**NVM Reference Manual**

1. INTRODUCTION
   1. PURPOSE

The stream of the document is to provide the overview of how to configure and provide NVM Services for Algos in GWM project.

1. Overview

In GWM project we have an External EEPROM. So, we will try to cover the EA and NVM together. In this document we will discuss in detail w.r.to

* + - Memory Stack
    - Algo Input PIM’s and NV Block Needs Config
    - NVM and EA Config
    - Layout Updation, .cfg, s37 Generation
    - Testing

1. Types of Memories

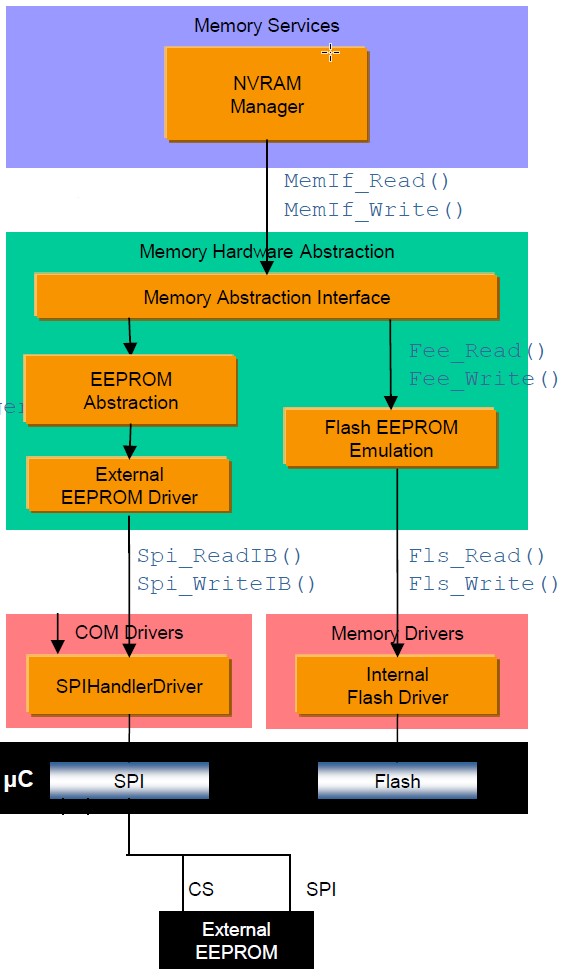
▪ The Memory Stack deals with

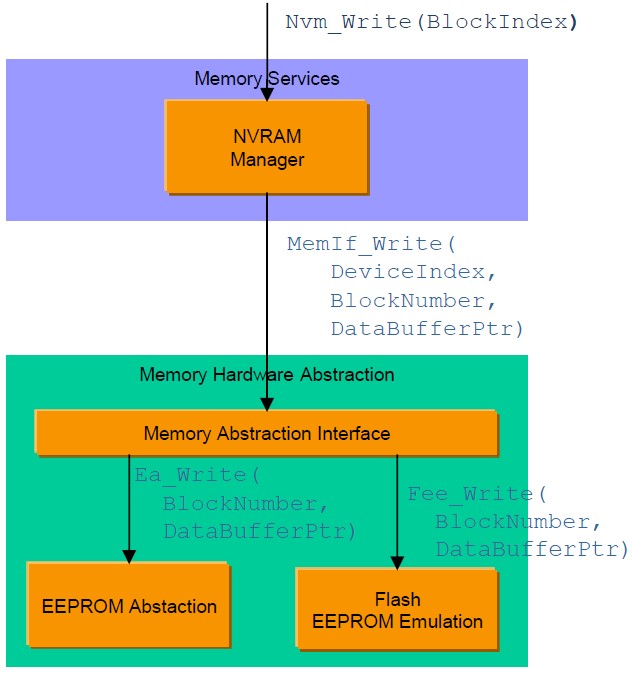
* + - * + ROM
        + RAM
        + NV Memory (Flash/ Eeprom)
      * ROM memory exists in FLASH
      * RAM memory as usual in RAM ▪ NV memory (Flash & Eeprom) divided into pages, blocks etc.



1. Memory Stack

* + - The Stack of modules for memory access
    - Flash & Eeprom are available in AUTOSAR
    - Stack includes memory manager, abstraction & memories
    - External memories can be interfaced with SPI
    - Provides Synchronous/ Asynchronous services to memory access





NVM Data

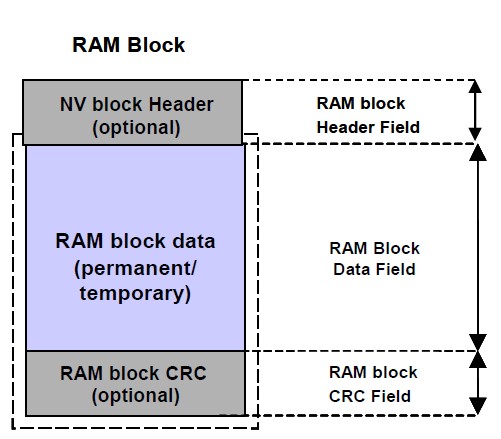
* + - Data whose values are changeable but available across the power cycles needs to be stored in the Non-volatile memory.
    - NV Data is that data inside the Non-volatile memory.
    - Application can access this Non-volatile memory only via the NVRAM Manager (NvM). This module provides the required services (synchronous / asynchronous) for the management and maintenance of the data.

* 1. Basic Storage Objects

A “Basic Storage Object” is the smallest entity of a “NVRAM block”. A “Basic Storage Object” can reside in different memory locations (RAM/ROM/NV memory).

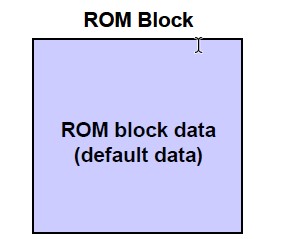
**4.1.1RAM Block**

It represents the part of a “NVRAM Block” which resides in the RAM.



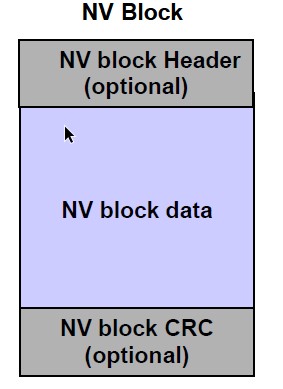
**4.1.2 ROM Block**

The ROM block is a basic storage object, resides in the ROM (FLASH) and is used to provide default data in case of an empty or damaged NV block. Contents of ROM Block are of persistent nature, which can’t be modified during program execution and resides in ROM/Flash.



**4.1.3 NV Block**

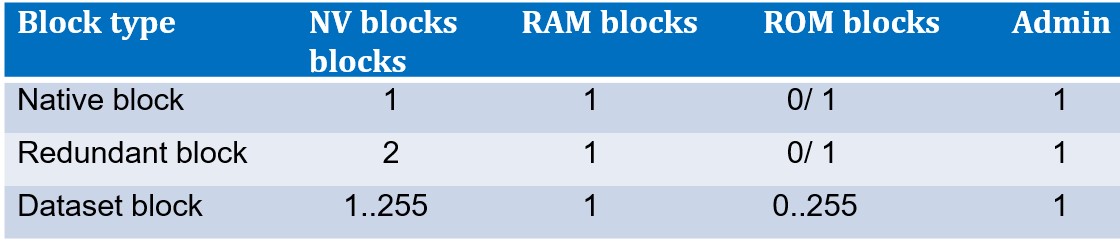
* + - The “NV Block” is a “Basic Storage Object”. It represents the part of a “NVRAM Block” which resides in the NV memory. The “NV Block” is a mandatory part of a “NVRAM Block”.
    - Contents of NV Block are of persistent nature that can be modified during program execution and resides in the Flash. It is composed of NV user data and (optionally) a CRC value and (optionally) a NV block header. It is used to hold the live data that are stored periodically/on request.



**4.1.4 Administrative Block**

* + - The Administrative block shall be located in RAM and shall contain a block index which is used in association with Dataset NV blocks. Additionally, attribute/error/status information of the corresponding NVRAM block shall be contained.
    - Contents of Administrative Block are of non-persistent nature and resides in the RAM.
    - Administrative block shall be invisible for the application and is used exclusively by the NvM module for security and administrative purposes of the RAM block and the NVRAM block itself.

4.2 Block Management Types



* + - The Native NVRAM block is the simplest block management type. It allows storage to/retrieval from NV memory with a minimal overhead.
    - In addition to the Native NVRAM block, the Redundant NVRAM block provides enhanced fault tolerance, reliability and availability. It increases resistance against data corruption. The Redundant NVRAM block consists of two NV blocks, a RAM block and an Administrative block.
    - The Dataset NVRAM block is an array of equally sized data blocks. The application can at onetime access exactly one of this data blocks.

1. Arxml Import and NV Block Needs Configuration

* + We get the Arxmls from Algo teams for every release. The Arxmls are to be Imported into Developer.
  + It may be required that a sequence of Algo Arxmls to be Imported which will be provided by the Algo Team.
  + The Arxmls have the PIMS that are to be mapped to NV Block needs and Port Assignments to be done.
  + Nv Blockneed should be created and the parameters highlited in fig NV Block Config to be ensured.
  + Any changes in the Signals/Parameters in the PIM of an Algo may be provided in Next version of Algo.
  + The only updates we have for Algo w.r.to NVM are o Addition of New PIM o Existing Block Size Update

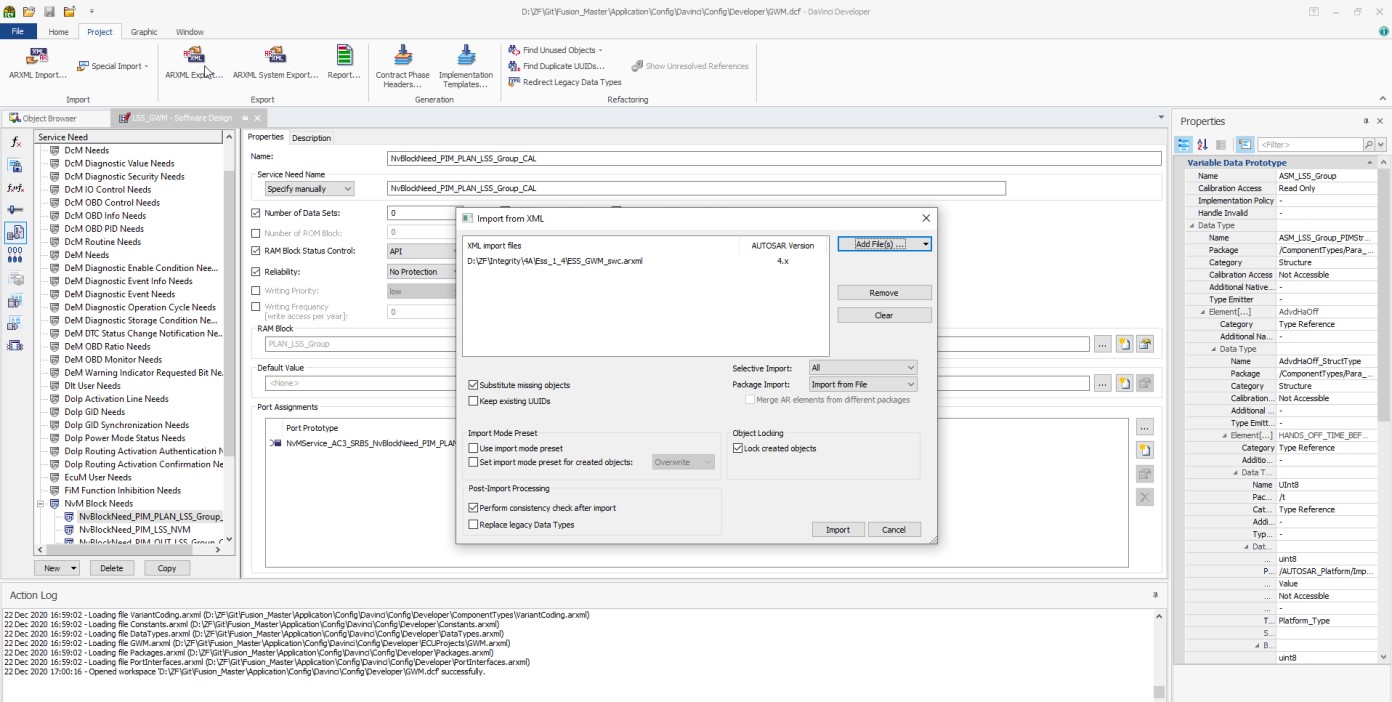
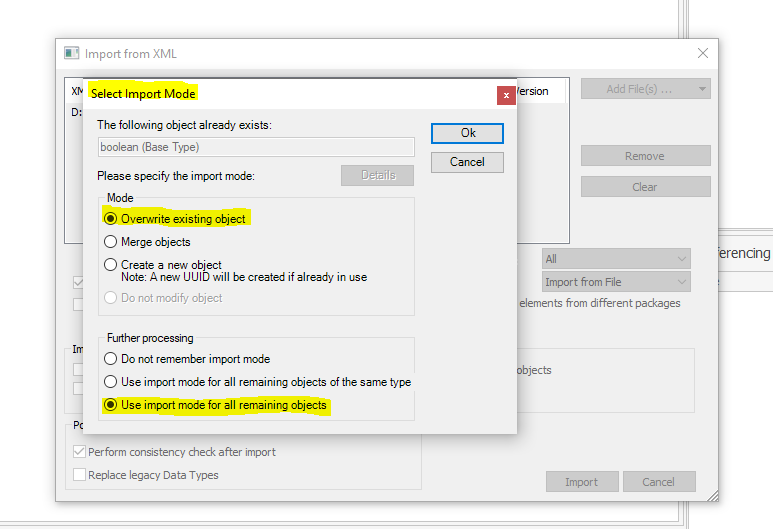
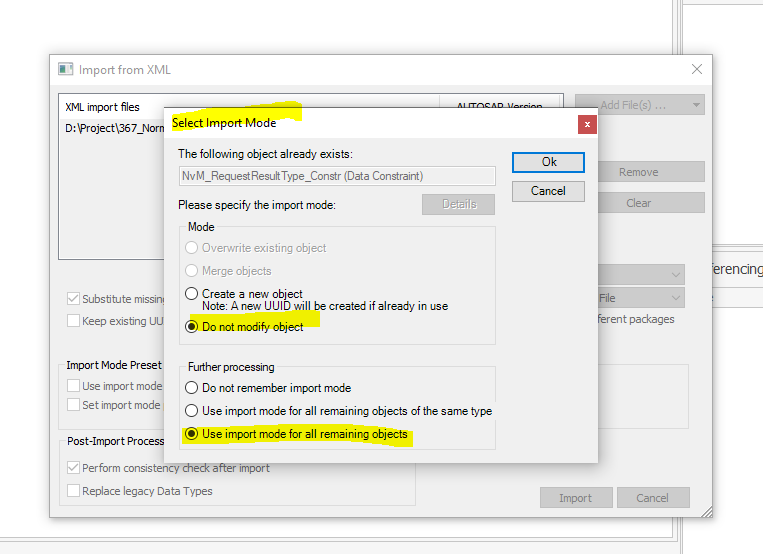


Fig: Arxml Import





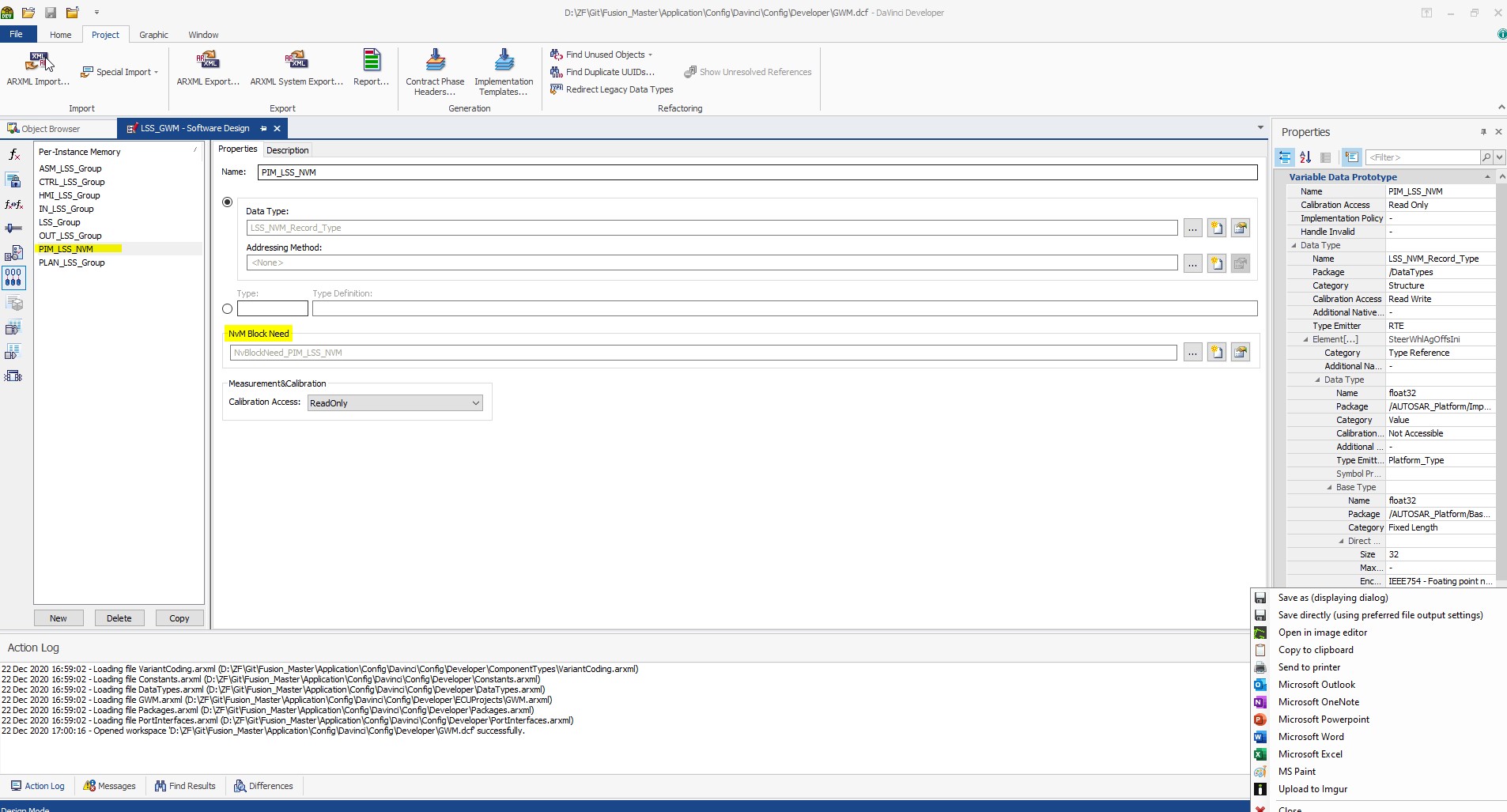


Fig: PIM

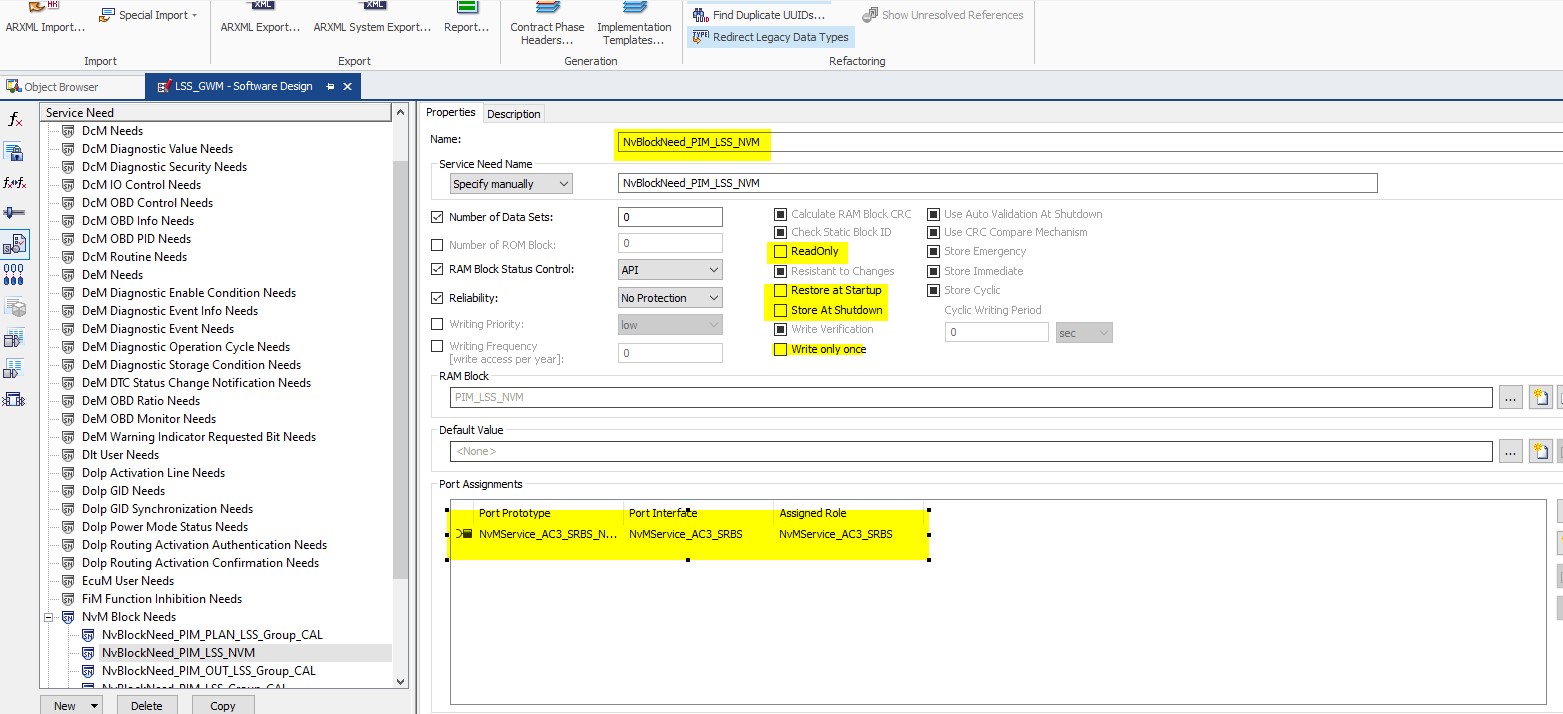


Fig: NV Block Need to be Configured

1. NVM and EA Configuration

Once the PIM is available and the NVBlock Need and Port Assignment are in place the moment we Synchronize the Configurator, we will have the New NVM and EA Blocks.

* + For EA we need to map the New Block to corresponding EA Partition.
  + The following parameters are to be ensured of NVM configuration.
    - CRC Required/Not
    - Select for ReadAll
    - Selection of Write All
    - Block length (Will be Automatically picked after validation)
    - RAM blocks reference (Will be Automatically picked after validation)
    - Block Management Type (Native or Redundant)

1. NVM Layout, .Cfg, .s37 Generation
   * There is a layout for GWM project where we have the different partitions for different blocks of Boot, EyeQ, BSW, NVM, CAL.
   * Also, there is a reserved block for each partition to accommodate any new blocks requested.
   * Whenever a new block is added the reserved block size needs t be reduced to accommodate the layout doesn’t exceed the last byte of the EEPROM.
   * There is a format for updating the default values to a NV Block, we need to add the following lines in .cfg file for every block.

HEX\_TOKEN = Reserved\_StartMarker+0, 0xF0

RHEX\_TOKEN = Reserved+0,0xFF,2051

HEX\_TOKEN = Reserved\_EndMarker+0,0xF0

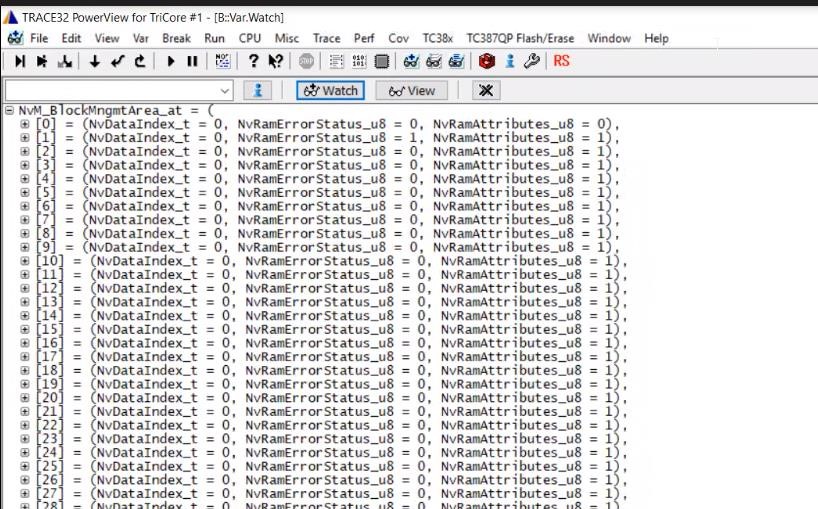
* + The Sart Marker and End Marker are also Mandatory for every block and there comes in a CRC item if it is requested.
  + The addresses generated in an layout Excel are to be placed in the .cfg file as well.
  + There is an exe “nvmsrecgen\_LittleEndian.exe” that will generate the .s37 out of the .cfg Input, we need to pass the following command in cmd to generate the .s37 flashable.

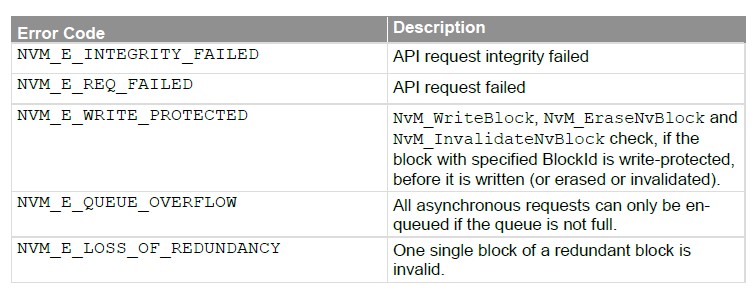
nvmsrecgen\_LittleEndian.exe NvmMap\_GWM\_Test\_64k.cfg

* + We have 3 to 4 .cfg files, the same process to be followed for all to generate .s37.

1. Testing Pointers:

* + Please check the NvRamErrorStatus\_u8 for all the NVM Blocks. The NVM blocks status is verified in the following variable in Trace.Also write to block power OFF and ON read from block written block values it should be same.
  + None of the blocks should have the NvRamErrorStatus as 3 or 5.





* + The status is updated in NVM static code with the following Macros.

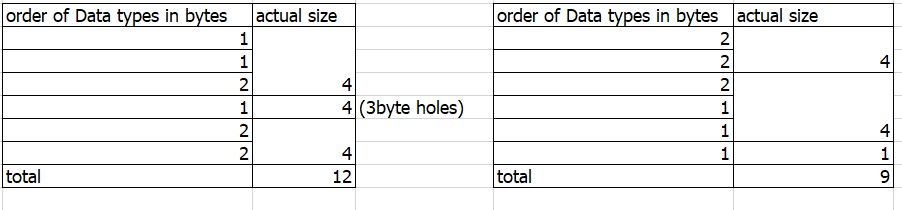


1. General NVM issues, important points and debugging steps:

9.1 **Holes in structures of PIMS in NVM**:

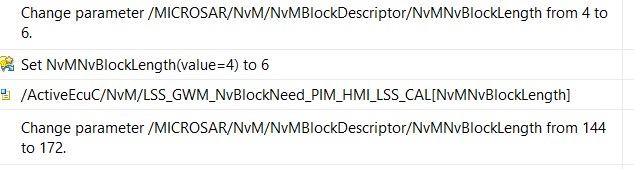
Holes in PIMS leads to difference in sizes provided in template by algo and generated sizes from developer and leads to wrong default values when we read them.

**Root cause and introduced at**: Owner of algorithm are resposibele while creating members of PIMS,if they do not take care of order of members like 32 bit,16bit followed by 8bit it will result in holes.



Identification of holes:

After importing the sizes of PIMS blocks may not be same as that mentioned in requirement.Plese verify the size of blocks in configurator after importing like below with actual allocated by compiler.



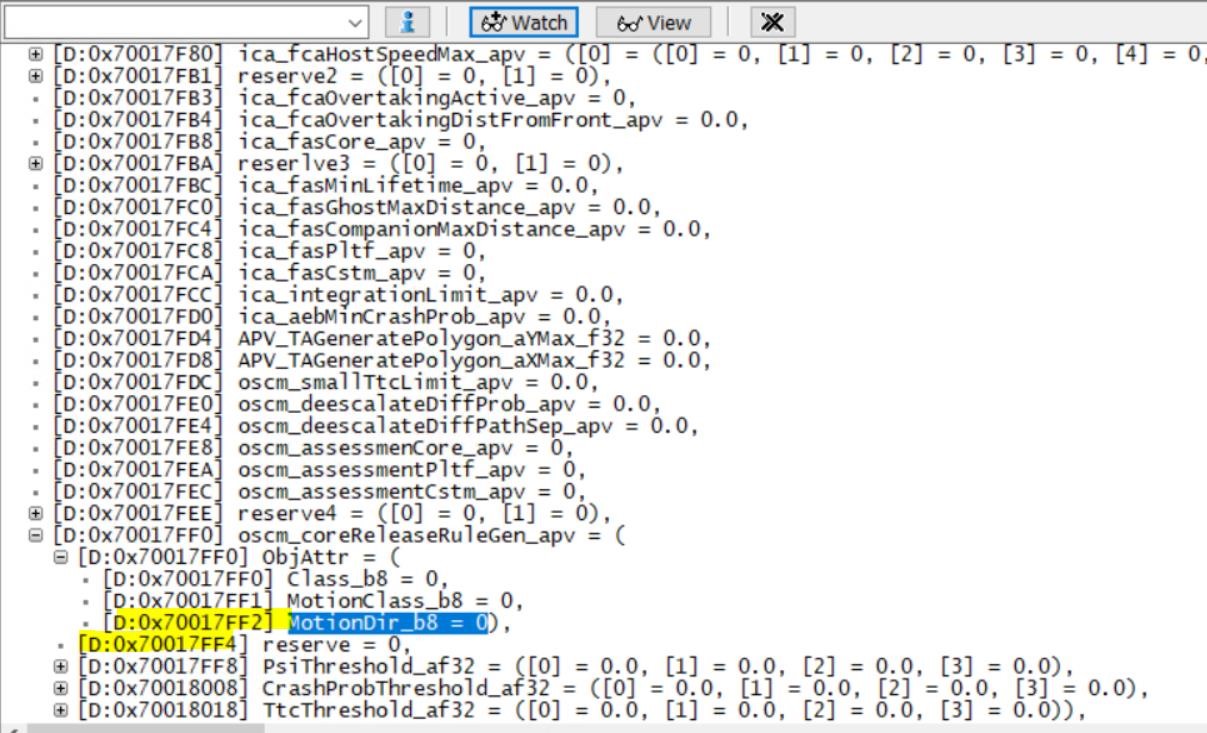
In trace we can find size of actual PIM allocated by using size of as below



If there is any difference in particular block we can inform algo owner to take of holes.

Example of holes in fusion project:

In below PIM the size reserve should be ideally 0x70017ff3, but it I allocated at 0x70017ff4.There id hole introduced at 0x70017ff3.

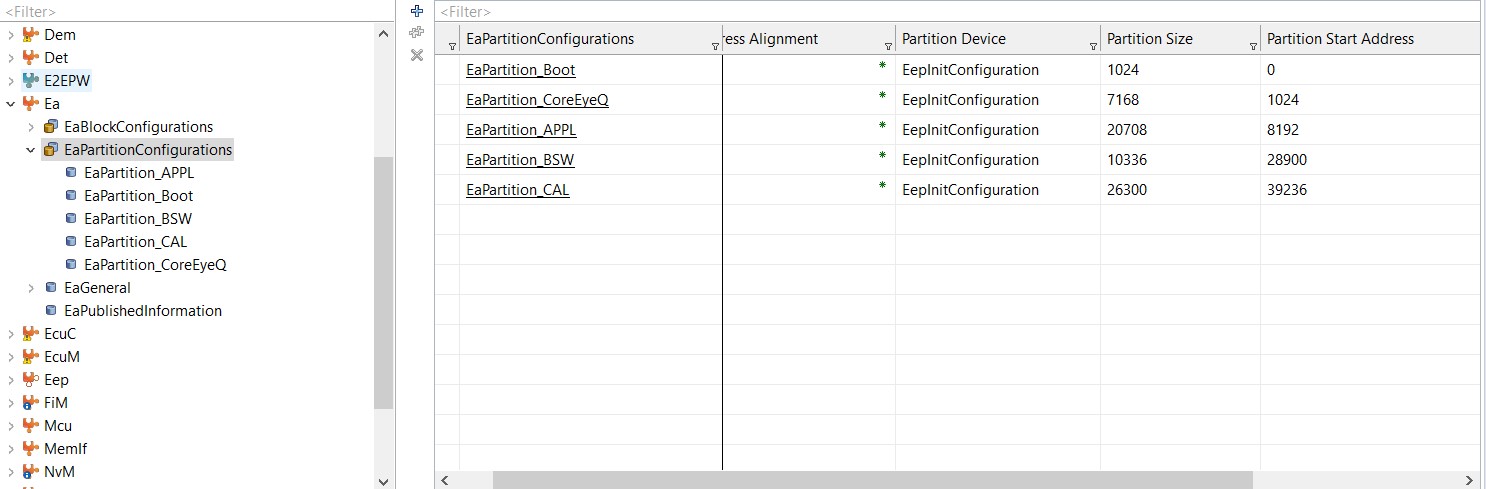


To avoid this allocate reserve into ObjAttr and verify address. Also verify size of PIM as below it should be same as configurator.



9.2 Check points to avoid issues:

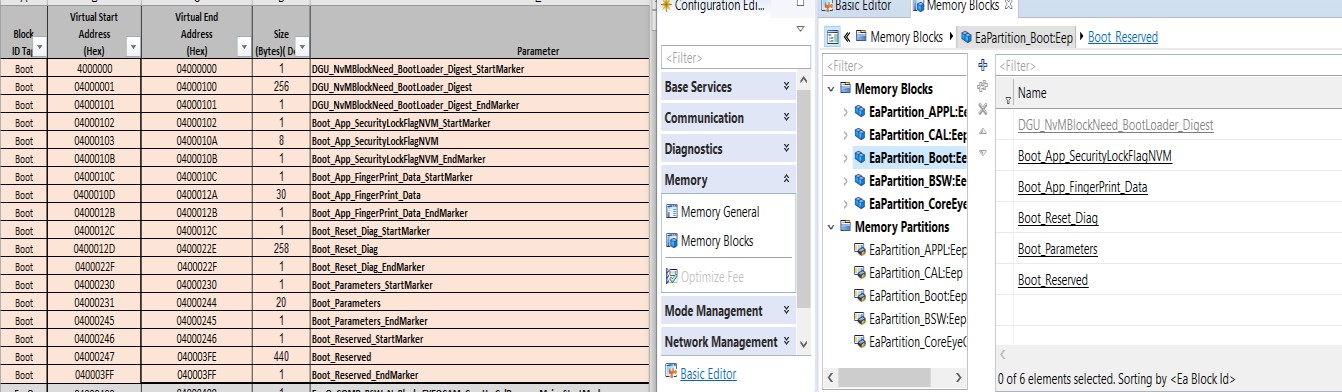
9.2.1 Partition order in configurator should be same as that in layout as shown below:

 Partation in configurator

|  |
| --- |
| BOOT BLOCKS |
|  |
| CORE EyeQ Blocks |
|  |
| Application Blocks |
|  |
| BSW |
|  |
| CAL |

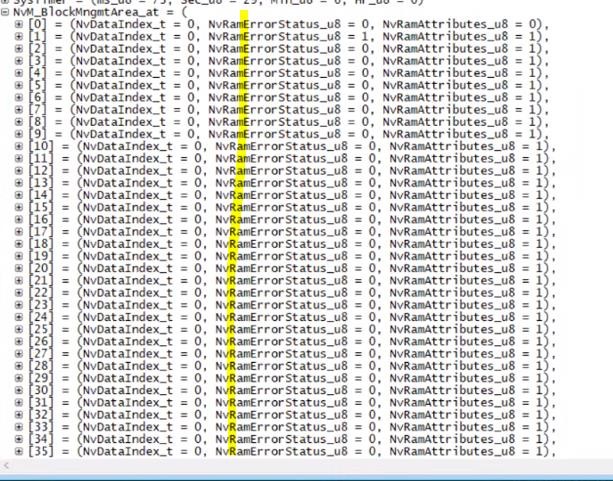
Representation of partations in layout

9.2.2 Block order in layout in a partition should be same as that of order of blocks in particular partations of configurator. Below is example in boot partition.



Left we have boot blocks order in layout same of that of order in boot partition of configurator.

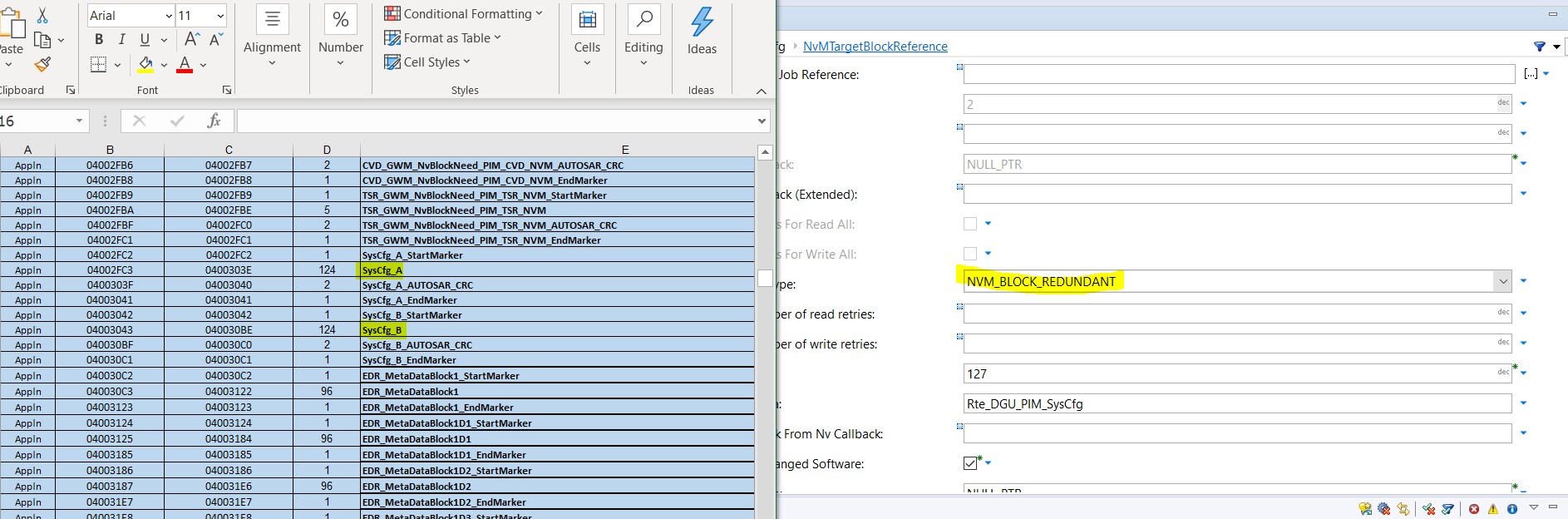
If this point is not taken care it will result in intigrity fail/invalidity error. We check in by checking “NvM\_BlockMngmtArea\_at” in trace and checking NvRamErrorStatus.



If any block is having NvRamErrorStatus as 3 or 5 find index in same structure and check the block in configurator with that index first.Verify the order of blocks above that block are same in layout and configurator if not please rearrange to solve issue.

9.2.3 Define the block management type same in layout and configurator below are examples:

In below example the block management type is redundant in configurator so the same block in layout is having Block A and BLOCK B. Similary if no redundancy only one block should be allocated in layout.

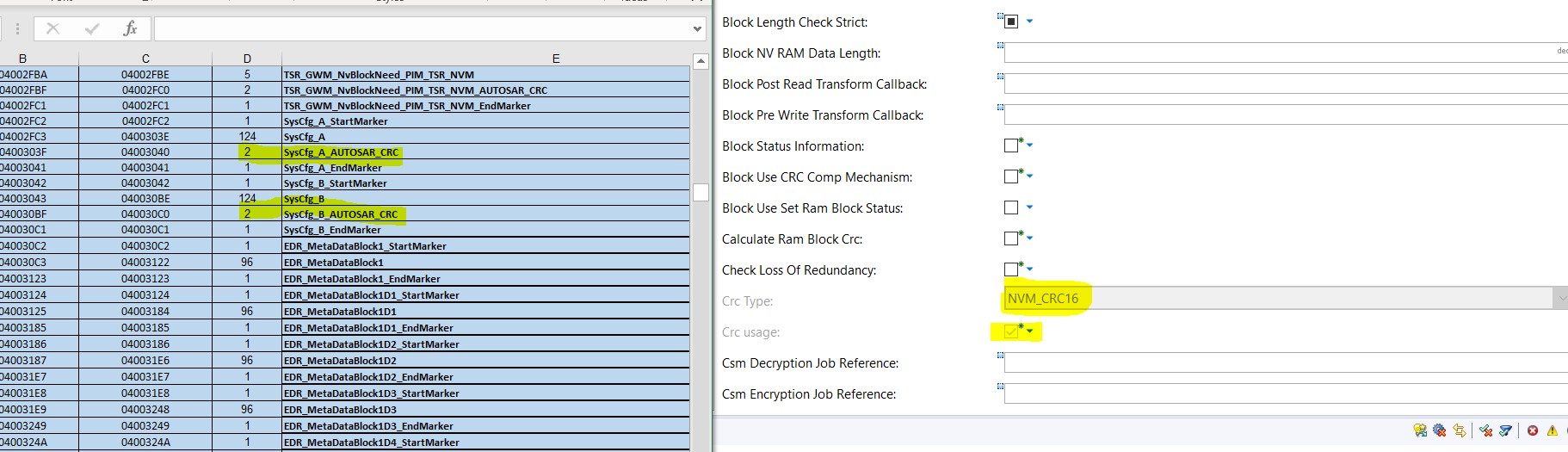


If above point is not taken care it will result in address mismatch of blocks below that in layout that is in .s37 and generated code of all blocks below that.We need can find issue same as discussed in 9.2.2 with help of

NvM\_BlockMngmtArea\_at” in trace and checking NvRamErrorStatus.

9.2.4 Define CRC usage same in layout and in configurator along with CRC length same:

In below example we enabled crc for block and crc types as NVM\_CRC16 ,so same should be defined in layout as shown below it is allocated 2 bytes for CRC .



If above point is not taken care it will result in address mismatch of blocks below that in layout that is in .s37 and generated code of all blocks below that.We need can find issue same as discussed in 9.2.2 with help of

NvM\_BlockMngmtArea\_at” in trace and checking NvRamErrorStatus.

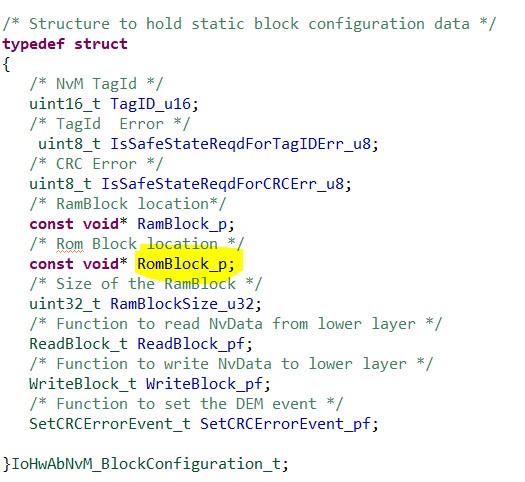
1010.NVM default values updates from core while integration:

In GWM and variants NVM defaults values are generally given in NVM .cfg files. Core do not fallow this, core use FEE they maintain default values in ROM. When there are any change in default values of NVM block parameters and parameters names this should also be updated in NVM .cfg files of application project to which we are integrating.

Below are ways to find differences in default values in core in latest release. Below are files where all NVM blocks with ROM defaults are defined.

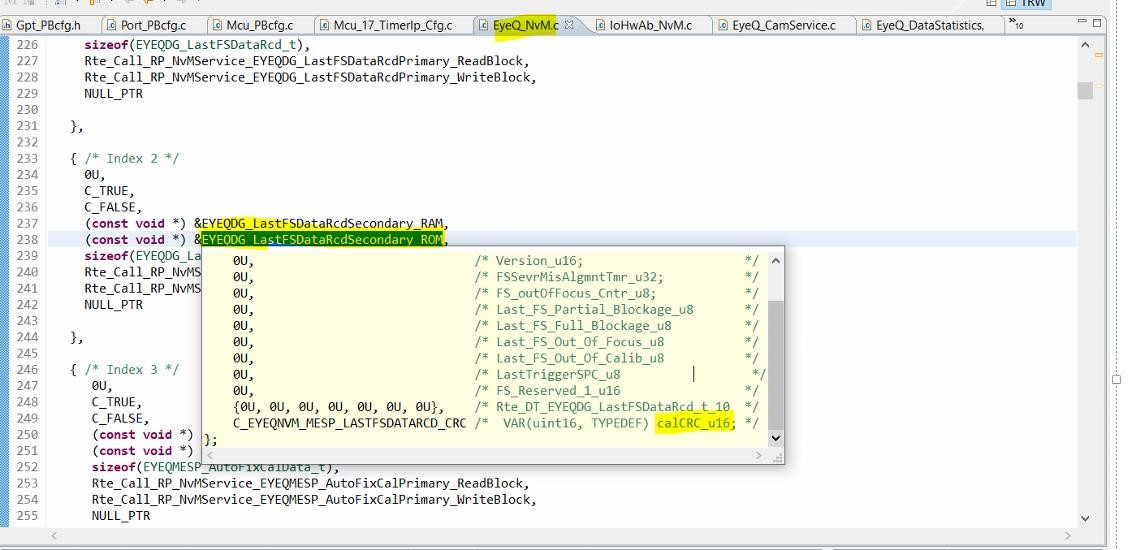
Core\_Application/SourceCode/CDD/EyeQ\_CDD/EyeQ\_NVM/src/EyeQ\_NvM.c Application\SourceCode\IoHwAbs\IoHwAb\_NvM\src\IoHwAb\_NvM.c

There are two structure each for EyeQ blocks and one for IoHwAB blocks defined as below .In EyeQ\_NvM.c and IoHwAb\_NvM.c files there is structure member /\* Rom Block location \*/ **const** **void**\* RomBlock\_p; .

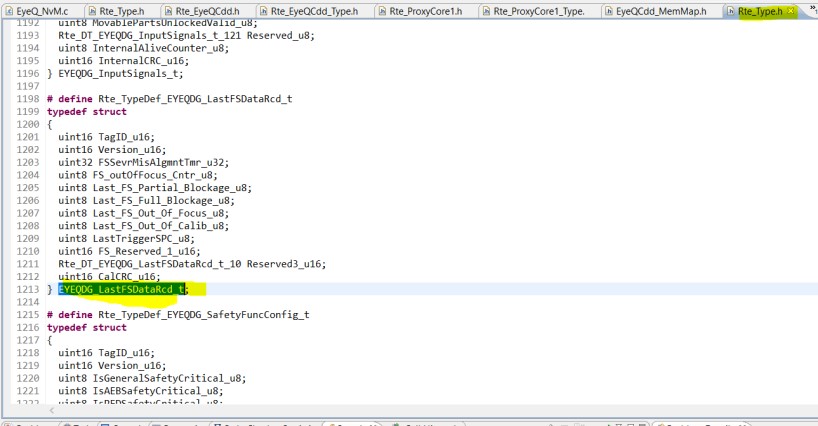


Example to find changes in default values:In below screen shot we can find

EYEQDG\_LastFSDataRcdSecondary blocks ROM member which defines default values of member of blocks ,the only way to find if any update in core is compare with earlier release of core default value.



Similarly for IoHwAb blocks we need to find any changes in ROM member default values and updating NVM .cfg files in application .We can also find any changes in members of structure elements names by comparing Application\SourceCode\BSW\Generated\MonoCam\Rte\_Type.h

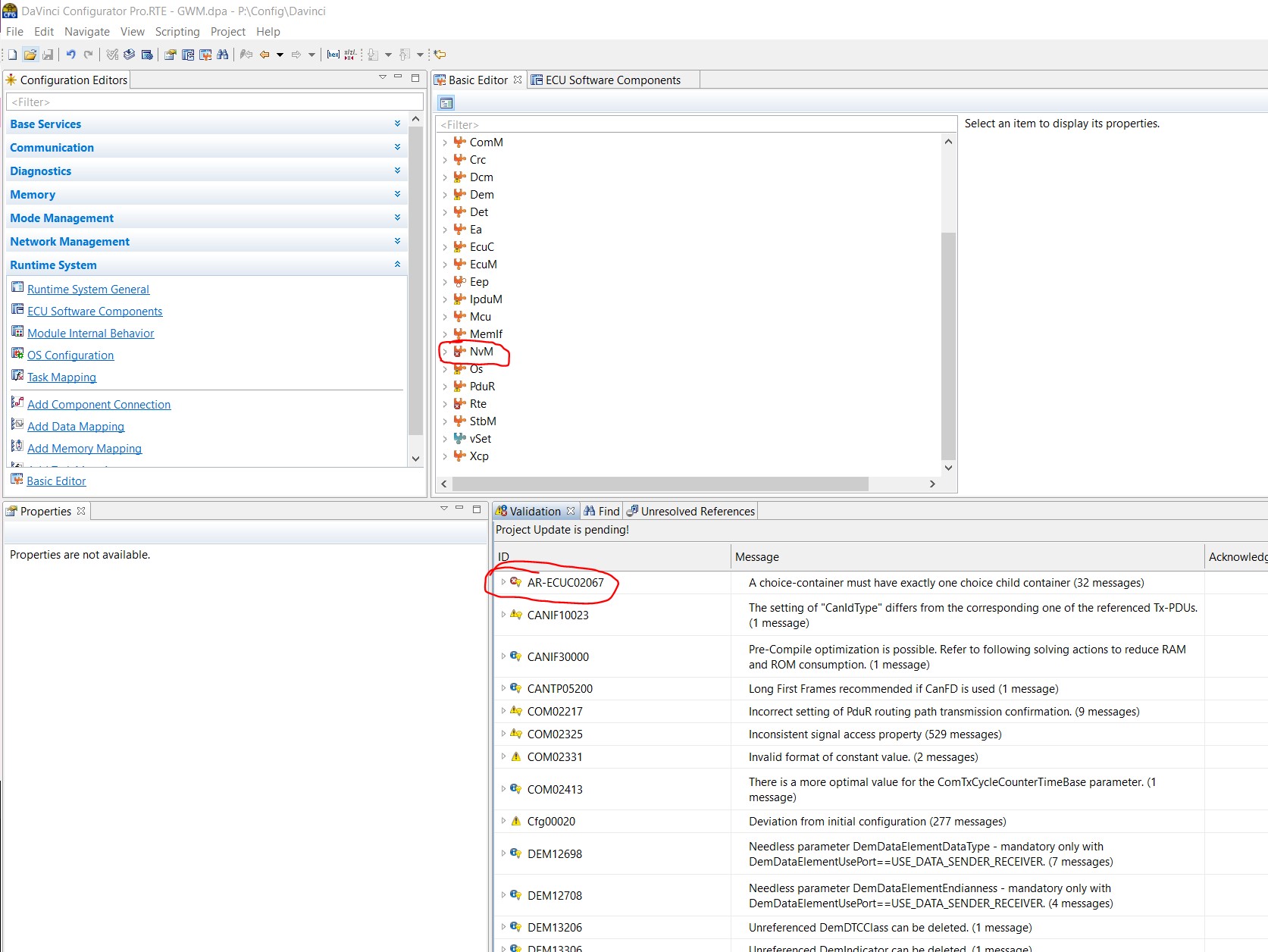


**11** 11.NvM Core Proxy ARXML Integration to Application project

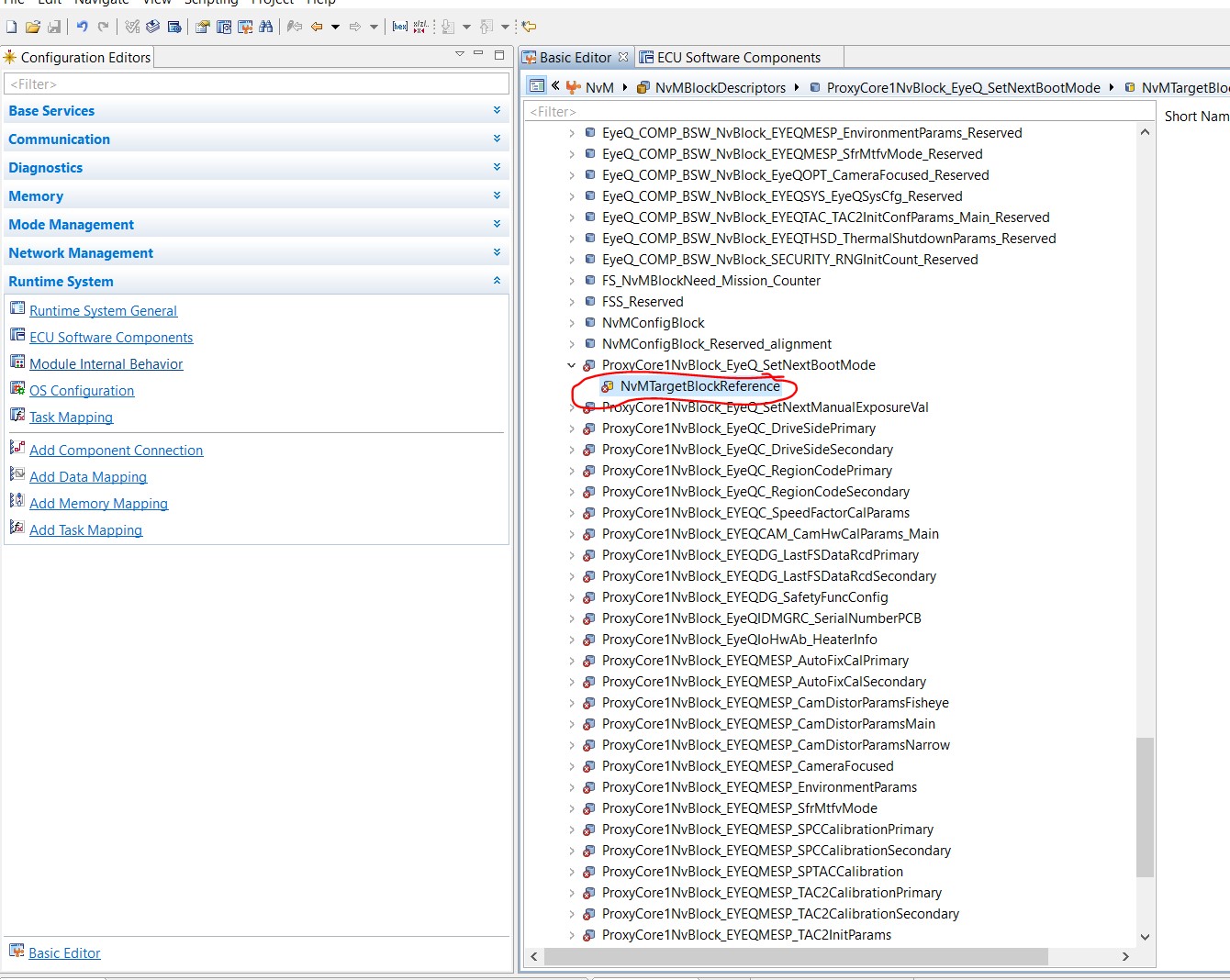
Proxy(Proxy0 or Proxy1 according architecure) arxmls conatins NVM related configuration .

Once the arxmls are imported Davinci configurator will synchronise with the Davinci developer project. After synchronization the NvM configuration needs will be updated. Follow the below procedure to cofigure NvM.

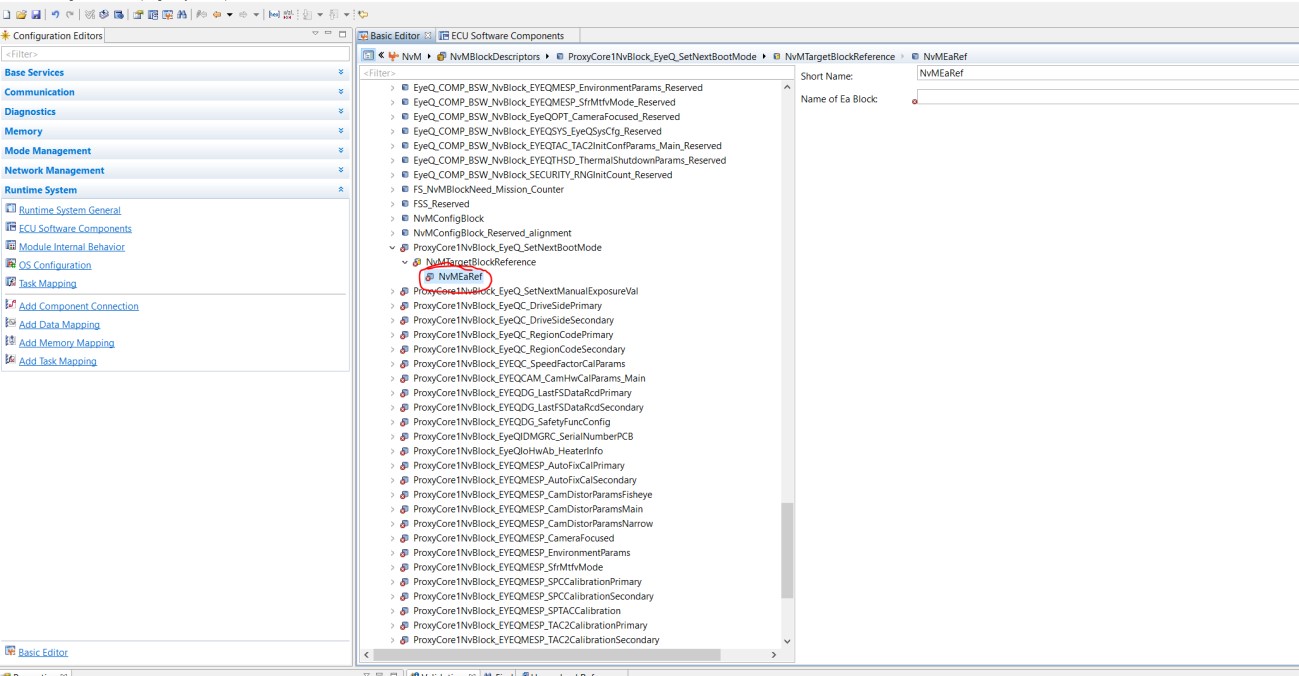
Check for NvM errors in **validation** tab.



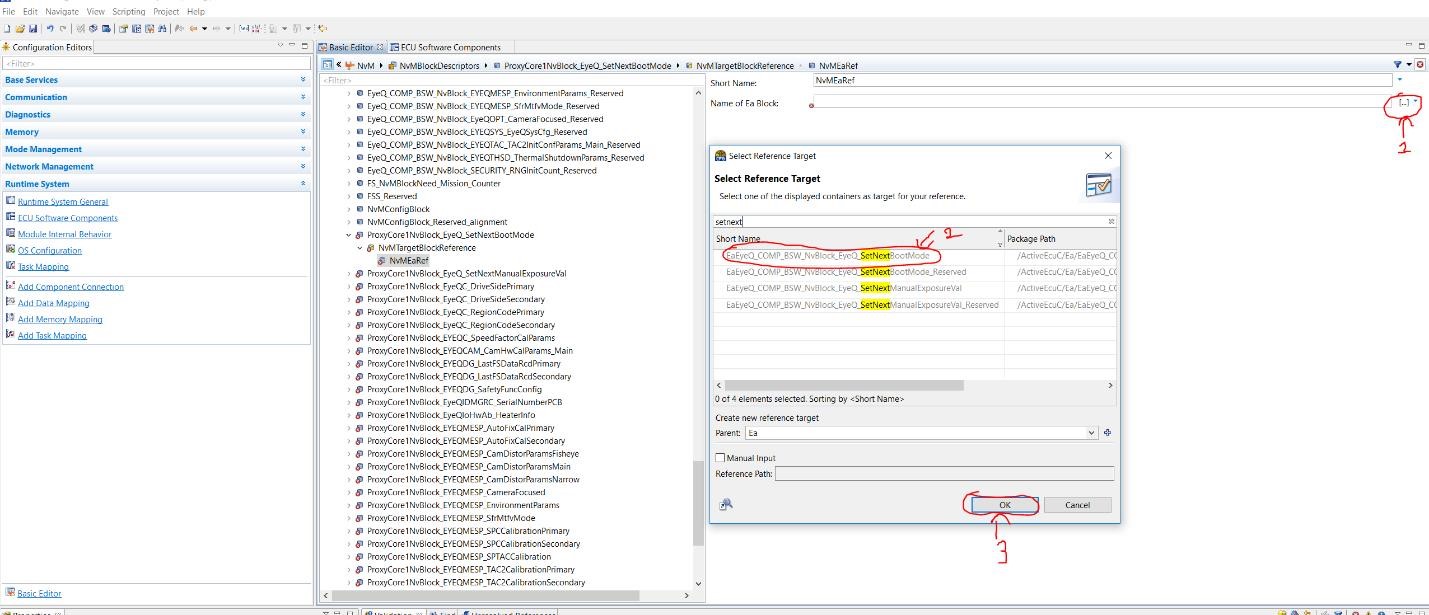
Go to faulty blocks in NVM Component and



Right click on the Block reference and Choose Ea Block or FEE block based on project requirement.



Select an Ea block by clicking on select a reference target and then select the respective block.(Ea block configuration is taken for example)



Repeat same steps for all faulty blocks and validate once finished.

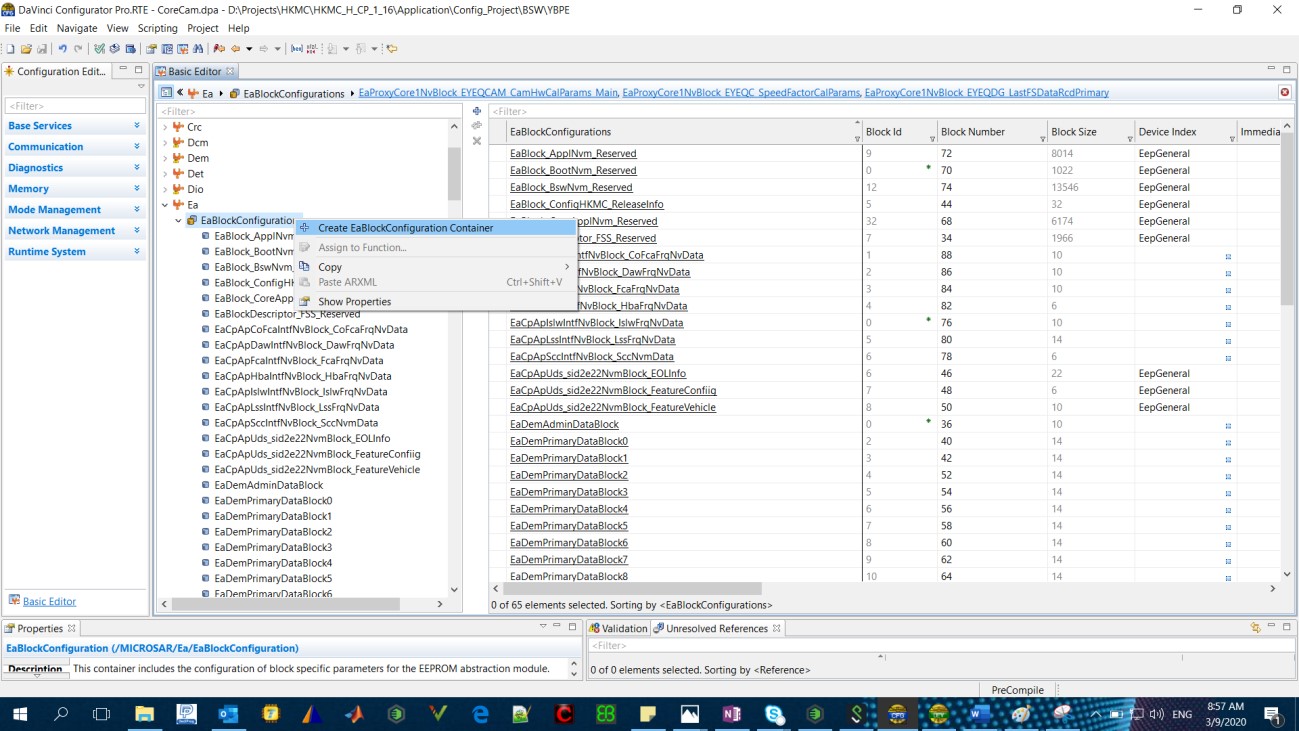
If some blocks are newly added/removed in core software then create/delete Ea/Fee blocks first in Ea/Fee in EyeQ CDD CORE partition in Application software and then give the Ea/Fee block reference in NvM Configuration.

Update the NvM block lengths as per CORE software requirement and adjust the Core NvM Reserved space in

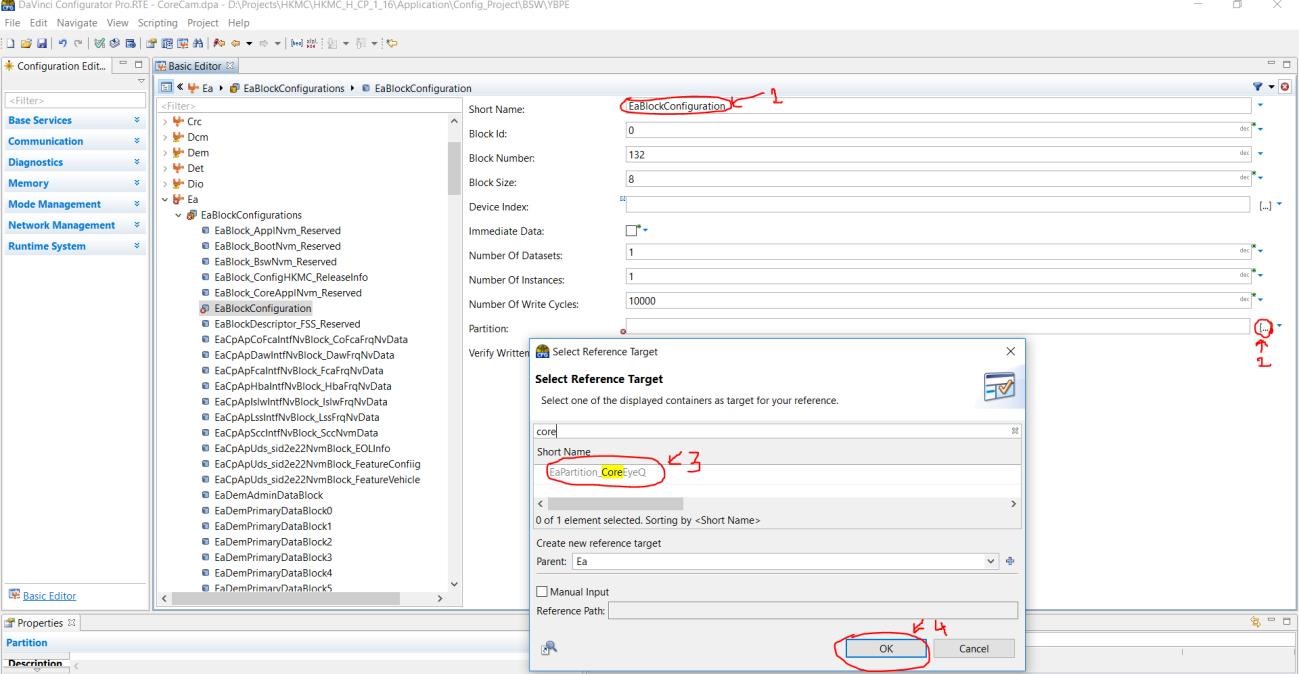
EyeQ CDD CORE partition

Creating and deleting Ea/Fee Blocks

In configurator open **Ea** module and the right click on **EaBlockConfigurations** and the select **CreateEaBlockConfigurationContainer.**

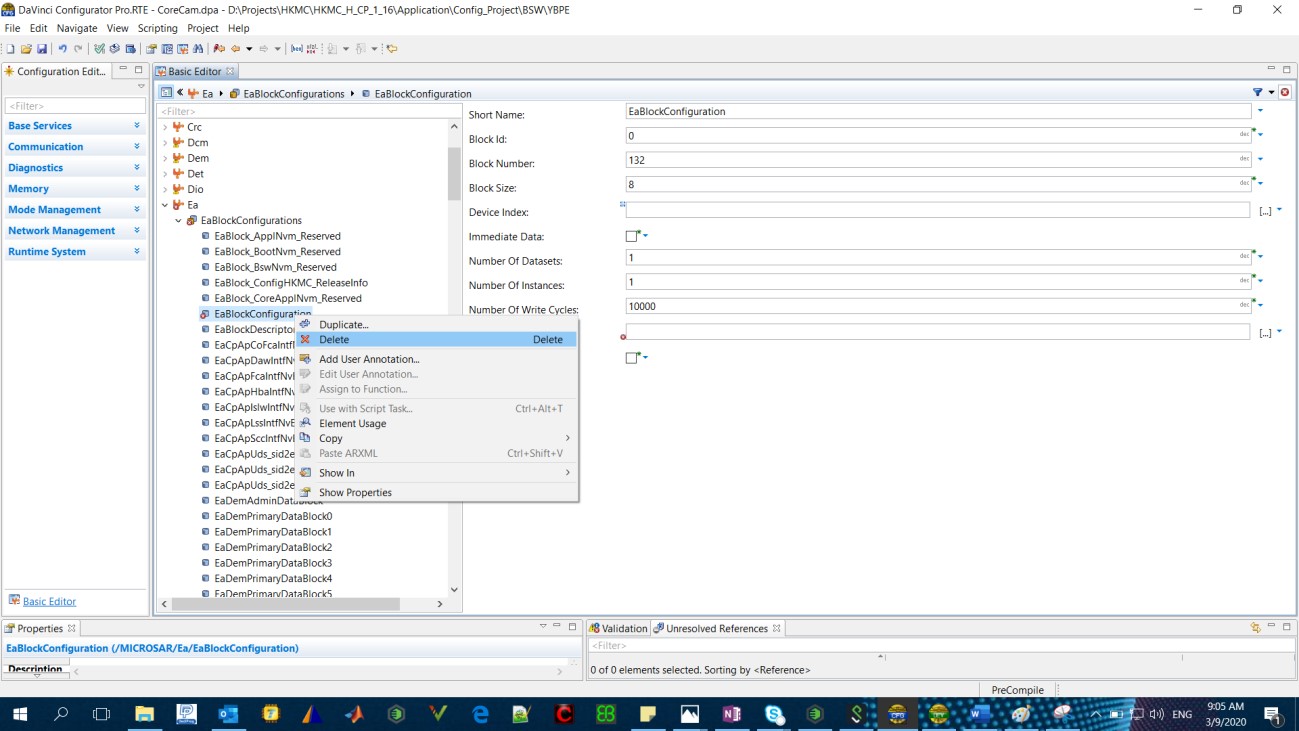


Click on created EaBlock and rename the block with Core specific block name and the select Core partition.

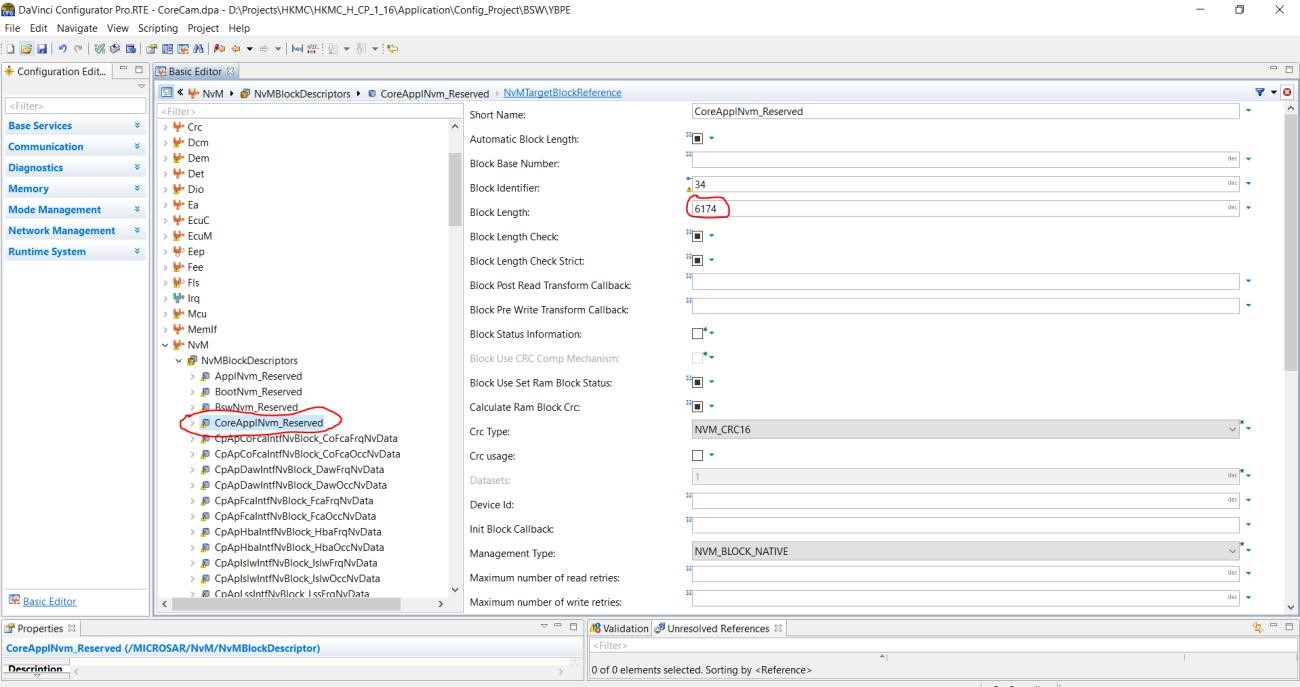


Repeat the above step for all newly created core blocks.

To Delete Ea block configuration which are removed from Core, Right click on the block to be removed and select delete.



Now adjust the Core reserved space according to new NvM layout of the Core.



If this step is not done then the size NvM partition for Core and the sum of configured core block lengths along with Core reserved block will not be same. Then Nvm and Ea modules will throw an error. To calculate the Core partition length, Subtract the lengths of blocks which are newly added from Core reserved block length and add the lengths of the blocks which are removed/deleted to Core reserved block length. The lengths of CRC and Start and End markers of the blocks should be included.

***Note****: To refer Core NvM Blocks the information will be given in EyeQCDD\_Platform\_Integration\_Guide in Core software.*

After this respective changes in layout and .cfg should be done.