

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

4/26/2023

Lab2

Embedded C Lesson 3

Several thin, curved lines in dark blue and light grey originate from the bottom left corner and sweep upwards and to the right.

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.....	3
ARM CORTEX-M3.....	3
Steps	3
Application File.....	4
Write app.h file	5
Write app.c file.....	6
Lab2 Version1	7
Makefile.....	7
Startup.s	8
Linker_script.ld	9
Build the project.....	10
Run the project on Proteus	10
Lab2 Version2	11
Makefile.....	11
startup.c.....	12
Linker_script.ld	13
Build the project.....	14
Run the project.....	15
Debug the project.....	16
Analyze Mapfile.....	17
Symbols table of object files	19
Conclusion	20

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 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 registers:

- 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

```
1  /**
2  ****
3  * @file      : app.h
4  * @author    : Mostafa Edrees
5  * @brief     : lab2 in lesson3 in Embedded C
6  * @date      : 26/4/2023
7  * @board     : STM32F103C8T6
8  * @processor  : ARM Cortex M3
9  ****
10 /**/
11
12 #ifndef _APP_H_
13 #define _APP_H_
14
15 //includes
16 #include "Platform_types.h"
17
18 //Base Addresses
19 #define RCC_Base      0x40021000
20 #define GPIO_PA_Base  0x40010800
21
22 //APB2ENR Register and IOPAEN pin
23 #define RCC_APB2ENR    *((vusint32_t *) (RCC_Base + 0x18))
24 #define RCC_APB2ENR_IOPAEN 2
25
26 //CRH Register
27 #define GPIO_PA_CRH    *((vusint32_t *) (GPIO_PA_Base + 0x04))
28
29 //ODR Register
30 typedef union
31 {
32     vusint32_t all_fields;
33     struct
34     {
35         vusint32_t reserved:13;
36         vusint32_t pin_13:1;
37     } Pins;
38 } R_GPIO_PA_ODR_t;
39
40 //pointer to ODR Register
41 volatile R_GPIO_PA_ODR_t *R_ODR = (volatile R_GPIO_PA_ODR_t *) (GPIO_PA_Base + 0x0C);
42
43
44 //Macro to SET & CLEAR bit
45 #define SET_BIT(Register,BIT_NO)    (Register |= (1 << BIT_NO))
46 #define CLR_BIT(Register,BIT_NO)    (Register &= (~(1 << BIT_NO)))
47
48 #endif
```

Write app.c file

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

app.c

```
1  /**
2  ****
3  * @file      : app.c
4  * @author    : Mostafa Edrees
5  * @brief     : lab2 in lesson3 in Embedded C
6  * @date      : 26/4/2023
7  * @board     : STM32F103C8T6
8  * @processor  : ARM Cortex M3
9  ****
10 /**/
11
12 //includes
13 #include "Platform_Types.h"
14 #include "app.h"
15
16 //to create .data & .rodata & .bss sections on memory
17 uint8_t g_variables[] = "Mostafa";           // 8Bytes
18 uint8_t const const_variables[] = " Edrees"; // 8Bytes
19 uint32_t uninitialized_variables;           // 4Bytes
20
21 //main function
22 int main(void)
23 {
24     //set IOPAEN in APB2ENR Register
25     SET_BIT(RCC_APB2ENR, RCC_APB2ENR_IOPAEN);
26
27     //SET "Mode pin13 = 2" from bit20 to bit24 on CRH Register on GPIO_PortA
28     GPIO_PA_CRH &= 0xFF0FFFFF; //zero bits from bit20 to bit24
29     GPIO_PA_CRH |= 0x00200000; //put the value of bits from bit20 to bit24 = 2
30
31     //toggle led
32     while(1)
33     {
34         R_ODR->Pins.pin_13 = 1; //turn on the led
35         for(int i = 0; i < 5000; i++); //delay
36         R_ODR->Pins.pin_13 = 0; //turn off the led
37         for(int i = 0; i < 5000; i++); //delay
38     }
39
40     return 0;
41 }
```

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

```
1  #*****
2  #* @file      : Makefile
3  #* @author    : Mostafa Edrees
4  #* @brief     : lab2 in lesson3 in Embedded C
5  #* @date      : 26/4/2023
6  #* @board     : STM32F103C8T6
7  #* @processor  : ARM Cortex M3
8  #* @level_debug : -gdwarf-2 to debug on proteus
9  #*****
10
11 CC = arm-none-eabi-
12 CFLAGS = -gdwarf-2 -mcpu=cortex-m3 -mthumb
13 INCS = -I .
14 LIBS =
15 SRC = $(wildcard *.c)
16 OBJ = $(SRC:.c=.o)
17 As = $(wildcard *.s)
18 AsObj = $(As:.s=.o)
19
20 Project_Name = learn-in-depth-Cortex-M3
21 Copyrights = Mostafa Edrees
22 Board = STM32F103C8T6
23 Arm_Processor = cortex-m3
24 date = 26/4/2023
25
26 all: $(Project_Name).bin
27     @echo -e "\n*****"
28     @echo -e "\tBuild is Done"
29     @echo -e "Project Name:" $(Project_Name)
30     @echo -e "Copyrights:" $(Copyrights)
31     @echo -e "date:" $(date)
32     @echo -e "Board:" $(Board)
33     @echo -e "Arm Processor:" $(Arm_Processor)
34     @echo -e "*****\n"
35
36 startup.o: startup.s
37     $(CC)as.exe $(CFLAGS) < -o $@
38
39 %.o: %.c
40     $(CC)gcc.exe -c $(INCS) $(CFLAGS) < -o $@
41
42 $(Project_Name).elf: $(OBJ) $(AsObj)
43     $(CC)ld.exe -T linker_script.ld $(LIBS) $(OBJ) $(AsObj) -Map=Map_file.map -o $@
44
45 $(Project_Name).bin: $(Project_Name).elf
46     $(CC)objcopy.exe -O binary < $@
47
48
49 clean_all:
50     rm *.o *.elf *.bin
51
52 clean:
53     rm *.elf *.bin
```


Startup.s

In this file we write startup code with assembly language.

startup.s

```
1  /**
2  ****
3  * @file      : startup.s
4  * @author    : Mostafa Edrees
5  * @brief     : lab2 in lesson3 in Embedded C
6  * @date      : 26/4/2023
7  * @board     : STM32F103C8T6
8  * @processor  : ARM Cortex M3
9  ****
10 /**/
11
12 /* Entry point of SRAM is 0x20000000 */
13
14 .section .vectors                /* .vectors section */
15
16 .word      0x20001000            /* address of sp (stack pointer) */
17 .word      _Reset               /* reset section */
18 .word      Vector_handler       /* 2 NMI (non maskable interrupt) */
19 .word      Vector_handler       /* 3 Hard Fault */
20 .word      Vector_handler       /* 4 MM Fault (Memory Management) */
21 .word      Vector_handler       /* 5 Bus Fault */
22 .word      Vector_handler       /* 6 Usage Fault */
23 .word      Vector_handler       /* 7 RESERVED */
24 .word      Vector_handler       /* 8 RESERVED */
25 .word      Vector_handler       /* 9 RESERVED */
26 .word      Vector_handler       /* 10 RESERVED */
27 .word      Vector_handler       /* 11 SV call */
28 .word      Vector_handler       /* 12 Debug reserved */
29 .word      Vector_handler       /* 13 RESERVED */
30 .word      Vector_handler       /* 14 PendSV */
31 .word      Vector_handler       /* 15 SysTick */
32 .word      Vector_handler       /* 16 IRQ0 */
33 .word      Vector_handler       /* 17 IRQ1 */
34 .word      Vector_handler       /* 18 IRQ2 */
35 .word      Vector_handler       /* 19 ... */
36      | | | | /* On to IRQ67 */
37
38
39 .section .text                  /* .text section */
40
41 _Reset:                        /* reset section */
42     bl main
43     b .
44
45 .thumb_func                    /* enable 16 bit instructions */
46
47 Vector_Handler:                /* vector_handler section */
48     b _Reset
```

Linker_script.ld

In this file we write linker_script but in this file we will put .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

```
1  /**
2  ****
3  * @file      : linker_script.ld
4  * @author    : Mostafa Edrees
5  * @brief     : lab2 in lesson3 in Embedded C
6  * @date      : 26/4/2023
7  * @board     : STM32F103C8T6
8  * @processor  : ARM Cortex M3
9  ****
10 **/
11
12 MEMORY
13 {
14     FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 128k
15     SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 20k
16 }
17
18
19 SECTIONS
20 {
21     .text :
22     {
23         *(.vectors*)
24         *(.text*)
25     }> FLASH
26
27     .data :
28     {
29         *(.data)
30     }> FLASH
31
32     .rodata :
33     {
34         *(.rodata)
35     }> FLASH
36
37     .bss :
38     {
39         *(.bss)
40     }> SRAM
41 }
```

Build the project

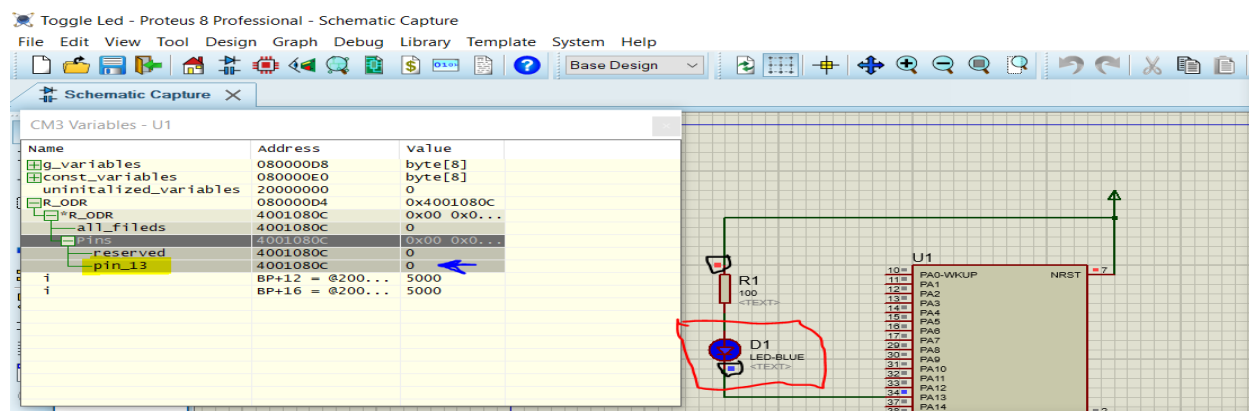
We will build the project by using Makefile.

```
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

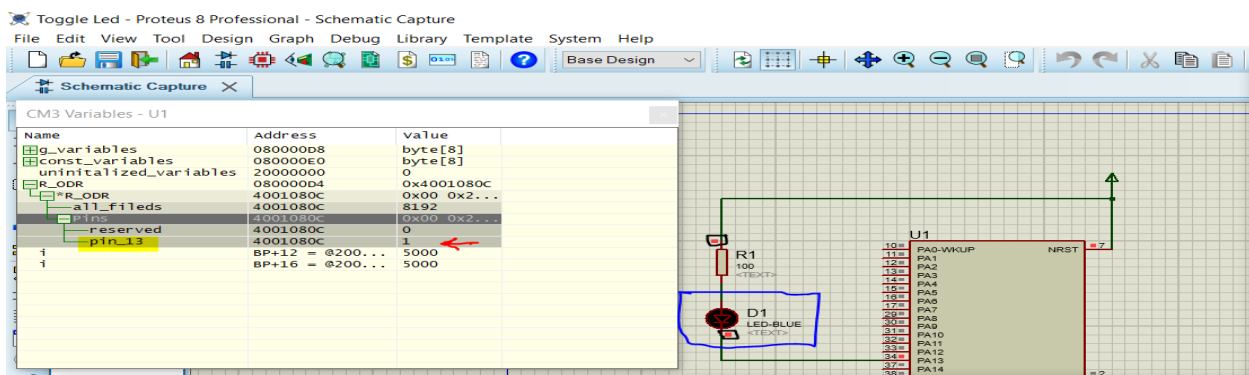
*****
Build is Done
Project Name: learn-in-depth-Cortex-M3
Copyrights: Mostafa Edrees
date: 26/4/2023
Board: STM32F103C8T6
Arm Processor: cortex-m3
*****
```

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

```
1  #*****
2  #* @file           : Makefile
3  #* @author          : Mostafa Edrees
4  #* @brief           : lab2 in lesson3 in Embedded C
5  #* @date            : 26/4/2023
6  #* @board           : STM32F103C8T6
7  #* @processor        : ARM Cortex M3
8  #* @level_debug     : -gdwarf-2 to debug on proteus
9  #*****
10
11 CC = arm-none-eabi-
12 CFLAGS = -gdwarf-2 -mcpu=cortex-m3 -mthumb
13 INCS = -I .
14 LIBS =
15 SRC = $(wildcard *.c)
16 OBJ = $(SRC:.c=.o)
17 As = $(wildcard *.s)
18 AsObj = $(As:.s=.o)
19
20 Project_Name = learn-in-depth-Cortex-M3
21 Copyrights = Mostafa Edrees
22 Board = STM32F103C8T6
23 Arm_Processor = cortex-m3
24 date = 26/4/2023
25
26 all: $(Project_Name).bin
27     @echo -e "\n*****"
28     @echo -e "\tBuild is Done"
29     @echo -e "Project Name:" $(Project_Name)
30     @echo -e "Copyrights:" $(Copyrights)
31     @echo -e "date:" $(date)
32     @echo -e "Board:" $(Board)
33     @echo -e "Arm Processor:" $(Arm_Processor)
34     @echo -e "*****\n"
35
36 %.o: %.c
37     $(CC) gcc.exe -c $(INCS) $(CFLAGS) $< -o $@
38
39 $(Project_Name).elf: $(OBJ) $(AsObj)
40     $(CC) ld.exe -T linker_script.ld $(LIBS) $(OBJ) $(AsObj) -Map=Map_file.map -o $@
41
42 $(Project_Name).bin: $(Project_Name).elf
43     $(CC) objcopy.exe -O binary $< $@
44
45 clean_all:
46     rm *.o *.elf *.bin
47
48 clean:
49     rm *.elf *.bin
50
51
52
```

startup.c

```
1  /**
2  ****
3  * @file      : startup.c
4  * @author    : Mostafa Edrees
5  * @brief     : lab2 in lesson3 in Embedded C
6  * @date      : 26/4/2023
7  * @board     : STM32F103C8T6
8  * @processor  : ARM Cortex M3
9  ****
10 */
11 //includes
12 #include "Platform_Types.h"
13
14 extern int main(void);
15 extern uint32_t _stack_top;
16
17 //Vector Handler
18 void Reset_Handler(void);
19 void NMI_Handler(void) __attribute__((weak,alias("Default_Handler")));
20 void Hard_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
21 void MM_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
22 void Bus_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
23 void Usage_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
24
25
26 vuint32_t vectors[] __attribute__((section(".vectors"))) =
27 {
28     (vuint32_t) &_stack_top,
29     (vuint32_t) &Reset_Handler,
30     (vuint32_t) &NMI_Handler,
31     (vuint32_t) &Hard_Fault_Handler,
32     (vuint32_t) &MM_Fault_Handler,
33     (vuint32_t) &Bus_Fault_Handler,
34     (vuint32_t) &Usage_Fault_Handler
35 };
36
37 extern uint32_t _E_text;
38 extern uint32_t _S_data;
39 extern uint32_t _E_data;
40 extern uint32_t _S_rodata;
41 extern uint32_t _E_rodata;
42 extern uint32_t _S_bss;
43 extern uint32_t _E_bss;
44
45
46 void Reset_Handler (void)
47 {
48     int i;
49
50     //copy .data section from flash to ram
51     uint32_t DATA_size = (uint8_t *)&_E_data - (uint8_t *)&_S_data ;
52     uint8_t *P_src = (uint8_t *)&_E_text ;
53     uint8_t *P_dst = (uint8_t *)&_S_data ;
54     for(i = 0; i < DATA_size; i++)
55     {
56         *((uint8_t *)P_dst++) = *((uint8_t *)P_src++);
57     }
58
59     //locate .bss section in ram and initialize it with zero
60     uint32_t BSS_size = (uint8_t *)&_E_bss - (uint8_t *)&_S_bss ;
61     P_dst = (uint8_t *)&_S_bss ;
62     for(i = 0; i < BSS_size; i++)
63     {
64         *((uint8_t *)P_dst++) = (uint8_t)0;
65     }
66
67     //jump to main()
68     main();
69 }
70
71 void Default_Handler (void)
72 {
73     Reset_Handler();
74 }
75
76
```

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

Toggle Led - Proteus 8 Professional - Schematic Capture

File Edit View Tool Design Graph Debug Library Template System Help

CM3 Source Code - U1

```
app.c
-----
//to create .data & .rodata & .bss sections on memory
//uint8_t g_variables[] = "mostafa"; // 8bytes
//uint8_t const_variables[] = "Edress"; // 8bytes
//uint32_t uninitialized_variables; // 4bytes

//main function
int main(void)
{
    int i;
    //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 |= 0x0F000000; //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->pin13 = 1; //turn on the led
        for(i = 0; i < 5000; i++); //delay
        R_ODR->pin13 = 0; //turn off the led
        for(j = 0; j < 5000; j++); //delay
    }

    return 0;
}
```

CM3 FLASH at 0x08000000 - U1

CM3 Variables - U1

Name	Address	Value
g_code	20000000	0x4001080C
r_odr	4001080C	0x00 0x0...
all_fileds	4001080C	0
pin13	4001080C	0x00 0x0...
reserved	4001080C	0
g_variables	20000004	byte[8]
const_variables	0800018C	byte[8]
uninitialized_variables	2000000C	0
vectors	08000000	dword[7]

CM3 RAM at 0x20000000 - U1

Stack top

D1 LED-YELLOW

PC13_RTC

PC14-OSC32_IN

PC15-OSC32_OUT

OSCIN_P00

OSCCOUT_F01

VBAT

BOOT0

STM32F103C8

Turn off the led

Toggle Led - Proteus 8 Professional - Schematic Capture

File Edit View Tool Design Graph Debug Library Template System Help

CM3 Source Code - U1

```
app.c
-----
//to create .data & .rodata & .bss sections on memory
//uint8_t g_variables[] = "mostafa"; // 8bytes
//uint8_t const_variables[] = "Edress"; // 8bytes
//uint32_t uninitialized_variables; // 4bytes

//main function
int main(void)
{
    int i;
    //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 |= 0x0F000000; //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->pin13 = 1; //turn on the led
        for(i = 0; i < 5000; i++); //delay
        R_ODR->pin13 = 0; //turn off the led
        for(j = 0; j < 5000; j++); //delay
    }

    return 0;
}
```

CM3 FLASH at 0x08000000 - U1

CM3 Variables - U1

Name	Address	Value
g_code	20000000	0x4001080C
r_odr	4001080C	0x00 0x2...
all_fileds	4001080C	8192
pin13	4001080C	1
reserved	4001080C	0
g_variables	20000004	byte[8]
const_variables	0800018C	byte[8]
uninitialized_variables	2000000C	0
vectors	08000000	dword[7]
i	BP+12 = ...	5000
j	BP+16 = ...	5000

CM3 RAM at 0x20000000 - U1

Stack top

D1 LED-YELLOW

PC13_RTC

PC14-OSC32_IN

PC15-OSC32_OUT

OSCIN_P00

OSCCOUT_F01

VBAT

BOOT0

STM32F103C8

Debug the project

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

Video

The screenshot displays the Proteus 8 Professional environment with three main windows:

- CM3 Source Code - U1:** A C program for toggling an LED. It defines variables for pin 13, sets IOPAEN, and configures GPIOA_CRLH and GPIOA_CRLH registers. The main function uses a while loop to toggle the LED by writing to R_ODR and R_ODR.
- CM3 Variables - U1:** A table showing the memory addresses and values of variables:

Name	Address	Value
R_ODR	20000000	0x4001080C
R_ODR	4001080C	0x00 0x00...
all_fileds	4001080C	0
pins	4001080C	0x00 0x00...
reserved	4001080C	0
pin_13	4001080C	0
g_variables	20000004	byte[8]
const_variables	0800018C	byte[8]
uninitialized_variables	2000000C	0
vectors	08000000	dword[7]
- CM3 RAM at 0x20000000 - U1:** A memory map showing the RAM address range from 20000000 to 200001F0. It includes a table of memory addresses and values, with a note indicating that the value of bits from bit20 to bit24 is 2.
- CM3 FLASH at 0x08000000 - U1:** A memory map showing the FLASH address range from 08000000 to 08000288. It includes a table of memory addresses and values, with a note indicating that the value of bits from bit20 to bit24 is 2.
- Circuit Diagram:** A schematic diagram of the CM3 microcontroller. It shows the connection of the LED to the GPIO pin 13 (pin_13) and the IOPAEN pin. The LED is labeled "LED-YELLOW" and is connected to the "pin_13" pin. The IOPAEN pin is connected to the "IOPAEN" pin. The diagram also shows the "R_ODR" pin connected to the "R_ODR" pin.

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
<code>.rodata</code>	<code>0x0800018c</code>	<code>0x8</code>
	<code>0x0800018c</code>	
	<code>0x0800018c</code>	<code>. = ALIGN (0x4)</code>
<code>*(.rodata*)</code>		<code>_S_rodata = .</code>
<code>.rodata</code>	<code>0x0800018c</code>	<code>0x8 app.o</code>
	<code>0x0800018c</code>	<code>const_variables</code>
	<code>0x08000194</code>	<code>. = ALIGN (0x4)</code>
	<code>0x08000194</code>	<code>_E_rodata = .</code>

.bss section

	RAM	size
<code>.bss</code>	<code>0x2000000c</code>	<code>0x4</code>
	<code>0x2000000c</code>	<code>. = ALIGN (0x4)</code>
	<code>0x2000000c</code>	<code>_S_bss = .</code>
<code>*(.bss*)</code>		
<code>.bss</code>	<code>0x2000000c</code>	<code>0x0 startup.o</code>
<code>.bss</code>	<code>0x2000000c</code>	<code>0x0 app.o</code>
	<code>0x2000000c</code>	<code>. = ALIGN (0x4)</code>
<code>*(COMMON*)</code>		
<code>COMMON</code>	<code>0x2000000c</code>	<code>0x4 app.o</code>
	<code>0x2000000c</code>	<code>uninitialized_variables</code>
	<code>0x20000010</code>	<code>. = ALIGN (0x4)</code>
	<code>0x20000010</code>	<code>_E_bss = .</code>
	<code>0x20000010</code>	<code>. = ALIGN (0x4)</code>
	<code>0x20001010</code>	<code>. = (. + 0x1000)</code>
	<code>0x20001010</code>	<code>_stack_top = .</code>

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 uninitialized_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 uninitialized_variables
080000cc W Usage_Fault_Handler
08000000 T vectors
```

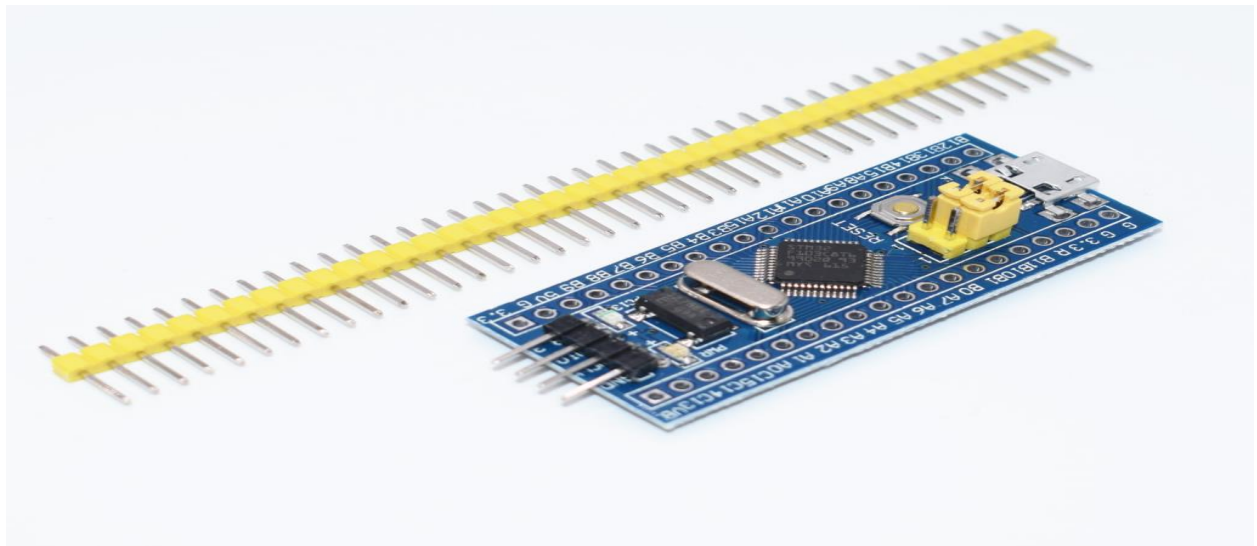
Conclusion

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