FLASH EEPROM Emulation from Simulink® – (F2837x)

This document will briefly summarize how to store data to Flash from Simulink using the demo model 'f2837x_eeprom_emulation.slx' on a Texas Instruments (TI) F28379D Launchpad using the *Embedded Coder® Hardware Support Package for TI C2000™ Processors:*

https://www.mathworks.com/matlabcentral/fileexchange/43096-embedded-coder-support-package-for-texas-instruments-c2000-processors

Overview of demo model

The model demostrates a basic framework for EERPOM Emulation over Flash. To get a background on EEPROM Emulation application please refer the following document from TI:

https://www.ti.com/lit/an/sprab69a/sprab69a.pdf

Based on above document, there is a need for storing calibration values in a non-volatile memory so that it can be used or modified and reused even after power cycling the system.

Through this demo, we define a sector of on-chip Flash memory as the emulated electrically erasable programmable read-only memory (EEPROM) by emulating the EEPROM functionality within the limitations of the Flash memory. Note that one Flash sector is entirely used as an emulated EEPROM; therefore, it is not available for the application code.

The demo model will showcase how one can define calibration parameters that will be loaded to Flash sector reserved for EEPROM emulation (achieved through a separate load and run addresses for these parameters, and a copy is performed to move from the Flash to the RAM at runtime such that modifying them is possible). The calibration values can then be updated in RAM and then copied back to Flash for reuse even after power cycle using the TI Flash API routines.

Refer to the demo video on this example here:

https://www.youtube.com/watch?v=fWZoXDdff3Q

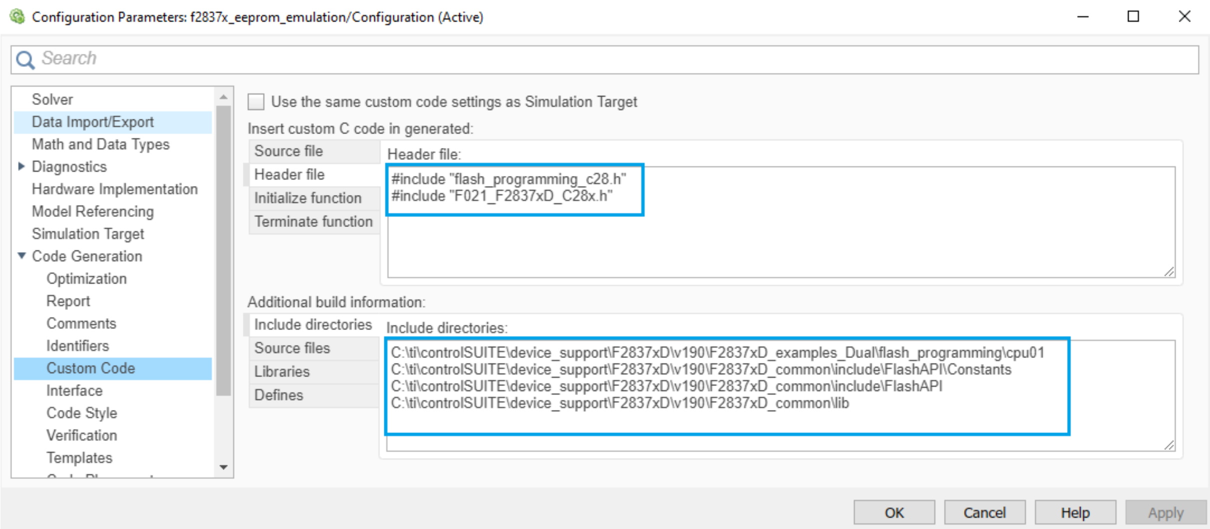
Note: The model is setup for external mode simulation to allow one to easily verify EEPROM emulation functionality for – use/modify and reuse, of calibration values. The calibration values are defined as scalar, vector and structure data types. However, the model is setup using scalar and vector types for demonstration, as there is a known limitation when

calibration parameters are defined as structures the same cannot be updated during external mode operation. However, defining calibration parameters as structure types is supported.

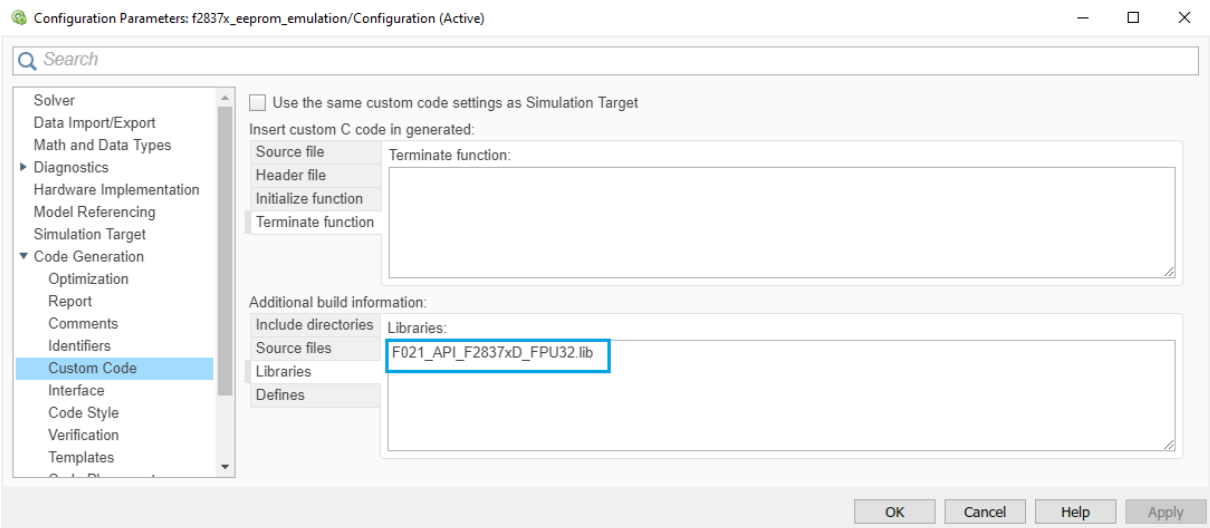
Implementation scheme

1. Using Flash API from TI

To begin with, the header and source files (Flash API library 'F021_API_F2837xD_FPU32.lib' as provided by TI) is used. We will use the Flash APIs to read/write/modify data. The files include the necessary definitions of variables, macros and functions to be able to program the Flash dynamically. To include the header and source files for use within the Simulink, use custom code under Code Generation option as shown below.



****modify/update the path to ControlSUITE if found different in your setup.***

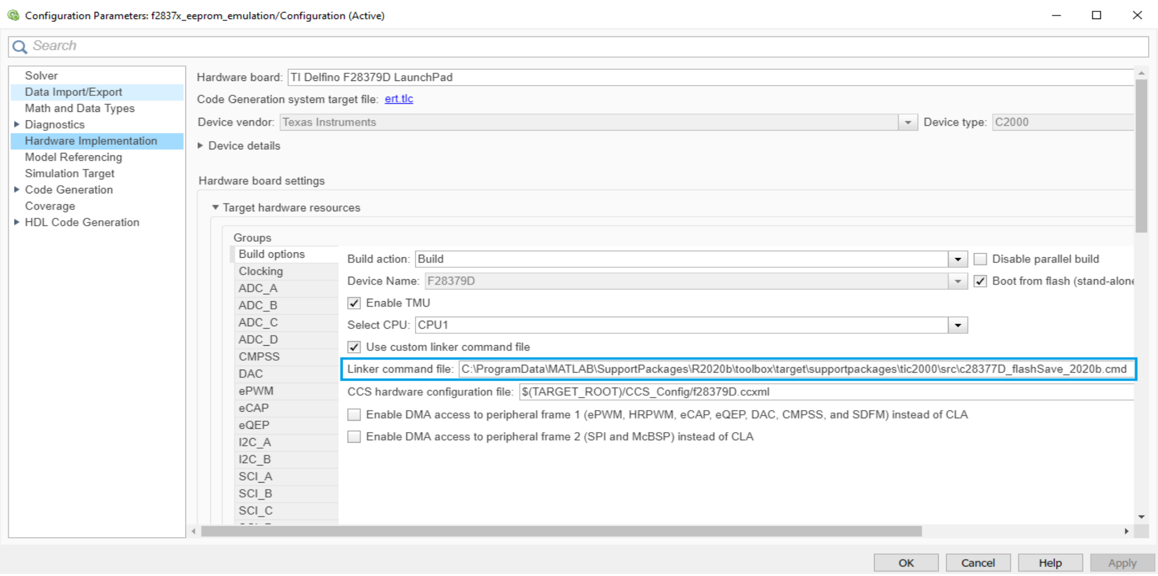


2. Mapping Flash API library to ramfunc

There should not be any read or fetch access from the Flash bank/OTP when an erase or program operation is in progress. Therefore, the Flash API functions must be executed from RAM. ***Please refer the TMS320F2837xD Flash API reference guide for more such restrictions when implementing a full design.*** The linker command file 'c28377D_flashSave_2020b.cmd' has updates to link F021_API_F2837xD_FPU32.lib to ramfunc.

```
SECTIONS
{
#if BOOT_FROM_FLASH
   /* Allocate program areas: */
       EEPROMdata            : LOAD = EEPROM_Flash,
                               RUN = EEPROMData_RAM,
                               LOAD_START(_eepromfuncsLoadStart),
                               LOAD_END(_eepromfuncsLoadEnd),
                               RUN_START(_eepromfuncsRunStart),
                               LOAD_SIZE(_eepromfuncsLoadSize),
                               PAGE = 1, ALIGN(8)
   .cinit                   : > FLASHA_M,                      PAGE = 0, ALIGN(8)
   .pinit                   : > FLASHA_M,                      PAGE = 0, ALIGN(8)
   .text                    : > FLASHA_M,                      PAGE = 0, ALIGN(8)
   codestart                : > BEGIN_FLASH
   .TI.ramfunc              : LOAD = FLASHA_M,
                              RUN = RAMLS_PROG,
                              LOAD_START(_RamfuncsLoadStart),
                              LOAD_SIZE(_RamfuncsLoadSize),
                              LOAD_END(_RamfuncsLoadEnd),
                              RUN_START(_RamfuncsRunStart),
                              RUN_SIZE(_RamfuncsRunSize),
                              RUN_END(_RamfuncsRunEnd),
                              PAGE = 0, ALIGN(8)
   {
       -lF021_API_F2837xD_FPU32.lib
   }
```

Place this c28377D_flashSave_2020b.cmd provided with model file at path as shown below. Run command 'matlabshared.supportpkg.getSupportPackageRoot' to get support package install path. Point to the path as shown below.



3. Define a Flash section for EEPROM Emulation

On the F28379D device, the Flash is divided into 14 sectors namely sector 0 to sector 13. The last sector, 'sector 13' is reserved for EEPROM emulation. The linker command file 'c28377D_flashSave_2020b.cmd' as discussed previously, has updates for the same. ***Note: for demo purpose, only 0x10 memory locations are allocated for storing calibration values from 'sector 13' as shown below.***

```
MEMORY
{
PAGE 0 :
   /* BEGIN is used for the "boot to SARAM" bootloader mode   */
   BEGIN                    : origin = 0x000000, length = 0x000002
   BEGIN_FLASH              : origin = 0x080000, length = 0x000002
   #ifdef CLA_BLOCK_INCLUDED
        RAMLS_PROG          : origin = 0x008000, length = 0x001800
        RAMLS_CLA_PROG      : origin = 0x00A800, length = 0x000800
   #else
        #if BOOT_FROM_FLASH
            RAMLS_PROG      : origin = 0x008000, length = 0x002000
        #else
            RAMLS_PROG      : origin = 0x008000, length = 0x003000
        #endif //BOOT_FROM_FLASH
   #endif //CLA_BLOCK_INCLUDED

   #ifdef CPU1
        #if (CPU1_RAMGS_PROG_LENGTH > 0)
            RAMGS_PROG      : origin = CPU1_RAMGS_PROG_START, length = CPU1_RAMGS_PROG_LENGTH
        #endif //(CPU1_RAMGS_PROG_LENGTH > 0)
   #else
        #if (CPU2_RAMGS_PROG_LENGTH > 0)
            RAMGS_PROG      : origin = CPU2_RAMGS_PROG_START, length = CPU2_RAMGS_PROG_LENGTH
       #endif //(CPU2_RAMGS_PROG_LENGTH > 0)
   #endif //CPU1

   RESET                    : origin = 0x3FFFC0, length = 0x000002
   /* Flash sectors */

   FLASHA_M             : origin = 0x080002, length = 0x03DFFD   /* on-chip Flash */

PAGE 1 :
    EEPROM_Bank         : origin = 0x0BE000, length = 0x00008    /* on-chip Flash */
    EEPROM_Page         : origin = 0x0BE008, length = 0x00008    /* on-chip Flash */
    EEPROM_Flash        : origin = 0x0BE010, length = 0x00010    /* on-chip Flash */
    FLASHN              : origin = 0x0BE020, length = 0x001FDF   /* on-chip Flash */
```

4. Define a RAM section for EEPROM Emulation

A RAM section is defined to copy the calibration values from the Flash to the RAM at runtime as shown below to be able to update/modify them.

```
#ifdef CPU1
        RAMGS_DATA              : origin = CPU1_RAMGS_DATA_START, length = CPU1_RAMGS_DATA_LENGTH-0x000050
#else
        RAMGS_DATA              : origin = CPU2_RAMGS_DATA_START, length = CPU2_RAMGS_DATA_LENGTH
#endif //CPU1

   RAMGS_IPCBuffCPU1            : origin = 0x00C000, length = 0x001000
   RAMGS_IPCBuffCPU2            : origin = 0x00D000, length = 0x001000

   CLA1_MSGRAMLOW               : origin = 0x001480, length = 0x000080
   CLA1_MSGRAMHIGH              : origin = 0x001500, length = 0x000080

   CPU2TOCPU1RAM                : origin = 0x03F800, length = 0x000400
   CPU1TOCPU2RAM                : origin = 0x03FC00, length = 0x000400

#ifdef EMIF1_CS0_INCLUDED
   EMIF1_CS0_MEMORY             : origin = 0x80000000, length = 0x10000000
#endif //EMIF1_CS0_INCLUDED
#ifdef EMIF1_CS2_INCLUDED
   EMIF1_CS2_MEMORY             : origin = 0x00100000, length = 0x00200000
#endif //EMIF1_CS2_INCLUDED
#ifdef EMIF1_CS3_INCLUDED
   EMIF1_CS3_MEMORY             : origin = 0x00300000, length = 0x00080000
#endif //EMIF1_CS3_INCLUDED
#ifdef EMIF1_CS4_INCLUDED
   EMIF1_CS4_MEMORY             : origin = 0x00380000, length = 0x00060000
#endif //EMIF1_CS4_INCLUDED
#ifdef EMIF2_CS0_INCLUDED
   EMIF2_CS0_MEMORY             : origin = 0x90000000, length = 0x10000000
#endif //EMIF2_CS0_INCLUDED
#ifdef EMIF2_CS2_INCLUDED
   EMIF2_CS2_MEMORY             : origin = 0x00002000, length = 0x00001000
#endif //EMIF2_CS2_INCLUDED

EEPROMData_RAM: origin = CPU1_RAMGS_DATA_START+CPU1_RAMGS_DATA_LENGTH-0x000050, length = 0x000050
```

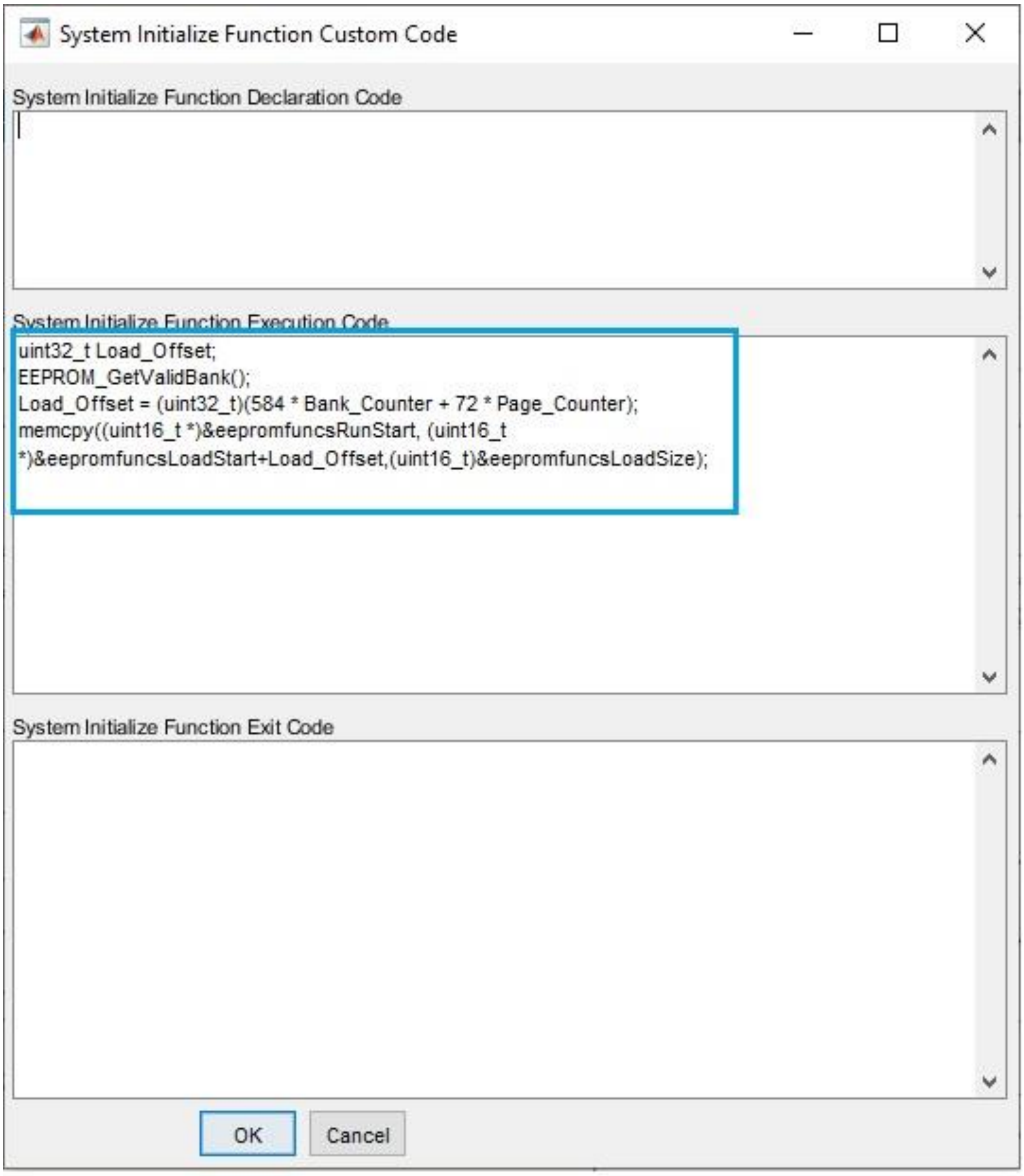
5. Define a global symbols for LOAD_START and RUN_START directives

As noted before, modifying the calibration values is achieved through a separate load and run addresses for these parameters, and a copy is performed to move from the Flash to the RAM at runtime. Hence a user defined section **EEPROMdata** is created for the same as shown below:

```
SECTIONS
{
#if BOOT_FROM_FLASH
   /* Allocate program areas: */
       EEPROMdata             : LOAD = EEPROM_Flash,
                                RUN = EEPROMData_RAM,
                                LOAD_START(_eepromfuncsLoadStart),
                                LOAD_END(_eepromfuncsLoadEnd),
                                RUN_START(_eepromfuncsRunStart),
                                LOAD_SIZE(_eepromfuncsLoadSize),
                                PAGE = 1, ALIGN(8)
       EEPROM_Bank_Flash      : LOAD = EEPROM_Bank,
                                PAGE = 1, ALIGN(8)
       EEPROM_Page_Flash      : LOAD = EEPROM_Page,
                                PAGE = 1, ALIGN(8)
   .cinit                   : > FLASHA_M,              PAGE = 0, ALIGN(8)
   .pinit                   : > FLASHA_M,              PAGE = 0, ALIGN(8)
   .text                    : > FLASHA_M,              PAGE = 0, ALIGN(8)
   codestart                : > BEGIN_FLASH
   .TI.ramfunc              : LOAD = FLASHA_M,
                              RUN = RAMLS_PROG,
                              LOAD_START(_RamfuncsLoadStart),
                              LOAD_SIZE(_RamfuncsLoadSize),
                              LOAD_END(_RamfuncsLoadEnd),
                              RUN_START(_RamfuncsRunStart),
                              RUN_SIZE(_RamfuncsRunSize),
                              RUN_END(_RamfuncsRunEnd),
                              PAGE = 0, ALIGN(8)

    {
        -lF021_API_F2837xD_FPU32.lib
    }
```

Finally, calibration values are be copied from its load address to its run address at runtime as shown below using System Initialize block.



6. Banks and Pages concept to ensure full Flash sector is used before Erasing:

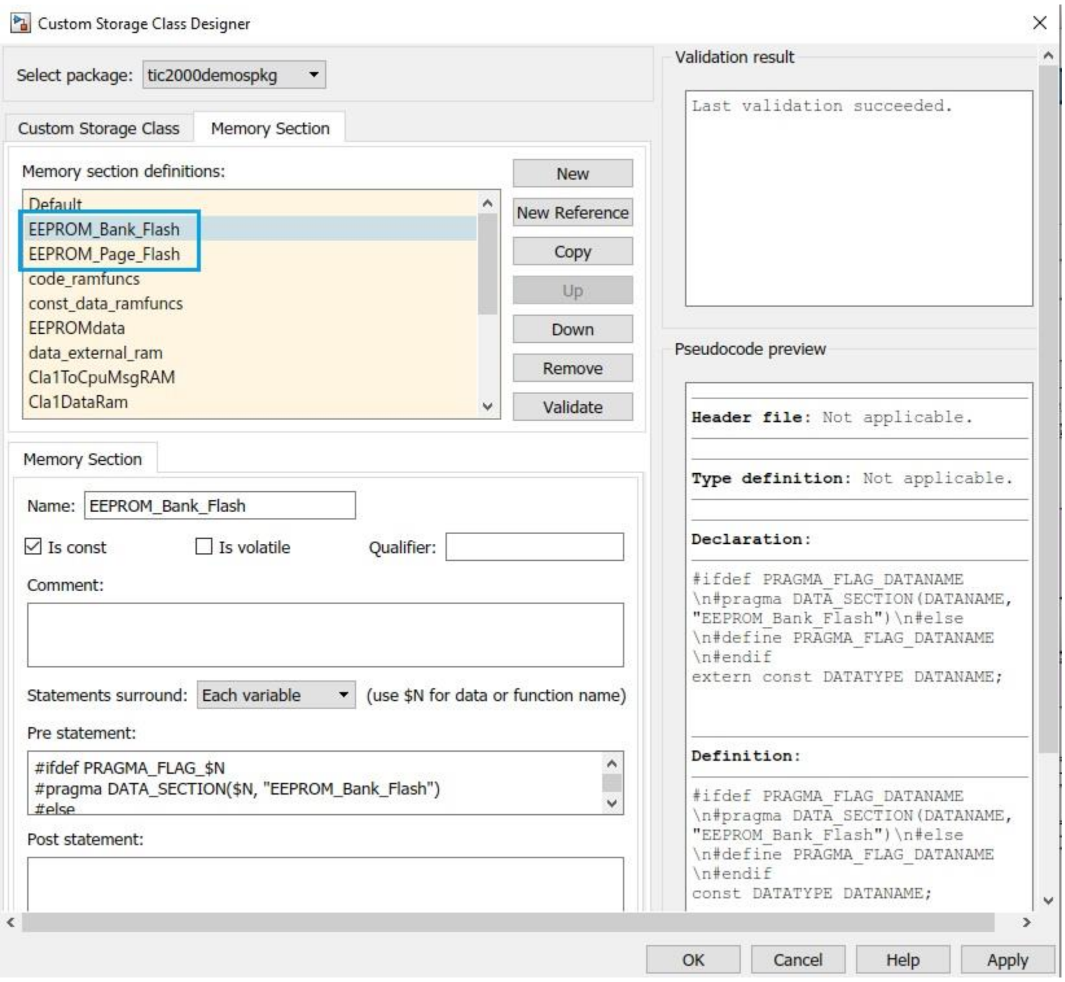
This is a new update made to the EEPROM Emulation example that will ensure we utilize the complete Flash (user defined length is possible) before we erase it completly and then begin from first. This will increase Flash enudrance for erase cycles.

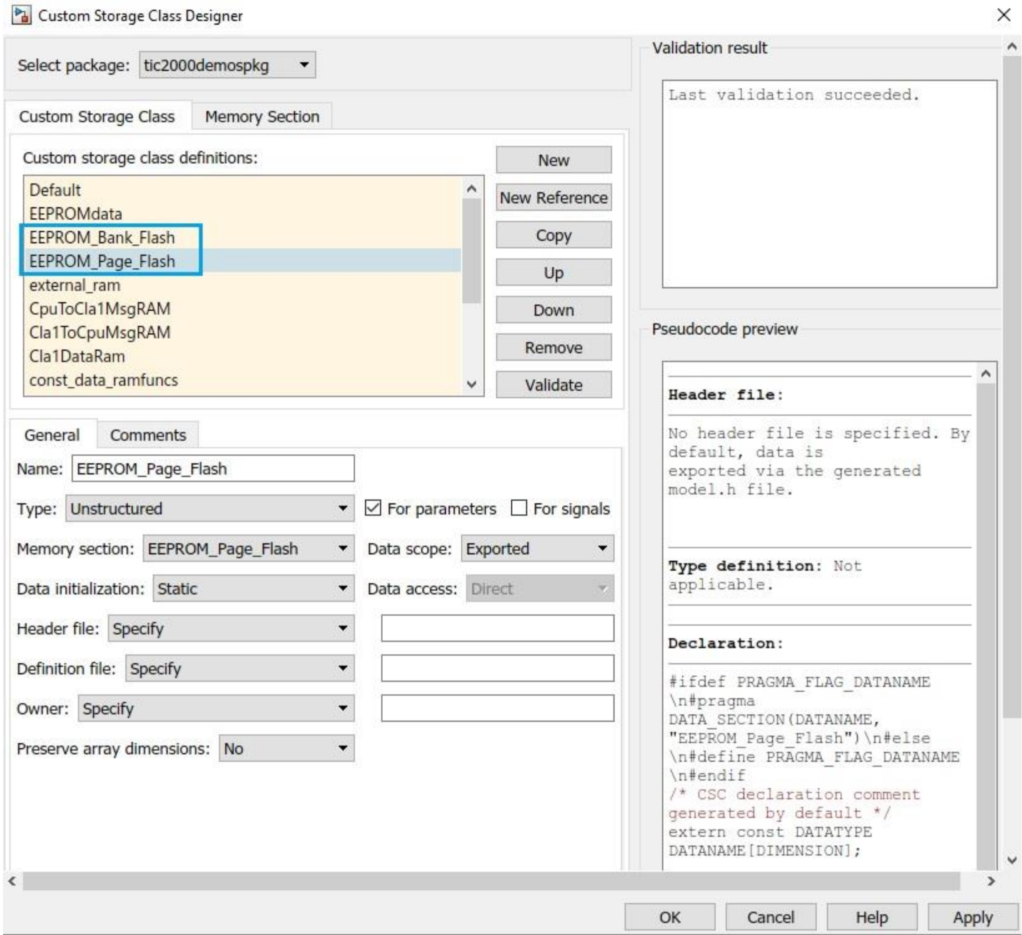
The following application note from TI is used as a base for implementing this example:

https://www.ti.com/lit/an/sprab69a/sprab69a.pdf

It should be noted that F28379D has Flash with ECC. Hence when updating the Bank and Page status we always do a 64bit data update on a 64 aligned memory to avoid triggering the ECC errors.
Each page holds 64 data words
Page size = Page_status word + 64W => 8W+64W = 72W
Bank Size = Bank_status word + 8 * Page size = 8W + 8*72 = 584W

We define a custom storage class to initially load the Flash calibration sector with CURRENT_BANK and CURRENT_PAGE.



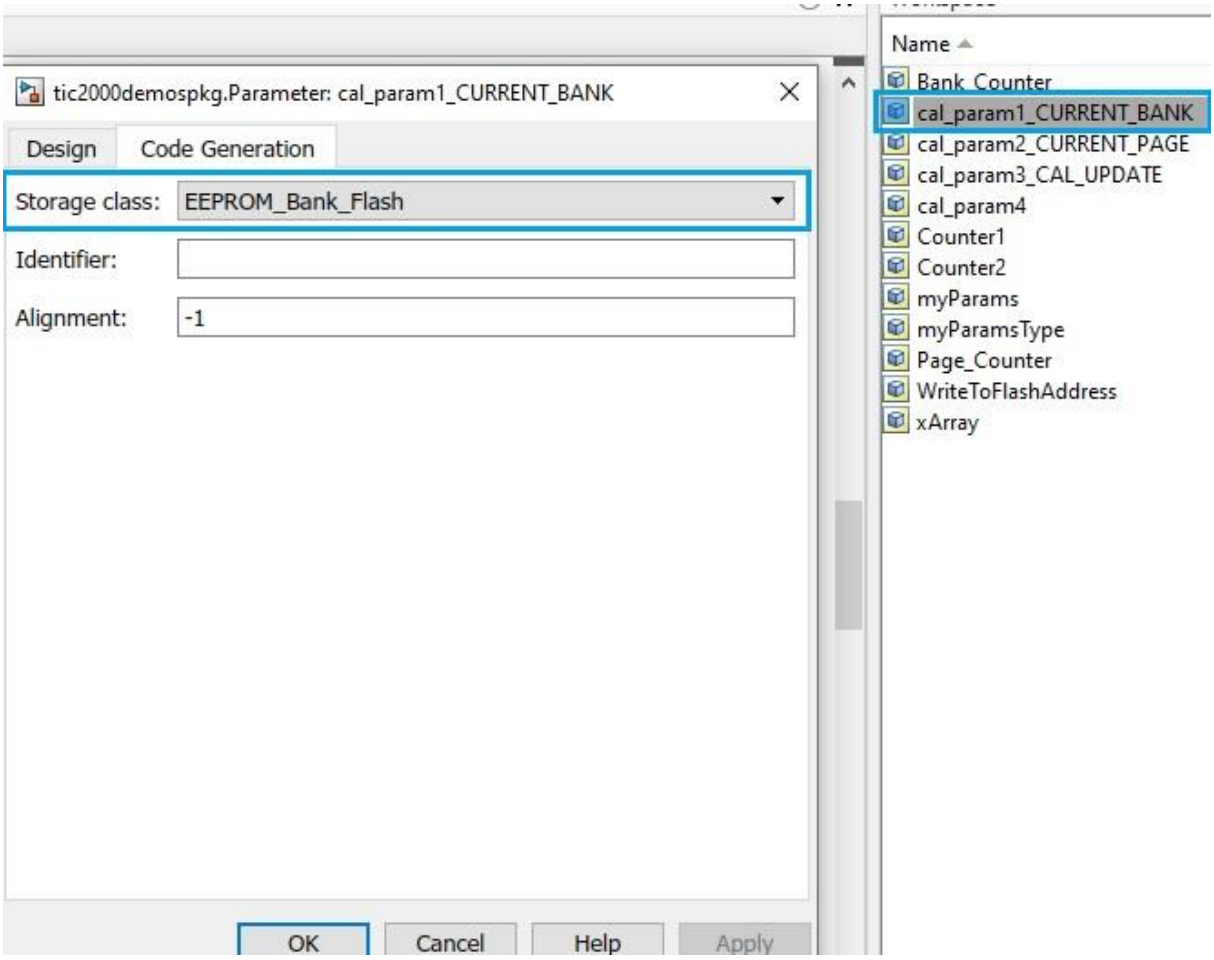



```
PAGE 1 :
   EEPROM_Bank         : origin = 0x0BE000, length = 0x00008    /* on-chip Flash */
   EEPROM_Page         : origin = 0x0BE008, length = 0x00008    /* on-chip Flash */
   EEPROM_Flash        : origin = 0x0BE010, length = 0x00010    /* on-chip Flash */
   FLASHN              : origin = 0x0BE020, length = 0x001FDF   /* on-chip Flash */
```

Define the Load start as shown below around the above memory region:

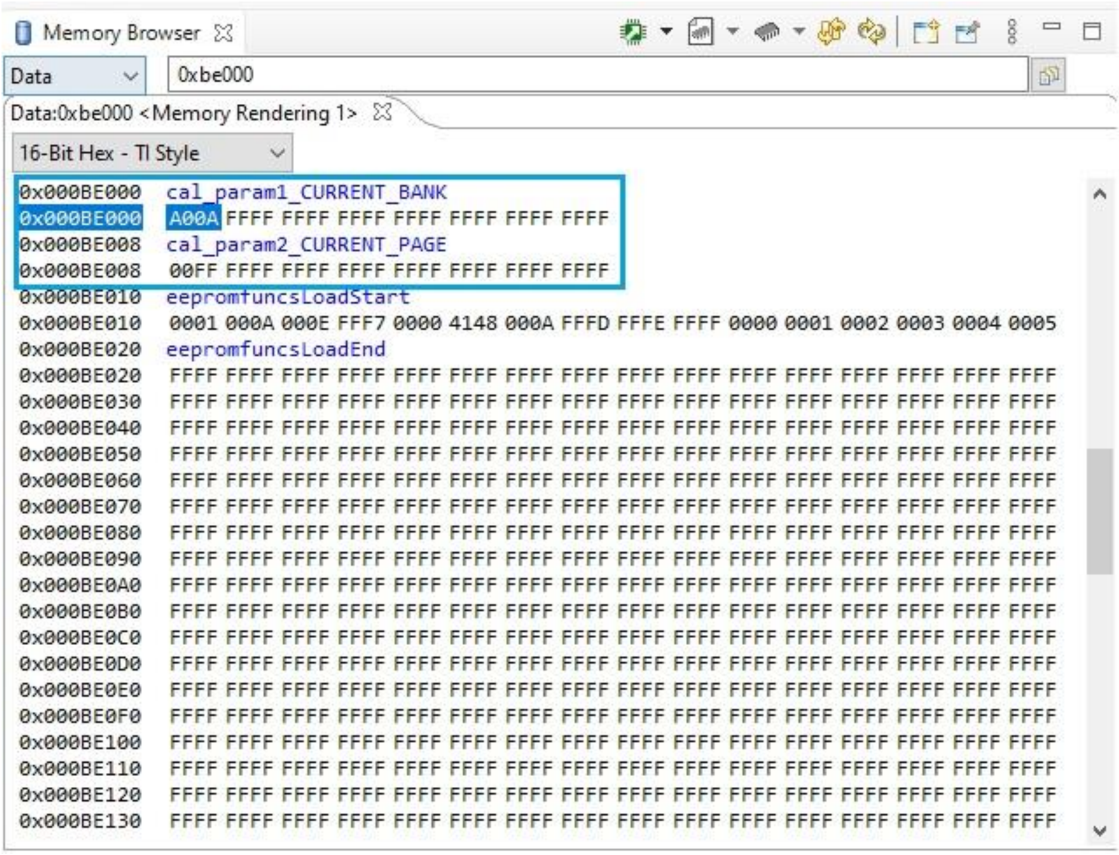
```
SECTIONS
{
#if BOOT_FROM_FLASH
   /* Allocate program areas: */
      EEPROMdata        : LOAD = EEPROM_Flash,
                          RUN = EEPROMData_RAM,
                          LOAD_START(_eepromfuncsLoadStart),
                          LOAD_END(_eepromfuncsLoadEnd),
                          RUN_START(_eepromfuncsRunStart),
                          LOAD_SIZE(_eepromfuncsLoadSize),
                          PAGE = 1, ALIGN(8)
      EEPROM_Bank_Flash : LOAD = EEPROM_Bank,
                          PAGE = 1, ALIGN(8)
      EEPROM_Page_Flash : LOAD = EEPROM_Page,
                          PAGE = 1, ALIGN(8)
   .cinit               : > FLASHA_M,             PAGE = 0, ALIGN(8)
   .pinit               : > FLASHA_M,             PAGE = 0, ALIGN(8)
   .text                : > FLASHA_M,             PAGE = 0, ALIGN(8)
   codestart            : > BEGIN_FLASH
   .TI.ramfunc          : LOAD = FLASHA_M,
                        RUN = RAMLS_PROG,
                        LOAD_START(_RamfuncsLoadStart),
                        LOAD_SIZE(_RamfuncsLoadSize),
                        LOAD_END(_RamfuncsLoadEnd),
                        RUN_START(_RamfuncsRunStart),
                        RUN_SIZE(_RamfuncsRunSize),
                        RUN_END(_RamfuncsRunEnd),
                        PAGE = 0, ALIGN(8)
```

Next we define 2 workspace parameters that each represent the CURRENT_BANK and CURRENT_PAGE and map it to above LOAD section.



*Similarly for CURRENT_PAGE.

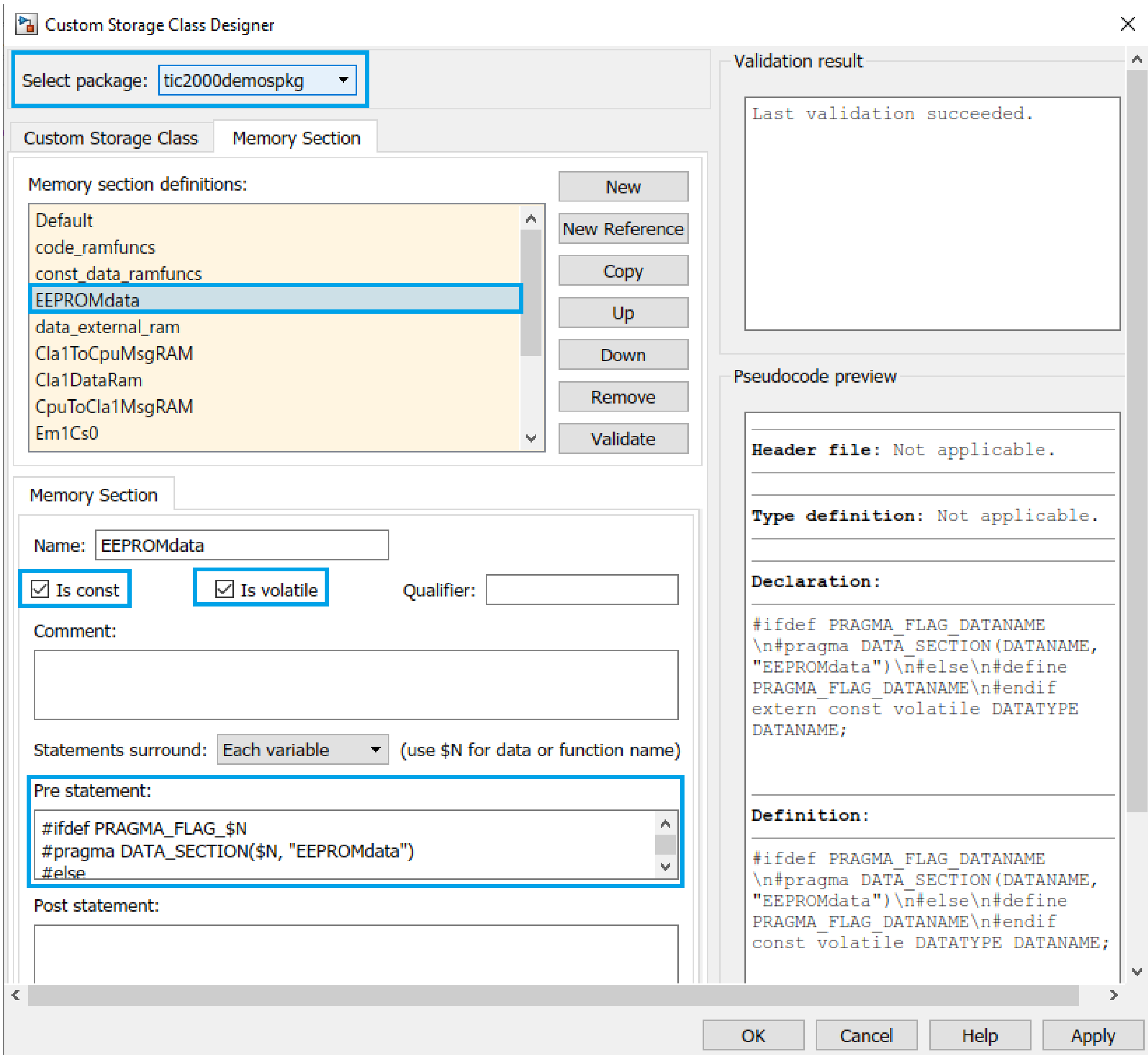
When we build the model and load into the Flash, we will have the initial set of Bank and Page ready for the algorithm. That is the idea of the approach. When we load the project in CCS, we should we able to see the 128 bit header for BANK status and 128bit for page status. As we progress updating the pages, we maintain the same header for both Banks and Pages.



Refer TI application note on the general idea of Banks and Pages:
https://www.ti.com/lit/an/sprab69a/sprab69a.pdf
It should be noted the Max Bank = 3(0,1,2) and Max page = 8(0-7) in the current setup.

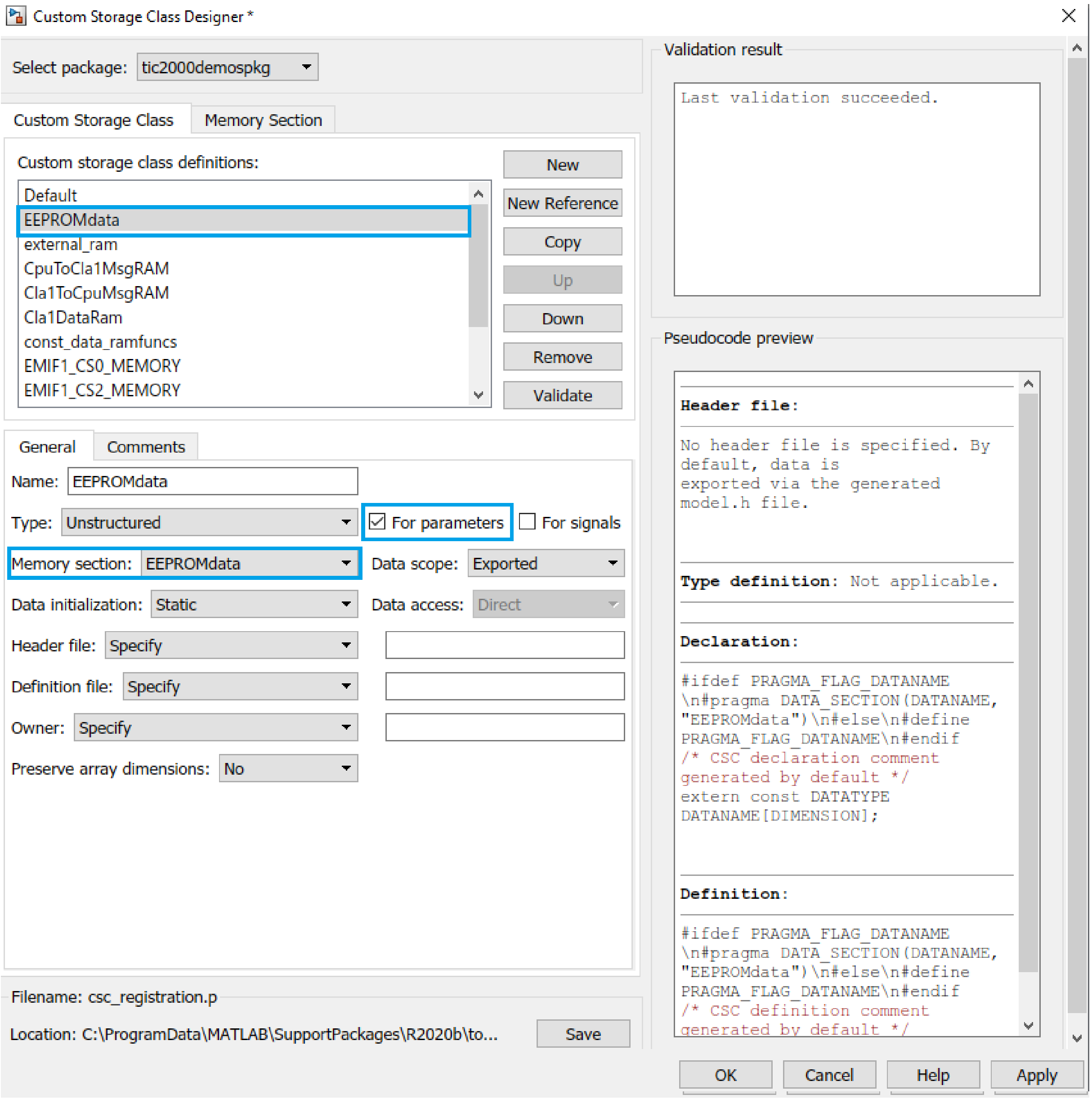
7. Define a custom storage class for the calibration parameters

Since we want to place the calibration parameters in a user defined section (**EEPROMdata**), we can use the DATA_SECTION pragma to achieve the same. Launch the custom storage class designer from the MATLAB command window by using the command 'cscdesigner'. Define the new memory section as shown below.



Custom Storage Class Designer
Select package: tic2000demospkg
Custom Storage Class
Memory Section
Memory section definitions:
Default
code_ramfuncs
const_data_ramfuncs
EEPROMdata
data_external_ram
Cla1ToCpuMsgRAM
Cla1DataRam
CpuToCla1MsgRAM
Em1Cs0
New
New Reference
Copy
Up
Down
Remove
Validate
Memory Section
Name: EEPROMdata
Is const
Is volatile
Qualifier:
Comment:
Statements surround: Each variable
(use $N for data or function name)
Pre statement:
#ifdef PRAGMA_FLAG_$N
#pragma DATA_SECTION($N, "EEPROMdata")
#else
Post statement:
Validation result
Last validation succeeded.
Pseudocode preview
Header file: Not applicable.
Type definition: Not applicable.
Declaration:
#ifdef PRAGMA_FLAG_DATANAME \n#pragma DATA_SECTION(DATANAME, "EEPROMdata")\n#else\n#define PRAGMA_FLAG_DATANAME\n#endif extern const volatile DATATYPE DATANAME;
Definition:
#ifdef PRAGMA_FLAG_DATANAME \n#pragma DATA_SECTION(DATANAME, "EEPROMdata")\n#else\n#define PRAGMA_FLAG_DATANAME\n#endif const volatile DATATYPE DATANAME;
OK
Cancel
Help
Apply

And define a new storage class using this memory section as shown below:

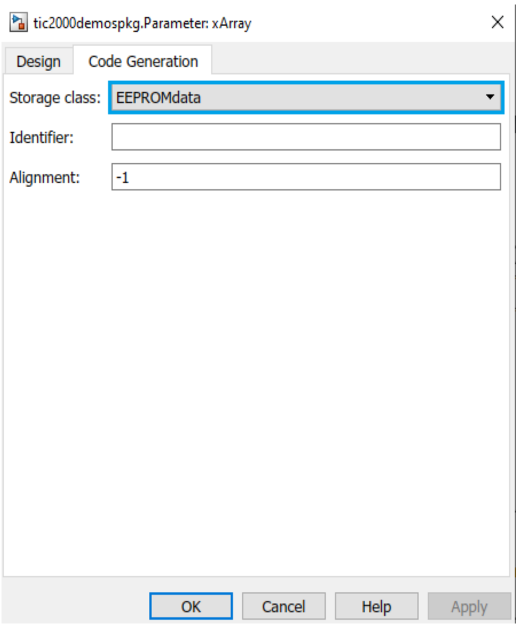


Alternately, just copy the contents of the zip -> +tic2000demospkg.zip file provided with the model, to path(depending on where your support package is installed)-> C:\ProgramData\MATLAB\SupportPackages\R2020b\toolbox\target\support packages\tic2000\dataclasses\ ->(take a backup of orginial folder before replacing & restart MATLAB after copy is done).

Doing above step will avoid the manually defining the storage class from step -6.

8. Defining the calibration parameters

The eeprom_cal.mat file as provided with the model contains calibration paramters namely cal_param3_CAL_UPDATE, cal_param4 that are scalar type, xArray[10] that is a vector type and myParams which is struct type data values. As an example xArray is shown below.



Notice the storage class is selected as EEPROMdata under code generation tab. *We also write 8 * 16bit word at a time to Flash to avoid triggiring the ECC errors. Hence the calibration data is defined as 8*2*16bit size.*

*Scalar type – 2*16bit*

*Vector type – 10 * 16bit*

*Struct type – 4 * 16bit*

9. Run model in external mode and view results

The model is configured to run in external mode and showcases the working of the basic EEPROM Emulation. The Memory Copy blocks contain the Flash address of the xArray[1] and cal_param4 elements. All calibration parameters are copied to RAM location where we can modify them. From the workspace, modify the xArray[1] and cal_param4 values.

Note: cal_param3_CAL_UPDATE is used as flag to copy back the modified calibration values to Flash.

1. When the model is running in external mode, notice xArray[1] = -4, hence count display for xArray[1] in RAM and xArray[1] in Flash will decrement with value -4 each step time. Note xArray[1] value in Flash i.e -4, is copied to xArray[1] in RAM during system initialization. Hence both display would decrement at the same value.
2. Same holds good for cal_param4 which has a value of -4.
3. Now update the cal_param3_CAL_UPDATE = 0 once in workspace and press ctrl+D in model.
4. Now update xArray[1] = 4 and cal_param4 = 4 in workspace followed by updating cal_param3_CAL_UPDATE = 1 in workspace and press ctrl+D in model.
5. With this, the modified calibration values from step-4 will be updated in Flash. And you will see the counters (both in RAM and Flash) counting in opposite direction.
6. Also notice the Page_Counter and Bank_Counter values getting updated.
7. Continue playing around the calibration values and see the effect on the counters. It should be noted the Max Bank's= 3(0,1,2) and Max page's = 8(0-7) in the current setup.

10. Create Calibration Parameter in the Generated code

Defining Calibration parameters is captured in ***point 8***. But one can find more details on the same at the below documentation page:

Create Tunable Calibration Parameter in the Generated Code - MATLAB & Simulink (mathworks.com)