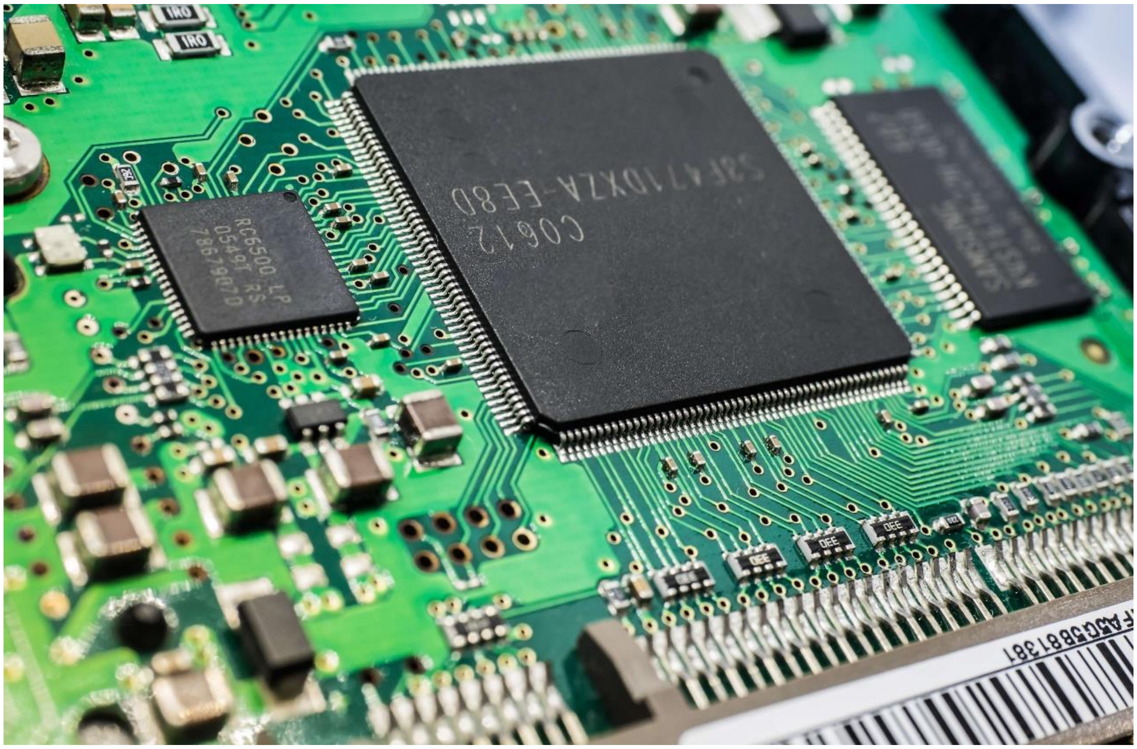
Mastering Embedded Systems
Learn in depth

Unit 3 Lesson 3

Lab 3



Submitted by: Omar Shawky Mohamed

Makefile:

This code automates the build process of any target files and the parameters can be adjusted through the variable at the start of the makefile

```
#Copyright Omar Shawky
#---------------------------------------------------------#
#Define target name
Target_Name = learn_in_depth_cortex_m3
#Define the cross-toolchain
CC=arm-none-eabi-
#Specify the C version
CSTD = -std=c99
#Define the flags for the cross-toolchain (Debugging info enabled , processor 
specified)
CFLAGS = -gdwarf-2 -mcpu=cortex-m3 -mthumb
#Define the includes
INCS = -I.
#Define the libraries
LIBS =
#---------------------------------------------------------#
#Get all .c files inside the folder
SRC = $(wildcard *.c)
#Get all .s files inside the folder
ASM = $(wildcard *.s)
#Get all .c and convert it to .o
OBJ = $(SRC:.c=.o)
#Get all .S and convert it to .o
OBJASM = $(ASM:.s=.o)
#---------------------------------------------------------#
#Build all
all: $(Target_Name).bin
    @echo "==========BUILD IS DONE=========="

#Assemble .o file from .s files
%.o: %.s
    $(CC)as.exe  $(CFLAGS)  $< -o $@

#Compile .o file from .c files
%.o: %.c
    $(CC)gcc.exe -c  $(CFLAGS) $(CSTD) $(INCS) $< -o $@

#Link all files with the linkerscript
$(Target_Name).elf:  $(OBJ) $(OBJASM)
    $(CC)ld.exe  -T  linkerscript.ld $(OBJ) $(OBJASM) $(LIBS) -o $@ -Map=Map_file
```

```

#Get the binary image of the .elf output
$(Target_Name).bin: $(Target_Name).elf
    $(CC)objcopy.exe  -O binary  $< $(Target_Name).bin
#---------------------------------------------------------#
#Clean the previous build
clean_all:
    rm *.o
    rm *.elf
    rm *.bin
clean:
    rm *.elf
    rm *.bin
```

Part1(Using Assembly for startup code)

startup.s code:

writing the startup code using assembly and defining the .vector section and the .text section for the handler implementation.

```
/* Omar Shawky startup.s code for cortex m3 */

/* SRAM begins at 0x20000000 (The next section will define the addresses for each
handler for the vector table )*/
.section .vectors            /* Defining a section called .vectors */
/*--------------------------------------------------------------------------------
--- */
.word _reset                 /* 01-    Reset        */
.word _Vector_handler        /* 02-     NMI         */
.word _Vector_handler        /* 03-  Hardware Fault */
.word _Vector_handler        /* 04-   MM Fault      */
.word _Vector_handler        /* 05-   Bus Fault     */
.word _Vector_handler        /* 06-  Usage Fault    */
.word _Vector_handler        /* 07-    RESERVED     */
.word _Vector_handler        /* 08-    RESERVED     */
.word _Vector_handler        /* 09-    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        /* 15-      IRQ_0      */
.word _Vector_handler        /* 15-      IRQ_1      */
.word _Vector_handler        /* 15-      IRQ_2      */
.word _Vector_handler        /* 15-      IRQ_3      */
.word _Vector_handler        /* 15-      IRQ_4      */
```

```
/*on to IRQ_67 */
/*--------------------------------------------------------------------------------
--- */

/*(The next section will define the symbols and the functionallity of each
handler of the vector table ) */

.section .text              /* Defining a section called .text */
/*--------------------------------------------------------------------------------
--- */
_reset:
    bl  main
    b   .

_Vector_handler:
    b   _reset
```

linkerscript:

this code is a step up from the previously made linkerscript as it has two memory segments (flash memory and the RAM).

```
/* Linker script Cortex M3
Eng. Omar Shawky */

MEMORY
{
    flash(RX): ORIGIN = 0x08000000 , LENGTH = 128K  /*Define flash memory from
address 0x08000000 with length 128K READ ONLY*/
    sram(RWX): ORIGIN = 0x20000000 , LENGTH = 20K   /*Define sram  memory from
address 0x20000000 with length 20k  READ WRITE*/
}

SECTIONS
{
    .text :{    /*output a section called .text*/
            *(.vectors*)    /*take input called .vectors section found in any .o
file */
            *(.rodata)      /*take input called .rodata  section found in any .o
file */
            *(.text*)       /*take input called .text    section found in any .o
file */
    } > flash               /*put the output section in the memory defined as
flash in both runtime and loadtime*/
```

```
    .data :{     /*output a section called .data*/
            *(.data)         /*take input called .data section found in any .o
file */
    } > sram AT> flash       /*put the output section in the memory defined as
flash loadtime then copy it to flash at runtime*/

    .bss  :{     /*output a section called .text*/
            *(.bss)          /*take input called .bss section found in any .o file
*/
    } > sram                 /*put the output section in the memory defined as
sram in both runtime and loadtime*/
}
```

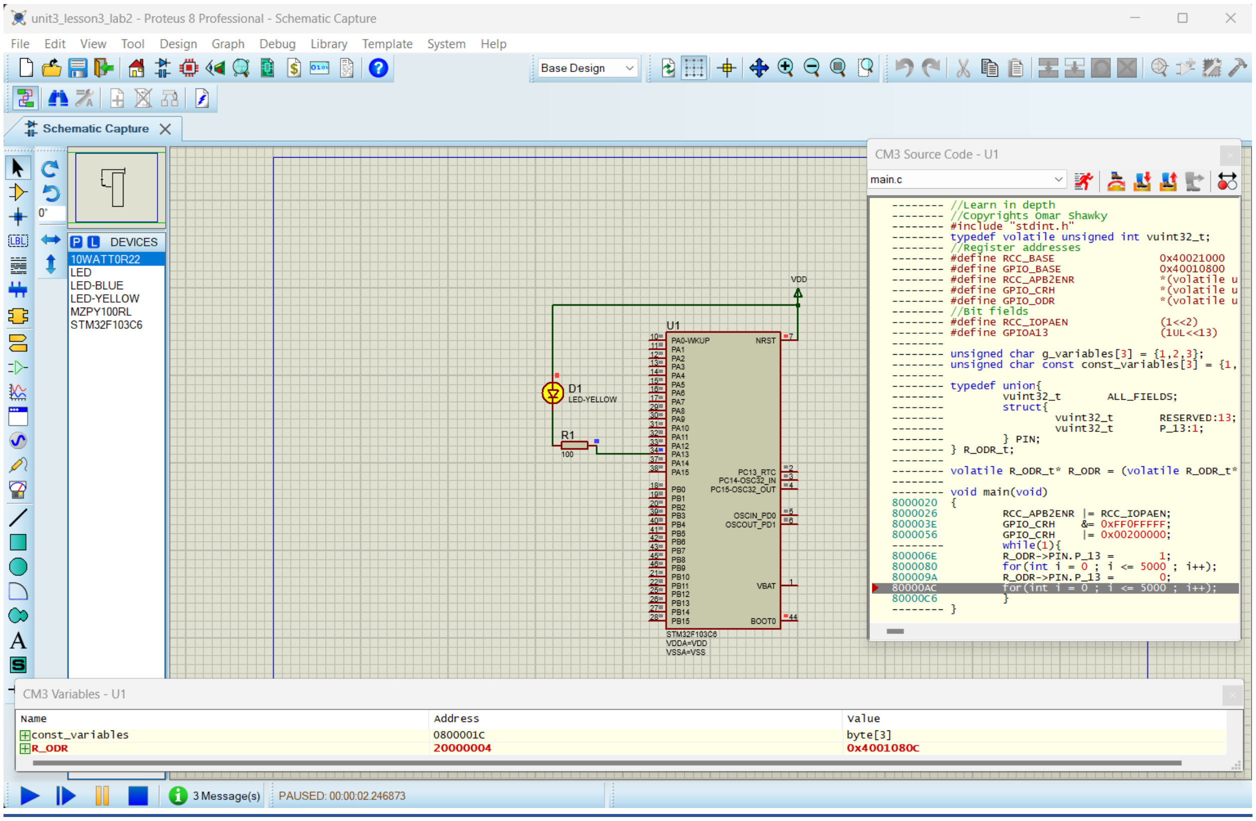
Main.c code:

this code is containing the registers for the GPIO. It initialize the GPIO peripheral and starts toggling a pin using arbitrary delay.

```
//Learn in depth
//Copyrights Omar Shawky
#include "stdint.h"
typedef volatile unsigned int vuint32_t;
//Register addresses
#define RCC_BASE        0x40021000
#define GPIO_BASE       0x40010800
#define RCC_APB2ENR     *(volatile uint32_t *)( RCC_BASE+   0x18)
#define GPIO_CRH        *(volatile uint32_t *)( GPIO_BASE+  0x04)
#define GPIO_ODR        *(volatile uint32_t *)( GPIO_BASE+  0x0C)
//Bit fields
#define RCC_IOPAEN      (1<<2)
#define GPIOA13         (1UL<<13)

unsigned char g_variables[3] = {1,2,3};
unsigned char const const_variables[3] = {1,2,3};

typedef union{
    vuint32_t   ALL_FIELDS;
    struct{
        vuint32_t   RESERVED:13;
        vuint32_t   P_13:1;
    } PIN;
} R_ODR_t;

volatile R_ODR_t* R_ODR = (volatile R_ODR_t*)(  GPIO_BASE+  0x0C);

void main(void)
```

```
{
    RCC_APB2ENR  |= RCC_IOPAEN;
    GPIO_CRH     &= 0xFF0FFFFF;
    GPIO_CRH     |= 0x00200000;
    int i;
    while(1){
    R_ODR->PIN.P_13 =   1;
    for(i = 0 ; i <= 5000 ; i++);        //Arbitrary delay
    R_ODR->PIN.P_13 =   0;
    for(i = 0 ; i <= 5000 ; i++);        //Arbitrary delay
    }
}
```

Output on proteus:



```
-------- //Learn in depth
-------- //Copyrights Omar Shawky
-------- #include "stdint.h"
-------- typedef volatile unsigned int vuint32_t;
-------- //Register addresses
-------- #define RCC_BASE                0x40021000
-------- #define GPIO_BASE               0x40010800
-------- #define RCC_APB2ENR             *(volatile u
-------- #define GPIO_CRH                *(volatile u
-------- #define GPIO_ODR                *(volatile u
-------- //Bit fields
-------- #define RCC_IOPAEN              (1<<2)
-------- #define GPIOA13                 (1UL<<13)
--------
-------- unsigned char g_variables[3] = {1,2,3};
-------- unsigned char const const_variables[3] = {1,
--------
-------- typedef union{
--------         vuint32_t       ALL_FIELDS;
--------         struct{
--------                 vuint32_t       RESERVED:13;
--------                 vuint32_t       P_13:1;
--------         } PIN;
-------- } R_ODR_t;
--------
-------- volatile R_ODR_t* R_ODR = (volatile R_ODR_t*
--------
-------- void main(void)
8000020  {
8000026          RCC_APB2ENR  |= RCC_IOPAEN;
800003E          GPIO_CRH     &= 0xFF0FFFFF;
8000056          GPIO_CRH     |= 0x00200000;
--------         while(1){
800006E          R_ODR->PIN.P_13 =       1;
8000080          for(int i = 0 ; i <= 5000 ; i++);
800009A          R_ODR->PIN.P_13 =       0;
80000AC          for(int i = 0 ; i <= 5000 ; i++);
80000C6          }
-------- }
```

Name	Address	Value
⊞const_variables	0800001C	byte[3]
⊞R_ODR	20000004	0x4001080C

Part2(Using C for startup code)

startup.c code:

- this code takes advantage of the fact that the first address that is saved on the cortex-m3 is the stack pointer this makes the c code a viable option as the stack pointer can be initialized using an array called vector.
- The rest of the vector can be filled with addresses to the handlers of different fault handler / interrupts.
- The default handler can be used as weak alias for all these handlers if a clear definition is not provided.
- Given that the code is now .c it is easier to implement the initialization of the .bss in the ram at runtime and copying the data section from the flash memory to the RAM.

```
//Startup.c
//Eng. Omar Shawky

#include <stdint.h>

extern void main(void);
extern unsigned int _STACK_TOP ;

void Reset_Handler();
void NMI_Handler()__attribute__((weak ,alias("Default_Handler")));
void H_fault_Handler()__attribute__((weak ,alias("Default_Handler")));
void MM_fault_Handler()__attribute__((weak ,alias("Default_Handler")));
void BUS_fault_Handler()__attribute__((weak ,alias("Default_Handler")));
void Usage_fault_Handler()__attribute__((weak ,alias("Default_Handler")));

uint32_t vectors[] __attribute__((section(".vectors")))={
    (uint32_t)  &_STACK_TOP,
    (uint32_t)  &Reset_Handler,
    (uint32_t)  &NMI_Handler,
    (uint32_t)  &H_fault_Handler,
    (uint32_t)  &MM_fault_Handler,
    (uint32_t)  &BUS_fault_Handler,
    (uint32_t)  &Usage_fault_Handler
};

extern uint32_t _E_TEXT ;
extern uint32_t _S_DATA ;
extern uint32_t _E_DATA ;
extern uint32_t _S_BSS ;
extern uint32_t _E_BSS ;

void Reset_Handler()
{
    //copy data from flash to ram
    uint32_t DATA_SIZE = (unsigned char*)&_E_DATA  -  (unsigned char*)&_S_DATA ;
    unsigned char * P_src  = (unsigned char*)&_E_TEXT;
    unsigned char * P_dest = (unsigned char*)&_S_DATA;
    for(int i = 0 ; i < DATA_SIZE ; i++){
        *((unsigned char*)P_dest++) = *((unsigned char*)P_src++);
    }

    //initialize .bss section in sram with zeros
    uint32_t BSS_SIZE = (unsigned char*)&_E_BSS  -  (unsigned char*)&_S_BSS ;
    P_dest = (unsigned char*)&_S_BSS;
    for(int i = 0 ; i < BSS_SIZE ; i++){
        *((unsigned char*)P_dest++) = (unsigned char) 0;
    }
    //Jump to main
    main();
}

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

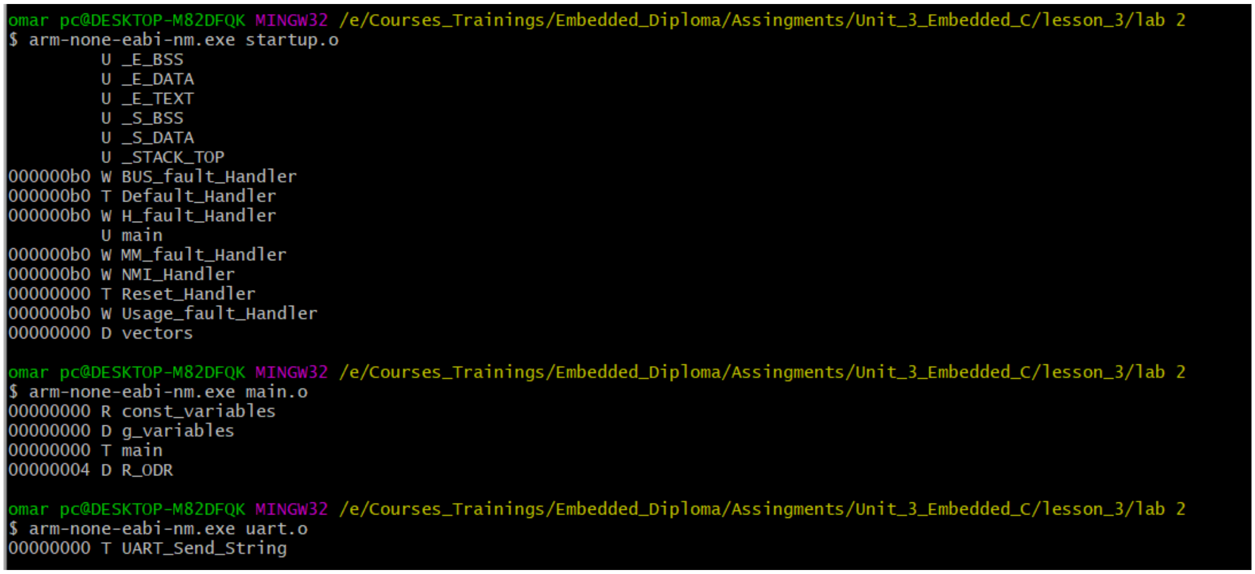
Linkerscript:

This linkerscript assign each section to the correct memory address and align the counter with 4 bytes to insure no misalignment takes place and maximizing the efficiency in the assembly instructions.

```
/* Linker script Cortex M3
Eng. Omar Shawky */

MEMORY
{
    flash(RX): ORIGIN = 0x08000000 , LENGTH = 128K  /*Define flash memory from
address 0x08000000 with length 128K READ ONLY*/
    sram(RWX): ORIGIN = 0x20000000 , LENGTH = 20K   /*Define sram  memory from
address 0x20000000 with length 20k  READ WRITE*/
}

SECTIONS
{
    .text :{    /*output a section called .text*/
            *(.vectors*)    /*take input called .vectors section found in any .o
file */
            *(.rodata)      /*take input called .rodata  section found in any .o
file */
            *(.text*)       /*take input called .text    section found in any .o
file */
            _E_TEXT = . ;   /*save the current location to the variable name
_E_TEXT  */
    } > flash               /*put the output section in the memory defined as
flash in both runtime and loadtime*/

    .data :{    /*output a section called .data*/
            _S_DATA = . ;   /*save the current location to the variable name
_S_DATA  */
            *(.data)        /*take input called .data section found in any .o
file */
            . = ALIGN(4);   /*Align the counter to a 4 byte address*/
            _E_DATA = . ;   /*save the current location to the variable name
_E_DATA  */
    } > sram AT> flash      /*put the output section in the memory defined as
flash loadtime then copy it to flash at runtime*/

    .bss  :{    /*output a section called .text*/
            _S_BSS = . ;    /*save the current location to the variable name
_S_BSS  */
            *(.bss)         /*take input called .bss section found in any .o file
*/
            _E_BSS = . ;    /*save the current location to the variable name
_E_BSS  */
            . = ALIGN(4);   /*Align the counter to a 4 byte address*/
            . = . + 0x1000; /*Increment the counter by the stack size*/
            _STACK_TOP = .;
    } > sram                /*put the output section in the memory defined as
sram in both runtime and loadtime*/
}
```

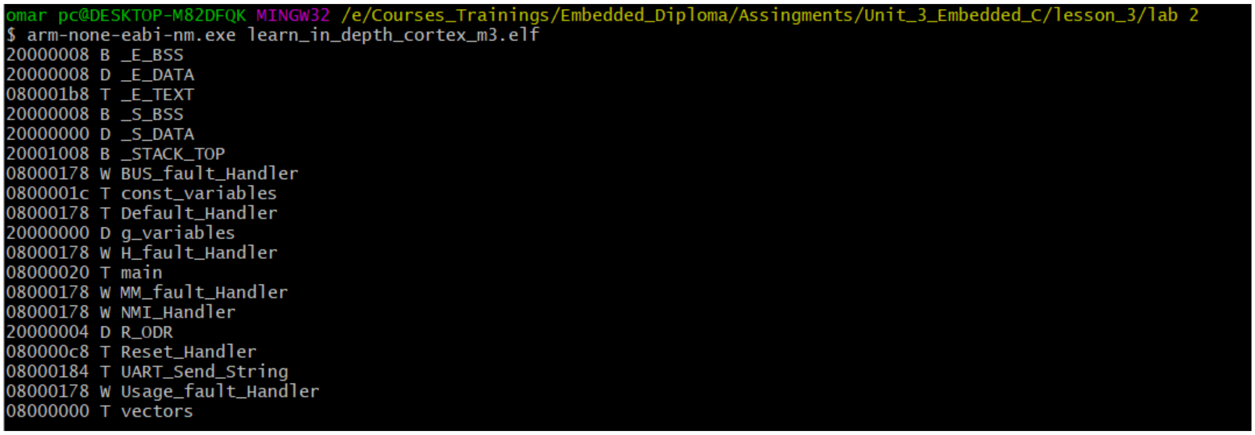
Symbols:

Before relocating



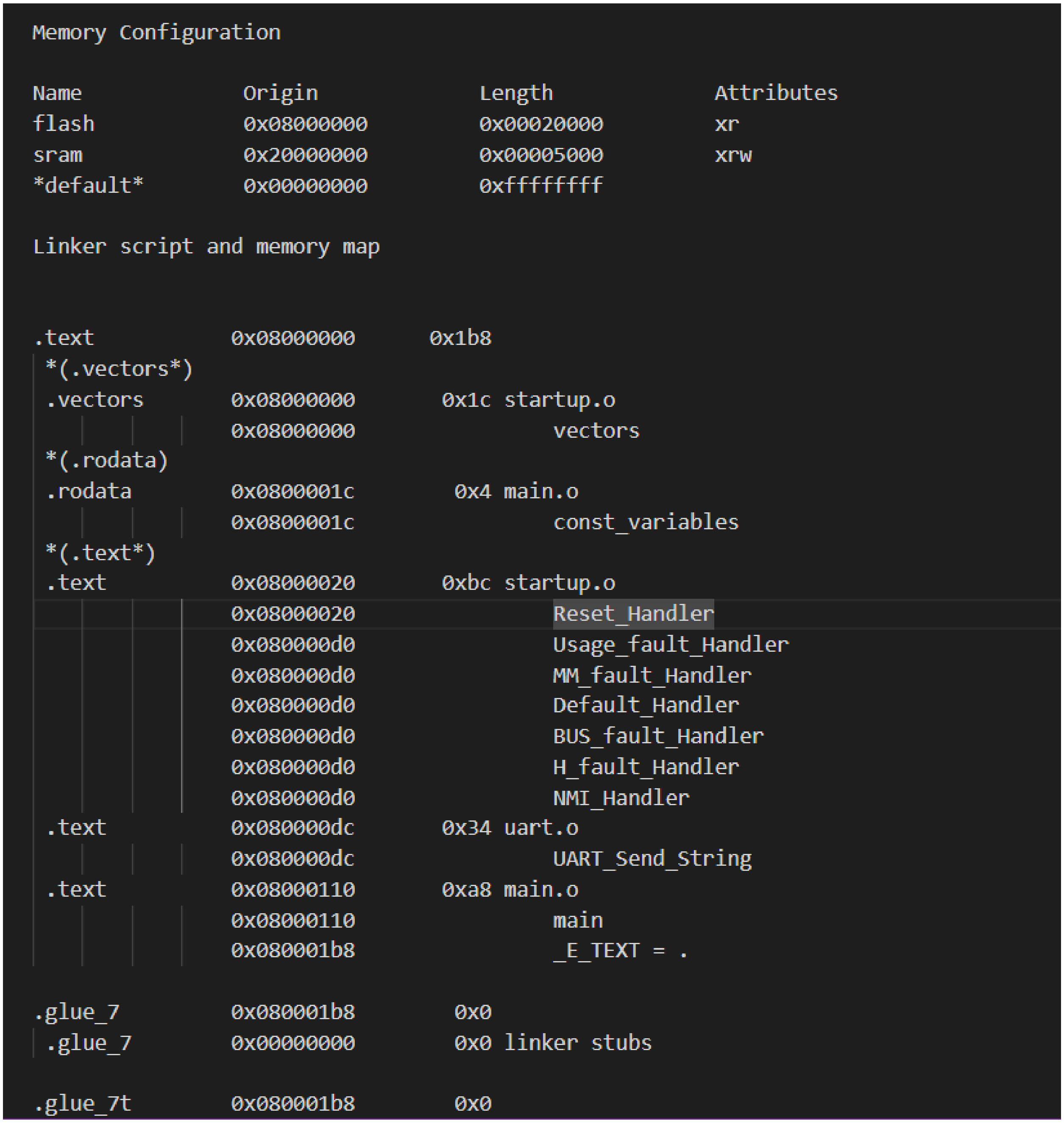
```
omar pc@DESKTOP-M82DFQK MINGW32 /e/Courses_Trainings/Embedded_Diploma/Assingments/Unit_3_Embedded_C/lesson_3/lab 2
$ arm-none-eabi-nm.exe startup.o
         U _E_BSS
         U _E_DATA
         U _E_TEXT
         U _S_BSS
         U _S_DATA
         U _STACK_TOP
000000b0 W BUS_fault_Handler
000000b0 T Default_Handler
000000b0 W H_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

omar pc@DESKTOP-M82DFQK MINGW32 /e/Courses_Trainings/Embedded_Diploma/Assingments/Unit_3_Embedded_C/lesson_3/lab 2
$ arm-none-eabi-nm.exe main.o
00000000 R const_variables
00000000 D g_variables
00000000 T main
00000004 D R_ODR

omar pc@DESKTOP-M82DFQK MINGW32 /e/Courses_Trainings/Embedded_Diploma/Assingments/Unit_3_Embedded_C/lesson_3/lab 2
$ arm-none-eabi-nm.exe uart.o
00000000 T UART_Send_String
```

After relocating



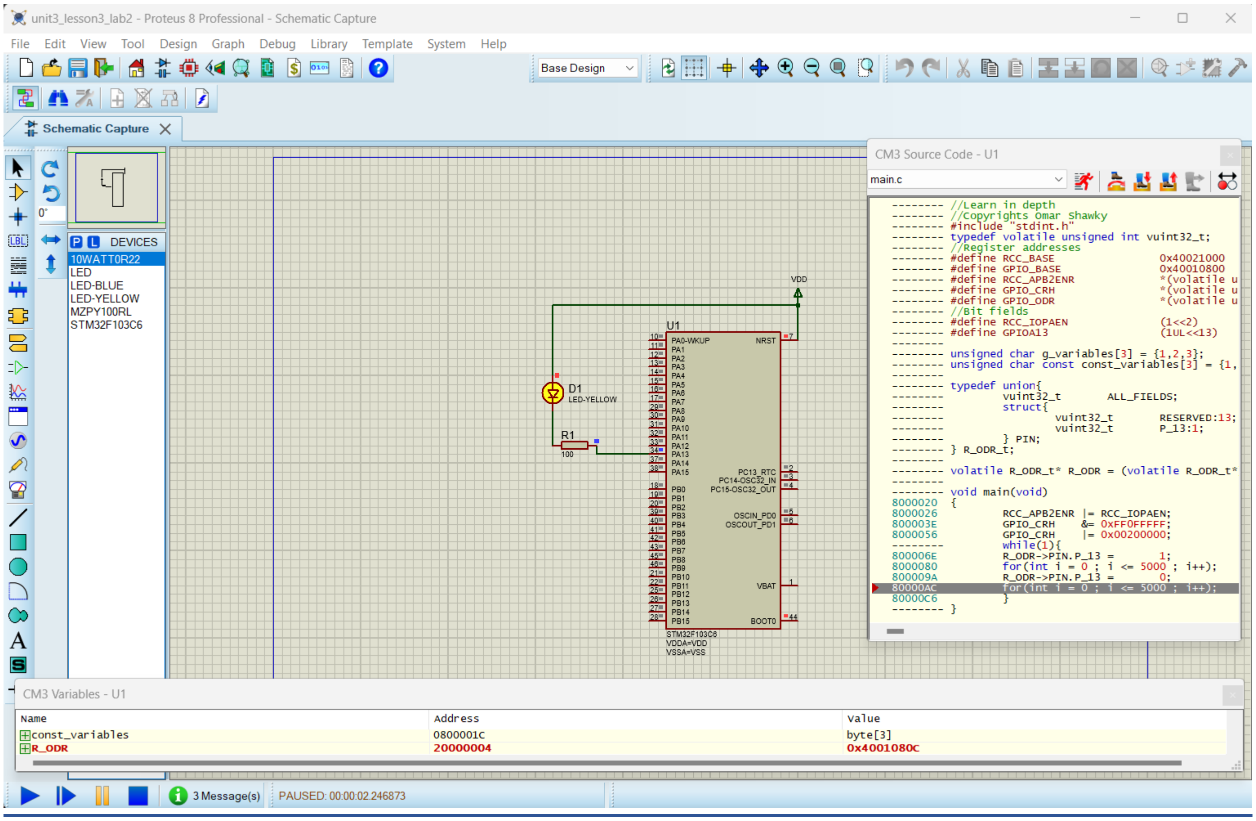
```
omar pc@DESKTOP-M82DFQK MINGW32 /e/Courses_Trainings/Embedded_Diploma/Assingments/Unit_3_Embedded_C/lesson_3/lab 2
$ arm-none-eabi-nm.exe learn_in_depth_cortex_m3.elf
20000008 B _E_BSS
20000008 D _E_DATA
080001b8 T _E_TEXT
20000008 B _S_BSS
20000000 D _S_DATA
20001008 B _STACK_TOP
08000178 W BUS_fault_Handler
0800001c T const_variables
08000178 T Default_Handler
20000000 D g_variables
08000178 W H_fault_Handler
08000020 T main
08000178 W MM_fault_Handler
08000178 W NMI_Handler
20000004 D R_ODR
080000c8 T Reset_Handler
08000184 T UART_Send_String
08000178 W Usage_fault_Handler
08000000 T vectors
```

Mapfile:



```
Memory Configuration

Name             Origin             Length             Attributes
flash            0x08000000         0x00020000         xr
sram             0x20000000         0x00005000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map


.text           0x08000000      0x1b8
 *(.vectors*)
 .vectors       0x08000000       0x1c startup.o
                0x08000000                vectors
 *(.rodata)
 .rodata        0x0800001c        0x4 main.o
                0x0800001c                const_variables
 *(.text*)
 .text          0x08000020       0xbc startup.o
                0x08000020                Reset_Handler
                0x080000d0                Usage_fault_Handler
                0x080000d0                MM_fault_Handler
                0x080000d0                Default_Handler
                0x080000d0                BUS_fault_Handler
                0x080000d0                H_fault_Handler
                0x080000d0                NMI_Handler
 .text          0x080000dc       0x34 uart.o
                0x080000dc                UART_Send_String
 .text          0x08000110       0xa8 main.o
                0x08000110                main
                0x080001b8                _E_TEXT = .

.glue_7         0x080001b8        0x0
 .glue_7        0x00000000        0x0 linker stubs

.glue_7t        0x080001b8        0x0
```

Proteus output:



```
-------- //Learn in depth
-------- //Copyrights Omar Shawky
-------- #include "stdint.h"
-------- typedef volatile unsigned int vuint32_t;
-------- //Register addresses
-------- #define RCC_BASE                 0x40021000
-------- #define GPIO_BASE                0x40010800
-------- #define RCC_APB2ENR              *(volatile u
-------- #define GPIO_CRH                 *(volatile u
-------- #define GPIO_ODR                 *(volatile u
-------- //Bit fields
-------- #define RCC_IOPAEN               (1<<2)
-------- #define GPIOA13                  (1UL<<13)
--------
-------- unsigned char g_variables[3] = {1,2,3};
-------- unsigned char const const_variables[3] = {1,
--------
-------- typedef union{
--------         vuint32_t       ALL_FIELDS;
--------         struct{
--------                 vuint32_t       RESERVED:13;
--------                 vuint32_t       P_13:1;
--------         } PIN;
-------- } R_ODR_t;
--------
-------- volatile R_ODR_t* R_ODR = (volatile R_ODR_t*
--------
-------- void main(void)
8000020  {
8000026         RCC_APB2ENR |= RCC_IOPAEN;
800003E         GPIO_CRH    &= 0xFF0FFFFF;
8000056         GPIO_CRH    |= 0x00200000;
--------         while(1){
800006E         R_ODR->PIN.P_13 =       1;
8000080         for(int i = 0 ; i <= 5000 ; i++);
800009A         R_ODR->PIN.P_13 =       0;
80000AC         for(int i = 0 ; i <= 5000 ; i++);
80000C6         }
-------- }
```

Name	Address	Value
⊞const_variables	0800001C	byte[3]
⊞R_ODR	20000004	0x4001080C

3 Message(s)   PAUSED: 00:00:02.246873