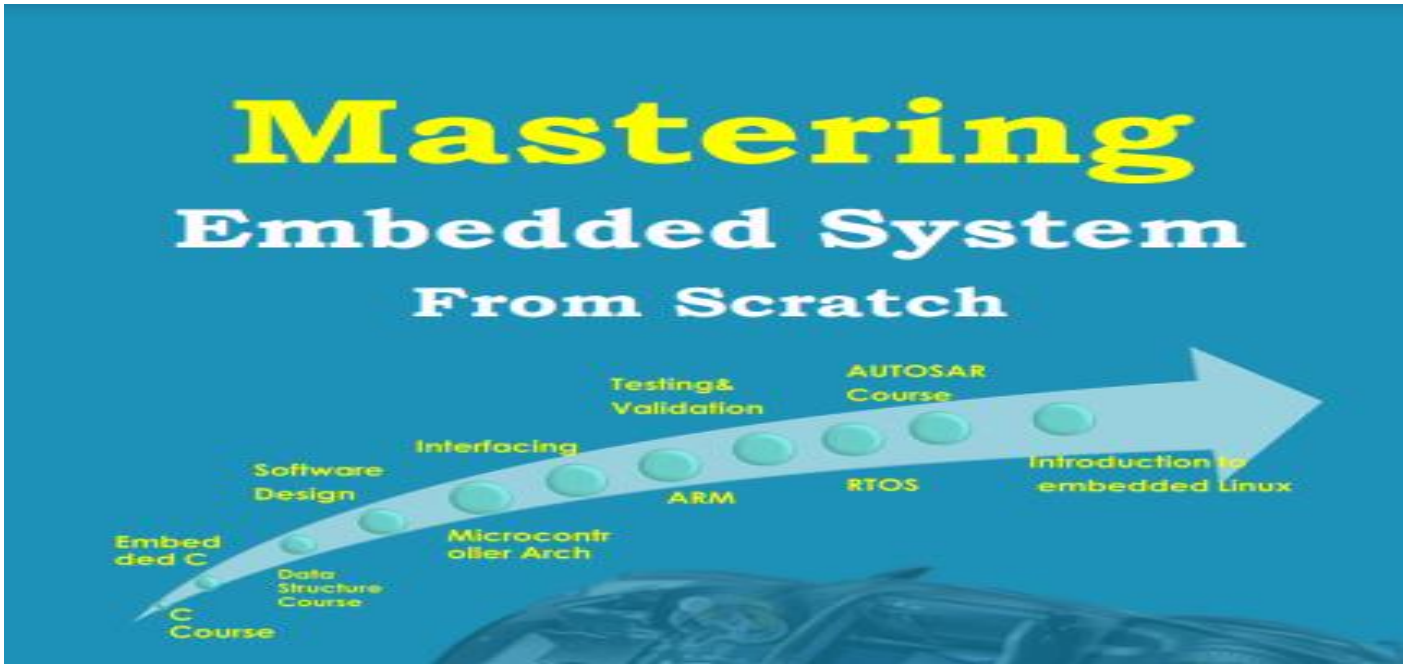
# Mastering Embedded System Diploma

## Embedded C- Lab 2: Write Bare metal SW on ARM Cortex-M3 32-bit microcontroller STM32F103C8T6 chip form scratch





**by:**

**Ehab Mohamed Abdelhamed**

**Under the supervision of:**

**Eng. Keroles Shenouda**

## Content of lab:

1) Makefile 2) C code files 3) Startup 4) Linker script 5) Simulation

## 1) Makefile

Makefile used to automate building process and minimize building time when we change some files.

**Consist of:**

(1) TARGET: (2) DEPENDENCY LIST

Command1…….

Command2……. (3) RULES

**Notes in Makefile**:

1) $<: dependencies , $@: target 2) (Generic) Rules % (%.c >> every file end with .c)
4) $(wildcard *.c) every file end with .c 5) $(var:.o=.c) every file.o >> file.c

**Dependency tree for building process**

File.bin >> File.elf >> files.o + startup.o +linker_script.ld , Files.o >> files.c ,

Startup.o >> startup.s

**Makefile Configuration**

**CC**= tool chain name (arm-none-eapi-) **CF**= compiler flags (-g -mcpu=)
**INFS**= include files (-I .<folder path>) **LIBS**=libraries object files
**SRC**=source files (files.c) **As**=assembly files (file.s)
**OBJ**=object files (file.o) **NAME**=Output file name

```
#Makefile prabared by Eng.Ehab Moahmed Abdelhamed
CC=arm-none-eabi-
CF=-g -mcpu=arm926ej-s
INCS=-I .
LIBS=
SRC= $(wildcard *.c)
As= $(wildcard *.s)
OBJ= $(SRC:.c=.o) $(As:.s=.o)
NAME=Lab1

all: $(NAME).bin
	@echo "****** Building is done *******"

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

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

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

%.o: %.s
	$(CC)as.exe $(CF) $< -o $@

clear_all:
	rm *.o *.bin *.elf

clear:
	rm *.bin *.elf
```

## 2) C code:

We will Write c code for toggle led at STM32F103C8T6 board at pin 13 of port A:
We will access registers in two modules.
(1) RCC at base address 0x40021000.

Register APB2ENR at offset 0x18, write 1 at bit 2 to enable clock for Port A

2) GPIO Port A at base address 0x40010800

1) Register CRH at offset 0x04, write num 2 at [20 -23] bits to select output pin mode
2) Register ODR at offset 0x0c, data register for port A

```c
/*******************************************************************************
 * file           : main.c
 * author         : Ehab Mohamed Abdlehamed
 * brief          : Toggle led using STM32F103C8T6 board
 ******************************************************************************/


#include "Platform_Types.h"

#define RCC_BASE      0x040021000
#define PortA_BASE    0x040010800

#define RCC_APB2ENR   *(volatile uint32*)(RCC_BASE+0x18)   //to enable clock of GPIO module
#define GPIOA_CRH     *(volatile uint32*)(PortA_BASE+0x04) //to select mode of pins of port A
#define GPIOA_ODR     *(volatile uint32*)(PortA_BASE+0x0C) //Data register for port A

typedef union{
    volatile uint32 all_port;
    struct{
        volatile uint32 rserved:13;
        volatile uint32 pin13:1;
    }pins;
}ODR_t;

ODR_t * PortA=(ODR_t *)(PortA_BASE+0x0C);

int main(void)
{
    static uint32 bss_var[3];
    uint32 i;
    /*Enable clock for gpio port A*/
    RCC_APB2ENR|=(1<<2);
    /*to select modo for pin 13 at port A as output pin*/
    GPIOA_CRH=(GPIOA_CRH&(0xff0fffff))|(0x00200000);
    /* Super loop */
    for(;;){
        /*Write 1 to turn on Led*/
        PortA->pins.pin13 = 1;
        /*delay*/
        for(i=0;i<5000;i++);
        /*Write 0 to turn off Led*/
        PortA->pins.pin13 = 0;
        for(i=0;i<5000;i++);
    }
    return 0;
}
```

## 3) Startup:

Startup is the code that runs before main function to do basic initialization of CPU and memory.
Startup code actins:
1) Disable interrupts
2) Create vector table for microcontroller.
3) Copy .data section that contains initialized data form ram to rom.
4) Reserve .bss section in ram and initialize it by zero.
5) initialize stack pointer by address of top stack.
6) Create and initialize heap (optional).
7) Enable interrupts.
8) jump to main function.

1- Startup.s

```
/*******************************************************************************
 * File Name: startup.s
 * Description: Startup file for arm cortex-m3
 * Author: Ehab Mohamed Abdelhamed
 ******************************************************************************/

 /*intialze stack pointer and create vectors table*/

.section .vectors

.word 0x20001000      //stack_top
.word _reset            // Reset_handler
.word Defualt_handler   //NMI_handlr
.word Defualt_handler   //HardFault_handler
.word Defualt_handler   //MemManage_handlr
.word Defualt_handler   //BusFault_handler
.word Defualt_handler   //Reserved
.word Defualt_handler   //SVCall_handler
.word Defualt_handler   //DebugMointor_handlr
.word Defualt_handler   //Reserved
.word Defualt_handler   //PendSV_handlr
.word Defualt_handler   //SysTick_handler
.word Defualt_handler   //WWDG_handlr
.word Defualt_handler   //PVD_handler
.word Defualt_handler   //TAMPER_handlr
.word Defualt_handler   //FLASH_handler
.word Defualt_handler   //RCC_handler
.word Defualt_handler   //ECTI0_handler
.word Defualt_handler   //ECTI1_handler
.word Defualt_handler   //ECTI2_handler

.section .text
/*for thumb instruction*/
.thumb_func

/* jump to main */
_reset:
        bl main
stop:   b stop

/*infinite loop when interrupt occur*/
Defualt_handler :
loop:   bl loop
```

## 2- Full C Startup file (Startup.c)

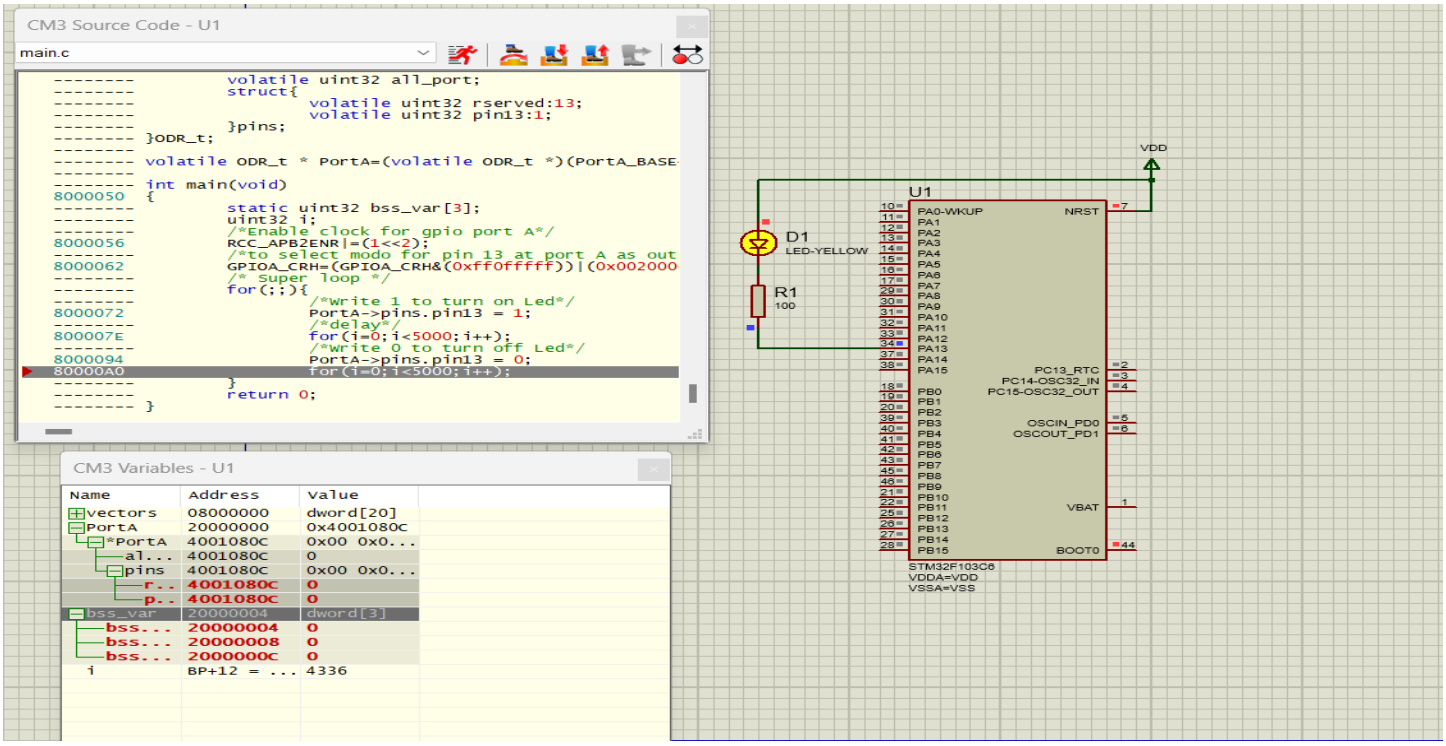
```c
/*******************************************************************************
 * File Name: startup.s
 * Description: Startup file for arm cortex-m3
 * Author: Ehab Mohamed Abdelhamed
 ******************************************************************************/
#include <Platform_Types.h>

extern uint32 _STACK_TOP;

#define START_STACK_SP _STACK_TOP
/* Enable Exceptions ... This Macro enable IRQ interrupts, by clearing the I-bit in the PRIMASK. */
#define Enable_Exceptions()     __asm("CPSIE I")

/* Disable Exceptions ... This Macro disable IRQ interrupts,by clearing the I-bit in the PRIMASK. */
#define Disable_Exceptions()    __asm("CPSID I")

int main(void);
void Defualt_handler(void);

extern uint32 _END_TEXT;
extern uint32 _START_DATA;
extern uint32 _END_DATA;
extern uint32 _START_BSS;
extern uint32 _END_BSS;



/*Create interrupt vector table*/
void _reset(void);                  // Reset_handler
void NMI_handler(void)__attribute__((weak, alias ("Defualt_handler")));           // NMI_handler
void HardFault_handler(void)__attribute__((weak, alias ("Defualt_handler")));     //HardFault_handler
void MemManage_handler(void)__attribute__((weak, alias ("Defualt_handler")));      //MemManage_handlr
void BusFault_handler(void)__attribute__((weak, alias ("Defualt_handler")));      //BusFault_handler
//Reserved
void SVCall_handler(void)__attribute__((weak, alias("Defualt_handler")));        //SVCall_handler
void DebugMointor_handler(void)__attribute__((weak, alias ("Defualt_handler"))); //DebugMointor_handler
//Reserved
void PendSV_handler(void)__attribute__((weak, alias ("Defualt_handler")));         //PendSV_handlr
void SysTick_handler(void)__attribute__((weak, alias ("Defualt_handler")));       //SysTick_handler
void WWDG_handler(void)__attribute__((weak, alias ("Defualt_handler")));          //WWDG_handlr
void PVD_handler(void)__attribute__((weak, alias ("Defualt_handler")));           //PVD_handler
void TAMPER_handler(void)__attribute__((weak, alias ("Defualt_handler")));        //TAMPER_handler
void FLASH_handler(void)__attribute__((weak, alias ("Defualt_handler")));         //FLASH_handler
void RCC_handler(void)__attribute__((weak, alias ("Defualt_handler")));           //RCC_handler
void ECTI0_handler(void)__attribute__((weak, alias ("Defualt_handler")));         //ECTI0_handler
void ECTI1_handler(void)__attribute__((weak, alias ("Defualt_handler")));         //ECTI1_handler
void ECTI2_handler(void)__attribute__((weak, alias ("Defualt_handler")));         //ECTI2_handler

/*Load Addresses of handlers in vectors section*/
const uint32 vectors[] __attribute__((section(".vectors"))) =
{
    (uint32) &START_STACK_SP,
    (uint32) &_reset,
    (uint32) &NMI_handler,
    (uint32) &HardFault_handler,
    (uint32) &MemManage_handler,
    (uint32) &BusFault_handler,
    (uint32) 0,
    (uint32) &SVCall_handler,
    (uint32) &DebugMointor_handler,
    (uint32)0,
    (uint32)&PendSV_handler,
    (uint32)&SysTick_handler,
    (uint32)&WWDG_handler,
    (uint32)&PVD_handler,
    (uint32)&TAMPER_handler,
    (uint32)&FLASH_handler,
    (uint32)&RCC_handler,
    (uint32)&ECTI0_handler,
    (uint32)&ECTI1_handler,
    (uint32)&ECTI2_handler
};

 void _reset(void){
     /* Disable interrupts */
    Disable_Exceptions();
    uint32  _SIZE_= (uint8*) &_END_DATA - (uint8*) &_START_DATA;
    uint8* _Ptr_Source=(uint8*) &_END_TEXT;
    uint8* _Ptr_Destination =(uint8*) &_START_DATA;
    /*Copy .data section that contains initialized data form ram to rom*/
    for(int i=0;i<_SIZE_;i++){
        *(uint8*)_Ptr_Destination=*(uint8*)_Ptr_Source;
        (uint8*)_Ptr_Destination++;
        (uint8*)_Ptr_Source++;
    }
    /*Reserve .bss section in ram and initialize it by zero*/
    _SIZE_= (uint32*) &_END_BSS - (uint32*) &_START_BSS;
    _Ptr_Destination =(uint8*) &_START_BSS;
    for(int i=0;i<_SIZE_;i++){
        *(uint8*)_Ptr_Destination=0;
        (uint8*)_Ptr_Destination++;
    }
    /*Enable interrupts */
    Enable_Exceptions();
    /*jump to main function.*/
    main();
    while(1){}
}

 void Defualt_handler(void){
    while(1){}
```

## 4) Linker_Script.ld

Linker script file describes the memory resources and memory map of the target microcontroller like number of memory and start address and size of each memory.
Define sections and start address of each section and define stack top address.
From linker script we can load section in specific address as reset section in entry point of CPU.

```
/*****************************************************************
 * File Name: linker_script.ld
 * Description: linker script for STM32F103C8T6 board
 * Author: Ehab Mohamed Abdelhamed
 *****************************************************************/

ENTRY(_reset)

MEMORY
{
    flash (RX) : ORIGIN = 0x08000000 , LENGTH = 128K
    sram  (RWX): ORIGIN = 0x20000000 , LENGTH = 20K
}

SECTIONS
{
    .text :
    {
        *(.vectors*)
        *(.text*)
        . = ALIGN(4);
        _END_TEXT = .;
    } > flash

    .data :
    {
        _START_DATA = .;
        *(.data);
        . = ALIGN(4);
        _END_DATA = .;
    }> sram AT> flash

    .bss :
    {
        _START_BSS = .;
        *(.bss);
        . = ALIGN(4);
        _END_BSS = .;
        _STACK_TOP = . +0x1000 ;
    }> sram

}
```

## 5) Simulation



# Binary Analysis

## #Symbols

1) main.o symbols:

```
Ehab Mohamed@LAPTOP-OQ38O3RQ MINGW64 /e/Embedded_System_KS/2-Uint3 Embedded C/Le
sson 3/Lab2
$ arm-none-eabi-nm.exe main.o
00000000 b bss_var.4123
00000000 T main
00000000 D PortA
```

2) startup. o symbols

```
Ehab Mohamed@LAPTOP-OQ38O3RQ MINGW64 /e/Embedded_System_KS/2-Uint3 Embedded C/Le
sson 3/Lab2
$ arm-none-eabi-nm.exe startup.o
         U _END_BSS
         U _END_DATA
         U _END_TEXT
00000000 T _reset
         U _STACK_TOP
         U _START_BSS
         U _START_DATA
0000008c W BusFault_handler
0000008c W DebugMointor_handler
0000008c T Defualt_handler
0000008c W ECTI0_handler
0000008c W ECTI1_handler
0000008c W ECTI2_handler
0000008c W FLASH_handler
0000008c W HardFault_handler
         U main
0000008c W MemManage_handler
0000008c W NMI_handler
0000008c W PendSV_handler
0000008c W PVD_handler
0000008c W RCC_handler
0000008c W SVCall_handler
0000008c W SysTick_handler
0000008c W TAMPER_handler
00000000 R vectors
0000008c W WWDG_handler
```

4) Ehab_lab2.elf symbols

```
Ehab Mohamed@LAPTOP-OQ38O3RQ MINGW64 /e/Embedded_System_KS/2-Uint3 Embedded C/Le
sson 3/Lab2
$ arm-none-eabi-nm.exe Ehab_Lab2.elf
20000010 B _END_BSS
20000004 D _END_DATA
08000158 T _END_TEXT
080000c4 T _reset
20001010 B _STACK_TOP
20000004 B _START_BSS
20000000 D _START_DATA
20000004 b bss_var.4123
08000150 W BusFault_handler
08000150 W DebugMointor_handler
08000150 T Defualt_handler
08000150 W ECTI0_handler
08000150 W ECTI1_handler
08000150 W ECTI2_handler
08000150 W FLASH_handler
08000150 W HardFault_handler
08000050 T main
08000150 W MemManage_handler
08000150 W NMI_handler
08000150 W PendSV_handler
20000000 D PortA
08000150 W PVD_handler
08000150 W RCC_handler
08000150 W SVCall_handler
08000150 W SysTick_handler
08000150 W TAMPER_handler
08000000 T vectors
08000150 W WWDG_handler
```

**#Sections**

**.text** Start_add=0x08000000 Size=0x158 End_add=0x08000158

**.data** Start_add=0x20000000 Size=0x04 End_add=0x20000004

**.bss** Start_add=0x20000004 Size=0x0c End_add=0x20000010

**.stack** Start_add=0x20001010 Size=0x1000 End_add=0x20000010

```
Ehab Mohamed@LAPTOP-OQ38O3RQ MINGW64 /e/Embedded_System_KS/2-Uint3 Embedded C/Lesson 3/Lab2
$ arm-none-eabi-objdump.exe -h Ehab_Lab2.elf

Ehab_Lab2.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000158  08000000  08000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  20000000  08000158  00010000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          0000000c  20000004  0800015c  00010004  2**2
                  ALLOC
  3 .debug_info   000002e3  00000000  00000000  00010004  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 000001bf  00000000  00000000  000102e7  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    0000009c  00000000  00000000  000104a6  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000040  00000000  00000000  00010542  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   000000ff  00000000  00000000  00010582  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    000001d4  00000000  00000000  00010681  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      0000007b  00000000  00000000  00010855  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000108d0  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000074  00000000  00000000  00010904  2**2
                  CONTENTS, READONLY, DEBUGGING
```