4/26/2023

Lab2

Embedded C Lesson 3

Mostafa Mohamed Edrees
LEARN-IN-DEPTH

Lab2

Required:

WRITE BAREMETAL SW ON ARM CORTEX-M3 32-BIT MICROCONTROLLER STM32F103C8T6 CHIP TO TOGGLE LED.

Physical Board:

STM32F103C6

Processor:

Arm CORTEX-M3

Name:

Mostafa Mohamed Edrees

Supervisor:

Eng. Keroles Shenouda

Table of Contents

Introduction

Hello everyone, we will write a bare metal SW on STM32F103C6 Board and ARM CORTEX-M3 Processor to toggle led.

We will write this lab from scratch so we will write app.c, app.h, startup, linker_script.ld and Makefile and we will run this lab on Proteus and debug it on Proteus too.

ARM CORTEX-M3

We should know important information about this processor at first.

This processor when it is starts to work the PC (program counter) will point to the entry point and the entry point should contain the address of SP (stack pointer) so this processor initialize SP by itself without we write assembly code to do that so we can write startup code by C and put this code at the next address after the address that we put SP in it.

So we will do this lab with two versions:

1st Version: do this lab with **startup.s**

2nd Version: do this lab with **startup.c**

Steps

Steps to do this lab:

- Write application file.
- Write startup.c or startup.s.
- Write linker_script.
- Write makefile.
- Build the project by using makefile.
- Run the project on proteus.
- Debug the project on proteus.

Notes:

In the two versions we do the same thing so:

The application file will become common between the two versions.

So let's start to write application file.

Application File

We want to write application to connect led to **GPIO_PortA pin13** in **STM32F103C6** to toggle this led.

To Toggle Led we need to work with two peripherals:

- RCC (reset and clock control).
- GPIO (general purpose input/output).

We should open datasheet and get the base address of each peripheral:

- Base address of RCC is **0x40021000**
- Base address of GPIO PortA is **0x40010800**

RCC:

It has a bit2 (IOPAEN) in APB2ENR Register used to enable PortA.

Offset of APB2ENR Register is **0x18**

Bit Number of IOPAEN is **2**

GPIO_PortA:

It has two regiseters:

- CRH Register
  We should write 2 “0010” on it from bit20 to bit24.
  Offset of CRH Register is **0x04**
- ODR Register
  It has pin13 that will connect led with it so we can turn on/off the led by sending 1/0 to this pin.
  Offset of ODR Register is **0x0C**

We will put a delay between turn on and turn off the led to notice the toggle.

We do the delay by using for loop.

Delay

```
for(i = 0; i < 5000; i++);                    //delay
```

Write app.h file

In this file we will put the definition of base addresses and registers.

app.h

```
/**
 ******************************************************************************
 * @file           : app.h
 * @author         : Mostafa Edrees
 * @brief          : lab2 in lesson3 in Embedded C
 * @date           : 26/4/2023
 * @board          : STM32F103C8T6
 * @processor      : ARM Cortex M3
 ******************************************************************************
**/

#ifndef _APP_H_
#define _APP_H_

//includes
#include "Platform_types.h"

//Base Addresses
#define RCC_Base            0x40021000
#define GPIO_PA_Base        0x40010800

//APB2ENR Register and IOPAEN pin
#define RCC_APB2ENR         *((vusint32_t *)(RCC_Base + 0x18))
#define RCC_APB2ENR_IOPAEN  2

//CRH Register
#define GPIO_PA_CRH         *((vusint32_t *)(GPIO_PA_Base + 0x04))

//ODR Register
typedef union
{
    vusint32_t all_fileds;
    struct
    {
        vusint32_t reserved:13;
        vusint32_t pin_13:1;
    }Pins;
}R_GPIO_PA_ODR_t;

//pointer to ODR Register
volatile R_GPIO_PA_ODR_t *R_ODR = (volatile R_GPIO_PA_ODR_t *)(GPIO_PA_Base + 0x0C);


//Macro to SET & CLEAR bit
#define SET_BIT(Register,BIT_NO)    (Register |= (1 << BIT_NO))
#define CLR_BIT(Register,BIT_NO)    (Register &= (~(1 << BIT_NO)))

#endif
```

Write app.c file

In this file we will implement the application to toggle the led.

app.c

```c
/**
 ********************************************************************************
 * @file           : app.c
 * @author         : Mostafa Edrees
 * @brief          : lab2 in lesson3 in Embedded C
 * @date           : 26/4/2023
 * @board          : STM32F103C8T6
 * @processor      : ARM Cortex M3
 ********************************************************************************
**/

//includes
#include "Platform_Types.h"
#include "app.h"

//to create .data & .rodata & .bss sections on memmory
usint8_t g_variables[] = "Mostafa";                 // 8Bytes
usint8_t const const_variables[] = " Edrees";       // 8Bytes
usint32_t uninitalized_variables;                   // 4Bytes

//main function
int main(void)
{
    //set IOPAEN in APB2ENR Register
    SET_BIT(RCC_APB2ENR,RCC_APB2ENR_IOPAEN);

    //SET "Mode pin13 = 2" from bit20 to bit24 on CRH Register on GPIO_PortA
    GPIO_PA_CRH &= 0xFF0FFFFF;      //zero bits from bit20 to bit24
    GPIO_PA_CRH |= 0x00200000;      //put the value of bits from bit20 to bit24 = 2

    //toggle led
    while(1)
    {
        R_ODR->Pins.pin_13 = 1;             //turn on the led
        for(int i = 0; i < 5000; i++);      //delay
        R_ODR->Pins.pin_13 = 0;             //turn off the led
        for(int i = 0; i < 5000; i++);      //delay
    }

    return 0;
}
```

Lab2 Version1

We will write Makefile & startup.s & linker_script.ld files for this version then build the project by using make file then run it on proteus.

Makefile

In this file we write makefile to auto build the lab.

Makefile

```
#*******************************************************************************
#* @file           : Makefile
#* @author         : Mostafa Edrees
#* @brief          : lab2 in lesson3 in Embedded C
#* @date           : 26/4/2023
#* @board          : STM32F103C8T6
#* @processor      : ARM Cortex M3
#* @level_debug    : -gdwarf-2 to debug on proteus
#*******************************************************************************

CC = arm-none-eabi-
CFLAGS = -gdwarf-2 -mcpu=cortex-m3 -mthumb
INCS = -I .
LIBS =
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o)
As = $(wildcard *.s)
AsOBj = $(As:.s=.o)

Project_Name = learn-in-depth-Cortex-M3
Copyrights = Mostafa Edrees
Board = STM32F103C8T6
Arm_Processor = cortex-m3
date = 26/4/2023

all: $(Project_Name).bin
    @echo -e "\n*****************************************"
    @echo -e "\tBuild is Done"
    @echo -e "Project Name:" $(Project_Name)
    @echo -e "Copyrights:" $(Copyrights)
    @echo -e "date:" $(date)
    @echo -e "Board:" $(Board)
    @echo -e "Arm Processor:" $(Arm_Processor)
    @echo -e "*****************************************\n"

startup.o: startup.s
    $(CC)as.exe $(CFLAGS) $< -o $@

%.o: %.c
    $(CC)gcc.exe -c $(INCS) $(CFLAGS) $< -o $@

$(Project_Name).elf: $(OBJ) $(AsOBj)
    $(CC)ld.exe -T linker_script.ld $(LIBS) $(OBJ) $(AsOBj) -Map=Map_file.map -o $@

$(Project_Name).bin: $(Project_Name).elf
    $(CC)objcopy.exe -O binary $< $@


clean_all:
    rm *.o *.elf *.bin

clean:
    rm *.elf *.bin
```

Startup.s

In this file we write startup code with assembly language.

startup.s

```
/**
 ******************************************************************************
 * @file           : startup.s
 * @author         : Mostafa Edrees
 * @brief          : lab2 in lesson3 in Embedded C
 * @date           : 26/4/2023
 * @board          : STM32F103C8T6
 * @processor      : ARM Cortex M3
 ******************************************************************************
**/

/* Entry point of SRAM is 0x20000000 */

.section .vectors                                   /* .vectors section                */

.word       0x20001000                              /* address of sp (stack pointer)   */
.word       _Reset                                  /* reset section                   */
.word       Vector_handler                          /* 2  NMI (non maskable interrupt) */
.word       Vector_handler                          /* 3  Hard Fault                   */
.word       Vector_handler                          /* 4  MM Fault (Memory Management) */
.word       Vector_handler                          /* 5  Bus Fault                    */
.word       Vector_handler                          /* 6  Usage Fault                  */
.word       Vector_handler                          /* 7  RESERVED                     */
.word       Vector_handler                          /* 8  RESERVED                     */
.word       Vector_handler                          /* 9  RESERVED                     */
.word       Vector_handler                          /* 10 RESERVED                     */
.word       Vector_handler                          /* 11 SV call                      */
.word       Vector_handler                          /* 12 Debug reserved               */
.word       Vector_handler                          /* 13 RESERVED                     */
.word       Vector_handler                          /* 14 PendSV                       */
.word       Vector_handler                          /* 15 SysTick                      */
.word       Vector_handler                          /* 16 IRQ0                         */
.word       Vector_handler                          /* 17 IRQ1                         */
.word       Vector_handler                          /* 18 IRQ2                         */
.word       Vector_handler                          /* 19 ...                          */
                    /* On to IRQ67 */


.section .text                                      /* .text section                   */

_Reset:                                             /* reset section                    */
    bl main
    b .

.thumb_func                                         /* enable 16 bit instructions       */

Vector_Handler:                                     /* vector_handler section           */
    b _Reset
```

Linker_script.ld

In this file we write linker_script but in this file we will but .data section in FLASH and don't copy it to RAM and we don't initialize .bss section in RAM with zero we will do this in Lab2_Version2.

linker_script.ld

```
/**
 ******************************************************************************
 * @file           : linker_script.ld
 * @author         : Mostafa Edrees
 * @brief          : lab2 in lesson3 in Embedded C
 * @date           : 26/4/2023
 * @board          : STM32F103C8T6
 * @processor      : ARM Cortex M3
 ******************************************************************************
**/

MEMORY
{
    FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 128k
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 20k
}


SECTIONS
{
    .text :
    {
        *(.vectors*)
        *(.text*)
    }> FLASH

    .data :
    {
        *(.data)
    }> FLASH

    .rodata :
    {
        *(.rodata)
    }> FLASH

    .bss :
    {
        *(.bss)
    }> SRAM
}
```

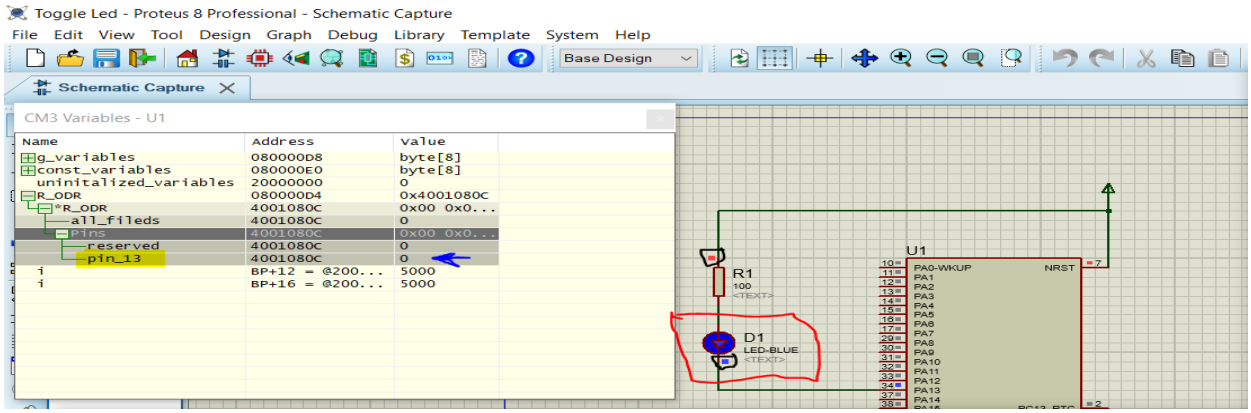
Build the project

We will build the project by using Makefile.

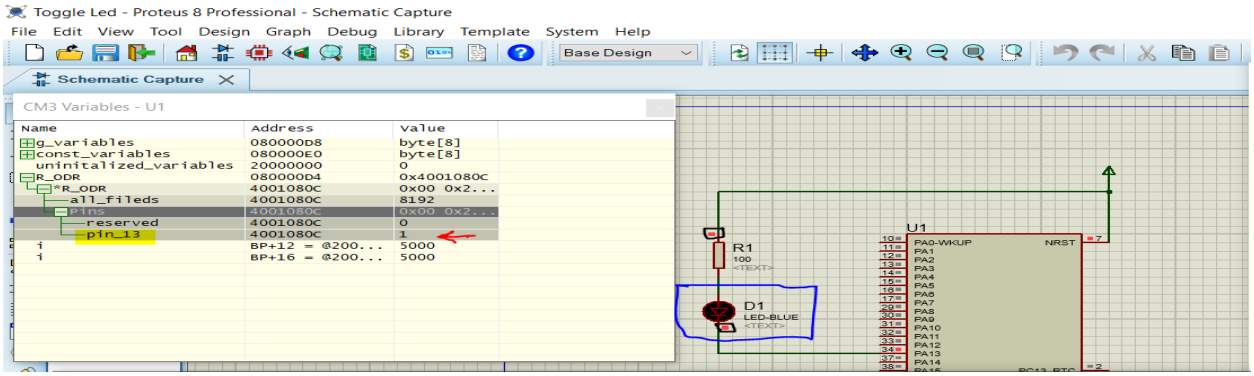
```
lenovo@MostafaEdrees MINGW32 /d/Mastering Embedded System/GitHub_Repo/Mastering_Embedded_System_Online_Diploma/Embedded C/Lesson 3/Lab2/Lab2_Version1 (master)
$ mingw32-make.exe all
arm-none-eabi-gcc.exe -c -I . -gdwarf-2 -mcpu=cortex-m3 -mthumb app.c -o app.o
arm-none-eabi-as.exe -gdwarf-2 -mcpu=cortex-m3 -mthumb startup.s -o startup.o
arm-none-eabi-ld.exe -T linker_script.ld  app.o startup.o -Map=Map_file.map -o learn-in-depth-Cortex-M3.elf
arm-none-eabi-objcopy.exe -O binary learn-in-depth-Cortex-M3.elf learn-in-depth-Cortex-M3.bin

****************************************
       Build is Done
Project Name: learn-in-depth-Cortex-M3
Copyrights: Mostafa Edrees
date: 26/4/2023
Board: STM32F103C8T6
Arm Processor: cortex-m3
****************************************
```

Run the project on Proteus

led is turn on:



Led is turn off:



You can see the video of running project from this link:

Video

Lab2 Version2

We will modify something in Makefile and write startup.c & linker_script.ld files for this version then build the project by using make file then run it on proteus and debug it on proteus.

Makefile

We will delete this two lines from Makefile because we will write startup code with C.

```
startup.o: startup.s
	$(CC)as.exe $(CFLAGS) $< -o $@
```

Makefile

```
#*****************************************************************************
#* @file           : Makefile
#* @author         : Mostafa Edrees
#* @brief          : lab2 in lesson3 in Embedded C
#* @date           : 26/4/2023
#* @board          : STM32F103C8T6
#* @processor      : ARM Cortex M3
#* @level_debug    : -gdwarf-2 to debug on proteus
#*****************************************************************************

CC = arm-none-eabi-
CFLAGS = -gdwarf-2 -mcpu=cortex-m3 -mthumb
INCS = -I .
LIBS =
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o)
As = $(wildcard *.s)
AsOBj = $(As:.s=.o)

Project_Name = learn-in-depth-Cortex-M3
Copyrights = Mostafa Edrees
Board = STM32F103C8T6
Arm_Processor = cortex-m3
date = 26/4/2023

all: $(Project_Name).bin
    @echo -e "\n*****************************************"
    @echo -e "\tBuild is Done"
    @echo -e "Project Name:" $(Project_Name)
    @echo -e "Copyrights:" $(Copyrights)
    @echo -e "date:" $(date)
    @echo -e "Board:" $(Board)
    @echo -e "Arm Processor:" $(Arm_Processor)
    @echo -e "*****************************************\n"


%.o: %.c
    $(CC)gcc.exe -c $(INCS) $(CFLAGS) $< -o $@

$(Project_Name).elf: $(OBJ) $(AsOBj)
    $(CC)ld.exe -T linker_script.ld $(LIBS) $(OBJ) $(AsOBj) -Map=Map_file.map -o $@

$(Project_Name).bin: $(Project_Name).elf
    $(CC)objcopy.exe -O binary $< $@


clean_all:
    rm *.o *.elf *.bin

clean:
    rm *.elf *.bin

```

startup.c

```c
/**
 ******************************************************************************
 * @file           : startup.c
 * @author         : Mostafa Edrees
 * @brief          : lab2 in lesson3 in Embedded C
 * @date           : 26/4/2023
 * @board          : STM32F103C8T6
 * @processor      : ARM Cortex M3
 ******************************************************************************
**/
//includes
#include "Platform_Types.h"

extern int main(void);
extern usint32_t _stack_top;

//Vector Handler
void Reset_Handler(void);
void NMI_Handler(void) __attribute__((weak,alias("Default_Handler")));
void Hard_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
void MM_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
void Bus_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
void Usage_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));


vusint32_t vectors[] __attribute__((section(".vectors"))) =
{
    (vusint32_t) &_stack_top,
    (vusint32_t) &Reset_Handler,
    (vusint32_t) &NMI_Handler,
    (usint32_t)  &Hard_Fault_Handler,
    (usint32_t)  &MM_Fault_Handler,
    (usint32_t)  &Bus_Fault_Handler,
    (usint32_t)  &Usage_Fault_Handler
};

extern usint32_t _E_text;
extern usint32_t _S_data;
extern usint32_t _E_data;
extern usint32_t _S_rodata;
extern usint32_t _E_rodata;
extern usint32_t _S_bss;
extern usint32_t _E_bss;


void Reset_Handler (void)
{
    int i;

    //copy .data section from flash to ram
    usint32_t DATA_size = (usint8_t *)&_E_data - (usint8_t *)&_S_data ;
    usint8_t *P_src = (usint8_t *)&_E_text ;
    usint8_t *P_dst = (usint8_t *)&_S_data ;
    for(i = 0; i < DATA_size; i++)
    {
        *((usint8_t *)P_dst++) = *((usint8_t *)P_src++);
    }

    //locate .bss section in ram and initalize it with zero
    usint32_t BSS_size = (usint8_t *)&_E_bss - (usint8_t *)&_S_bss ;
    P_dst = (usint8_t *)&_S_bss ;
    for(i = 0; i < BSS_size; i++)
    {
        *((usint8_t *)P_dst++) = (usint8_t)0;
    }

    //jump to main()
    main();
}

void Default_Handler (void)
{
    Reset_Handler();
}
```

In startup.c:

- We put the address of SP (stack pointer) at the entry point of flash memory.
- We copy .data section from FLASH to RAM.
- We locate .bss section in RAM and initialize it with zero.

Linker_script.ld

In this file we put symbols to define start and end of each section to use it to copy .data section from FLASH to RAM and locate .bss section in RAM.

```
12  MEMORY
13  {
14      FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 128k
15      SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 20k
16  }
17
18  SECTIONS
19  {
20      .text :
21      {
22          *(.vectors*)
23          *(.text)
24          . = ALIGN(4) ;
25          _E_text = . ;
26      }> FLASH
27
28      .data :
29      {
30          . = ALIGN(4) ;
31          _S_data = . ;
32          *(.data*)
33          . = ALIGN(4) ;
34          _E_data = . ;
35      }> SRAM AT> FLASH
36
37      .rodata :
38      {
39          . = ALIGN(4) ;
40          _S_rodata = . ;
41          *(.rodata*)
42          . = ALIGN(4) ;
43          _E_rodata = . ;
44      }> FLASH
45
46      .bss :
47      {
48          . = ALIGN(4) ;
49          _S_bss = . ;
50          *(.bss*)
51          . = ALIGN(4) ;
52          *(COMMON*)
53          . = ALIGN(4) ;
54          _E_bss = . ;
55      }> SRAM
56
57      . = ALIGN(4) ;
58      . = . + 0x1000 ;
59      _stack_top = .;
60  }
```

Build the project

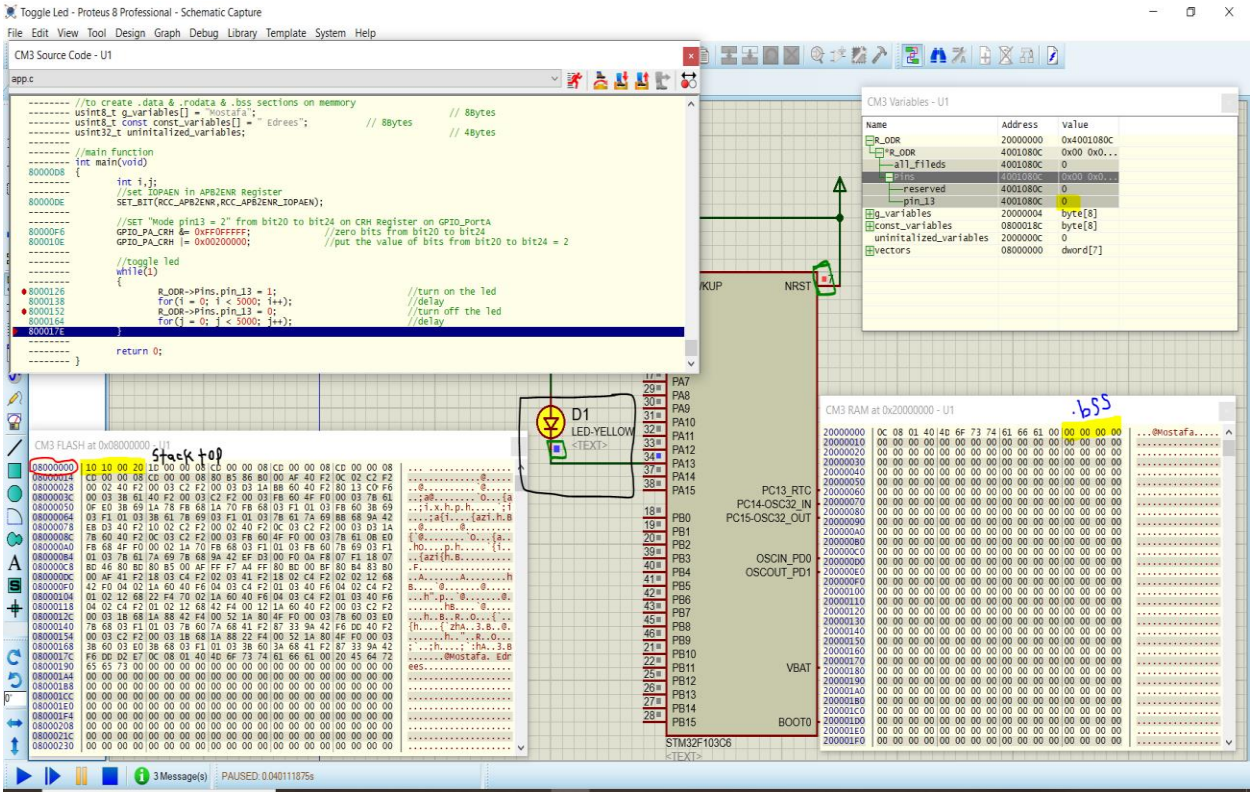
We will build the project by using Makefile.

```
lenovo@MostafaEdrees MINGW32 /d/Mastering Embedded System/GitHub_Repo/Mastering_
Embedded_System_Online_Diploma/Embedded C/Lesson 3/Lab2/Lab2 Version2 (master)
$ mingw32-make.exe all
arm-none-eabi-gcc.exe -c -I . -gdwarf-2 -mcpu=cortex-m3 -mthumb startup.c -o startup.o
arm-none-eabi-gcc.exe -c -I . -gdwarf-2 -mcpu=cortex-m3 -mthumb app.c -o app.o
arm-none-eabi-ld.exe -T linker_script.ld  startup.o app.o  -Map=Map_file.map -o learn-in-depth-Cortex-M3.elf
arm-none-eabi-objcopy.exe -O binary learn-in-depth-Cortex-M3.elf learn-in-depth-Cortex-M3.bin

****************************************
        Build is Done
Project Name: learn-in-depth-Cortex-M3
Copyrights: Mostafa Edrees
date: 26/4/2023
Board: STM32F103C8T6
Arm Processor: cortex-m3
****************************************
```

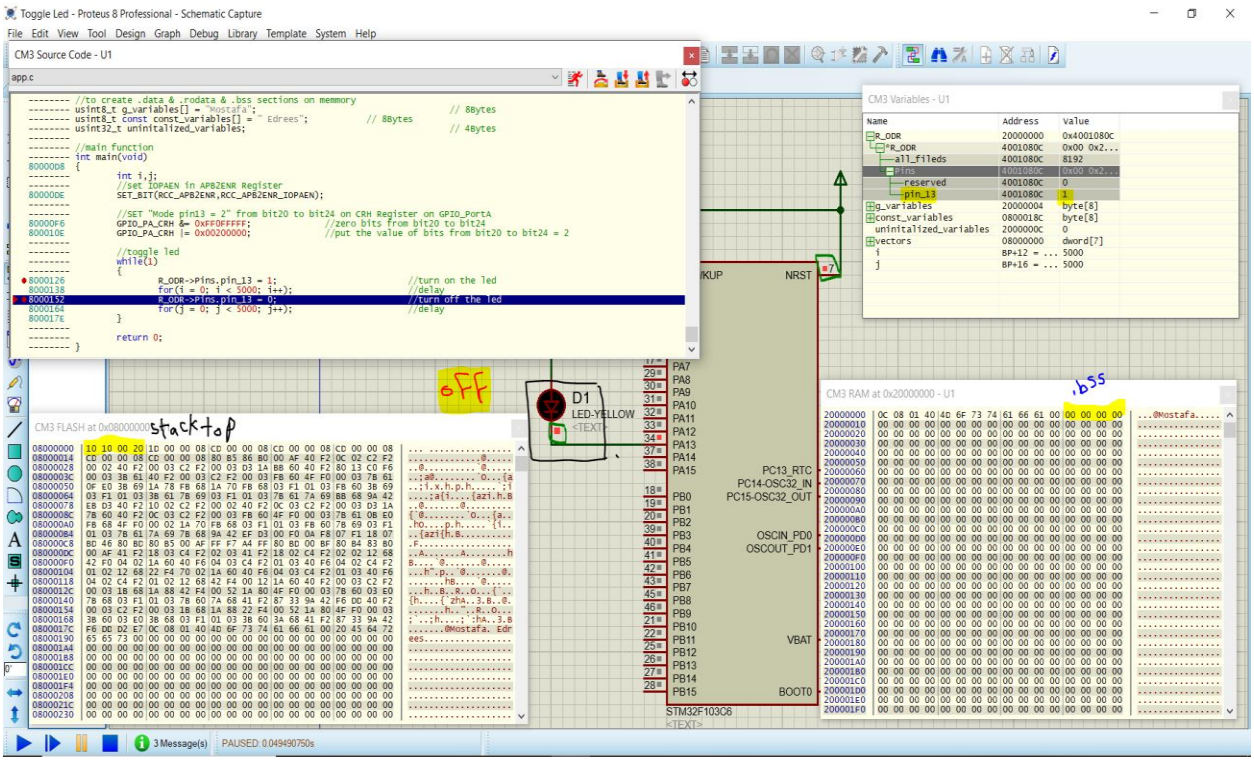
After that we will run & debug lab on Proteus.

Run the project

Turn on the led



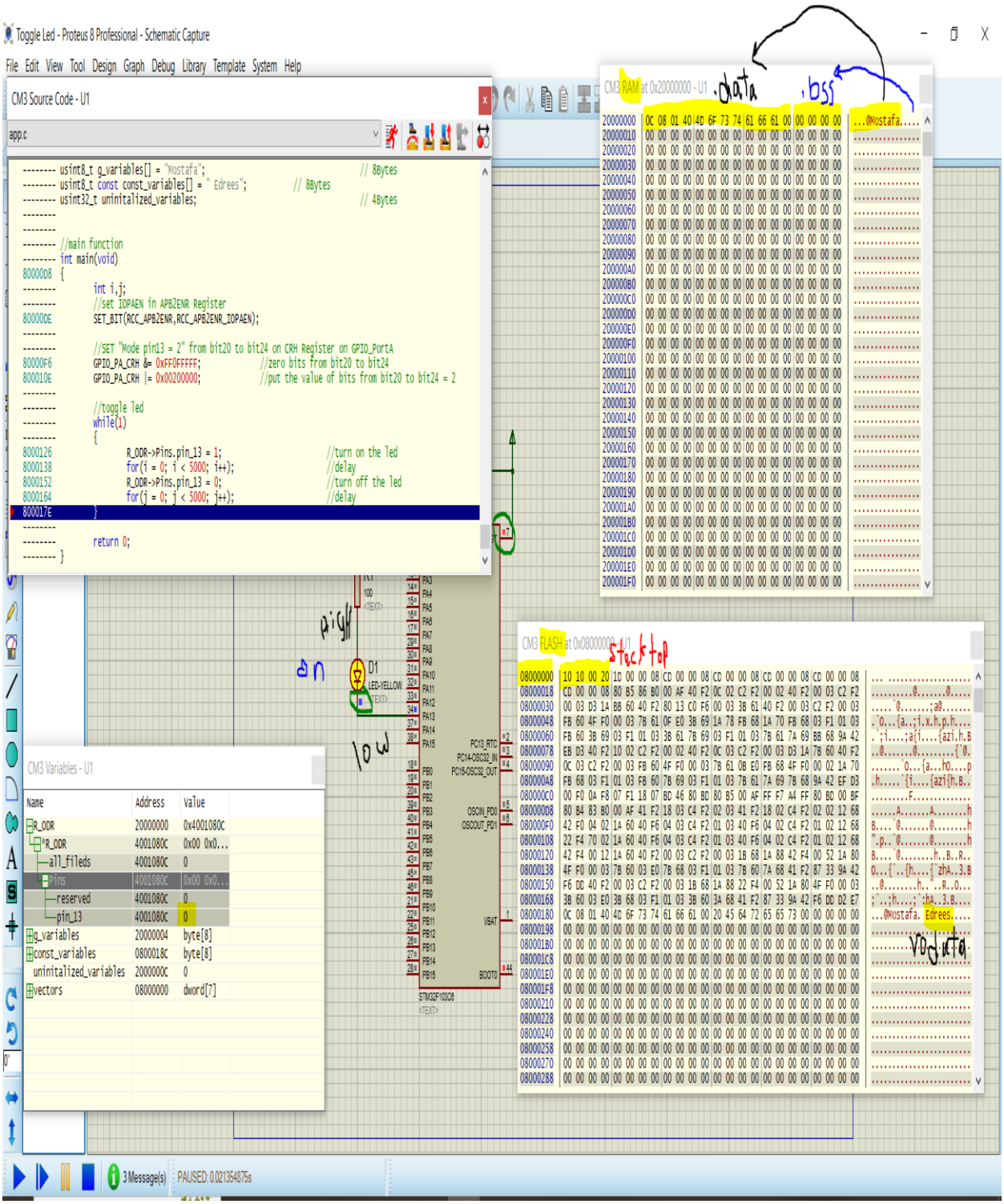
Turn off the led



Debug the project

You can see the debug of the lab on proteus in this video:

Video



Analyze Mapfile

Memory

```
Memory Configuration

Name             Origin             Length             Attributes
FLASH            0x08000000         0x00020000         xr
SRAM             0x20000000         0x00005000         xrw
*default*        0x00000000         0xffffffff
```

.text section

```
.text           0x08000000      0x180
 *(.vectors*)
 .vectors       0x08000000      0x1c startup.o
                0x08000000                vectors
 *(.text)
 .text          0x0800001c      0xbc startup.o
                0x0800001c                Reset_Handler
                0x080000cc                MM_Fault_Handler
                0x080000cc                Bus_Fault_Handler
                0x080000cc                Default_Handler
                0x080000cc                Usage_Fault_Handler
                0x080000cc                Hard_Fault_Handler
                0x080000cc                NMI_Handler
 .text          0x080000d8      0xa8 app.o
                0x080000d8                main
                0x08000180                . = ALIGN (0x4)
                0x08000180                _E_text = .
```

.data section

```
.data           0x20000000      0xc load address 0x08000180
                0x20000000                . = ALIGN (0x4)
                0x20000000                _S_data = .
 *(.data*)
 .data          0x20000000      0x0 startup.o
 .data          0x20000000      0xc app.o
                0x20000000                R_ODR
                0x20000004                g_variables
                0x2000000c                . = ALIGN (0x4)
                0x2000000c                _E_data = .
```

.rodata section

Flash size

```
.rodata         0x0800018c      0x8
                0x0800018c              . = ALIGN (0x4)
                0x0800018c              _S_rodata = .
 *(.rodata*)
 .rodata        0x0800018c      0x8 app.o
                0x0800018c              const_variables
                0x08000194              . = ALIGN (0x4)
                0x08000194              _E_rodata = .
```

.bss section

RAM size

```
.bss            0x2000000c      0x4
                0x2000000c              . = ALIGN (0x4)
                0x2000000c              _S_bss = .
 *(.bss*)
 .bss           0x2000000c      0x0 startup.o
 .bss           0x2000000c      0x0 app.o
                0x2000000c              . = ALIGN (0x4)
 *(COMMON*)
 COMMON         0x2000000c      0x4 app.o
                0x2000000c              uninitalized_variables
                0x20000010              . = ALIGN (0x4)
                0x20000010              _E_bss = .
                0x20000010              . = ALIGN (0x4)
                0x20001010              . = (. + 0x1000)
                0x20001010              _stack_top = .
```

SP

Symbols table of object files

Startup.o

```
$ arm-none-eabi-nm.exe startup.o
         U _E_bss
         U _E_data
         U _E_text
         U _S_bss
         U _S_data
         U _stack_top
000000b0 W Bus_Fault_Handler
000000b0 T Default_Handler
000000b0 W Hard_Fault_Handler
         U main
000000b0 W MM_Fault_Handler
000000b0 W NMI_Handler
00000000 T Reset_Handler
000000b0 W Usage_Fault_Handler
00000000 D vectors
```

app.o

```
$ arm-none-eabi-nm.exe app.o
00000000 R const_variables
00000004 D g_variables
00000000 T main
00000000 D R_ODR
00000004 C uninitalized_variables
```

learn-in-depth-Cortex-M3.elf

```
$ arm-none-eabi-nm.exe learn-in-depth-Cortex-M3.elf
20000010 B _E_bss
2000000c D _E_data
08000194 R _E_rodata
08000180 T _E_text
2000000c B _S_bss
20000000 D _S_data
0800018c R _S_rodata
20001010 B _stack_top
080000cc W Bus_Fault_Handler
0800018c R const_variables
080000cc T Default_Handler
20000004 D g_variables
080000cc W Hard_Fault_Handler
080000d8 T main
080000cc W MM_Fault_Handler
080000cc W NMI_Handler
20000000 D R_ODR
0800001c T Reset_Handler
2000000c B uninitalized_variables
080000cc W Usage_Fault_Handler
08000000 T vectors
```

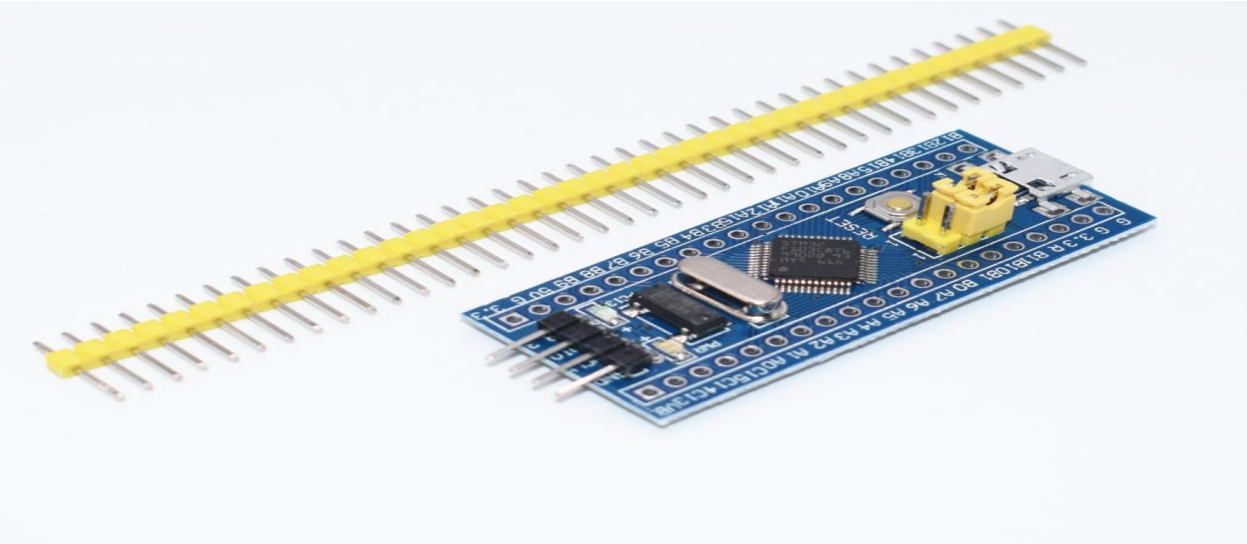
Conclusion

After doing this lab we know important note:

“We can write startup code with C not only assembly.”

Explanation:

In some processor like CORTEX-M3 we should put address of SP (stack pointer) at the entry point of the program so the processor define SP without we write assembly code to do that so after this operation the CPU will understand C so we can write startup with C and put it after the address containing the address of SP if the processor don’t have this feature we will write startup code with assembly and we will define SP by using assembly before we jump to main function.



Good Luck