Aim of this project was to test the boot-loader design.

Section	Addess	Size
BootloaderCode	0x0800_0000 0x0800_5FFF	24KB
metadata_img_0	0x0800_6000 0x0800_6FFF	4KB
Img0_code	0x0800_7000 0x0800_CFFF	24KB
metadata_img_1	0x0800_D000 0x0800_DFFF	4KB
Img1_code	0x0800_E000 0x0801_3FFF	24KB

Figure 1: Organisation of Flash

Img0_code and Img1_code represents application code which could be selected by Bootloader for execution.

metadata_img_0 and metadata_img_1 are the meta-data about corresponding images. This would be used by bootloader.

Step1: Img_0 Setup

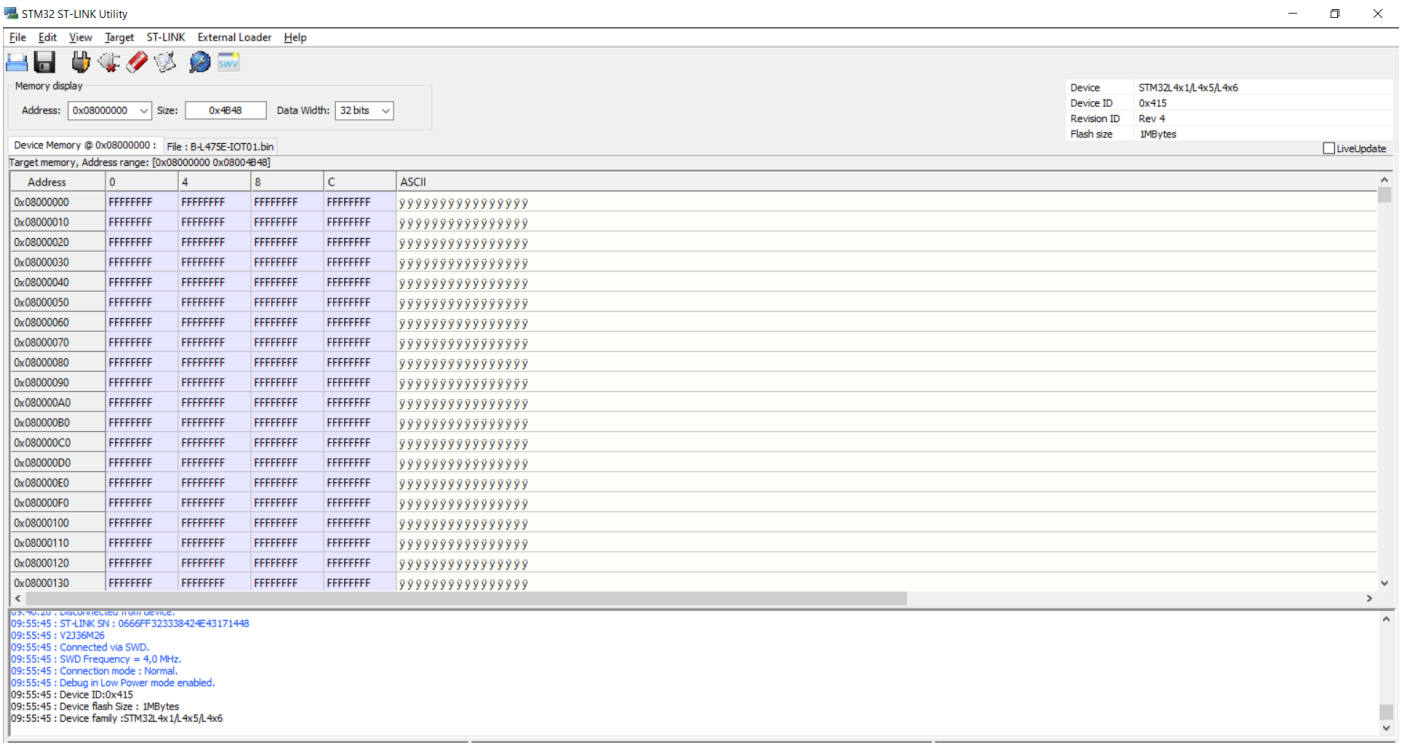


Figure 2:Erase the whole Flash Memory using STM-tool

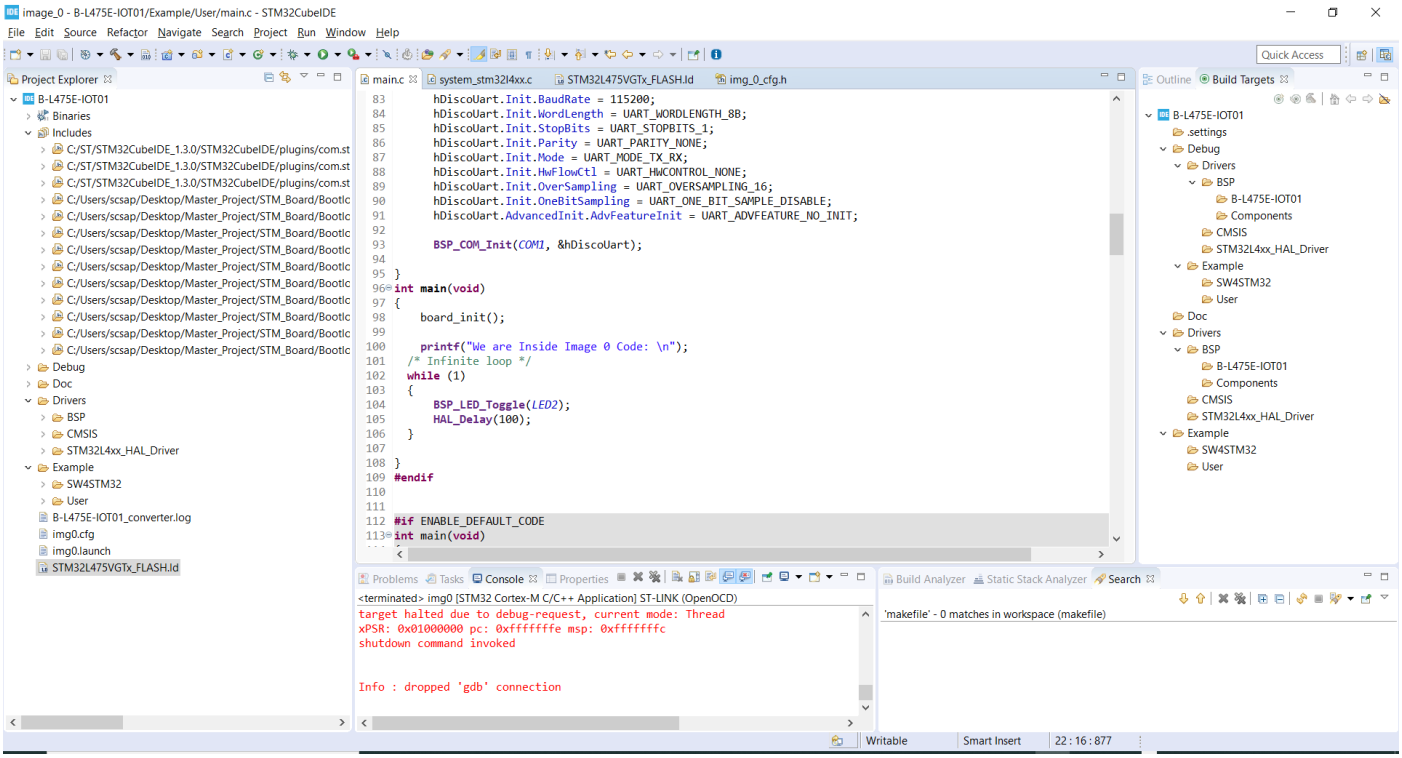


Figure 3: Image_0 Application code.

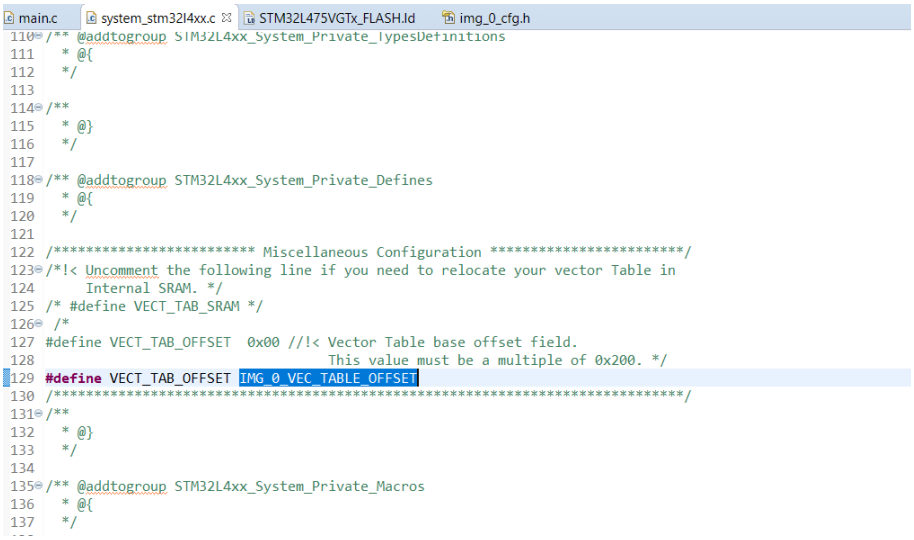


Figure 4:Img_0 vector Table offset

main.c | system_stm32l4xx.c | STM32L475VGTx_FLASH.ld | img_0_cfg.h

```
34
35 /* Highest address of the user mode stack */
36 _estack = 0x20018000;    /* end of RAM */
37 /* Generate a link error if heap and stack don't fit into RAM */
38 _Min_Heap_Size = 0x200;      /* required amount of heap  */
39 _Min_Stack_Size = 0x400; /* required amount of stack */
40
41 /* Specify the memory areas */
42 MEMORY
43 {
44 RAM (xrw)      : ORIGIN = 0x20000000, LENGTH = 96K
45 RAM2 (xrw)      : ORIGIN = 0x10000000, LENGTH = 32K
46 FLASH (rx)      : ORIGIN = 0x8007000, LENGTH = 1024K
47 }
48
49 /* Define output sections */
50 SECTIONS
51 {
52   /* The startup code goes first into FLASH */
53   .isr_vector :
54   {
55     . = ALIGN(8);
56     KEEP(*(.isr_vector)) /* Startup code */
57     . = ALIGN(8);
58   } >FLASH
59
60   /* The program code and other data goes into FLASH */
61   .text :
62   {
```

Figure 5: Change in Img_0 Load Script.

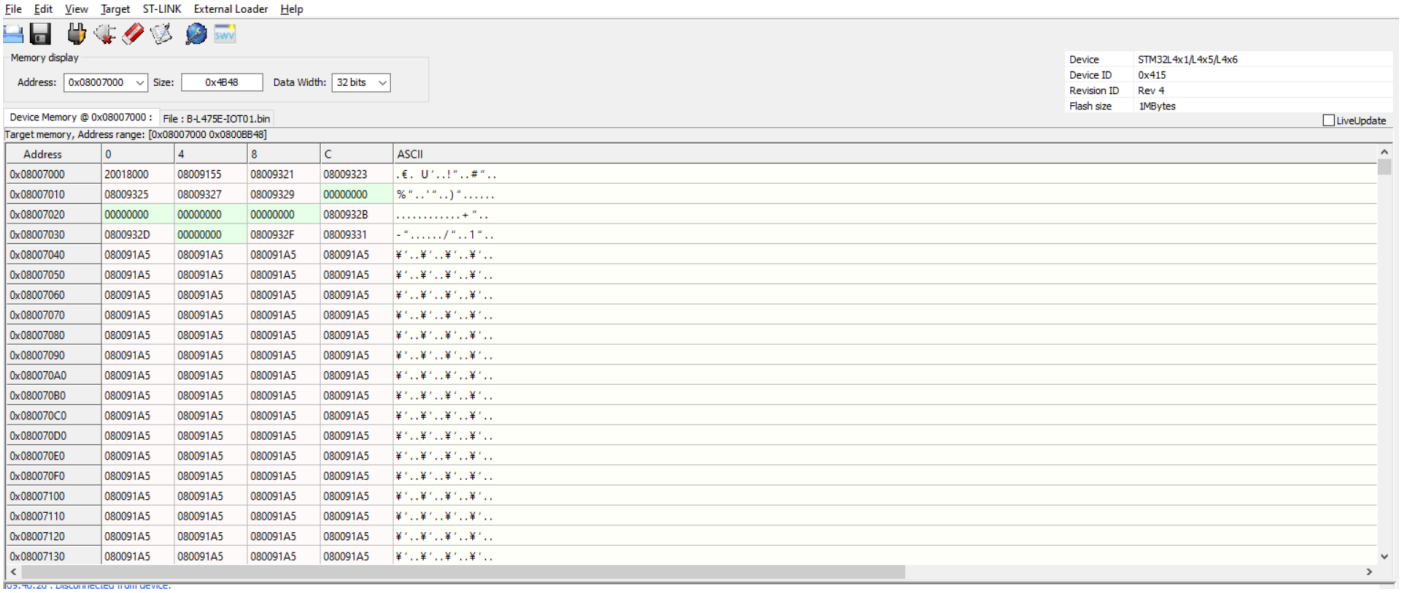


Figure 6:Verify Img_0 code is programmed into Flash.

Step 2: Img_1 Setup

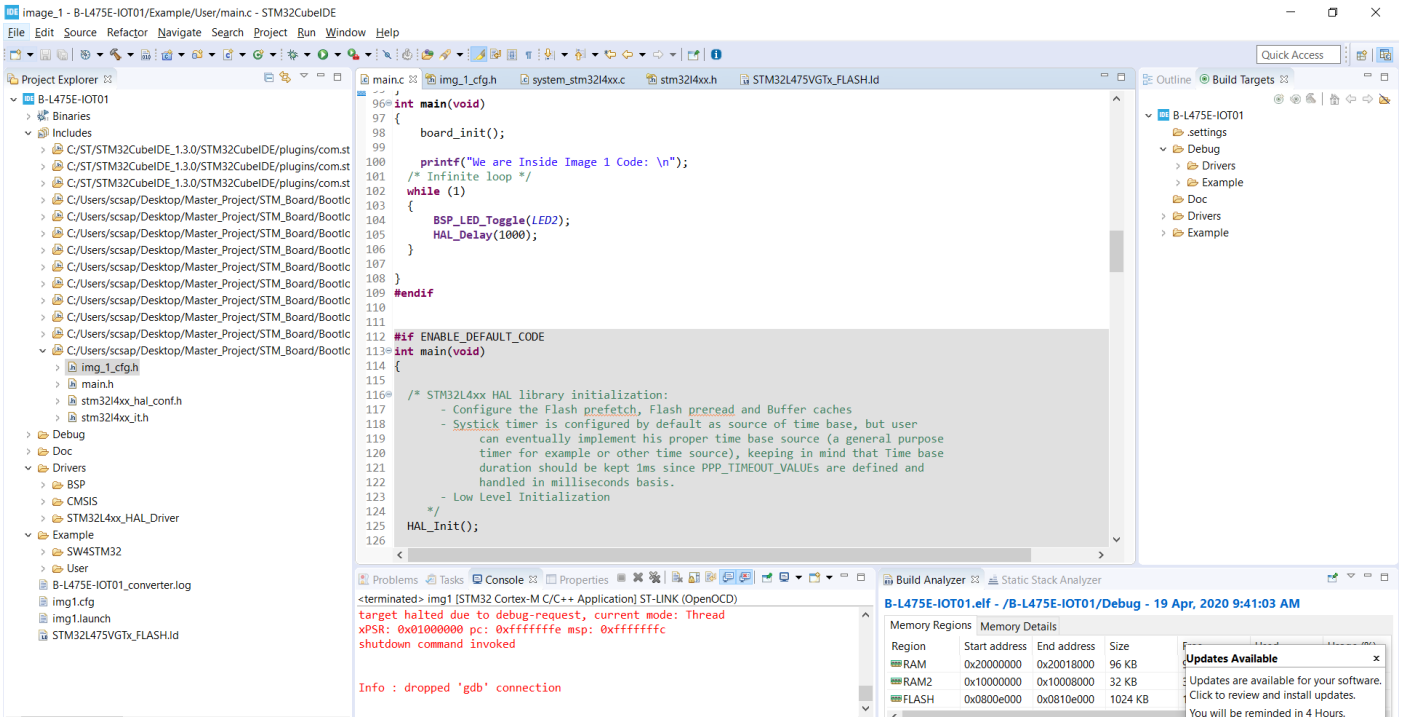


Figure 7: Image_1 Application Code.

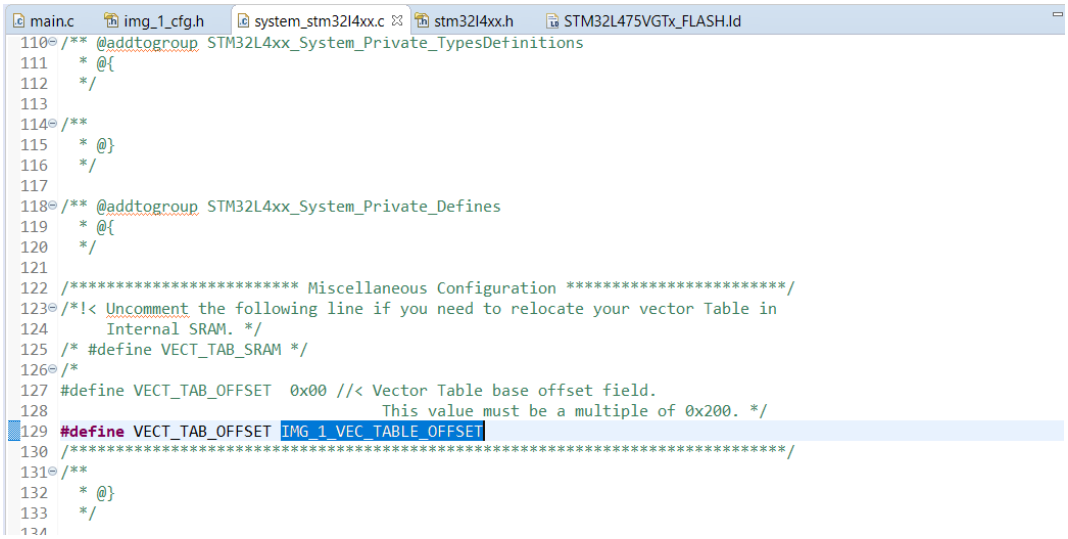


Figure 8: Img_1 Vector Offset.

main.c | img_1_cfg.h | system_stm32l4xx.c | stm32l4xx.h | STM32L475VGTx_FLASH.ld

```
MEMORY
{
RAM (xrw)      : ORIGIN = 0x20000000, LENGTH = 96K
RAM2 (xrw)      : ORIGIN = 0x10000000, LENGTH = 32K
FLASH (rx)      : ORIGIN = 0x800E000, LENGTH = 1024K
}

/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(8);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(8);
  } >FLASH

  /* The program code and other data goes into FLASH */
  .text :
  {
    . = ALIGN(8);
    *(.text)           /* .text sections (code) */
    *(.text*)          /* .text* sections (code) */
    *(.glue_7)         /* glue arm to thumb code */
    *(.glue_7t)        /* glue thumb to arm code */
    *(.eh_frame)

    KEEP (*(.init))
    KEEP (*(.fini))
```

Figure 9:Img_1 Load script changes.

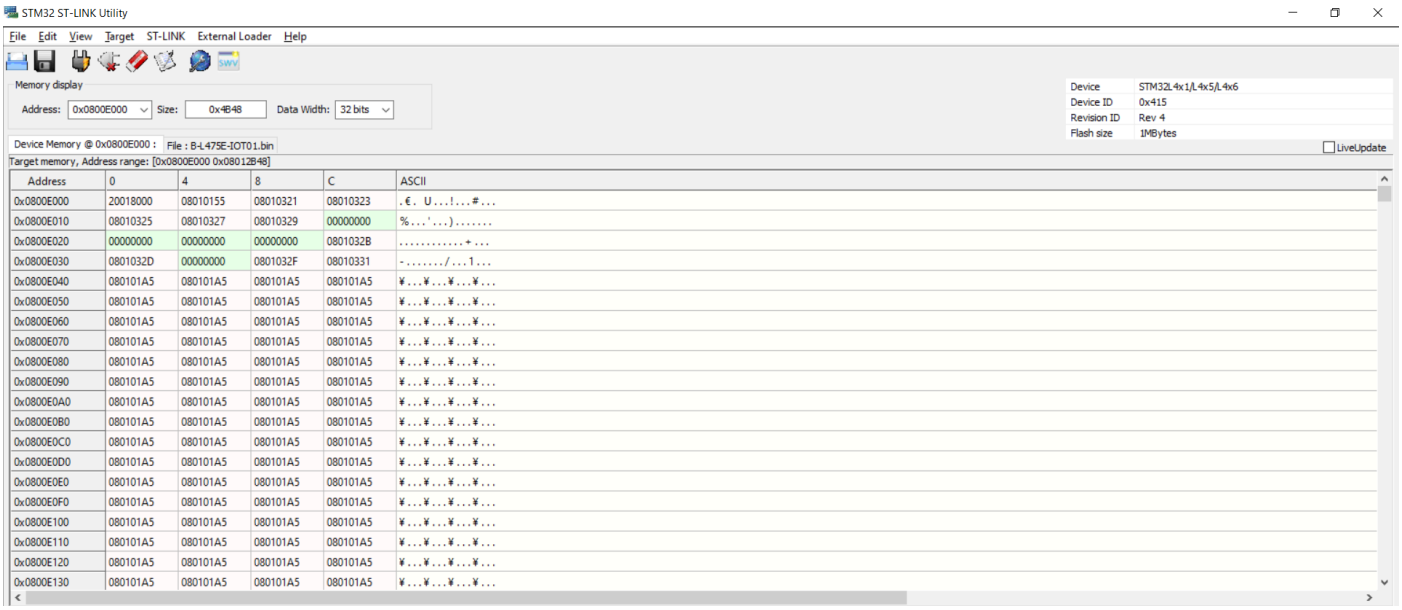


Figure 10: Verify the programmed Img_1 in Flash

Step 3: Bootloader Setup

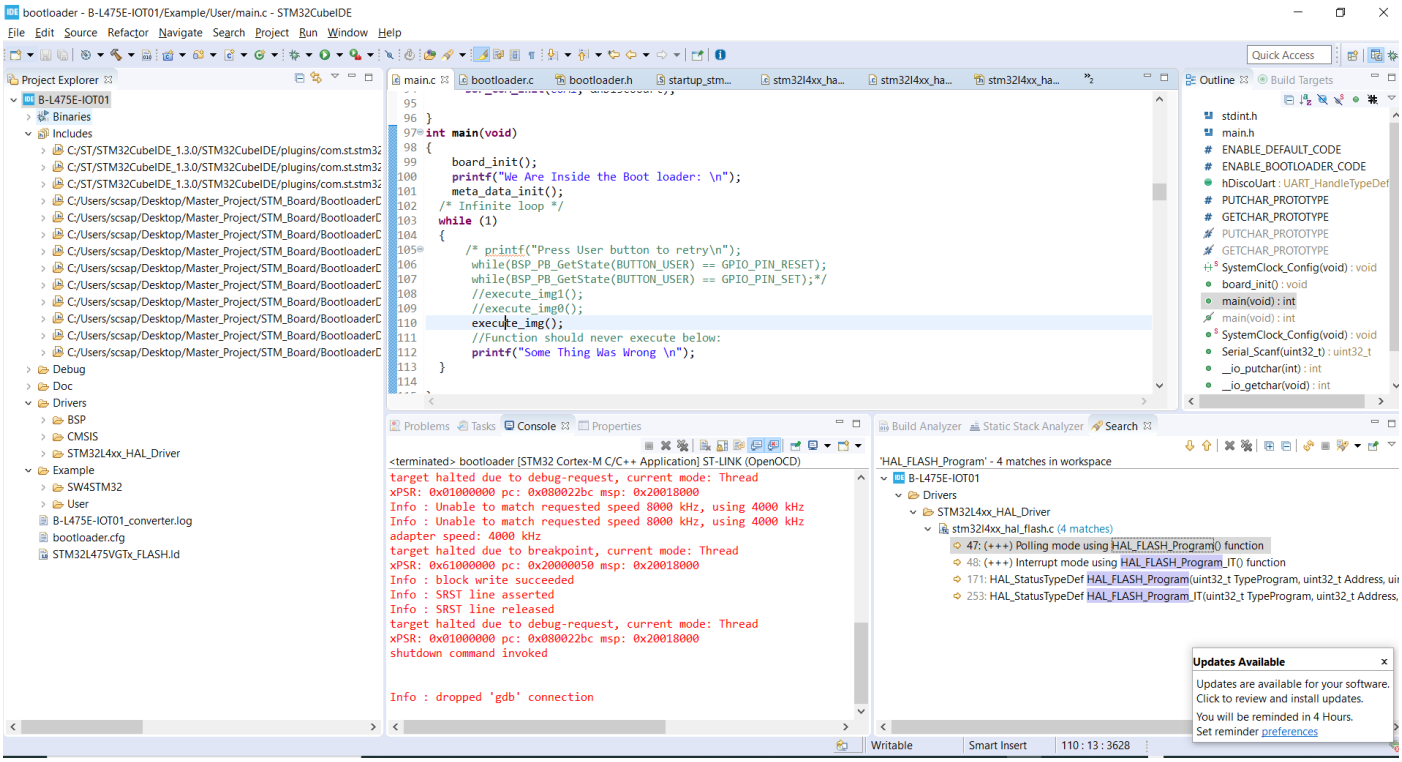


Figure 11: Bootloader Code

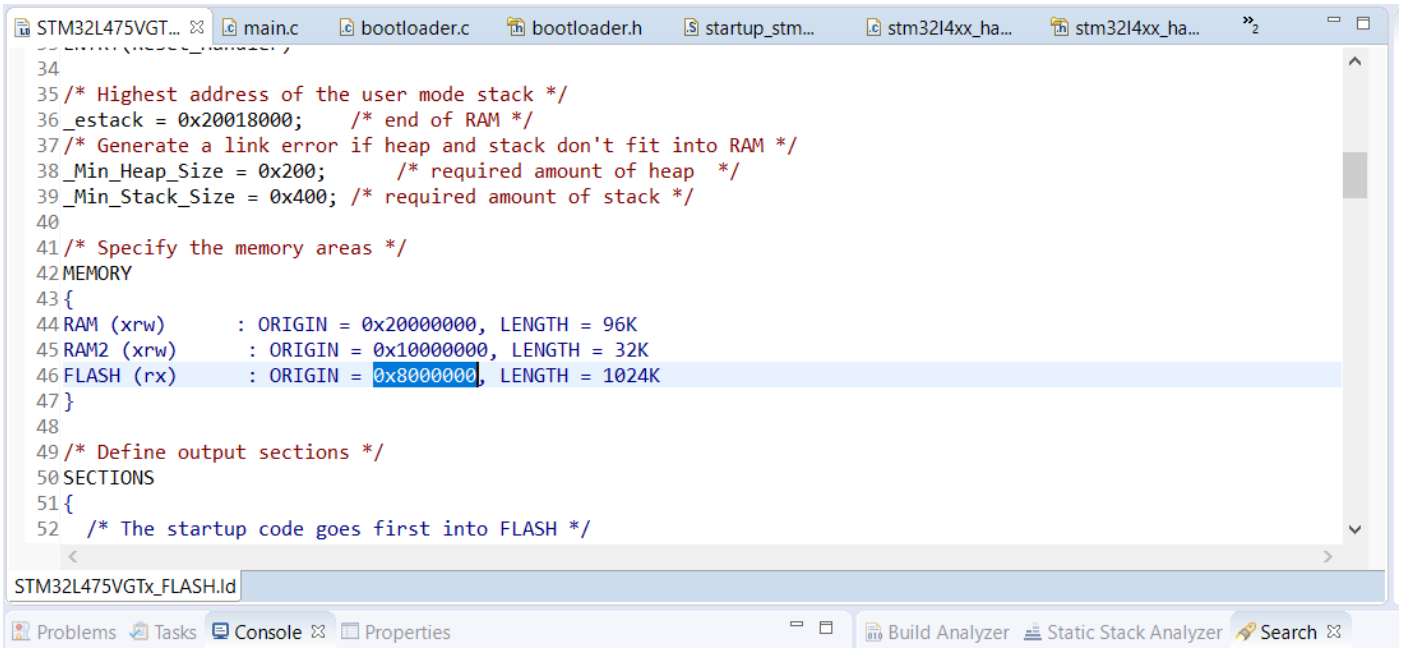
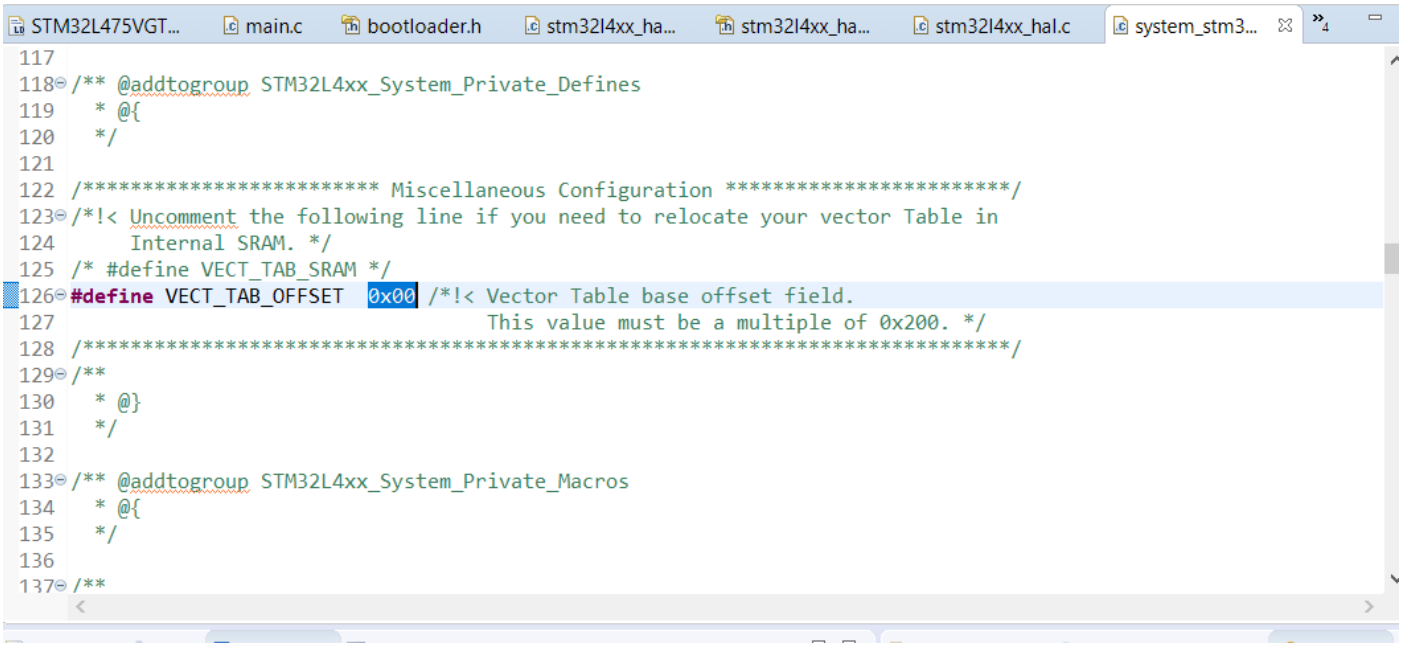
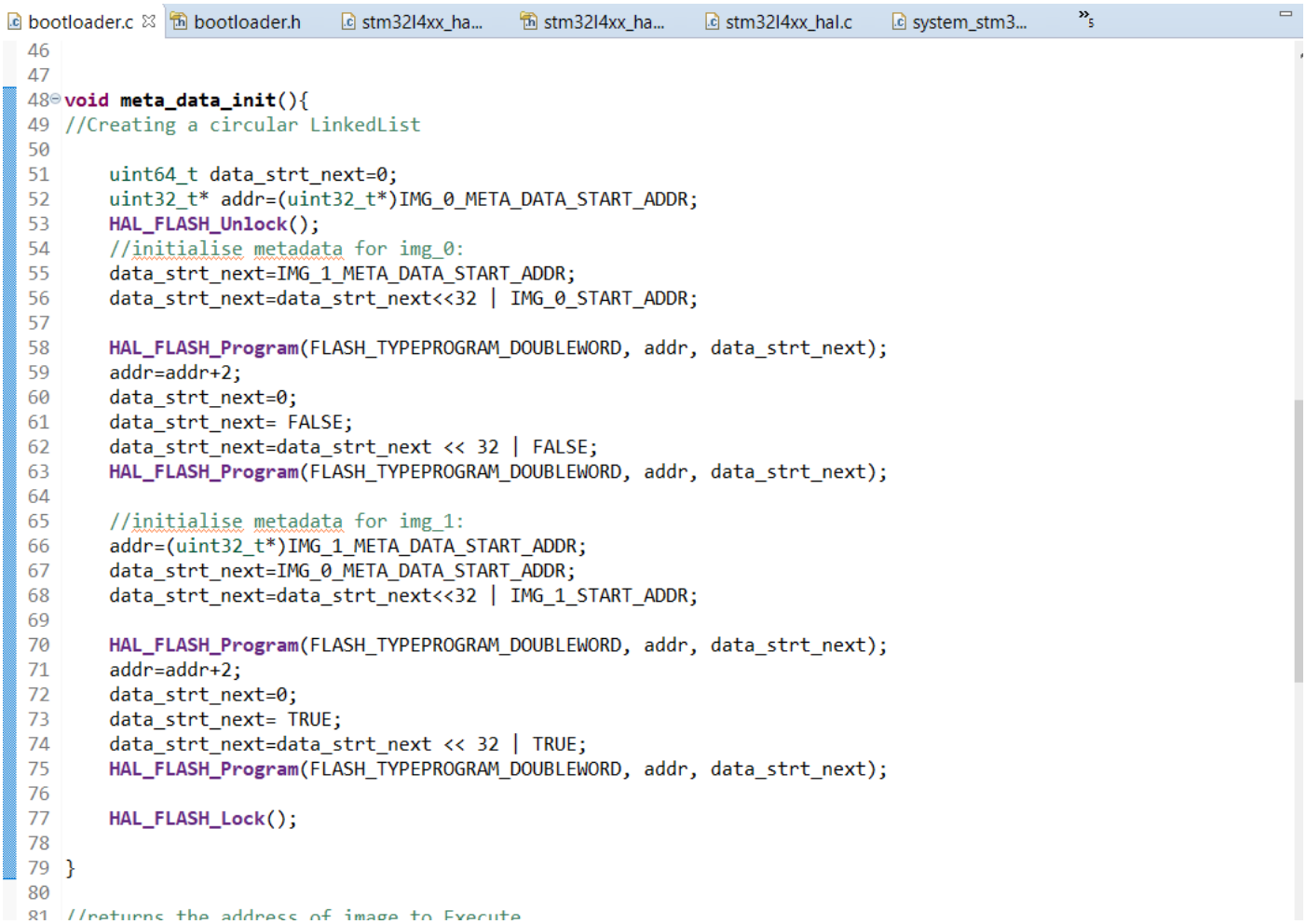


Figure 12: Bootloader Loader Scrip Changes.



```
117
118 /** @addtogroup STM32L4xx_System_Private_Defines
119   * @{
120   */
121
122 /************************ Miscellaneous Configuration ************************/
123 /*!< Uncomment the following line if you need to relocate your vector Table in
124      Internal SRAM. */
125 /* #define VECT_TAB_SRAM */
126 #define VECT_TAB_OFFSET  0x00 /*!< Vector Table base offset field.
127                                   This value must be a multiple of 0x200. */
128 /******************************************************************************/
129 /**
130   * @}
131   */
132
133 /** @addtogroup STM32L4xx_System_Private_Macros
134   * @{
135   */
136
137 /**
```

Figure 13: Bootloader code Vect_Table Offset.



```
46
47
48 void meta_data_init(){
49 //Creating a circular LinkedList
50
51     uint64_t data_strt_next=0;
52     uint32_t* addr=(uint32_t*)IMG_0_META_DATA_START_ADDR;
53     HAL_FLASH_Unlock();
54     //initialise metadata for img_0:
55     data_strt_next=IMG_1_META_DATA_START_ADDR;
56     data_strt_next=data_strt_next<<32 | IMG_0_START_ADDR;
57
58     HAL_FLASH_Program(FLASH_TYPEPROGRAM_DOUBLEWORD, addr, data_strt_next);
59     addr=addr+2;
60     data_strt_next=0;
61     data_strt_next= FALSE;
62     data_strt_next=data_strt_next << 32 | FALSE;
63     HAL_FLASH_Program(FLASH_TYPEPROGRAM_DOUBLEWORD, addr, data_strt_next);
64
65     //initialise metadata for img_1:
66     addr=(uint32_t*)IMG_1_META_DATA_START_ADDR;
67     data_strt_next=IMG_0_META_DATA_START_ADDR;
68     data_strt_next=data_strt_next<<32 | IMG_1_START_ADDR;
69
70     HAL_FLASH_Program(FLASH_TYPEPROGRAM_DOUBLEWORD, addr, data_strt_next);
71     addr=addr+2;
72     data_strt_next=0;
73     data_strt_next= TRUE;
74     data_strt_next=data_strt_next << 32 | TRUE;
75     HAL_FLASH_Program(FLASH_TYPEPROGRAM_DOUBLEWORD, addr, data_strt_next);
76
77     HAL_FLASH_Lock();
78
79 }
80
81 //returns the address of image to Execute
```

Figure 14: Bootloader Function to initialise meta-data for both images

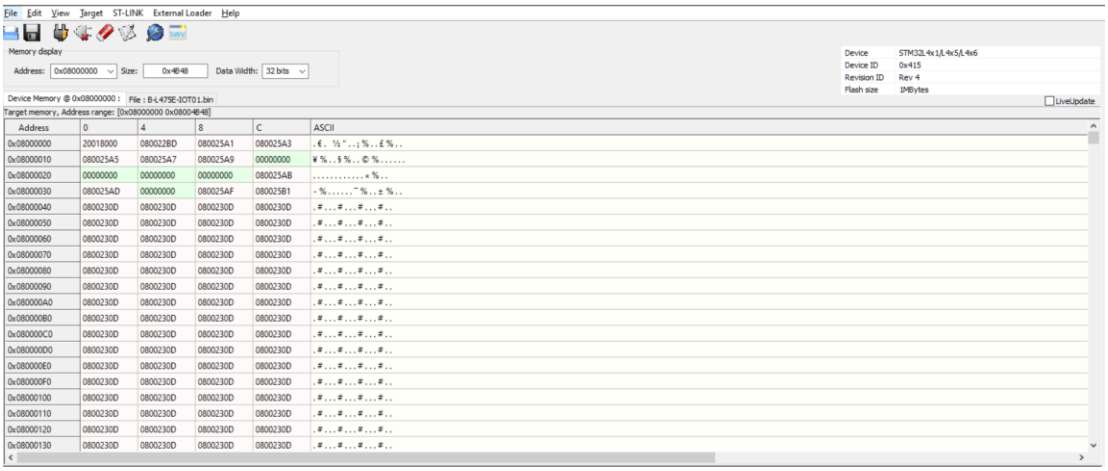


Figure 15: Verify Bootloader Code in Flash

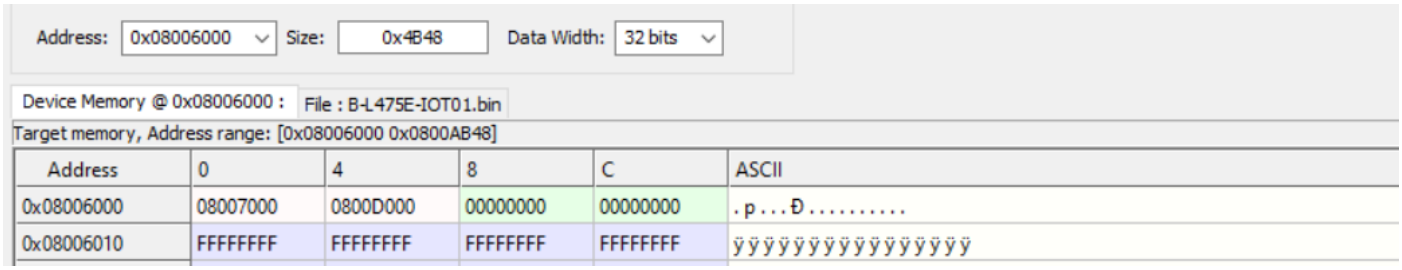


Figure 16: Verify Metadata for Img-0 (Img_0 is set to be invalid for Demo Purpose)*

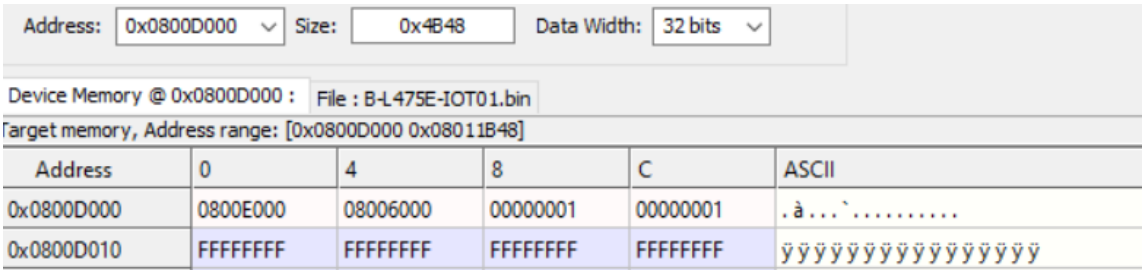


Figure 17: Verify Meta-data for Img-1 (Img_1 is set to be valid and updated for Demo Purpose)*

```
//Every image will have a meta-data denoting below information.
typedef struct meta_data{

    //this is the address of the Image in FLASH
    uint32_t img_strt_add;

    //this is pointer to metadata of next image
    struct meta_data *next;

    //This field indicates if the image pointed by this metadata is valid or not
    uint32_t isValid;

    //This field indicates if the image is updated or not
    uint32_t isUpdated;

}meta_data_t;
```

Figure 18: Meta-Data Structure in Bootloader Code

Step 4: Verify the Working of Bootloader

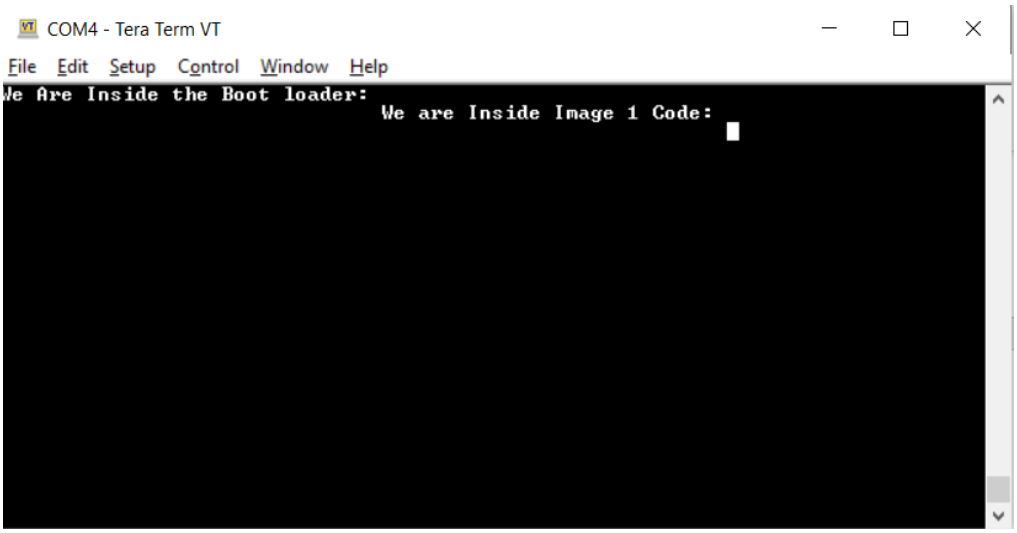


Figure 19: Boot loader Executed Img_1 , since Img_0 was set to be invalid.