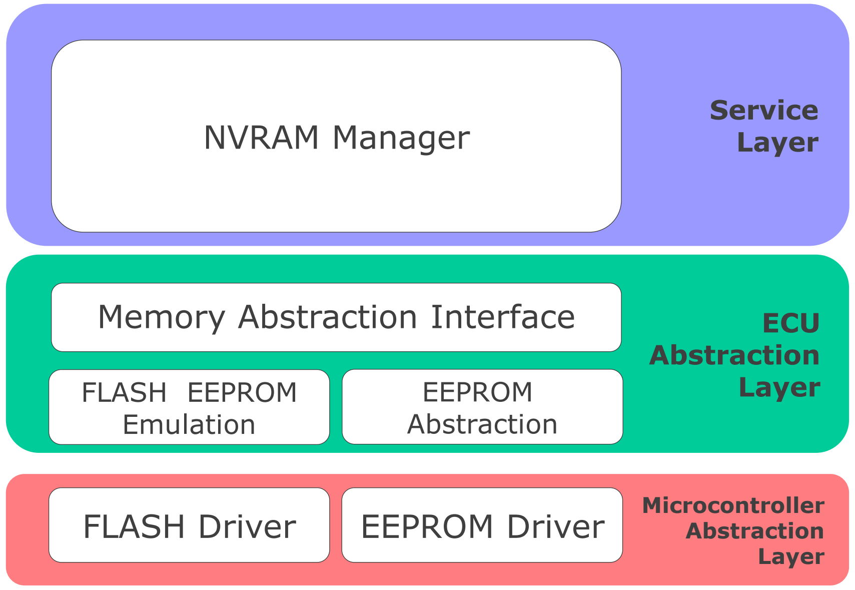
EB tresos classic AUTOSAR training - Memory stack

Chapter overview

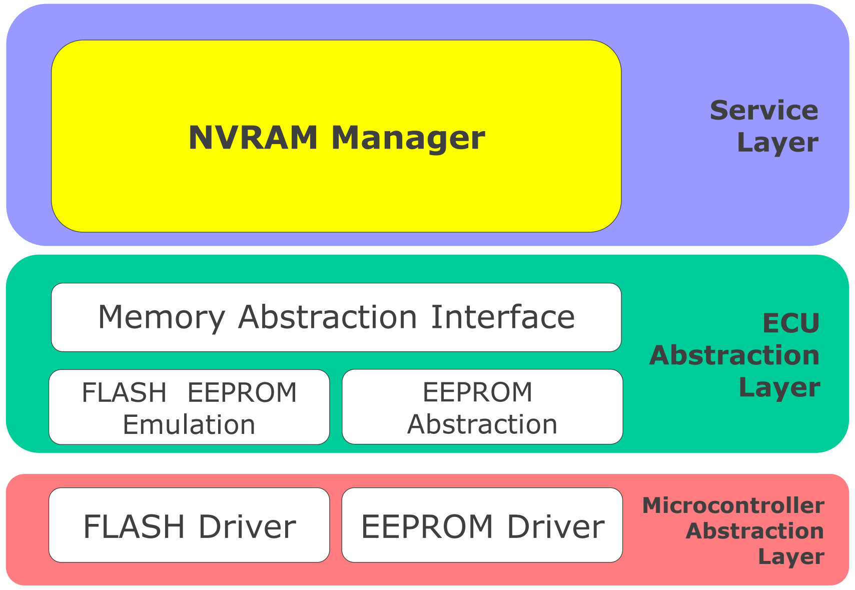
- NVRAM Manager (NvM)
- Memory Abstraction Interface (MemIf)
- FLASH EEPROM emulation stack
- EEPROM stack



Non volatile Ram manager (NvM)

NVRAM Manager overview

- NVRAM Manager manages all data requests to the EEPROM / Flash used as non-volatile memory



Elektrobit

NVRAM Manager features

- NVRAM Manager handles async./sync. memory requests like:

$$\left.\begin{array}{l}\text{– Read}\\ \text{– Write}\\ \text{– Erase}\\ \text{– Invalidate}\end{array}\right\}\text{ asynchronous}$$

$$\left.\begin{array}{l}\text{– Validation}\\ \text{– Status reports}\\ \text{– Data management settings}\end{array}\right\}\text{ synchronous}$$

What is a block?

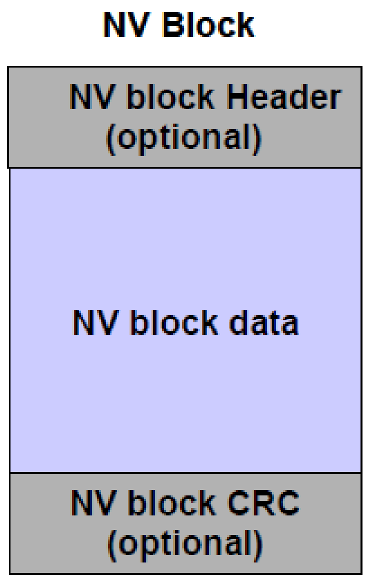
- There are NvM blocks and related to them there are Fee/Ea blocks;
- A block is an abstract storage concept that is uniquely identified by a number/identifier, and represents a way of filing user data
- Each NvM block consists of 2 or 3 storage entities:
  - a RAM block = a dedicated space in the RAM memory;
  - a NV block = a dedicated space in non-volatile memory;
  - and optionally a ROM block = represents default data in ROM memory;
- The NV block (non-volatile block) as seen by the NvM, actually translates as one or many Fee/Ea blocks;
- Ea/Fee blocks have a direct connection to the physical non-volatile memory space dedicated for them;
- From Ea/Fee block perspective the RAM component is always temporary, given by NvM as a pointer on job request;

NvM blocks

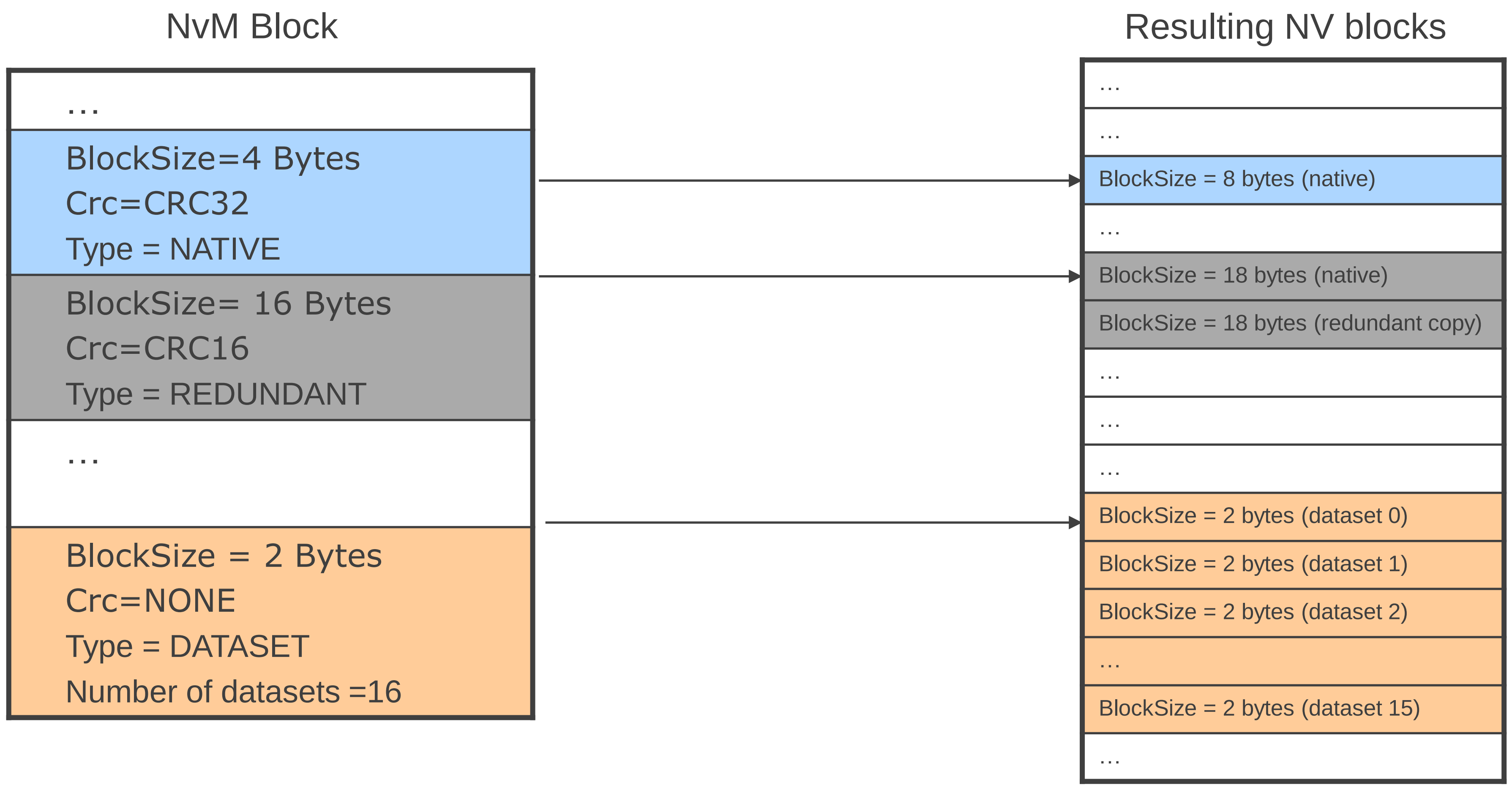
- The NVRAM Manager defines three types of blocks
  - – Native Block
    - - Translates as single Ea/Fee block
  - – Redundant Block
    - - Translates as two Ea/Fee blocks that keep the same content
  - –Dataset Block
    - - Consist of multiple Ea/Fee blocks
    - - This type is useful to store data arrays, like in the array structure you access the block by using a data index

- The structure of a NV block is composed from three parts
  - – Header (optional) - StaticId
  - – Data
  - – CRC (optional)

Note: The lower layer Ea/Fee will consider this entire structure as “data”

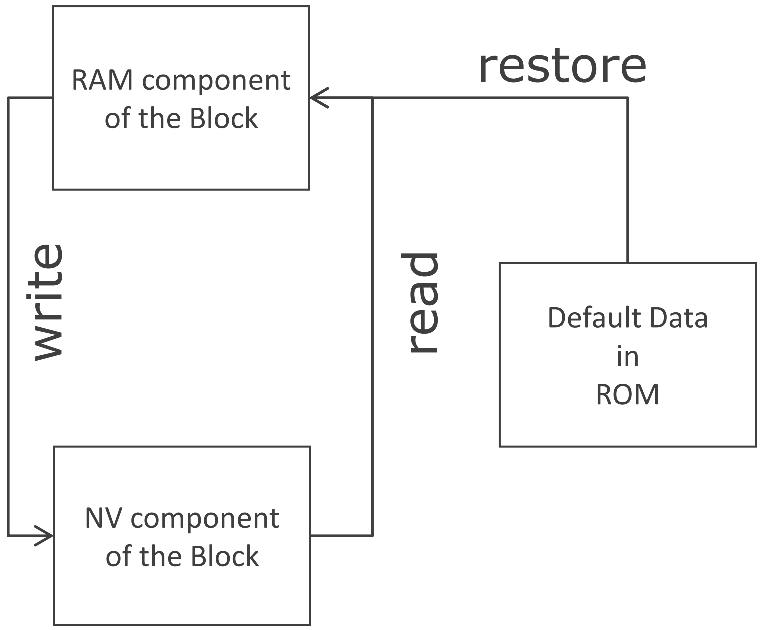


Mapping and Block Sizes


NvM Block
…
BlockSize=4 Bytes
Crc=CRC32
Type = NATIVE
BlockSize= 16 Bytes
Crc=CRC16
Type = REDUNDANT
…
BlockSize = 2 Bytes
Crc=NONE
Type = DATASET
Number of datasets =16
Resulting NV blocks
…
…
BlockSize = 8 bytes (native)
…
BlockSize = 18 bytes (native)
BlockSize = 18 bytes (redundant copy)
…
…
…
BlockSize = 2 bytes (dataset 0)
BlockSize = 2 bytes (dataset 1)
BlockSize = 2 bytes (dataset 2)
…
BlockSize = 2 bytes (dataset 15)
…

RAM / ROM / NVRAM components of a block

- Read Request
  - NvM copies the persistently stored data into the variable in RAM for the requested block
  - In case of data corruption, the NvM may copy the default data which is stored in the ROM block
- Write Request
  - NVM copies the data from the variable in RAM into the corresponding NV block

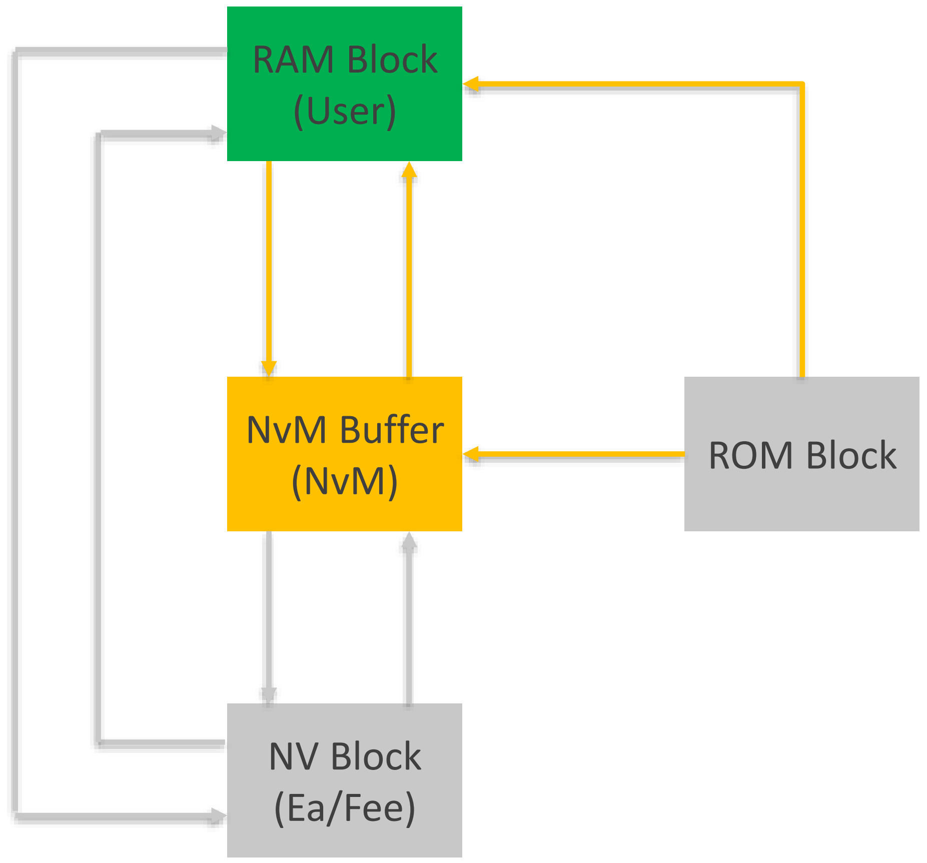


Note: the data flow between user variables and the non-volatile memory through Memory Stack is called “synchronization”.

The synchronization types are depicted in the next slides.

Implicit synchronization – Permanent/Temporary RAM Block

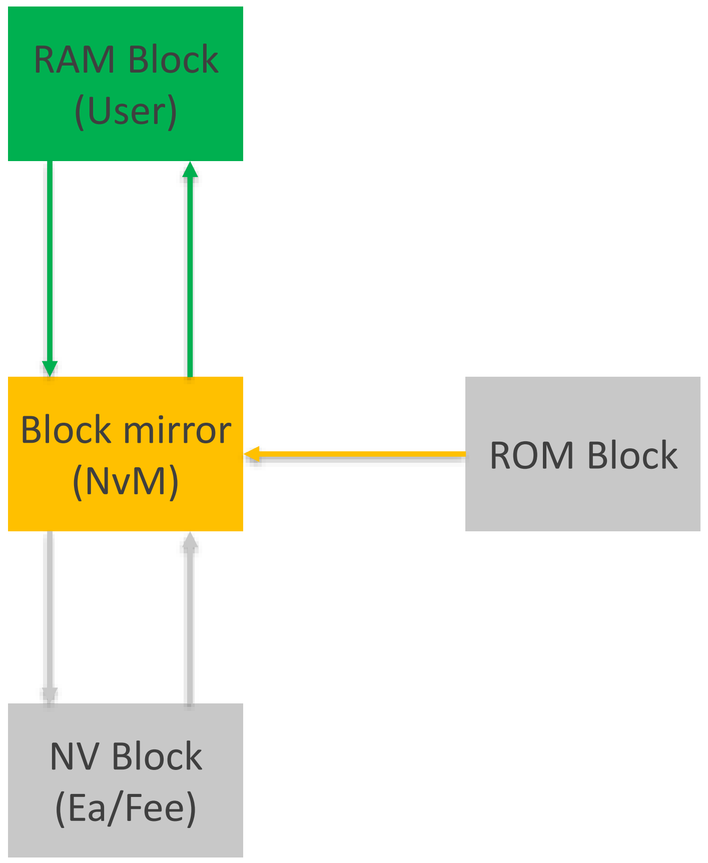
- For permanent RAM block, NvM "knows" the address of the RAM block from configuration
- For temporary RAM block, the user must pass the address of the RAM block on job request
- The data transits through NvM's buffer if additional processing is needed e.g., CRC needs to be computed and appended to the data.
- If no additional processing is needed the lower layer (Ea/Fee) performs the data flow under NvM's control.



Elektrobit

Explicit synchronization

- The user is responsible for copying the data to/from its own RAM block
  When NvM requests it through explicit sync callbacks
- NvM doesn't have direct access to the user's data
- NvM buffer is used as a mirror of the user's RAM block



Startup and shutdown

- Startup:
  - `NvM_ReadAll()` is called by BswM during startup
  - Only blocks with `NvMSelectBlockForReadAll= true` are read

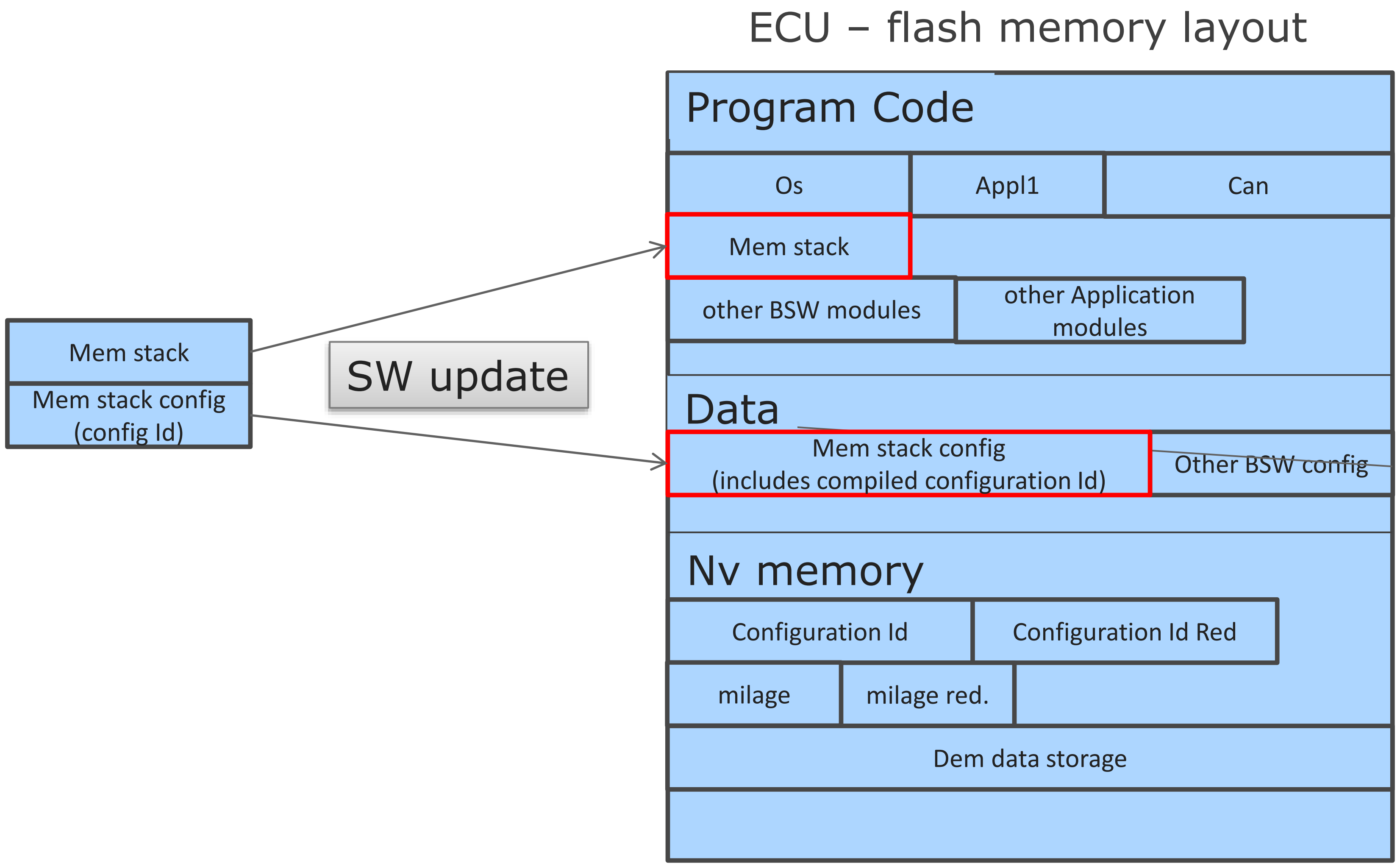
- Shutdown:
  - `NvM_WriteAll()` is called by BswM during shutdown
  - `NvM_WriteAll()` writes block to NVRAM if
    - `NvMSelectBlockForWriteAll = true`, and
    - a block is marked as changed by function `NvM_SetRamBlockStatus()`, and
    - a block is not written as protected and valid
  - Can be canceled with `NvM_CancelWriteAll()`

Default blocks: the configuration ID

One NvM block is always necessary for a valid configuration

- Configuration ID block
  - Stores the Configuration ID (2 bytes)
  - Configuration ID identifies the memory layout of the data stored in the NV memory
  - Requires at least two NV blocks since it must be a redundant block with CRC

(Compiled-) and configuration ID

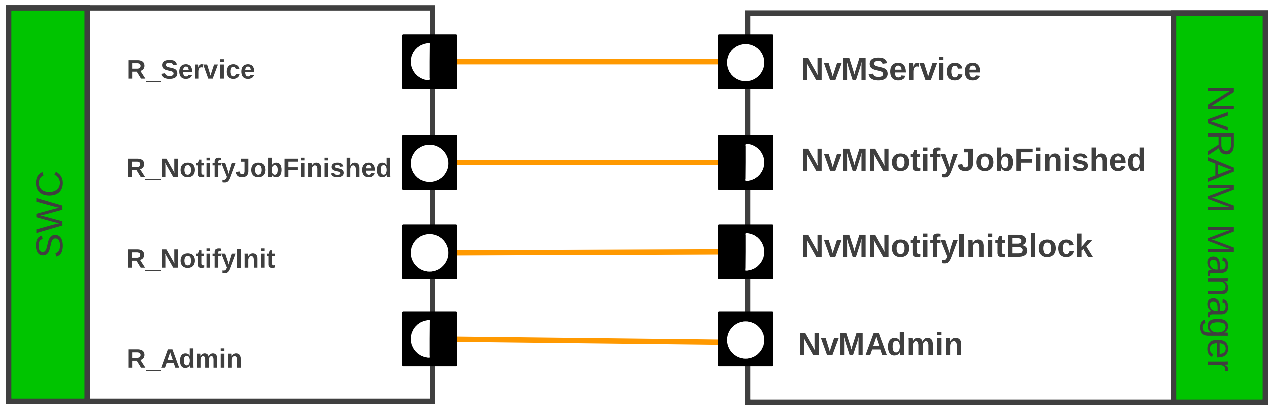


Compiled Configuration ID

- The Compiled Configuration ID identifies the memory layout of the current configuration
- During `NvM_ReadAll()` the NvM compares the Compiled Configuration ID with the Configuration ID stored in the first NvM block
- If the two values are different it means that NvM configuration has changed
- In this case
  - All “non resistant to SW change” blocks (configuration checkbox for each block) will be initialized with default values (from configured ROM block)
  - All “resistant to SW change” blocks will be read out form non-volatile memory (if possible – as NvM memory layout might have changed)

NvM – ports (individually configurable for each Block)

- Port NvMService with the following operations:
  - Write, WritePRAMBlock
  - ReadBlock, ReadPRAMBlock
  - EraseBlock
  - InvalidateNvBlock
  - GetDataIndex, SetDataIndex
  - GetErrorStatus
  - RestoreBlockDefaults, RestorePRAMBlockDefaults
- Port NvMMirror
  - ReadRamBlockFromNvM
  - WriteRamBlockToNvM
- Port NvMAdministration:
  - SetBlockProtection
- Port NvMNotifyJobFinished:
  - JobFinished (Callback that is called when a job has finished)
- Port NvMNotifyInitBlock:
  - InitBlock (This callback is called if the initialization of a block has completed)



NvM – further features

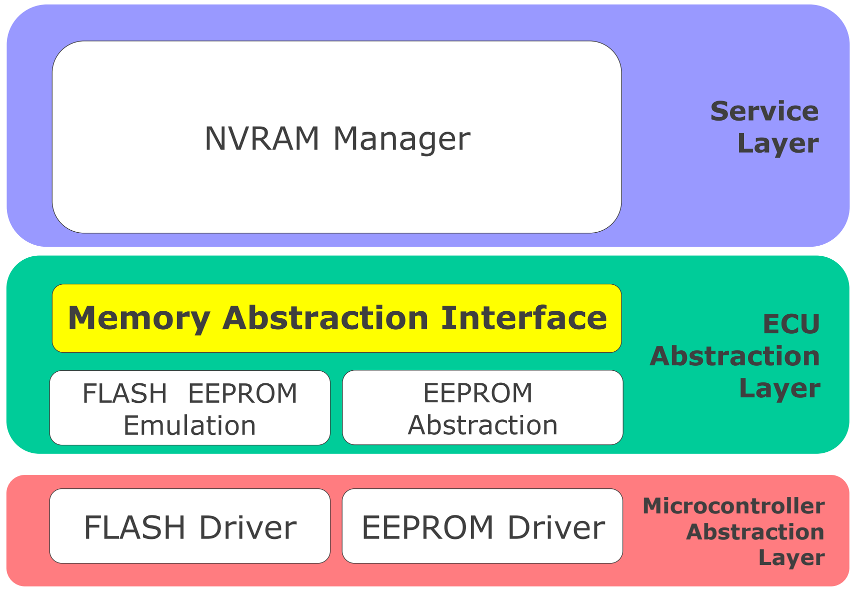
- Write Prioritization for each block (priorities 0..255)
- Write Protection after First Write
- Write Protection of individual blocks
- Automatic Data Repair for redundant blocks
- Four CRC settings for each block (None, CRC8, CRC16, CRC32)
- Static Block Id Check
- Explicit Synchronization
- *Background block check*

Memory abstraction interface (MemIf)

Elektrobit

Memory Abstraction Interface - MemIf

- MemIf abstracts function calls to the underlying memory abstraction modules
  - EEPROM abstraction (Ea)
  - Flash EEPROM Emulation (Fee)
- MemIf doesn‘t require any initialization
- MemIf doesn‘t support run-time configuration

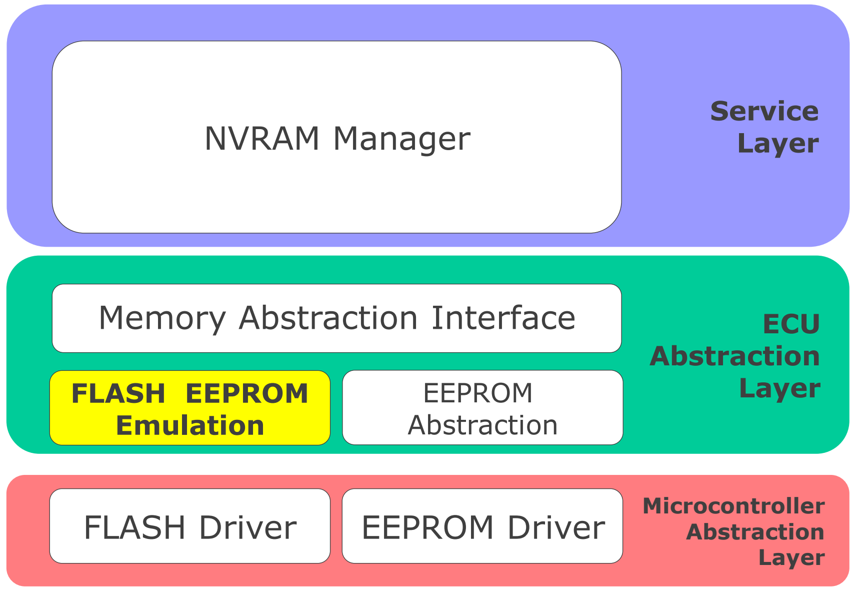


EEPROM stack
Flash EEPROM emulation stack

Elektrobit

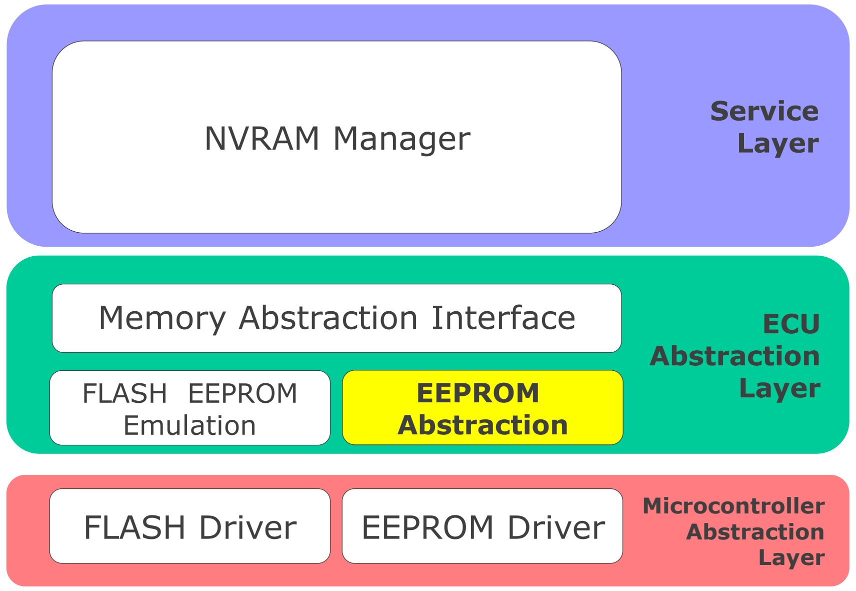
Flash EEPROM emulation (Fee)

- Abstracts device specific addressing scheme and segmentation
- Provides a virtual addressing scheme and segmentation
- Provides a „unlimited“ number of write/erase cycles to the MemIf / NvM



EEPROM Abstraction(Ea)

- Manage limitations of erase/write cycles for extended EEPROM write-cycles
- Check for valid data
- Mapping of NVRAM blocks to physical addresses

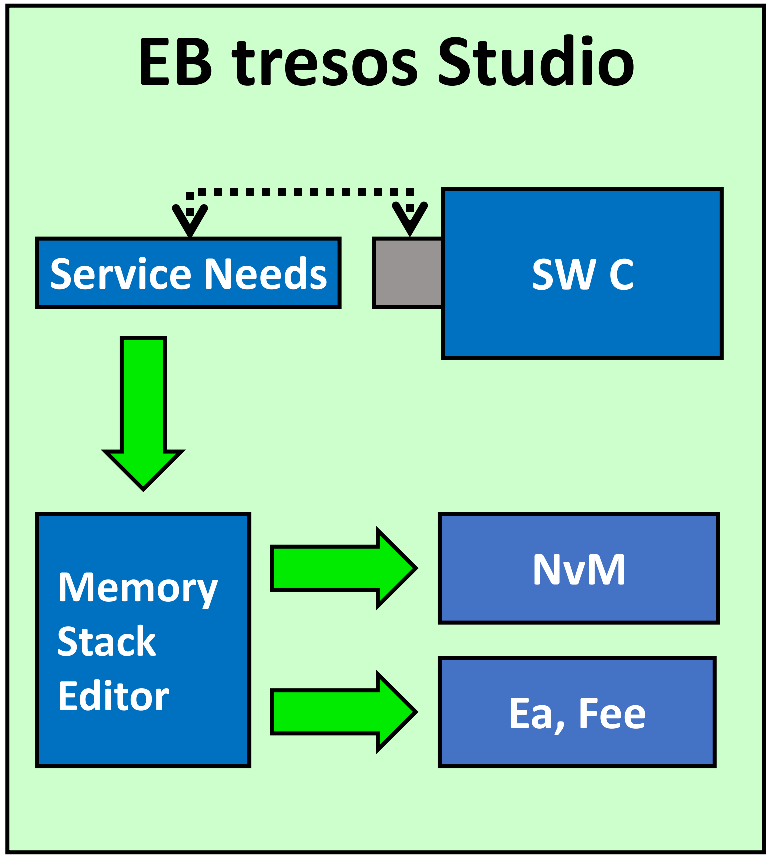


EB tresos Studio Memory Stack Editor

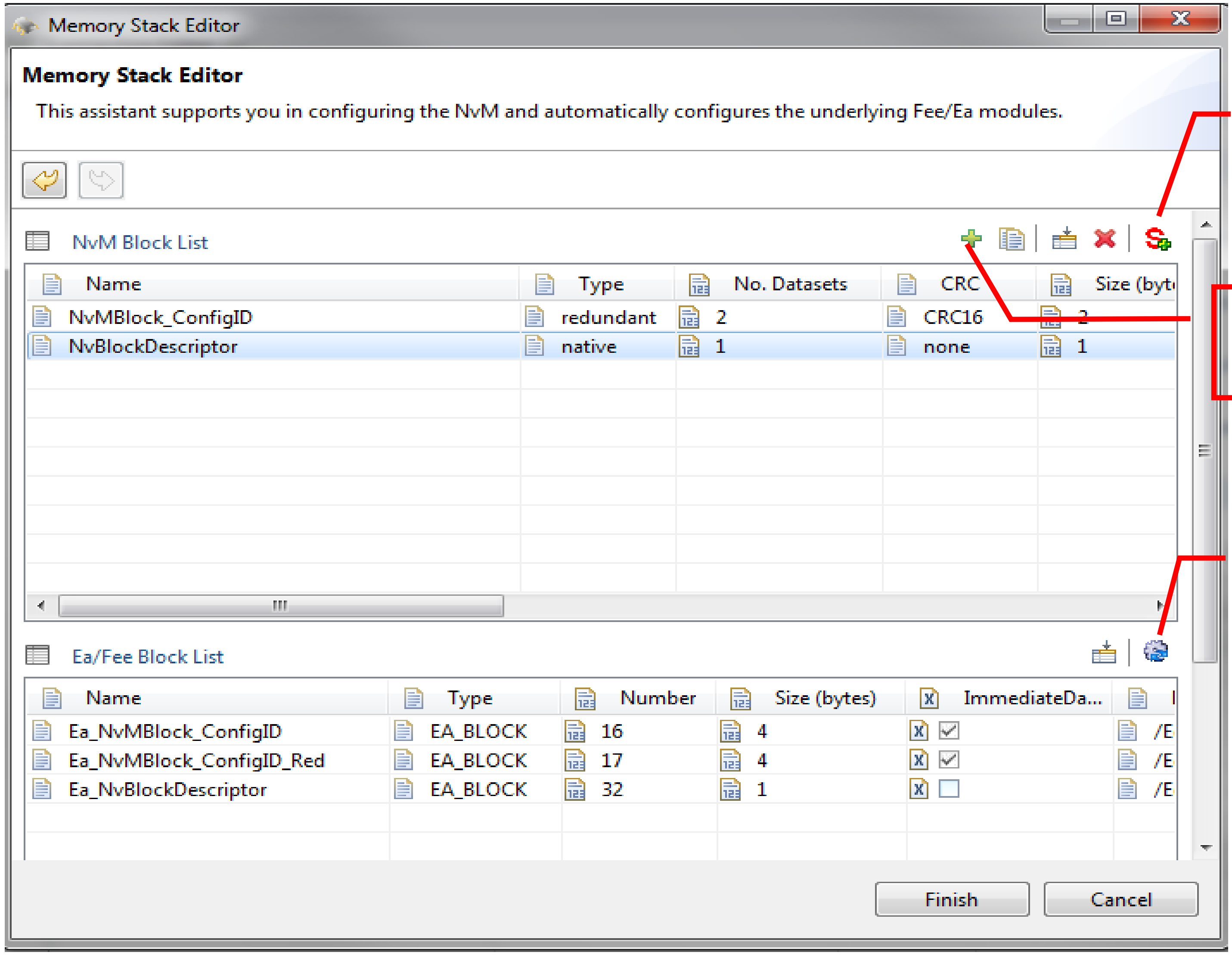
Elektrobit

Memory Stack Editor

- Evaluate NvM Service Dependencies of Software Components
- Convenient manual configuration of memory stack modules (NvM, Ea, Fee)



Memory Stack Editor



Memory Stack Editor

This assistant supports you in configuring the NvM and automatically configures the underlying Fee/Ea modules.

NvM Block List

Name	Type	No. Datasets	CRC	Size (byt
NvMBlock_ConfigID	redundant	2	CRC16	2
NvBlockDescriptor	native	1	none	1

Ea/Fee Block List

Name	Type	Number	Size (bytes)	ImmediateDa...	I
Ea_NvMBlock_ConfigID	EA_BLOCK	16	4	✓	/E
Ea_NvMBlock_ConfigID_Red	EA_BLOCK	17	4	✓	/E
Ea_NvBlockDescriptor	EA_BLOCK	32	1	☐	/E

Evaluate Service Dependencies

Add new NvM Block

Automatically create and update Ea & Fee Blocks

Summary

- Nonvolatile memory manager (NvM)
- Memory Interface (MemIf)
- Flash EEPROM emulation stack (Fee, Fls)
- EEPROM stack (Ea, Eep)
- Memory Stack Editor

Get in touch!

sales@elektrobit.com
www.elektrobit.com