Content: Documentation for Bootloader project, AlbertaSat, Spring 2021

Prepared by: Farough Motamed

Student ID: 1490721

University of Alberta

Bootloader features (version 1):

1. **Two binary images can be written at two different locations in the flash memory. It can be extended to more than two images, but the idea was to have two binary images for working and golden images.**
2. **The code has the ability to calculate the CRC of each image.**
3. **The code checks the CRC of each binary image before executing the image but the CRC has to be manually set into the code. If the CRC of the golden image fails, it jumps to the working image and vice versa.**
4. **The code has the ability to prevent overwriting the golden image by checking a flag that is set when a binary image is written into the flash.**
5. **If the MCU is power reset, or if it is reset using the button on the development kit, it will jump to the address that is assigned in the code (details are provided below).**

Note: Please watch the demo first for version0 and version1 for easy understanding of the work procedure.

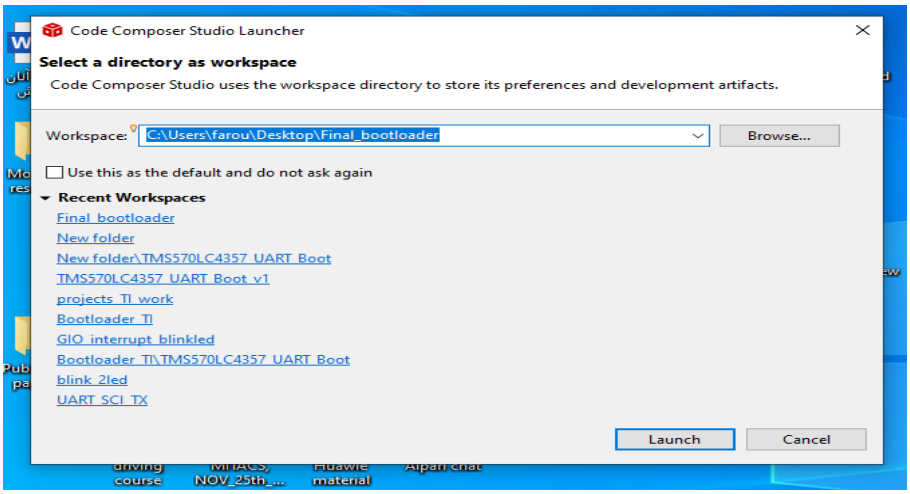
For easier start, please refer to bootloader version 0 on page 10. It explains how to set the address in the binary image that is going to be written into the microcontroller by the bootloader and how to generate the .bin file for the binary image.

Note: I have included two already prepared sample projects and binary images, blinky_led and 2led_blink in the folder for easy start.

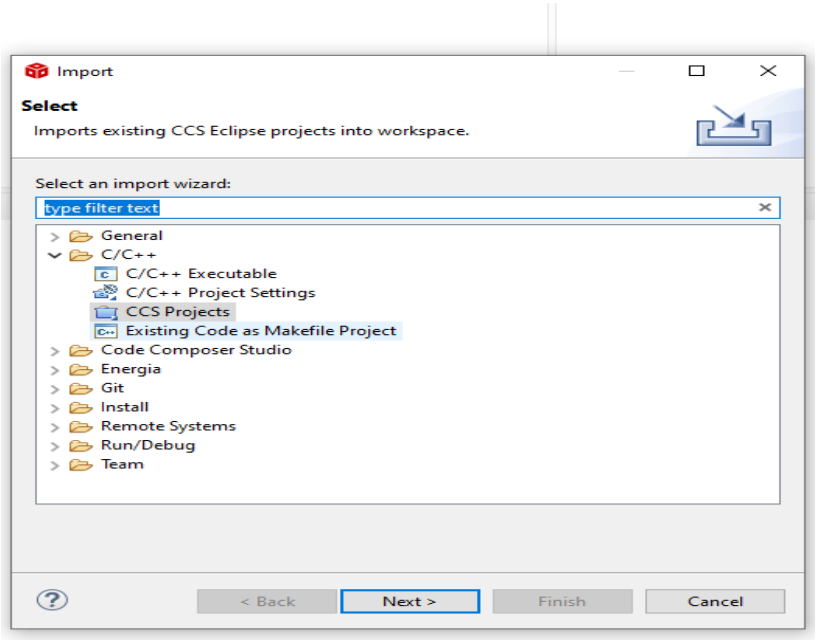
How to use the boot loader version 1:

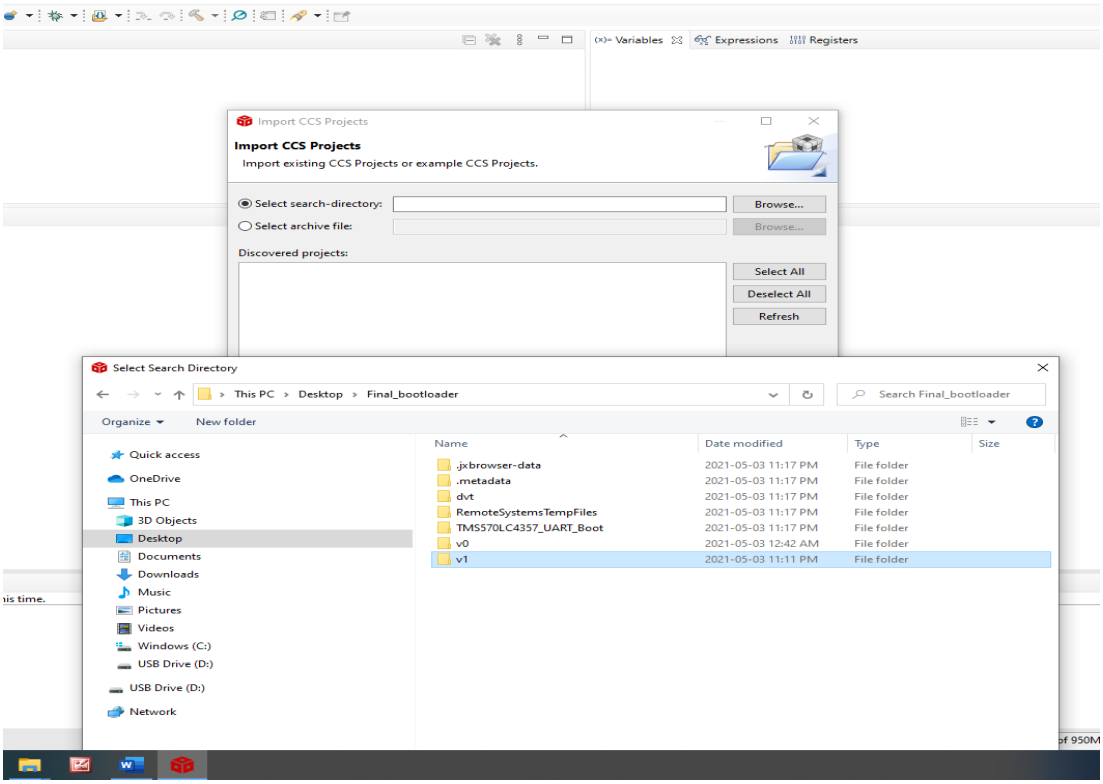
Connect the development kit to a computer using the USB cable. CSS software needs to be installed on the computer.

1. Open the CSS software, select the folder where the bootloader project exist as your workspace and click launch.

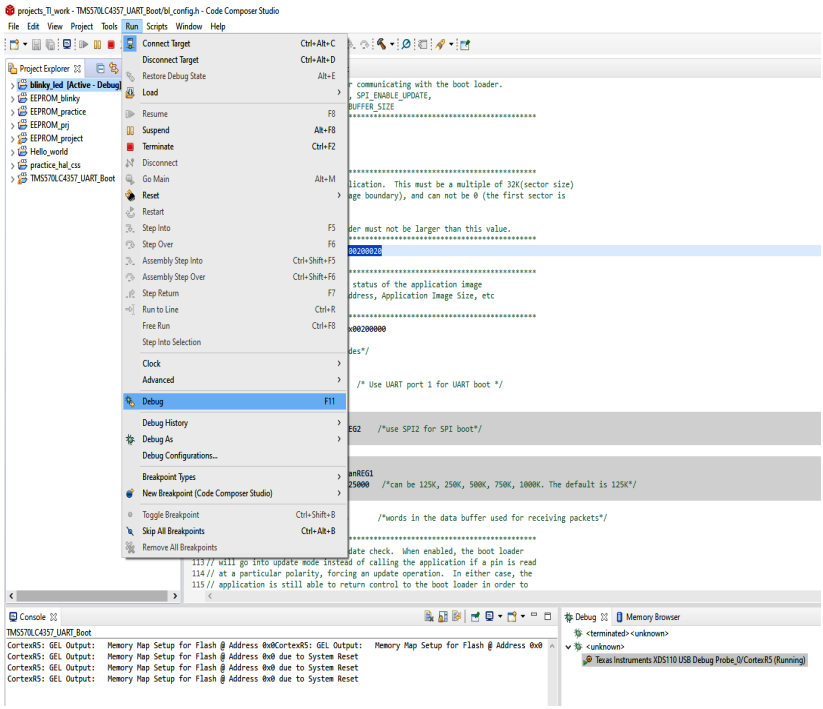


2. At the top left corner, click File, import, C/C++, CSS projects, browse, click on the project folder and click finish. Must tick the box at the bottom that says copy project to workspace.
   Important Note: Once you import the project, another folder named "**TMS570LC4357_UART_Boot**" will be generated in the workspace. This folder is virtual and must be deleted from the workspace folder when you want to import the v0 project unless it will conflict with v0.

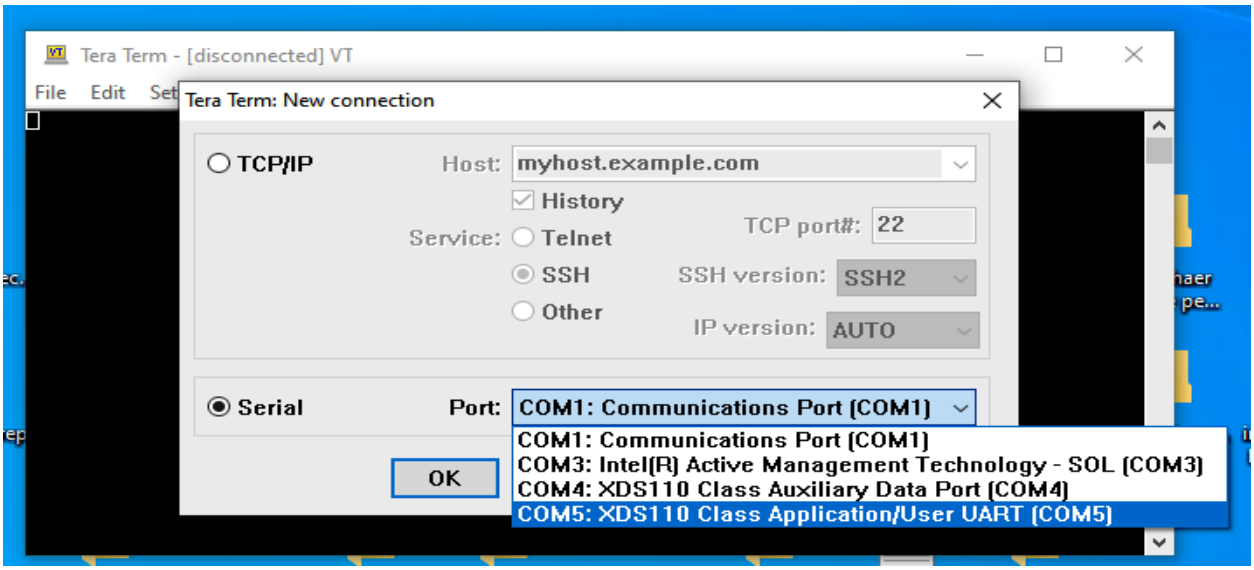




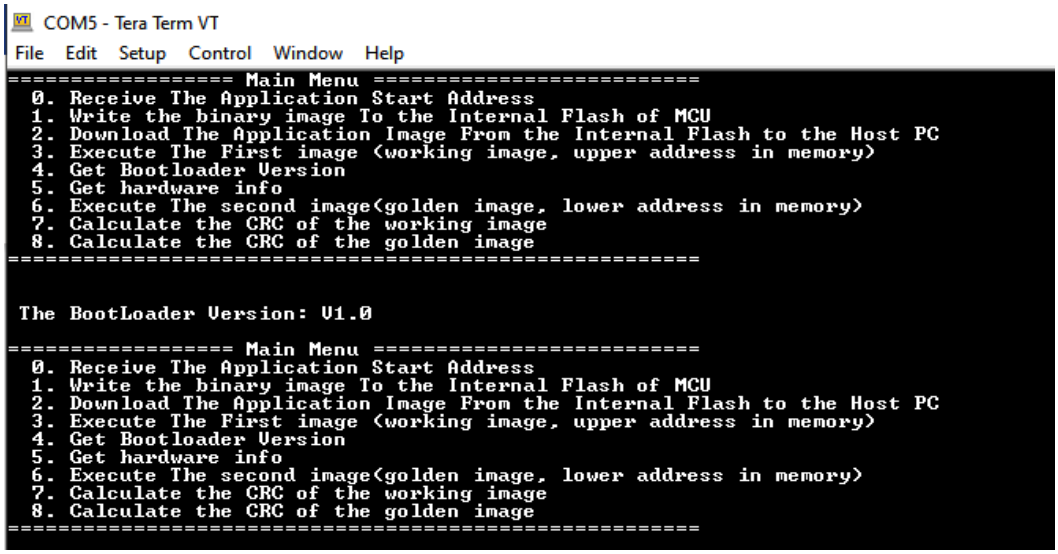
3. Click on run, then debug. The code will be written to the MCU.



4. Install the free software tera term on the computer. Click on serial and select the port corresponding the MCU. (COM5 in my case).

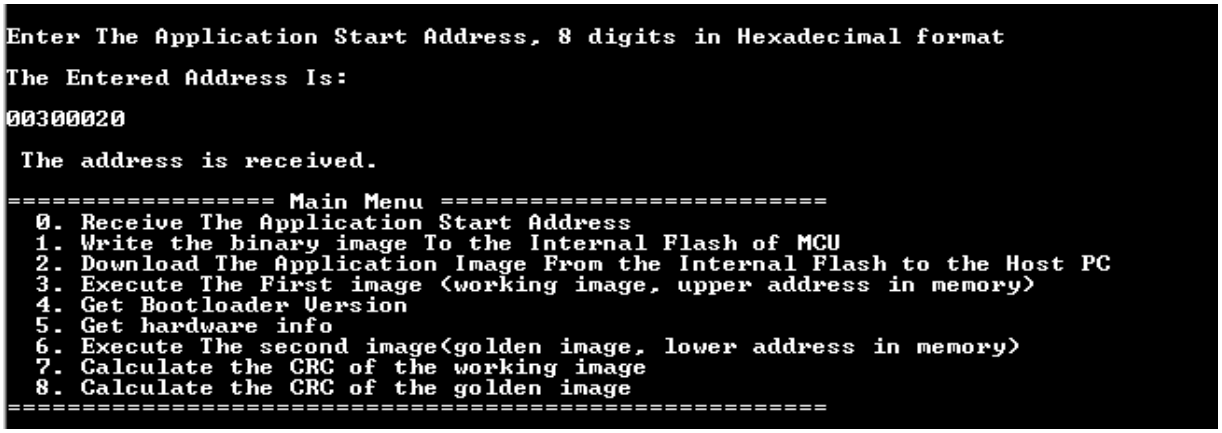


5. Press 4 or 5 on the keyboard to see some information about the hard ware and bootloader version. Note that bootloader version does not change and it is hardcoded, unless it is changed within the code.



6. Each number between 0 to 8 does a task in the bootloader.

7. First push the number 0 and then type the address of the working image (higher address in the memory compared to the address of the golden image). It has to be an 8 digit hex number. For example: 00300020
   This is the address where the binary image will be written in the flash memory. 00300020 means the same as 0x00300020 but what should be typed in the tera term terminal is 00300020.

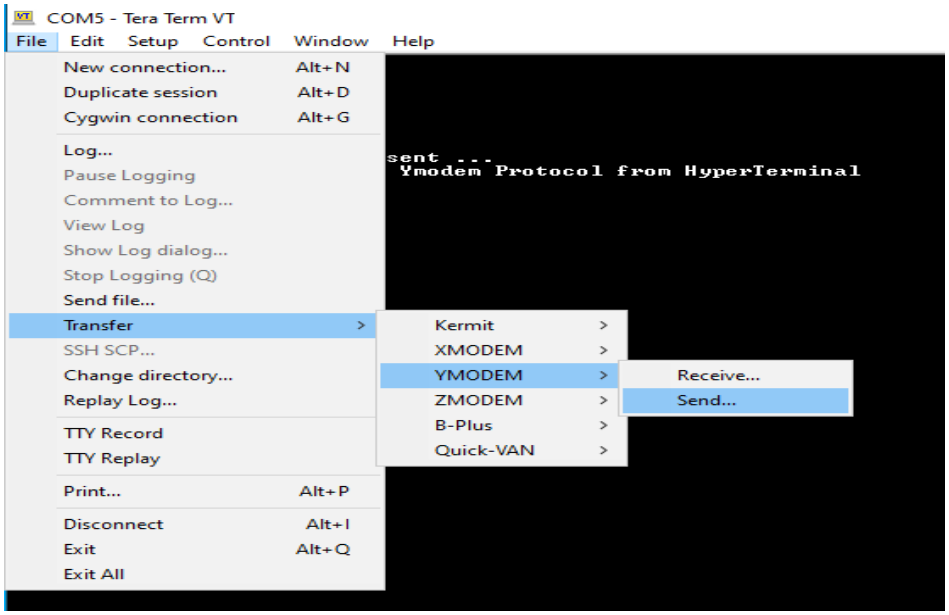


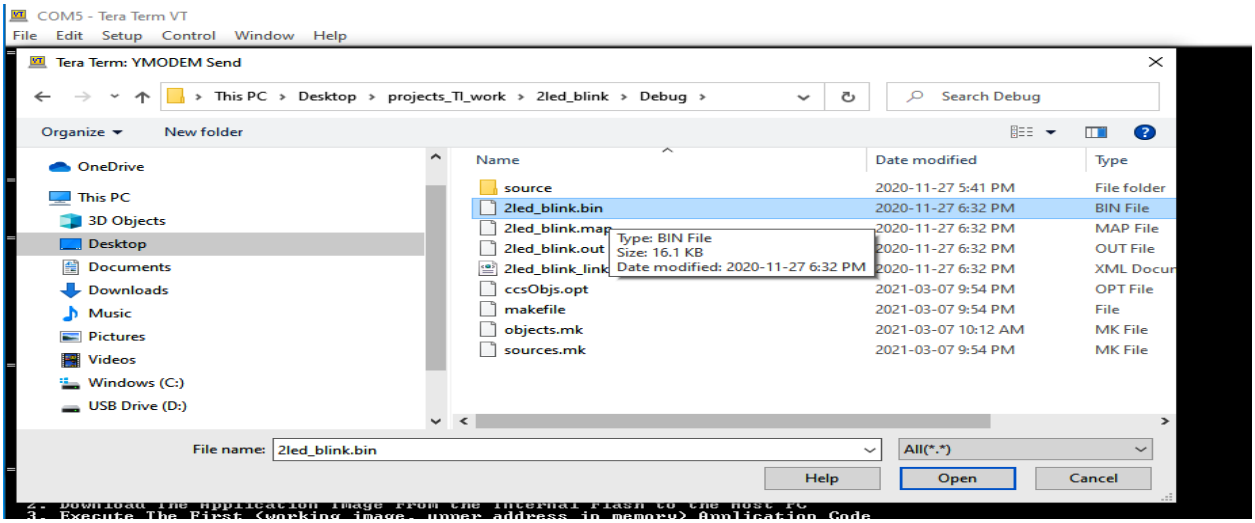
   Note: I have included two binary images. The blinky_led.bin image should be written in the address 0x00200020 and the 2led_blink.bin should be written in the address 0x00300020.

8. Two binary images can be written into the bootloader. I have noted that the image with the higher address (working image, 0x00300020 for 2led_blink.bin) should be written first and then

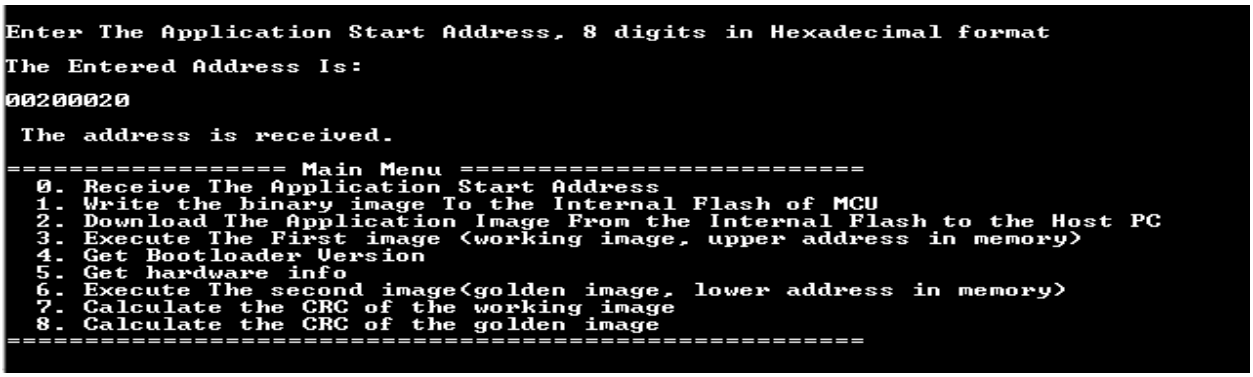
the image for the lower address (golden image, 0x00200020 for blinky_led.bin). It seems that there is a glitch in the code provided by TI that if the lower address is written first and then the higher address, the image for the higher address will glitch during the UART transfer and the write process will not complete.

9. Press button 1 in order to write the working image into the flash. Then click on file, transfer, Ymodem, send. Then click on the 2led_blink.bin binary image in the debug folder of the 2led_blink project. When the message is written successfully, the confirmation message and the size of the bin image will be displayed in tera term terminal.

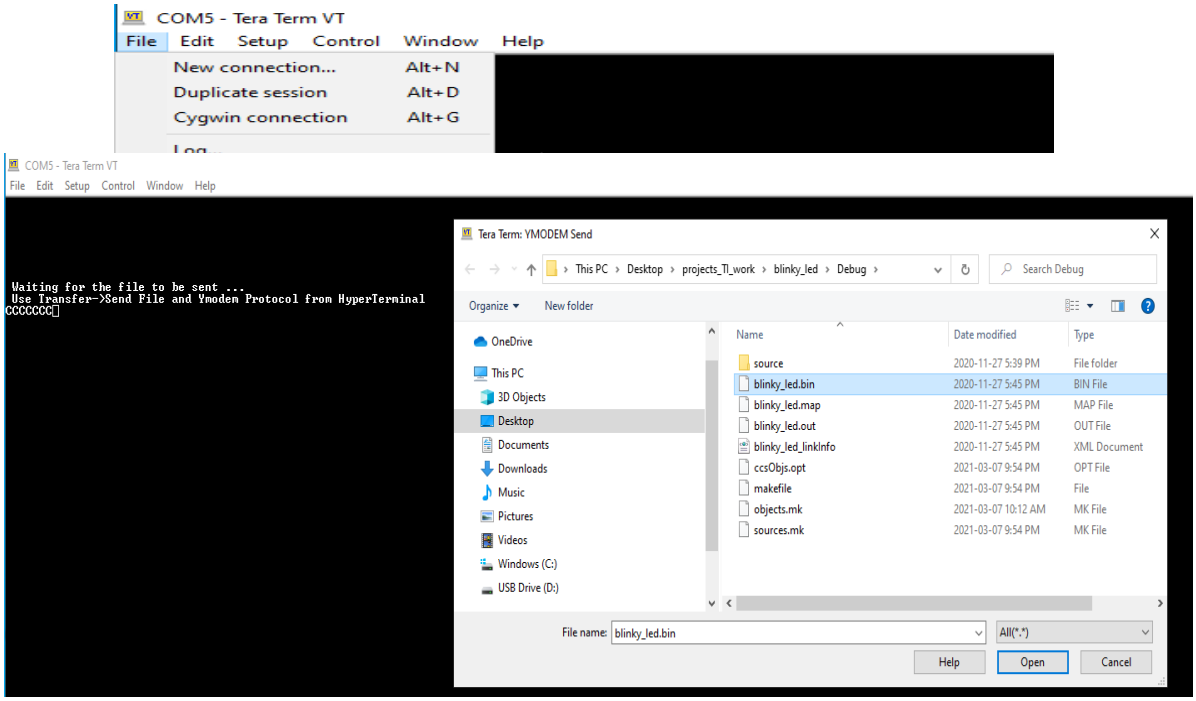




10. Next, push 0 on the keyboard again to send the address for the golden image. Type 00200020.



11. Press 9 to write the golden image to 00200020. Then go to tera term terminal, file, transfer, Ymodem, send. Then select the binary image for the blinky_led.bin from the debug folder.



12. Now if you try to overwrite the golden image, if you first press 0 to type the address of the golden image (0x00200020) and then press 1 on the keyboard to write the image, it will say the image is already written and prevents overwriting , since it checks three items at line 304 of bl_uart.ca

```
  if (  count_write>=3 &&    APP_START_ADDRESS==image_num_arr[1]    &&
(*(uint32_t *)APP_STATUS_ADDRESS) == 0x5A5A5A5A      )
```

    The address sent to the MCU on the second time (golden image address written at image_num_arr[1]), a flag that is set when an image is written in the flash (*(uint32_t *)APP_STATUS_ADDRESS) == 0x5A5A5A5A), and the number of times an image is written in the flash (count_write>=3). Since overwriting the golden image happens on the third write trial (after writing the working and golden images), the overwrite address corresponds to the golden image and the write flag is already set on the first write trial, it will prevent overwriting the golden image. Notice that the write flag 0x5A5A5A5A is written at the address `0x00200000, line 92 of bl_congfig.h (#define APP_STATUS_ADDRESS 0x00200000).`

13. **Note: Never try to overwrite the golden image at the address of the working images and vice versa, since the binary image will be corrupted.**

CRC calculation and CRC check before execution:

14. After writing both working and golden images, the CRC of the images should be calculated. If you press 7, CRC of the working image and if you press 8, the CRC of the golden images will be calculated and printed on the CSS console (not tera terminal). Write down the CRC values and manually write them in the code in bl_uart.ca in lines 50 and 51 (`int CRC_WI=0x4D07;int CRC_GI=0xB371,` WI: working image, GI: golden image). Write the size of the working and golden images in lines 52 and 53: `(int size_GI= 16372; int size_WI= 16564).` Next time when the code wants to execute the images, it will calculate the CRC values and if it is correct, it will execute. I have already followed this procedure and written the CRC and size of the binary and golden images in the code. You can repeat this process for any other binary images you have).

15. After both binary images are written, if you press 3, the first written image (working image at address 0x00300020) will be executed and if you press 6, the second written image (golden image at 0x00200020) will be executed. If the CRC of the golden image fails(due to radiation for instance) the working image will be executed and vice versa.

16. Now if you push the reset button on the board or power cycle the board, the code jumps to the image where its address is assigned to a variable in the code. The variable is in the bl_main.c in line 42:

```
long long int APP_START_ADDRESS=0x00200020;
```

I have set it to be 0x00200020. This is the address I used as the golden image address for the blinky_led.bin image.

Note: It is possible to write a single image into the bootloader and jump to that single image and set the variable in the bl_main.c in line 42 `to be that address.`

```
long long int APP_START_ADDRESS=0x00200020;
```

Run the boot loader version 0: it has a fixed memory location for the golden image.

How to Generate the .bin file for the binary image (blinkey LED) and set the address in the .cmd file of the binary image.

Set the address in the .cmd file of the binary image.

0. Open the CCS software and write the code that you want to convert to .bin image (for instance blinky_LED).

1. How to set the write_address in the .cmd file of the binary image.

Go to the source folder of the project for the binary image (blinky_LED in my example). Open the Hl_sys_link.cmd file. See the example code below. The red line shows the address where the code is going to be written in the MCU flash.

Note: In the example code below for a Hl_sys_link.cmd file of a blinky LED project, the address written in `VECTORS (X): origin=0x00200020 length=0x00000020` (line 59 in Hl_sys_link.cmd) must match the address written in the bl_config.h file of the bootloader code (line 59 of bl_config.h) `#define APP_START_ADDRESS   0x00200020`

The following is an example of the content of file Hl_sys_link.cmd in source folder of blinky_LED project.

```
/*----------------------------------------------------------------------------*/
/* Linker Settings                                                            */

--retain="*(.intvecs)"

/* USER CODE BEGIN (1) */
/* USER CODE END */

/*----------------------------------------------------------------------------*/
/* Memory Map                                                                 */

MEMORY
{
/* USER CODE BEGIN (2) */
/* USER CODE END */
    VECTORS (X)  : origin=0x00200020 length=0x00000020
//    FLASH0  (RX) : origin=0x00000020 length=0x001FFFE0
    FLASH1  (RX) : origin=0x00200040 length=0x00200000-0x40
    STACKS  (RW) : origin=0x08000000 length=0x00001500
```

```
    RAM     (RW) : origin=0x08001500 length=0x0007EB00

/* USER CODE BEGIN (3) */
/* USER CODE END */
}

/* USER CODE BEGIN (4) */
/* USER CODE END */

/*----------------------------------------------------------------------------*/
/* Section Configuration                                                      */

SECTIONS
{
/* USER CODE BEGIN (5) */
/* USER CODE END */
    .intvecs : {} > VECTORS
    .text   align(32) : {} > FLASH0 | FLASH1
    .const  align(32) : {} > FLASH0 | FLASH1
    .cinit  align(32) : {} > FLASH0 | FLASH1
    .pinit  align(32) : {} > FLASH0 | FLASH1
    .bss     : {} > RAM
    .data    : {} > RAM
    .sysmem  : {} > RAM


/* USER CODE BEGIN (6) */
/* USER CODE END */
}
```

2. Add a line of code to the CCS software to enable generating the .bin image.

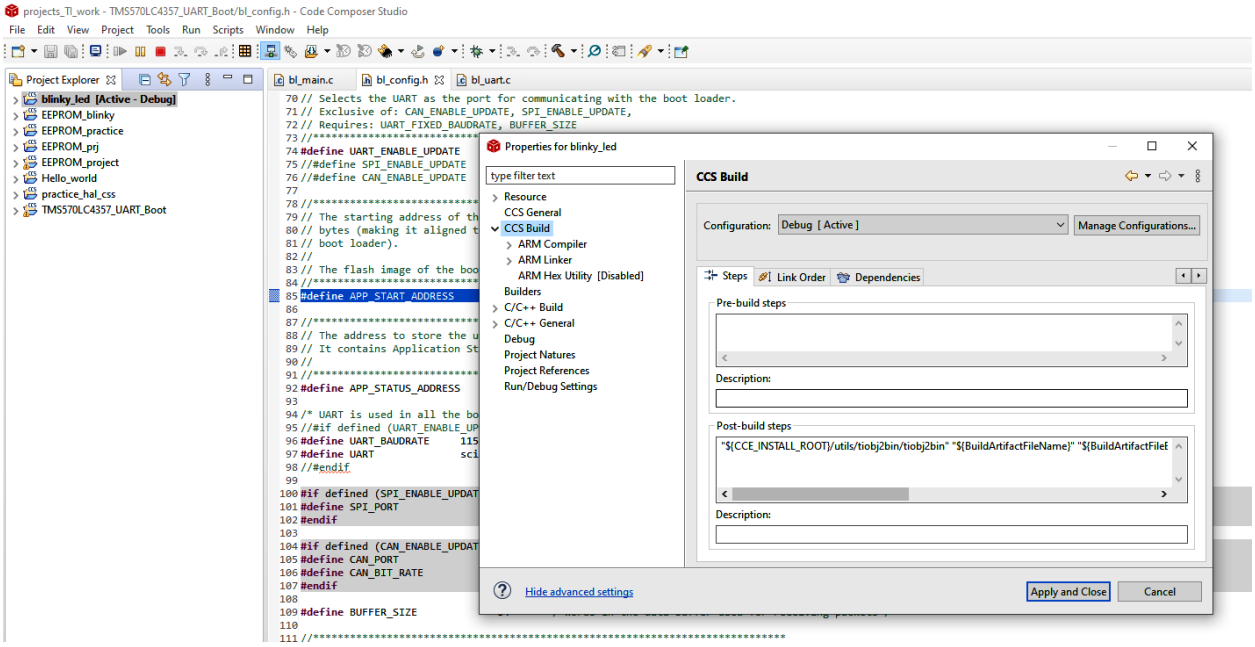
procedure: right click on the project name in the project explorer window in CCS, select the properties at the end of the list. click on CCS build. click on steps tab. Add the below line of code in the Post-build steps window.

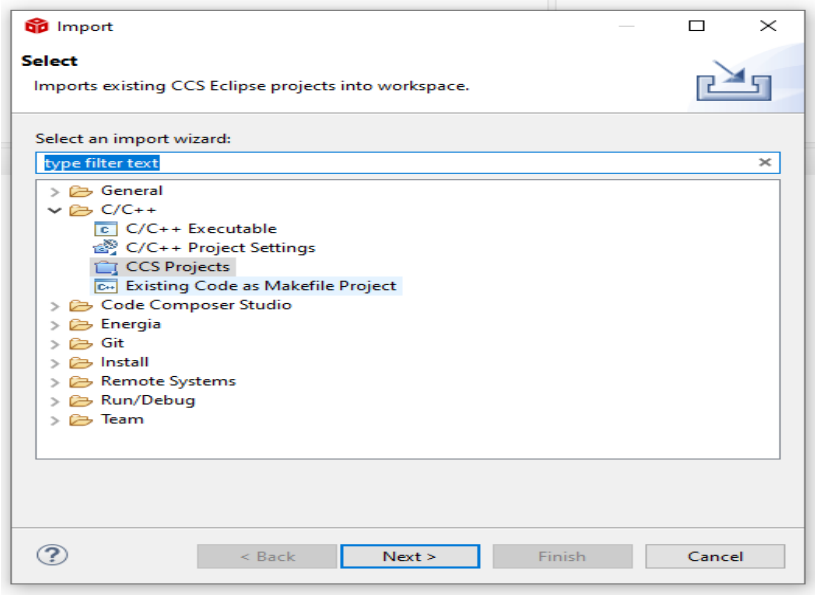
"${CCE_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin" "${BuildArtifactFileName}" "${BuildArtifactFileBaseName}.bin" "${CG_TOOL_ROOT}/bin/armofd" "${CG_TOOL_ROOT}/bin/armhex" "${CCE_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin"

3. Check the debug folder in the project folder to see if there is already a .bin image. Delete the .bin file if it exists.

Build the code to generate the .bin image. It will be generated in the Debug folder.

3. After generating the .bin for the blinky_LED, go to project, import CCS projects, from select search directory, select the project folder for the bootloader.
   Important Note: Once you import the project, another folder named "**TMS570LC4357_UART_Boot**" will be generated in the workspace. This folder is virtual and must be deleted from the workspace folder when you want to import the v1 project unless it will conflict with v1.

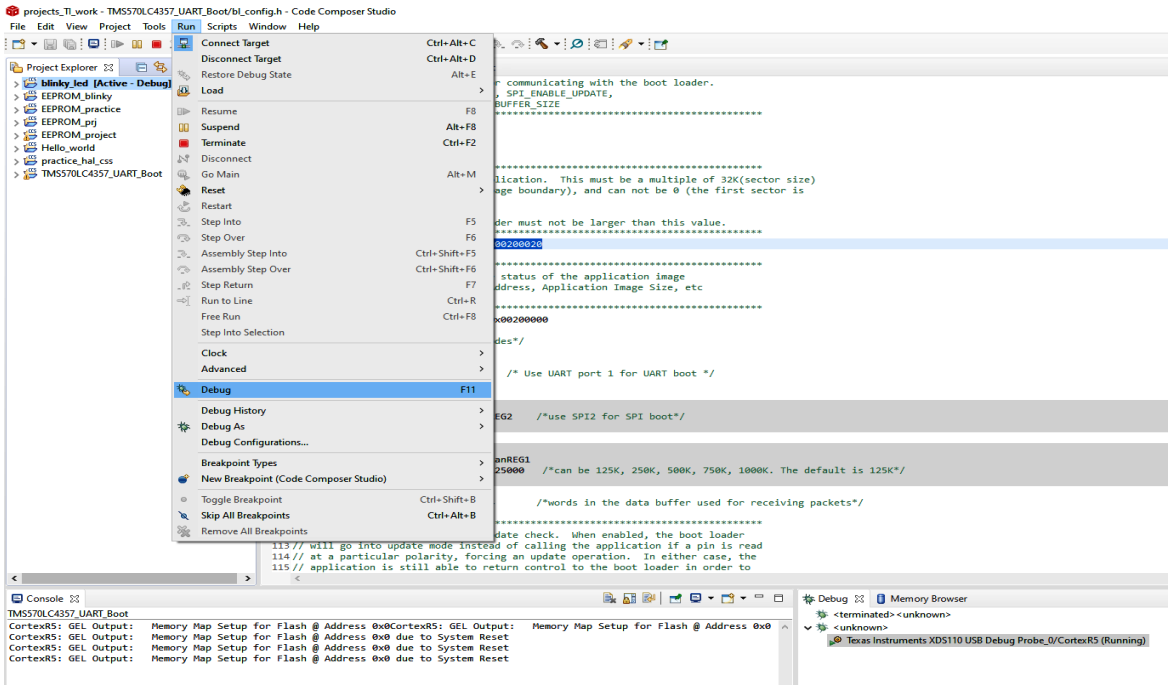




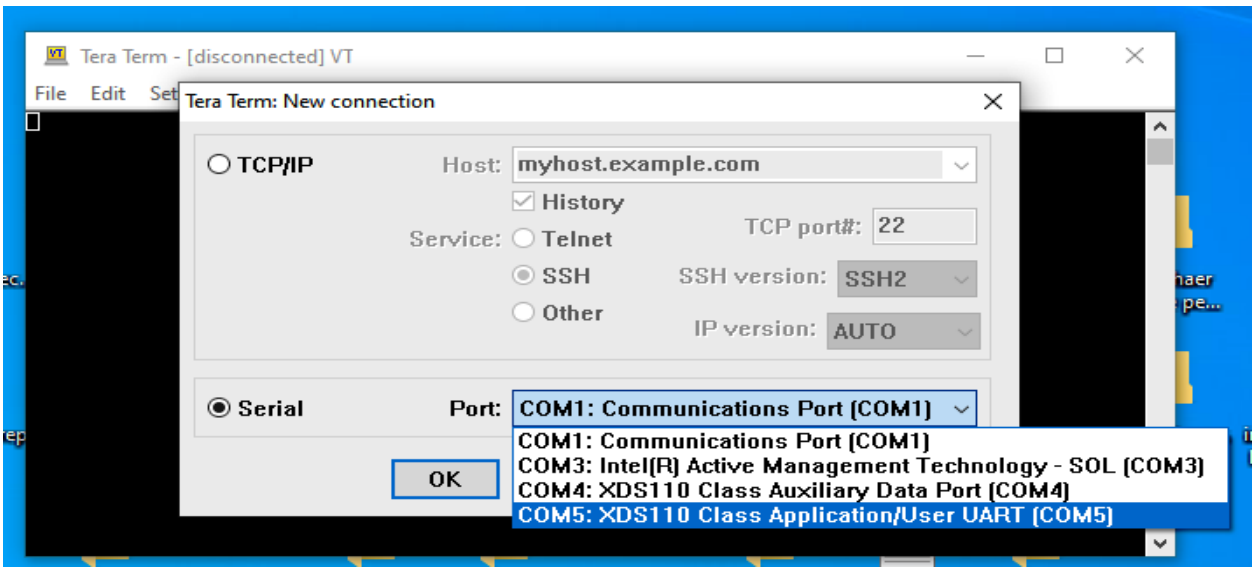
4. In the bootloader project, the file bl_config.h includes the `#define APP_START_ADDRESS 0x00200020 (line 85).`

APP_START_ADDRESS is the start address of the binary image in the Flash memory of the MCU. This address must match the address set in the .cmd file of the binary image (`0x00200020 in my case`).

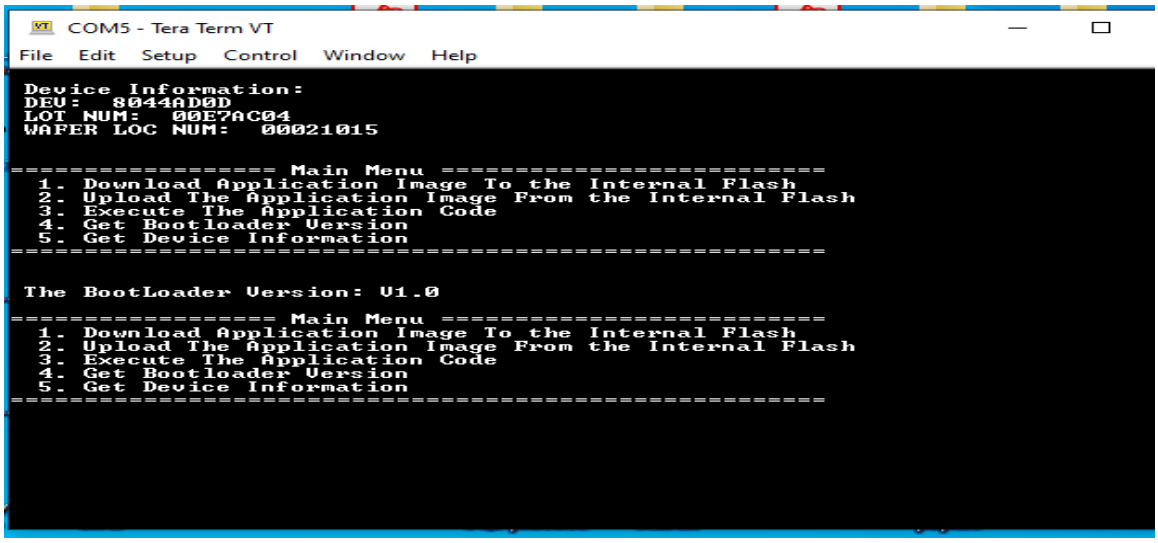
5. Go to Run, from drop down menu, select debug to write the bootloader code into the MCU.



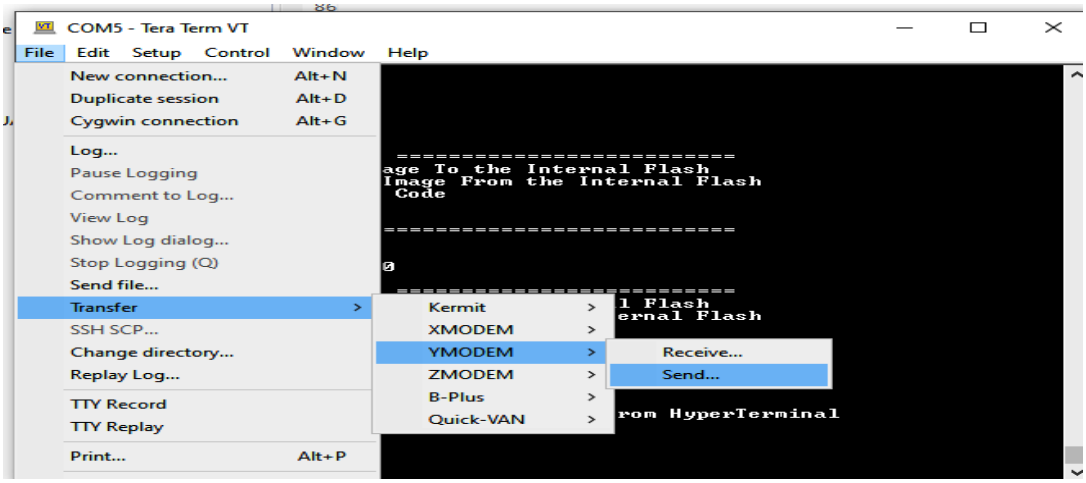
6. Then install the free software, tera term and open the terminal. Select serial and select the come port corresponding to the bootloader project.

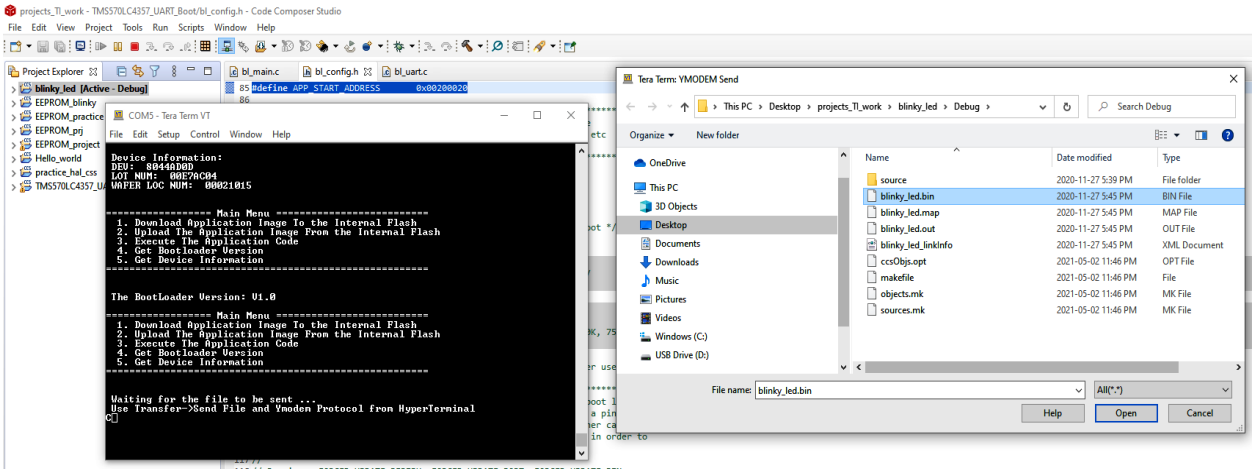


7. Press 4 or 5 on the keyboard to get some hardware and version info (this information is hard coded, you can change it within the code).



8. Press 1 to write the binary image to the flash. Then click file, transfer, Ymodem, send and then select the .bin image.





9. Press 3 to execute the binary image. If you reset or power cycle the MCU, the image will run again. Since after reset, it checks a flag (0x5A5A5A5A)written at bl_config.h (line 92, `#define APP_STATUS_ADDRESS  0x00200000),  if the flag is sets, if runs the code written at line 85 of` bl_config.h (`#define APP_START_ADDRESS  0x00200020 ). Notice that the flag is set, once the binary image in written in the flash of MCU.`

Appendix:

Hercules Tutorial: Using the SCI for UART Communication

https://www.youtube.com/watch?v=PpalANwuzIo&t=603s

Teaches how to integrate HALCoGen and CCS.

Steps to combine HALCoGen and Code Composer Studio:

In HALCoGen:

1. Open the HALCoGen software.
2. File, New, project.
3. Select the microcontroller family and then the microcontroller (For bootloader project TMS57043x, The first option)
4. In the name field, write a name for the project and click next to generate a folder for the project in the workspace,
5. Go to Driver enable tab and select the required drivers (GIO, SCI, FEE,…)
6. Go to the tab corresponding to each driver and change the settings if required.
7. Go to file and click generate code.

In code composer studio:

1. Open the CCS software. File, new, project, CCS project
2. Select the microcontroller family and the connection and click verify to make sure the connection works.
3. In the project name, write the exact name of the folder that was generated by HALCoGen in the same directory.
4. Delete the main.c file generated by CCS.
5. Right click on the project name, go to properties, ARM compiler, Include options.
6. In the above rectangle (Add dir to #include search path …) , click on the + green button. Then click on workspace, go to the project folder and select the include folder, then click apply and close.

To use the EEPROM, after doing above steps:

Add F021 Library files to CCS Project

* - Add/Link F021_API_CortexR4_BE_L2FMC_V3D16.lib from folder C:/ti/Hercules/F021 Flash API/2.01.01 to CCS Project

* - Add Path C:/ti/Hercules/F021 Flash API/2.01.01/include to Include Path in CCS Project

In CSS, right click on the project, go to properties, click on ARM compiler, include options, in the fist rectangle click on the + green button, from browse directory, add the following include file to include search path:

C:\ti\Hercules\F021 Flash API\02.01.01\include

2. Then click on the ARM linker, go to file search path, in the first box (include library file or command file as input), click browse and add the following .lib file

C:\ti\Hercules\F021 Flash API\02.01.01\F021_API_CortexR4_BE_L2FMC_V3D16.lib

On the same page, go to the next box (add <dir > to library search path), add the following line:

C:\ti\Hercules\F021 Flash API\02.01.01\include

Note: It is not clear to me yet how the EEPROM works. It seems that it is not possible to save several variables/parameters into the EEPROM and retrieve them after the power is cut off).

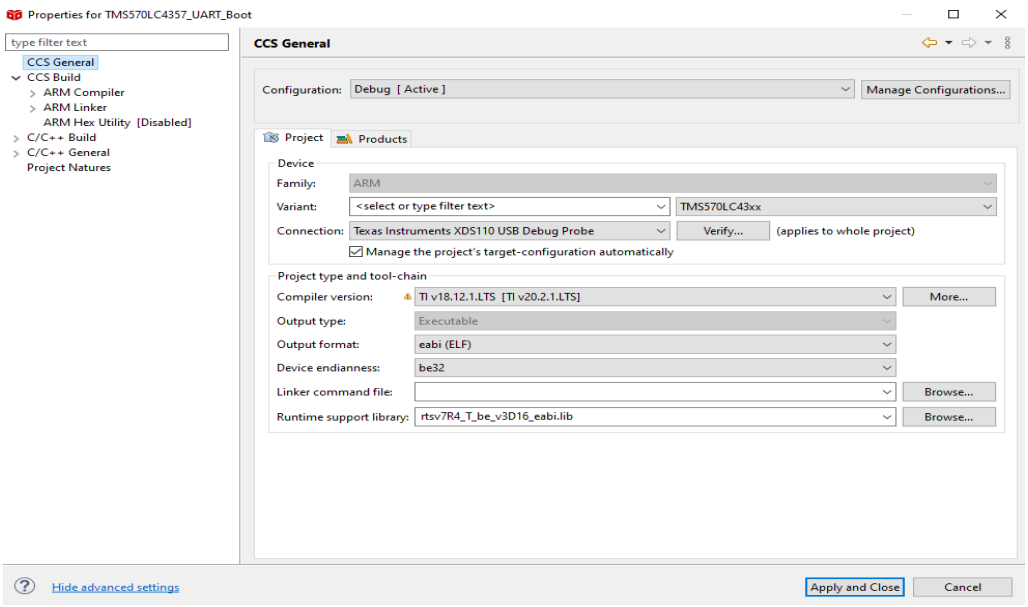
There is a PDF in the help section of the MCU that shows how to use the EEPROM, but as described above, it seems that it can save only a single variable.

How to resolve the following error after clicking on debug to write the code into the MCU.

Texas Instruments XDS100v2 USB Debug Probe_0/CortexR5 : Target must be connected before loading program.

After importing the project into the workspace, click on run then debug in the CCS software.

Then go to project, show build settings, and then select the right connection type: Texas instruments XDS 110 USB Debug probe



To erase the MCU flash fast (do not use this option for safer operation).

In CSS, right click on the project name, go to properties, go to debug, flash settings, then go to Erase options and select necessary sectors only. This option does not erase the EEPROM if EEPROM is used.

Online CRC calculator:

https://crccalc.com/

Getting Started with Code Composer Studio v9.3 - YouTube

https://www.youtube.com/watch?v=2W0ZHO0vzJE&list=PL3NIKJ0FKtw77BTkTKo_OXYLaT0lL9_zc&index=3