - Distribution of non-volatile data between SWCs is difficult to handle
    - PIMs or module-local static variables are private members of an SWC
  - Access over PIMs is not protected against concurrent access
- AUTOSAR 4.x addresses this situation by introducing optional NvBlockSwComponents

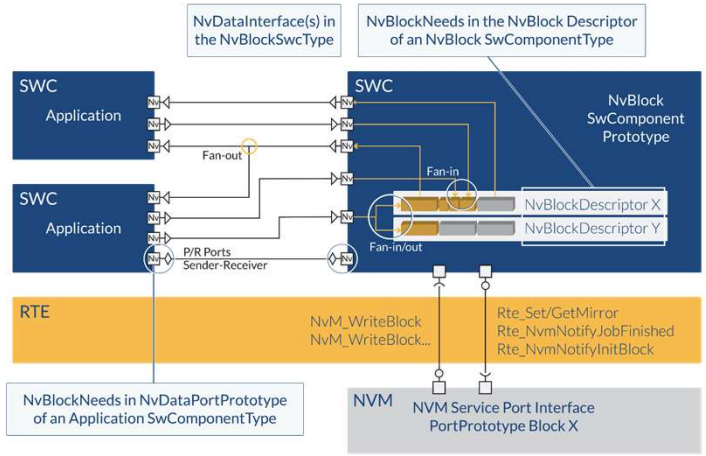
NvBlockSwComponents

Introduction to NvBlockSwComponents

- S/R oriented NVRAM handling
- Aggregation of several RAM variables into one NVRAM block (analogous to Signals and Signal Groups in Corn)
- "Fan-in" and "fan-out" of data elements to and from NVRAM blocks
  - Additional RAM is required because an extra buffer stage is required for handling the "fan-in"
  - "Fan-out": overall RAM consumption may be lower because the same data elements can be shared in the ECU with this feature (like calibration data)
  - NvData elements can fan into/out of the same (shared) NVRAM Block Element
- Flexible NvData distribution across multiple SWCs
- Protection against concurrent access by RTE

NvBlockSwComponents

Introduction to NvBlockSwComponents

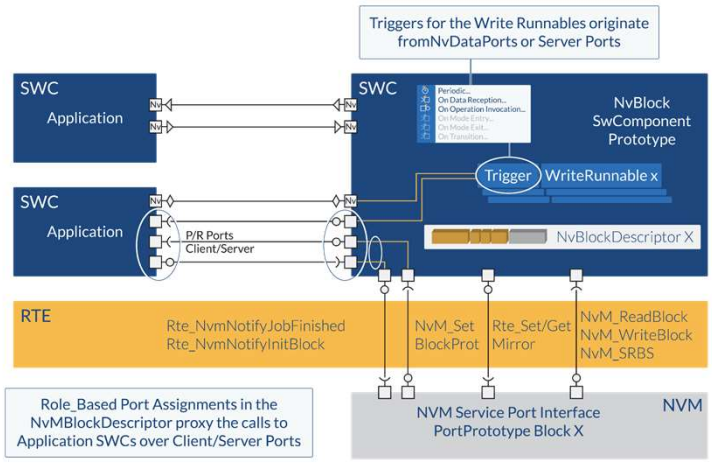


33 - CONFIDENTIAL MATERIAL -

The NvmBlockDescriptor maps 1:1 to an NVM Block. It contains a VariableDataPrototype (RAM mirror) and optionally a ParameterDataPrototype (default ROM data).

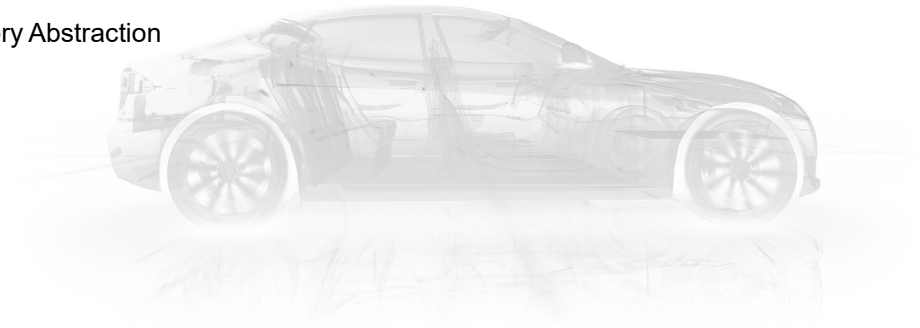
VariableDataPrototypes must exist in just one partition.

NvBlockSwComponents
Control Flow



Agenda

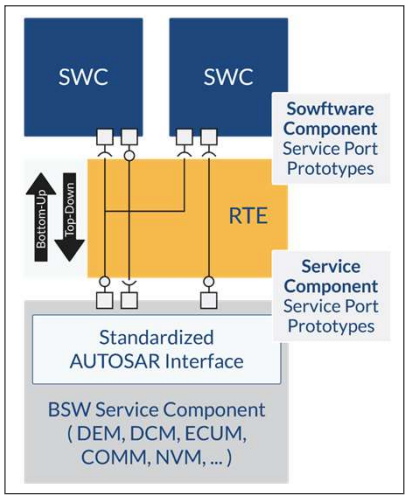
- Principles of Memory Technologies
- The MICROSAR.MEM solution
- Memory Handling
- NvBlockSwComponents
- **Service Mapping**
- Exercise 5 - Memory Abstraction



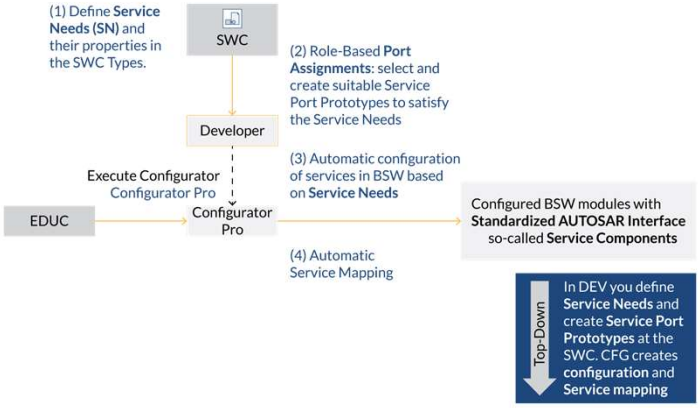
Service Mapping

Service Mapping

- Assigning Service Ports of Software Components to Service Ports of Service Components or vice versa
- Bottom-up service configuration
  1. Configure a service module in the basic software
  2. Assign the respective service port prototypes to the SWCs where appropriate
- Top-down service configuration
  1. Use Service Needs (Document at the SWC level what is required from the BSW as a service)
  2. Derive a service module configuration based on these requirements
  3. Configure the basic software service module in detail



Service Mapping
Service Component Configuration



Agenda

- Principles of Memory Technologies
- The MICROSAR.MEM solution
- Memory Handling
- NvBlockSwComponents
- Service Mapping
- **Exercise 5 - Memory Abstraction**